Oracle Utilities Testing Accelerator for Cloud

User Guide



Release 26.10
G62491-01
November 2026

ORACLE

Oracle Utilities Testing Accelerator for Cloud User Guide, Release 26.10

G62491-01

Copyright © 2026, Oracle and/or its affiliates.

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, software documentation, data (as defined in the Federal Acquisition Regulation), 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," "commercial computer software documentation," or "limited rights data" 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®, Java, MySQL, and NetSuite 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

1 Utilities Testing Accelerator for Cloud User Guide

2 Preface

Audience 1
Prerequisite Knowledge 1
Abbreviations 1
Related Documents 1
Updates to the Documentation 2
Documentation Accessibility 2
Conventions 3

3 Overview

Introduction 1
Terminology 1
Application Architecture 2
Application Features 3
What's New in 26.10 4
Deprecation Notice 4
Supported Oracle Utilities Applications 4

4 Oracle Utilities Testing Accelerator Features

Administration 1
Components 1
Dashboard 1
- Analytics 1
- Run List 4
- Summary Zone 5
- Notifications 5

Flows 6
Flow Sets 6

Copyright © 2026, Oracle and/or its affiliates.

Tools 6

5 Developing Metadata Driven Web Service Based Test Automation

Metadata Driven Automation Development Methodology 1
Planning 2
Design and Development 2
Test Run 3
Configuring the Automation Development Environment 3
Setting Up Flow and User Configuration Sets 3
Setting Up Application under Test 3

6 Oracle Utilities Testing Accelerator Administration

Overview 1
Administration Tab 1
Managing Products 2
Creating a Product 2
Updating a Product 3
Deleting a Product 3
Managing Modules 3
Creating a Module 4
Updating a Module 4
Deleting a Module 4
Purging Flow Run Data 4
Purging Notification Data 5
Custom Content Upgrade 5
Running the CM Content Upgrade Process for Flows 5
Clearing the Highlight Markers 7
Running the CM Content Upgrade Process for Integration Flows 8
Running the CM Content Upgrade Process for Flow Sets 9

7 Creating Components

Component Structure 1
Component Lifecycle 2
Locking/Unlocking Components 4
Component Types 4
Web Service Based Components 4
Generic REST Web Service Components 4
Creating Web Service Based Components 5
Creating a Component 5

Copyright © 2026, Oracle and/or its affiliates.

Creating a Component Definition 6
Defining Default Data at Component Level 8
Setting Up Operation Name for a Web Service 8
Using Runtime Variables in Components 9
file: prefix - csv file 9
Using Function Libraries 9
Resolving the Repeating Elements in Response XML 9
Adding Validations 10
Flow-Level Validations 10
Handling the List Elements 11
Providing Test Data 13
Extending the Base Component Definition 14
Creating Generic REST Web Service Components (Non OpenAPI Specification) 16
Creating a Generic REST Service Component Definition 16
Entering Test Data for a Generic REST Component 17
Copying Components 18

8 Creating Test Flows

Adding Products to Workspace Products 1
Creating Flow Modules 2
Creating Flows 3
Creating Flows by Dragging-and-Dropping Components 3
Adding Test Data in a Flow 4
Moving Data Between Components without Using Global Variables 12
Managing Flow Test Data Using Spreadsheets 16
Annotating Components in a Flow 18
Adding Documentation to a Flow 19
Adding Documents to a Flow 19
Deleting Documents from a Flow 20
Downloading Documents from a Flow 21
Using Global Variables 22
Using Container for Flow Variables 24
Adding Flow Variable Container to a Flow 24
Deleting Flow Variable Container from a Flow 24
Flow Lifecycle 24
Locking/Unlocking Flows 25
Copying Flows 26
Reordering Components in a Flow 26
Copying Test Data from One Component to Another in a Flow 27
Fetching Component Test Data from an Utilities Application 27
Unit Testing a Component in a Flow 28

Copyright © 2026, Oracle and/or its affiliates.

Bulk Replacing Component Test Data in Multiple Flows 29
Flow Subroutines 30
Adding Subroutines to a Flow 30
Defining Input-Output Parameters of a Subroutine 31
Running Subroutine in a Loop 33
Conditional Bypass of Components in a Flow Run (Skip Component) 36
Skipping more than one component in a flow 37
Skipping components based on outcome of a component step 37
Suspension/Pause and Conditional Resumption of Flow Run 37
Component Test Data Sets 39
Creating Reference Test Data for a Component 39
Loading Test Data from a Component Test Data Set 40
Deleting Component Test Data Sets 40
Flow Test Data Sets 40
Creating Flow Test Data Sets 41
Loading Test Data from Flow Test Data Sets 42
Deleting Flow Test Data Sets 42
Support for Integration Flows 42
Hybrid Integration Scenario 43
Oracle Utilities Cloud Services Integration Scenario 44
Running Test Flows 46
Running Test Flows 46
Iterative Flow Run 47
Stopping Flow Run on Validation Failure 48
Stopping Flow Run Manually 48
Viewing Flow Run Details 49
Viewing Flow Run Failure Details 49
Viewing Flow Run Summary Report 50
Conversational Test Data Management 50
Runtime Configuration for Flow Run 52
Creating a Flow Configuration Set 55
Creating a User Configuration Set 56
Editing a Flow Configuration Set 56
Editing a User Configuration Set 56
Copying a Flow Configuration Set 57
Copying a User Configuration Set 57

9 Creating Test Flow Sets

Creating Flow Sets 1
Adding Flows to a Flow Set 1
Deleting Flows from a Flow Set 2

Copyright © 2026, Oracle and/or its affiliates.

Running Flow Sets 2
Stopping Flow Set Run 3
Exporting Flow Sets 3
Viewing Flow Set Run History 3
Viewing Flow Set Execution Summary Report 4

10 Creating Test Plans

About Test Plans 1
Creating a Test Plan 2
Adding and Removing Flow Sets in a Test Plan 2
Managing Test Plan Lifecycle 3
Running a Test Plan 4
Viewing Test Plan Run Results 4

11 Development Accelerator Tools

Component Export Tool 1
Flow Export Tool 1
Component/ Flow Import Tool 2
    Product Pack Import 2
Component Generation Tool 3

12 Function Library Reference

OUTSPCORELIB 1
    setVariable 1
    getCurrentTimeInMilliSeconds 2
    Randomstring 2
    compare2Strings 2
    randomNumberUsingDateTime 3
    getCurrentDateTimeWithGivenDateFormat 3
    getDateDiffInSecsWithGivenDateFormat() 3
    getAdjustedTimeWithGivenDateTime 3
    getAdjustedTimeWithCurrentDateTime 4
    getAdjustedTimeWithGivenDateAndTime 4
    getAdjustedTimeWithGivenDateAndTime 5
    addDaysToCurrentDateWithGivenFormat 5
    waitForTime 5
    addDaysToAGivenDate 6
    randomNumber 6
    setVariableValueUsingListIndex 6

Copyright © 2026, Oracle and/or its affiliates.

appendStrings 7
getCurrentMonth 7
readAttachmentAsString 8
WSVALIDATELIB 8
elementListNotNull 8
elementListNull 8
validateXpathOccurrenceCount 9
elementNotNull 9
elementIsNull 9
elementValueEquals 10
elementValueNotEquals 10
elementValueGreaterThan 11
elementValueGreaterThanEqualTo 11
elementValueLesserThan 11
elementValueLesserThanEqualTo 12
elementContains 12
elementNotContains 12
reponseNotContains 13
responseContains 13
CORERESPONSEUTILLIB 13
setResponseIntoVariable 14
setVariableFromResponseList 14
setVariableFromResponseListWithFilter 14
setVariableFromResponseListWithFilters 17
getValueFromListWithIndex 17
getGroupsInIntervalFromResponse 18
getFirstGroupFromResponse 20
getFirstNGroupsFromResponse 22
getAllGroupsFromResponse 24
getLastGroupFromResponse 27
getLastNGroupsFromResponse 28
getGroupsFromResponseWithFilter 31
getResponseGroupSize 33
COREDATETIMELIB 33
getCurrentDatetimeWithGivenDateFormat 33
getFormattedDateWithGivenDate 34
getDateDiffInSecsWithGivenDateFormat 34
getAdjustedTimeWithGivenDateTime 34
getAdjustedTimeWithCurrentDateTime 35
addDaysToCurrentDateWithGivenFormat 35
getCurrentTimeinMilliSeconds 35
getEpochInGivenDateTimeFormat 35

Copyright © 2026, Oracle and/or its affiliates.

getNthDayOfCurrentMonth 36
getNthDayOfCurrentYear 36
COREDATAGENLIB 36
randomStringWithGivenRange 37
randomString 37
randomNumber 37
COREVALIDATEVARIABLELIB 37
elementListNotNull 38
elementListNull 38
validateXpathOccurrenceCount 38
elementNotNull 39
elementIsNull 39
elementValueEquals 40
elementValueNotEquals 40
elementValueGreaterThan 40
elementValueGreaterThanEqualTo 41
elementValueLesserThan 41
elementValueLesserThanEqualTo 42
elementContains 42
elementNotContains 42
variableNotContains 43
variableContains 43
COREVERIFYCONDITIONVARIABLELIB 43
elementListNotNull 44
elementListNull 44
Return Type: String (true|false) 44
elementNotNull 45
elementIsNull 45
elementValueEquals 46
elementValueNotEquals 46
elementValueGreaterThan 46
elementValueGreaterThanEqualTo 47
elementValueLesserThan 47
elementValueLesserThanEqualTo 48
elementContains 48
elementNotContains 48
variableNotContains 49
variableContains 49
CORESTOREVALUES 49
setVariable 49
appendValueToList 50
getListValueUsingIndex 50

Copyright © 2026, Oracle and/or its affiliates.

appendKeyValueToList 51
getListValueUsingKey 51
COREFILEOPS 52
readAttachmentAsString 52
CORESTRINGOPS 52
appendStrings 52
CORENUMBEROPS 53
getSumOfTwoNumbers 53
getDiffOfTwoNumbers 53
getProductOfTwoNumbers 53
getModOfTwoNumbers 54
getDivisionOfTwoNumbers 54
getMaxOfTwoNumbers 54
getMinOfTwoNumbers 55
getAbsoluteOfNumber 55
COREUTAOPS 55
waitForTime 56
customLog 56

13 Custom Libraries

Creating/Updating Custom Libraries 1
Exporting/Importing Custom Libraries 3
Using Custom Library Functions 4

14 User Management

Selecting User Time Zone 1
Selecting User Language 1
Removing User 2
Clearing Server Side Cache 2

A Web Service Component Keywords

WS-SETWEBSERVICENAME A-1
WS-SETXMLELEMENT A-1
WS-SETXMLLISTELEMENT A-2
WS-SETVARIABLE A-2
WS-SETVARIABLEFROMRESPONSE A-2
WS-SETTRANSACTIONTYPE A-3
WS-LOGMESSAGE A-3
WS-CREATEWSREQUEST A-3

Copyright © 2026, Oracle and/or its affiliates.

WS-PROCESSWSREQUEST A-3
WS-STARTPOLLWS A-4
WS-STOPPOLLWSIF A-4

B REST Component Keywords

RS-SETREQUESTHEADER B-1
RS-SETENDPOINT B-1
RS-ARGUMENT B-1
RS-SETMETHOD B-2
RS-PROCESSRESTREQUEST B-2

C Setting Up Inbound Web Services

Creating Inbound Web Services C-1
Importing Inbound Web Services C-1
Searching Inbound Web Services C-2

D Generating Re-runnable Test Data

E Configuring Authentication for Web Service Requests

F OUTA REST Services

Prerequisites F-1
Next Generation REST APIs F-1
Legacy REST APIs F-3
Flow Run F-3
Example F-4
Example F-4
Flow Set Run F-5
Example F-5
Example F-6
Flow Run Analytics F-6
Example F-7
Flow Set Run Analytics F-7
Example F-8
Flow Run Summary F-8
Example F-9
Flow Set Run Summary F-9

Copyright © 2026, Oracle and/or its affiliates.

Copyright © 2026, Oracle and/or its affiliates.

1

Utilities Testing Accelerator for Cloud User Guide

Welcome to the Oracle Utilities Testing Accelerator for Cloud User Guide . The guide explains how to use Oracle Utilities Testing Accelerator 26.10 to automate the business test flows for testing the Oracle Utilities' applications.

- Preface
- Overview
- Oracle Utilities Testing Accelerator Features
- Developing Metadata Driven Web Service Based Test Automation
- Oracle Utilities Testing Accelerator Administration
- Creating Components
- Creating Test Flows
- Creating Test Flow Sets
- Creating Test Plans
- Development Accelerator Tools
- Function Library Reference
- Custom Libraries
- User Management
- Web Service Component Keywords
- REST Component Keywords
- Setting Up Inbound Web Services
- Generating Re-runnable Test Data
- OUTA REST Services

For information about the other documentation included in this release, please refer to the Oracle Utilities Testing Accelerator for Cloud Services page on Oracle Help Center.

Have a question? Contact your Delivery Team or visit My Oracle Support.

Copyright © 2026, Oracle and/or its affiliates.

2

Preface

Welcome to the Oracle Utilities Testing Accelerator User's Guide for Cloud for release 26.10. 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 Automation Engineers who automate the business test flows for testing the Oracle Utilities' applications.

Prerequisite Knowledge

The metadata driven automation development paradigm of Oracle Utilities Testing Accelerator does not require any in-depth programming experience to develop scripts for testing. However, good understanding and working knowledge of Oracle Utilities Application Framework and its metadata based objects along with in-depth functional understanding of the application being tested, is required. The advanced programming features available in the application require experience with the programming concepts and groovy scripting language.

Abbreviations

The following terms are used in this document:

Term	Expanded Form
OUTA/UTA	Oracle Utilities Testing Accelerator

Related Documents

For more information, refer to the following Oracle resources.

Copyright © 2026, Oracle and/or its affiliates.

User and Reference Guides:

- Oracle Utilities Testing Accelerator Reference Guide for Core
- Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Customer Cloud Service
- Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Billing Cloud Service
- Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Customer Care and Billing Cloud Service
- Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Meter Solution Cloud Service
- Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Work and Asset Cloud Service
- Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Rate Cloud Service

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)

Training Material

- Training Material on Oracle Video Hub Oracle Utilities Testing Accelerator channel

Utility Reference Model Based Test Flows

The Utility Reference Model based Oracle Utilities Testing Accelerator test flows for supported Oracle Utilities cloud services, along with the necessary documentation, are available here: https://docs.oracle.com/cd/F25653_01/index.htm

Sample Data Generation Flows

The sample data generation Oracle Utilities Testing Accelerator flows for supported Oracle Utilities cloud services, along with necessary documentation, are available here: https://docs.oracle.com/cd/F42444_01/index.htm

Updates to the Documentation

Documentation updates are posted on Oracle Help Center 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

Copyright © 2026, Oracle and/or its affiliates.

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.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Copyright © 2026, Oracle and/or its affiliates.

3
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
- What's New in 26.10
- Deprecation Notice
- 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 for creating the web service based 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 avoids 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 Documentssection in the Preface.

Terminology

This table lists the different terms used in the document:

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.

Copyright © 2026, Oracle and/or its affiliates.

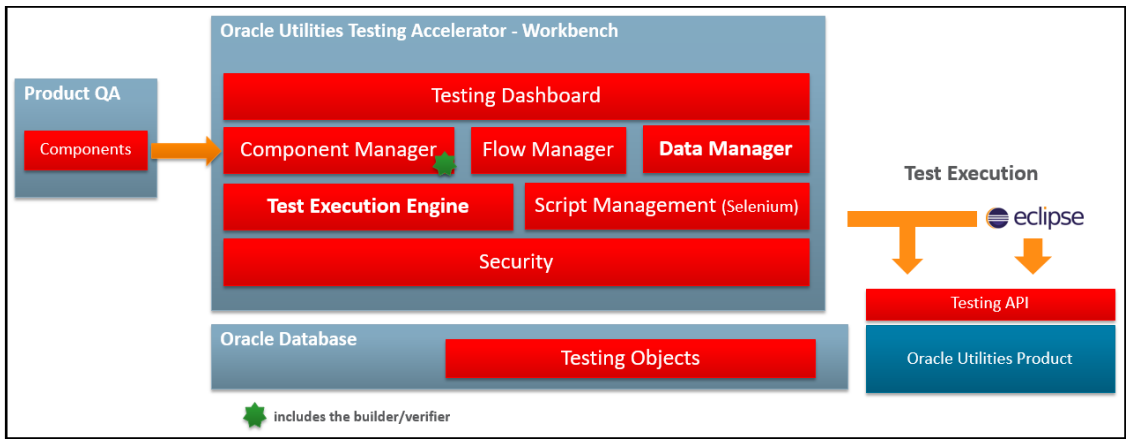
Term	Description
Flow	An automated test. A flow comprises one or more components and/or component sets that are called in a pre-determined sequence.
Flow Test Data	A test data set specific for a given flow.
All components and flows in Oracle Utilities Testing Accelerator are organized into hierarchy 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. Example: CCS A product contains one or more modules.
Module	Represents an Oracle Utilities application functional area. For example: Billing in CCS A module contains one or more components that are used to automate a specific functional area in an Oracle Utilities application. A module in the flow tree hierarchy can be used for logical grouping of custom flows for easier access. Note: The module in the hierarchy of flow and component tree structure is different.

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

Application Architecture

The following diagram depicts the high-level architecture of Oracle Utilities Testing Accelerator.

Figure 3-1 Oracle Utilities Testing Accelerator Architecture



Copyright © 2026, Oracle and/or its affiliates.

Oracle Utilities Testing Accelerator's workbench can be accessed using a web browser, such as Microsoft Edge, 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:

- **Dashboard** provides basic analytics on flow execution through time series visualizations, along with the overall status of flow development and usage.
- **Administration** provides various controls to manage logs and functions within Oracle Utilities Testing Accelerator.
- Utilities can be used to auto generate components, import-export components and flows from and to different instances of Oracle Utilities Testing Accelerator.
- **Component Manager** supports auto generation creation, update and delete of components.
- **Flow Manager** provides features to create and manage test flows in Oracle Utilities Testing Accelerator.
- **Test Planning** provides basic test planning functions to better manage test flow runs and review the overall results.

> **Note**
>
> Test Planning is designed only to provide basic test manager. It is not a test management suite.

- **Test Execution Engine** is used to run an Oracle Utilities Testing Accelerator test flow directly through the web browser.
- UTA APIs can be used to run Oracle Utilities Testing Accelerator test flows and flow sets remotely using the REST APIs. These can be used to integrate Oracle Utilities Testing Accelerator flow runs to external continuous integration/continuous delivery or test management systems.
- Testing APIs are used by Oracle Utilities Testing Accelerator flows to post messages to and receive responses from Oracle Utilities Application Framework based cloud services/ applications.
- **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.

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.

Application Features

Copyright © 2026, Oracle and/or its affiliates.

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 Oracle Utilities Testing Accelerator Features.

What's New in 26.10

The new features/enhancements in this release are:

- **New User Experience**: Oracle Utilities Testing Accelerator now provides an improved Oracle Redwood user experience that is consistent with other Oracle applications. The updated interface improves frontend performance and page-load time while retaining existing testing capabilities. If needed, you can select **Switch to Legacy UI** from the user menu at the top of the page to use the legacy user interface.
- **Selective Import of Product Packs**: During an upgrade, Oracle Utilities Testing Accelerator automatically imports the product pack only for the Oracle Utilities Cloud Service it is deployed with. The **Product Pack Import** option, available from **Tools > Import**, lets you import product packs for other supported Oracle Utilities Cloud Services as needed for integration testing.

Deprecation Notice

The following feature has been deprecated in this Oracle Utilities Testing Accelerator release:

- The legacy user interface has been deprecated. It remains available through the **Switch to Legacy UI** option, which will be removed in release 27.4.

Supported Oracle Utilities Applications

The following table lists the Oracle Utilities cloud services testing accelerator packs available as part this Oracle Utilities Testing Accelerator release.

Product	Version
Oracle Utilities Customer Cloud Service	26.10
Oracle Utilities Billing Cloud Service	26.10
Oracle Utilities Customer Care and Billing Cloud Service	26.10
Oracle Utilities Meter Solution Cloud Service	26.10
Oracle Utilities Work and Asset Cloud Service	26.10
Oracle Utilities Rate Cloud Service	26.10

Copyright © 2026, Oracle and/or its affiliates.

4

Oracle Utilities Testing Accelerator Features

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

- Administration
- Components
- Dashboard
- Flows
- Flow Sets
- Tools

Administration

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

- Create/edit release, portfolio, product and modules

Components

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

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

For more information about components, refer to Creating Components.

Dashboard

This is the **Home** page of the application. The **Dashboard** page includes the following tabs:

- Analytics
- Run List
- Summary Zone
- Notifications

Analytics

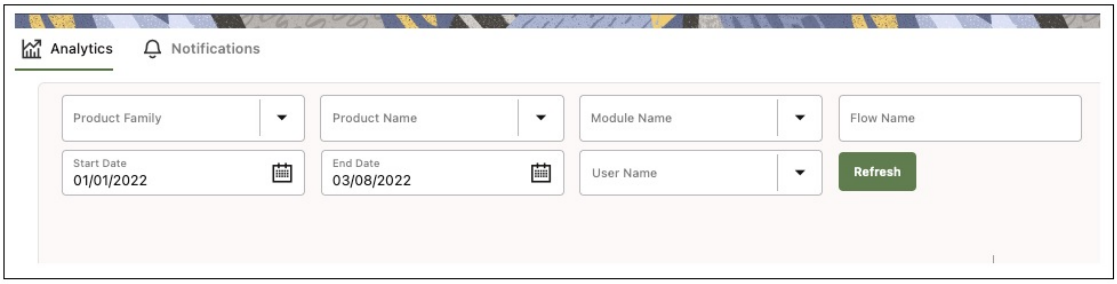
Copyright © 2026, Oracle and/or its affiliates.

This tab displays test run analytics with the ability to filter the data using several data points. The tab has three zones:

- The first zone has two visualizations that can be used to visualize the flow run data based on the filters provided at the top of the dashboard:
  - **Product Family**: Filters the visualization data to include flows that belong to the product family selected.
  - **Product Name**: Filters the visualization data based on the selected product name
  - **Module Name**: Filters the visualization data to include flows that belong to the module name selected.
  - **Flow Name**: Filters the visualization data to include flows that start or end with a set of characters.
  - **Start Date**: Filters the visualization data to include flow runs that are after the start date.
  - **End Date**: Filters the visualization data to include flow runs that are before the end date.

The following figure shows the filters applicable for visualizations in the first zone:

Figure 4-1 Dashboard visualization filters



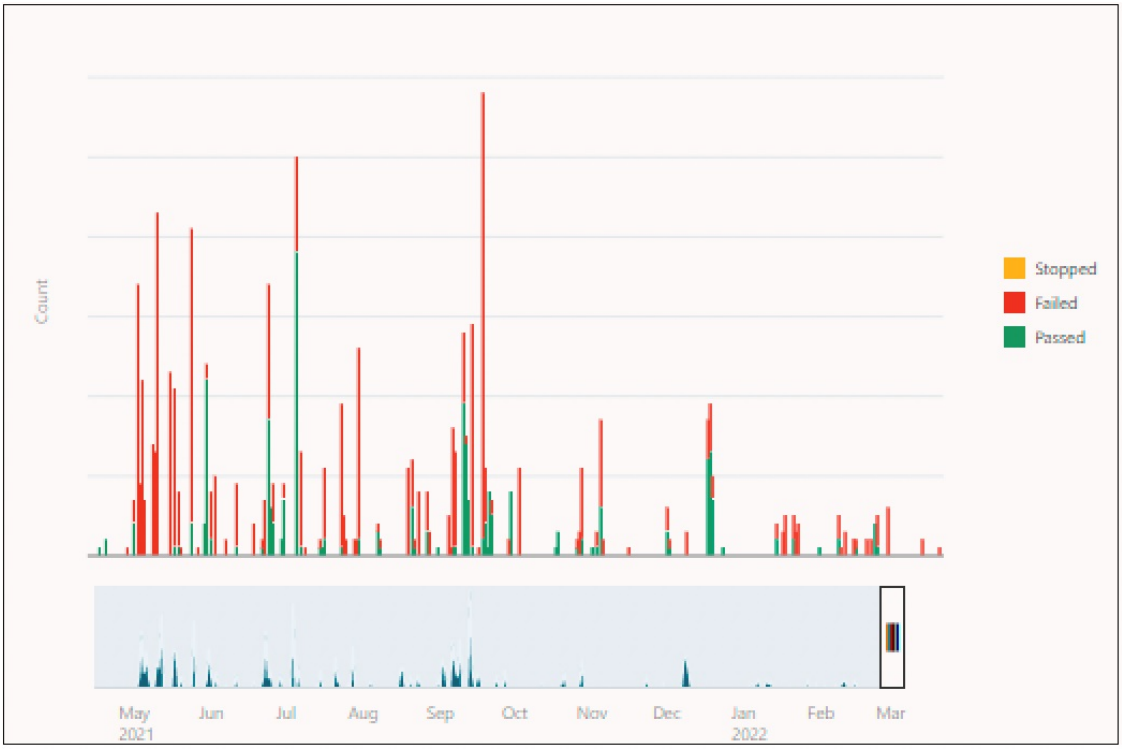
After selecting the filter criteria, click **Refresh** to refresh the visualizations in the first zone. The results are displayed based on the filtered data.

- The first visualization, **Flow Run History**, provides a view of the cumulative of test flow run results based on the filter criteria. This graph also supports rolling window based visualization, so results can be further filtered across time windows. This visualization shows the count of flow runs divided by their statuses (Stopped, Failed and Passed) subject to the filters.

The following figure shows the rolling window based visualization under the first zone in the dashboard.

Copyright © 2026, Oracle and/or its affiliates.

Figure 4-2 Flow execution results over time

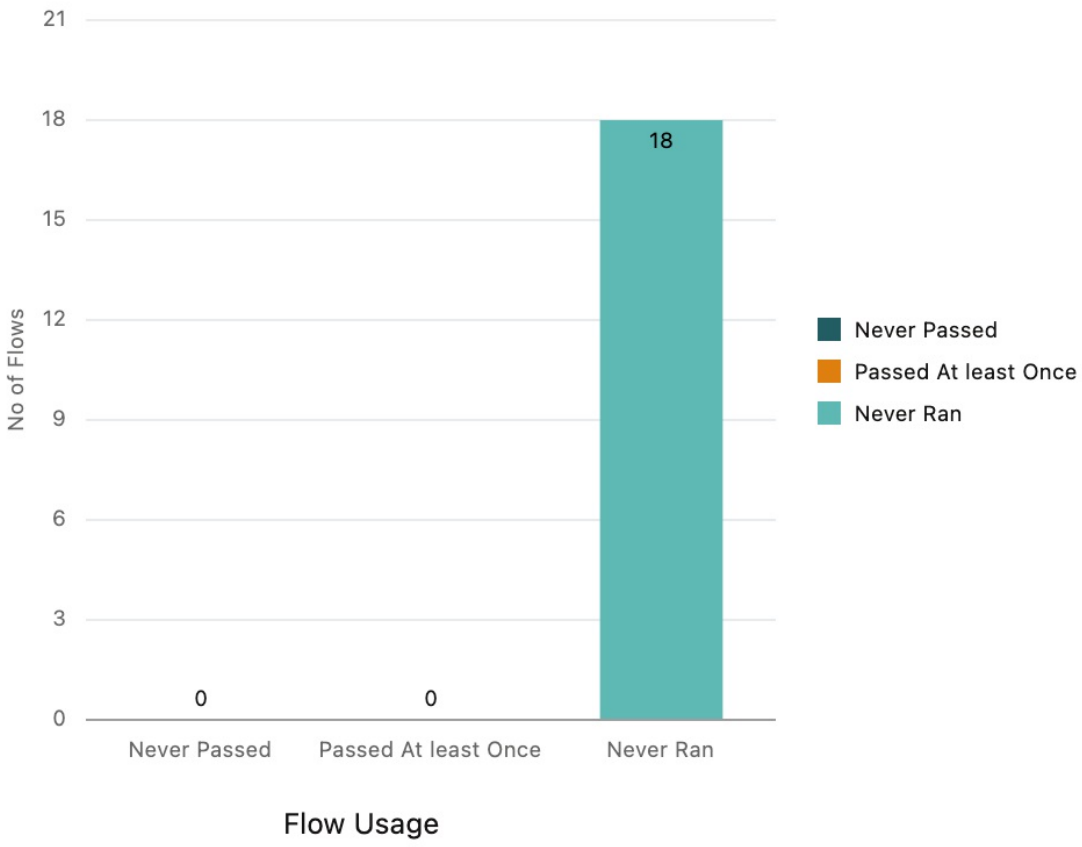


- The second visualization, **Flow Usage**, helps ascertain the count of flows between flows that were never run, flows that were run but had never passed, and flows that had passed at least once, subject to the filters.

The following figure shows the second visualization related to flow run status:

Copyright © 2026, Oracle and/or its affiliates.

Figure 4-3 Flow usage by run and pass status



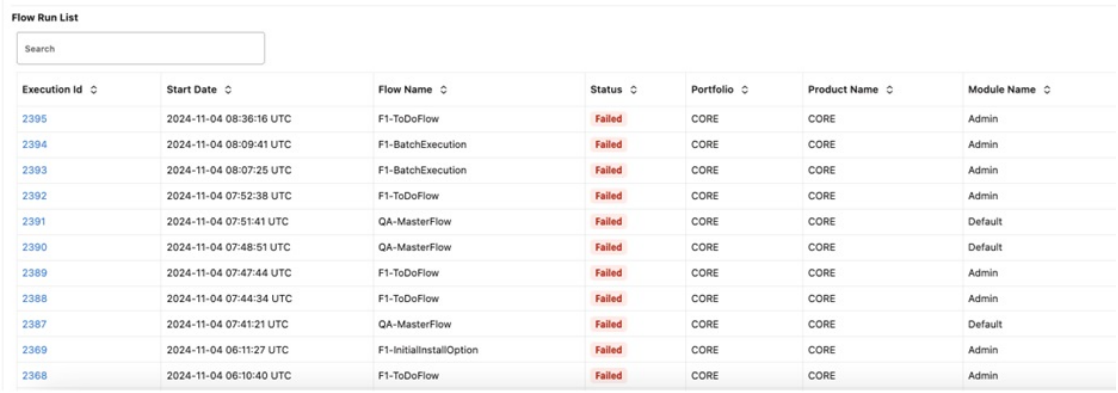
Run List

The second zone has a run list of flows that are dependent on the filters at the top of the dashboard. The run list shows the list of flow runs based on the values selected against the filters. The generic search provided at the top of the run list can be used to find appropriate results from within the displayed list of values, if required.

Click the **Execution ID** of a specific flow run to display the flow run results corresponding to that Execution ID.

Copyright © 2026, Oracle and/or its affiliates.

Figure 4-4 Flow run list



Summary Zone

The zone on the right holds the dashboard that details **Flow Status Summary** and **Component Status Summary**. The flow count displays the count of flows in the various states that are applicable to the flows. Similarly, the component count displays the count of components in various states that are applicable to the components.

> **Note**
>
> The filters provided in the dashboard are not applicable to this zone.

Notifications

The **Notifications** tab displays notifications of interest to the user currently logged in to Oracle Utilities Testing Accelerator.

From the drop-down list, you can select **Unread**, **Read**, or **All** to view the 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
  Example: Release/Portfolio/Product/Module
- Change in lifecycle state of a component/flow
  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.

Copyright © 2026, Oracle and/or its affiliates.

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

- To Do 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 To Do 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)). Select the **Read** column corresponding to the **To Do** to mark a To Do notification as 'read'.

Flows

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

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

For more details, refer to Creating Test Flows.

Flow Sets

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

- Create a new flow set
- Define/manage a flow set

For more details, refer to Creating Test Flow Sets.

Tools

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

For more details, refer to Development Accelerator Tools.

Copyright © 2026, Oracle and/or its affiliates.

5

Developing Metadata Driven Web Service Based Test Automation

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

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

- Metadata Driven Automation Development Methodology
- Configuring the Automation Development Environment

Metadata Driven Automation Development Methodology

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

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

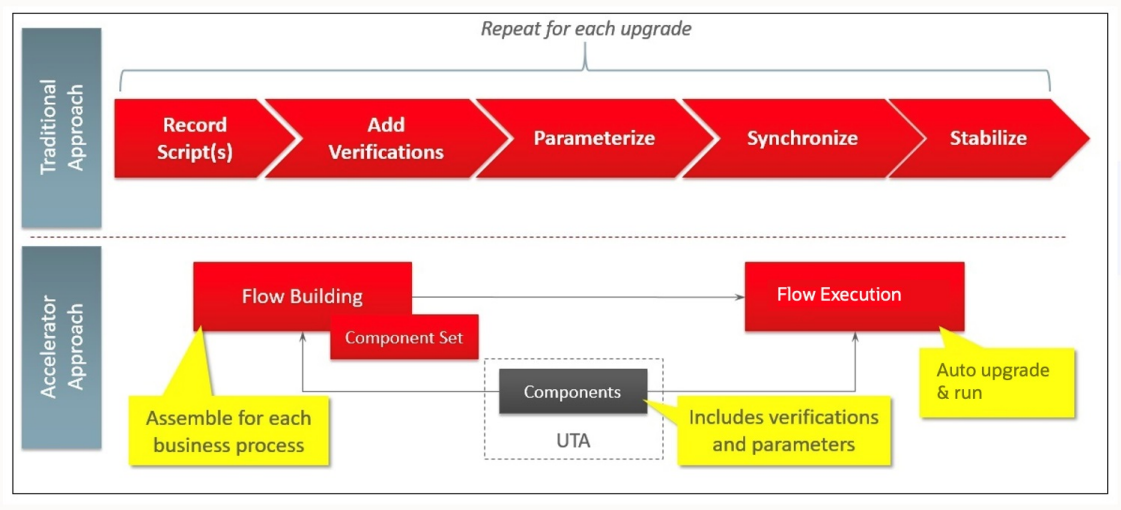
Typically, automation development is a time consuming exercise and teams have challenges in knowing and implementing the industry best practices and automation tools that work best for their product technology stack, helping them be successful in their efforts. Few of such challenges are:

- Selecting an automation tool
- Creating the automation framework
- Identifying the automation development methodology
- Ensuring the automated tests are updated for new releases
- Ensuring the coverage levels are up to date
- Configuration management of automated test programs

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

Copyright © 2026, Oracle and/or its affiliates.

Figure 5-1 Traditional and accelerator automation approaches



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 custom components or extend the delivered components.

For details about how to extend the components, refer to 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:

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 Creating Test Flows.
3. Drag and drop the required components into the flow.
4. Add the test data for the flow.

Copyright © 2026, Oracle and/or its affiliates.

The test data can be modified at the runtime using the standard Oracle Utilities Testing Accelerator databanks. For more details, refer to Creating Test Flows.

Test Run

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

For more details, refer to Running Test Flows in 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.

Configuring the Automation Development Environment

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

- Step 1: Setting Up Flow and User Configuration Sets
- Step 2: Setting Up Application under Test

Setting Up Flow and User Configuration Sets

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

Setting Up Application under Test

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

For more details about the flows and components, refer to the corresponding Oracle Utilities Testing Accelerator component reference guide for that Oracle Utilities cloud service.

Copyright © 2026, Oracle and/or its affiliates.

6

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 product and component modules
- Upgrade CM content (flows)from one version of an Oracle Utilities application to a later version.
  Example: From Oracle Utilities Customer Cloud Service 26.4 to Oracle Utilities Customer Cloud Service 26,10
- Purging old flow run logs/results
- Create and manage custom function libraries

Administration Tab

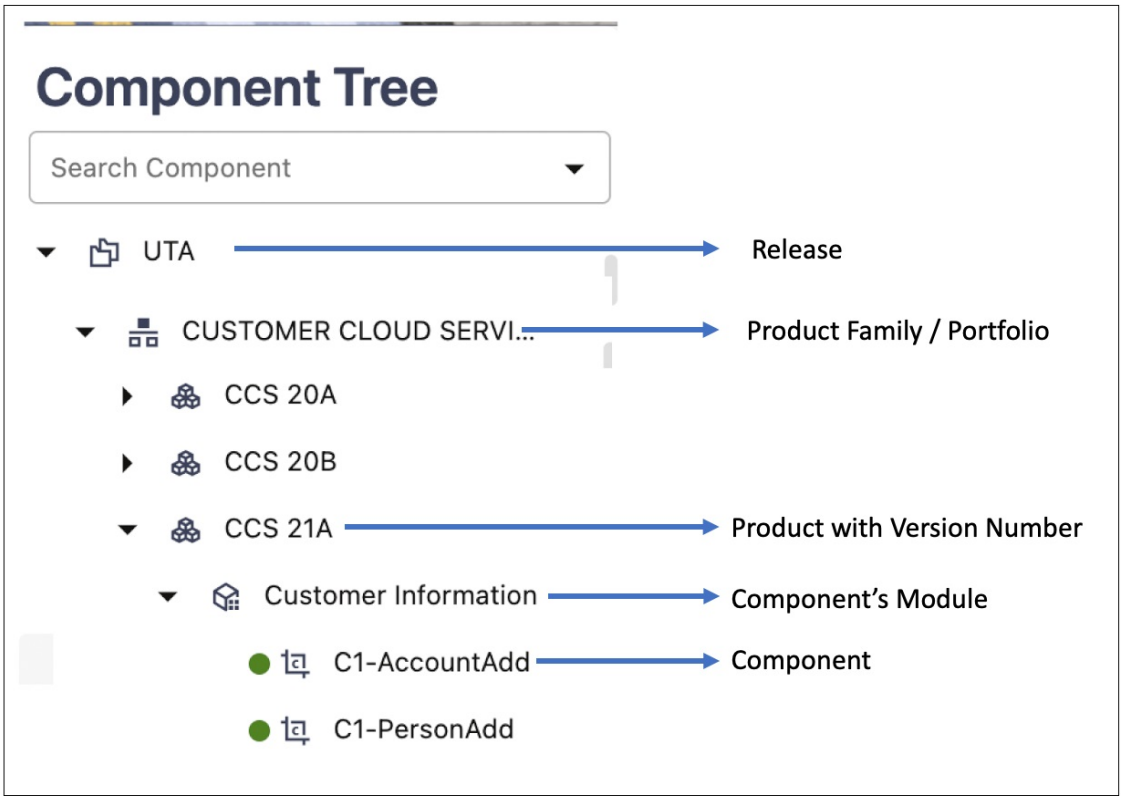
The **Administration** tab in the Oracle Utilities Testing Accelerator application allows users with Administrator role to perform the following actions:

- Managing Products
- Managing Modules
- 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.

Copyright © 2026, Oracle and/or its affiliates.

Figure 6-1 Component tree hierarchy



> **Note**
>
> Though flows are also organized under modules, flow modules are different from component modules and should be managed separately.

Managing Products

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

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.

Copyright © 2026, Oracle and/or its affiliates.

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 Cloud Service 25.10) includes flows that use components from another product (example: Oracle Utilities Meter Solution Cloud Service 25.10), to delete the Oracle Utilities Meter Solution Cloud Service, the flows in the first product (Oracle Utilities Customer Cloud Service 25.10) 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 CCS

Copyright © 2026, Oracle and/or its affiliates.

> **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).

Purging Flow Run Data

When the flow run logs and flow run history entries accumulate over time, it may impact the performance/usability. An administrator can decide to purge some of the existing flow run data

Copyright © 2026, Oracle and/or its affiliates.

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

Oracle Utilities cloud service's/application's version specific Oracle Utilities Testing Accelerator test components are released with each of the Oracle Utilities cloud service major version updates, such as 25.10, 26.4, and so on. The custom content upgrade process allows custom flows to be automatically upgraded to the latest component pack version that correspond to the latest version of Oracle Utilities cloud service/application.

Example: Flows may have been built using components from Oracle Utilities Customer Cloud Service 26.4 version. When a new version say Oracle Utilities Customer Cloud Service 26.10 is released, a corresponding set of components for 26.10 are also released as part of UTA. Using the **CM Content Upgrade** option in the administration, flows can be automatically upgraded to use the components from the latest 26.10 version instead of components from 26.4 version.

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

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 for Flows

To upgrade an existing set of flows:

1. Navigate to **CM Content Upgrade** page > **Flows** > **Administration**.
2. From the **From Product** field select the product version under which the flows exit.
3. From the **To Product** field select the product version to which the flows should be upgraded.
4. 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.
5. 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

Copyright © 2026, Oracle and/or its affiliates.

destination product that have the same name as the flows from the source product. Select either “Override” or “Skip” based on the requirement.

6. Click **Upgrade**.
   The upgrade process should run to completion with appropriate messages displayed.

   The history of CM Content Upgrade process runs are displayed in the **History** section.

   Click the **View Logs** button ( ) to display the logs corresponding to that specific CM Content Upgrade process run. The history table also shows the following information:

   - **From Product**: Product from which flows have been upgraded.
   - **To Product**: Product version to which flows have been upgraded.
   - **Tags**: List of tags provided by the user as a flow selection criteria for the upgrade process.
   - **Status**: Current status of the upgrade process (Completed, Error, In-progress)
   - **User Name**: User who has triggered the upgrade process.
   - **Start Date**: Start date and time of the upgrade process.
   - **End Date**: End date and time of the upgrade process.
   - **Logs**: To view the logs of the specific upgrade process run. The logs of the upgrade process also list the flows that have been upgraded.
     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.

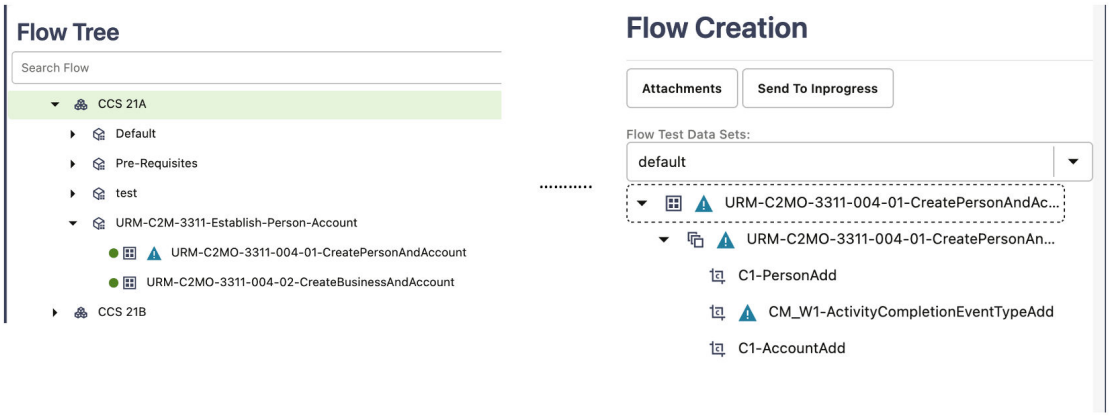
7. Click the **About** section on the top-right corner of the application and select **Clear Cache** to clear the cache after the upgrade process is complete.

> **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 to remain intact. This ensures that the flow works fine. But, if the name starts with “CM”, the upgrade process simply copies the custom component across from the source to the destination product.
> - The test data defined in the flows in the source product will remain intact in the destination product flows.

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

Copyright © 2026, Oracle and/or its affiliates.

Figure 6-2 Flows highlighted after a CM content upgrade



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 **More** > **Clear Highlight** to clear the highlight marker. Alternately, the highlight marker can be cleared for multiple flows at once at the module or product level. Right-click the module/product in the flow tree (in the left frame) and select **More** > **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**
>
> Note the following:
>
> - For a flow to be picked up by the upgrade process, the flow header should have a tag specified.
> - If a custom component is 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.

Copyright © 2026, Oracle and/or its affiliates.

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.

Before upgrading integration flows, import the product packs required for the destination product versions. During an upgrade, Oracle Utilities Testing Accelerator automatically imports the product pack for the Oracle Utilities Enterprise cloud service that it is part of. To import additional product packs used by the integration flows, navigate to **Tools > Import > Product Pack Import**.

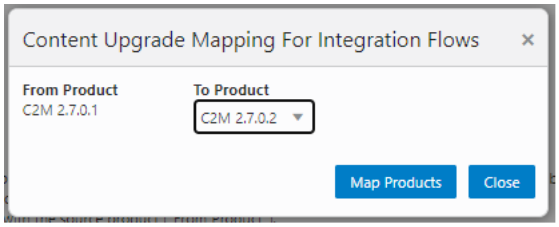
For details, refer to the Product Pack Import section in Development Accelerator Tools.

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 for Flows 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.

Figure 6-3 Product mapping for integration flow upgrades



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.

Example: If one or more flows are created in CCS 26.4 and they use components from WACS 26.4 along with CCS 26.4 components, then during the course of the CM content upgrade process, the product mapping screen will be displayed with the source product as WACS 26.4 and the destination product containing a list of available product packs in UTA, such as WACS 26.4/WACS 26.10, and so on. To proceed with the upgrade, the appropriate destination product needs to be selected. If WACS 26.10 is selected as the destination product for WACS 26.4 components, in this example, then the upgrade process will upgrade the flows from CCS 26.4 to CCS 26.10 and will also upgrade all the WACS 26.4 components being used in the upgraded CCS 26.10 flows to WACS 26.10.

Copyright © 2026, Oracle and/or its affiliates.

Running the CM Content Upgrade Process for Flow Sets

The automated upgrade of flow sets in the testing tool saves significant time and effort by eliminating manual recreation of flow sets during Oracle Energy and Water Cloud Service upgrades. By automating this process, users can quickly and efficiently manage their upgrades to increase productivity. This results in faster testing and improves overall efficiency.

The automated upgrade of flow sets is a feature that streamlines the testing process for Oracle Energy and Water Cloud Service upgrades. By specifying the desired flow set, users can initiate an automated upgrade process, eliminating the need to manually recreate the flow set with each upgrade.

To upgrade existing flow sets:

1. Navigate to the **Administration** tab.
2. In the **CM Content Upgrade** section, click the **Flows Sets** tab.
3. From the **Flow Set** drop-down list, select the flow set to be upgraded.
4. Click **Upgrade**.
5. Provide a name for the new flow set to be created.
6. If the flow set has flows from multiple product versions, then a corresponding source target mapping needs to be defined for each product version. Else, this mapping is not needed.
7. Click **Upgrade**.
   The upgrade process should run to completion displaying appropriate messages. The history of CM Content Upgrade process runs are displayed in the **History** section.
8. Click **View Logs** to display the logs corresponding to that specific CM Content Upgrade process run. The history table also shows the following information:
   - **Flow Set Name**: Name of the flow set which has been upgraded.
   - **New Flow Set Name**: Name of the upgraded flow set.
   - **Status**: Current status of the upgrade process (Completed, Failed, In- progress).
   - **Initiated By User**: User who triggered the upgrade process.
   - **Start Date Time**: Start date and time of the upgrade process.
   - **End Date Time**: End date and time of the upgrade process.
   - **Logs**: To view the logs of the specific upgrade process run. The logs of the upgrade process also list the flows in the have been upgraded.

> **Note**
>
> - Before a Flow Set upgrade can be run, ensure that all the flows under that flow set have already been upgraded to the target version/versions.
> - Flow Set upgrade creates a new flow set and maps this new flow set to flow that have already been upgraded. This process does not upgrade the flows.

Copyright © 2026, Oracle and/or its affiliates.

7

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.

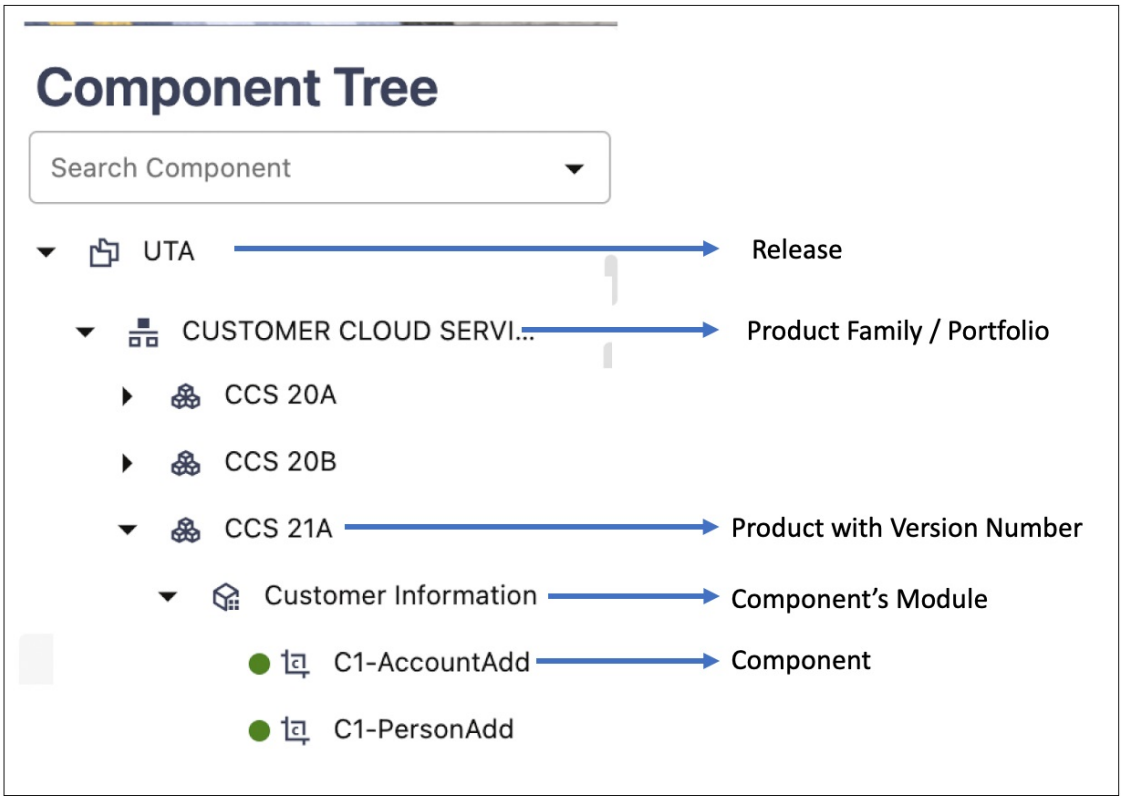
- Component Structure
- Component Lifecycle
- Component Types
- Creating Web Service Based Components
- Creating Generic REST Web Service Components (Non OpenAPI Specification)
- Copying Components

Component Structure

The following figure shows the high-level component structure.

Copyright © 2026, Oracle and/or its affiliates.

Figure 7-1 Component Tree Hierarchy

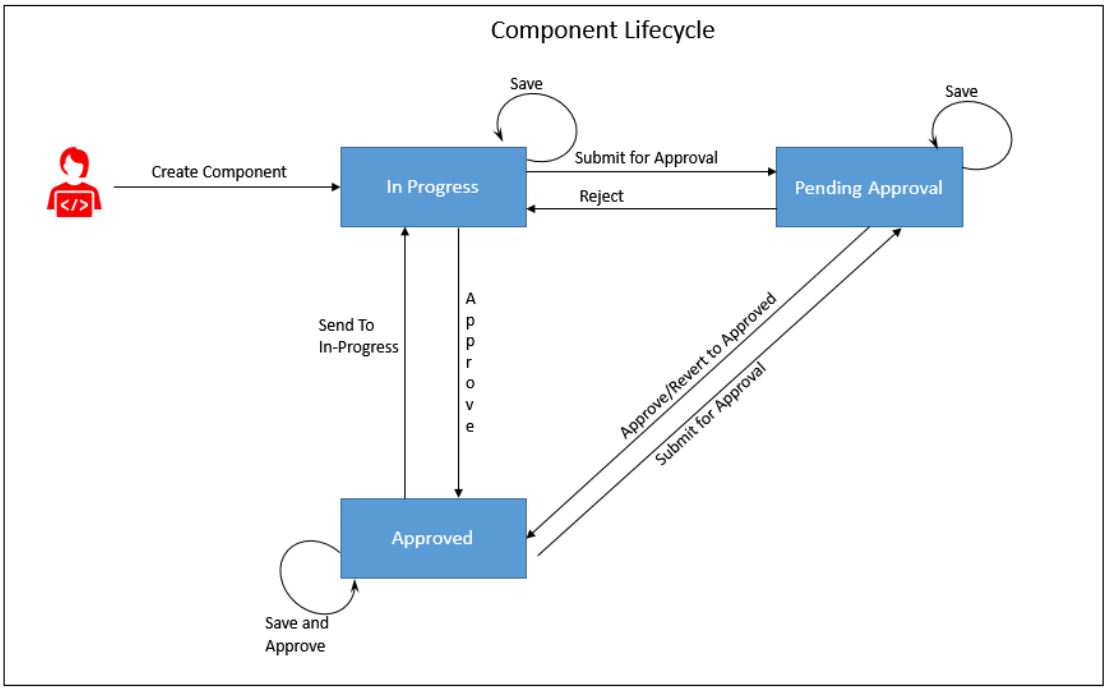


Component Lifecycle

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.

Copyright © 2026, Oracle and/or its affiliates.

Figure 7-2 Component lifecycle states



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

Copyright © 2026, Oracle and/or its affiliates.

Component Lifecycle State	Permitted Actions	Role	Resultant State (after action)
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 Lock  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

Make sure the component is created under the required hierarchy level.

Oracle Utilities Testing Accelerator supports the following types of components:

- Web Service Based Components
- Generic REST Web Service Components

Web Service Based Components

A web service based component corresponds to an Inbound Web Service in Oracle Utilities Customer Cloud Service.

The component type is defined as “WS” and the keywords used in defining it are specific to a web service request. Web service based components can be used for SOAP based services and REST based services that support OpenAPI specifications.

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

Generic REST Web Service Components

Copyright © 2026, Oracle and/or its affiliates.

A Generic REST web service component corresponds to a non-OpenAPI specification based REST end point. The component type is defined as “Web Services” and the keywords used in defining the component are specific to a REST web service.

For information about REST-specific keywords, refer to REST Component Keywords.

Creating Web Service Based Components

The web service component creation process for SOAP based and OpenAPI supported REST endpoints is the same. Flow run process in Oracle Utilities Testing Accelerator determines if the request for a component is SOAP based or REST based depending on the settings provided in the User/Flow configuration sets.

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 9: 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
- Handling the List Elements
- Extending the Base Component Definition

Creating a Component

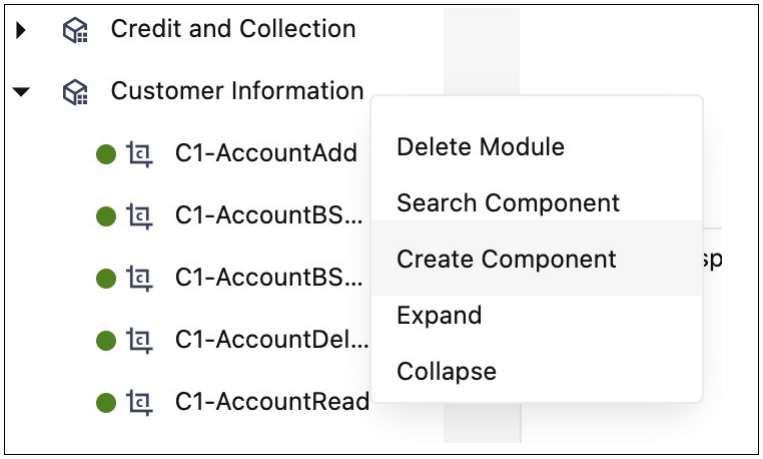
To create a web service based component manually:

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.

Copyright © 2026, Oracle and/or its affiliates.

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

Figure 7-3 Create Component Menu Action



3. Select **Create Component**.

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

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

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

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

Creating a Component Definition

A component consists of several component definition lines. Each component definition line comprises a keyword, object, display name, attribute values, default data, function name, and output parameters. All keywords for this type of component start with ‘WS’.

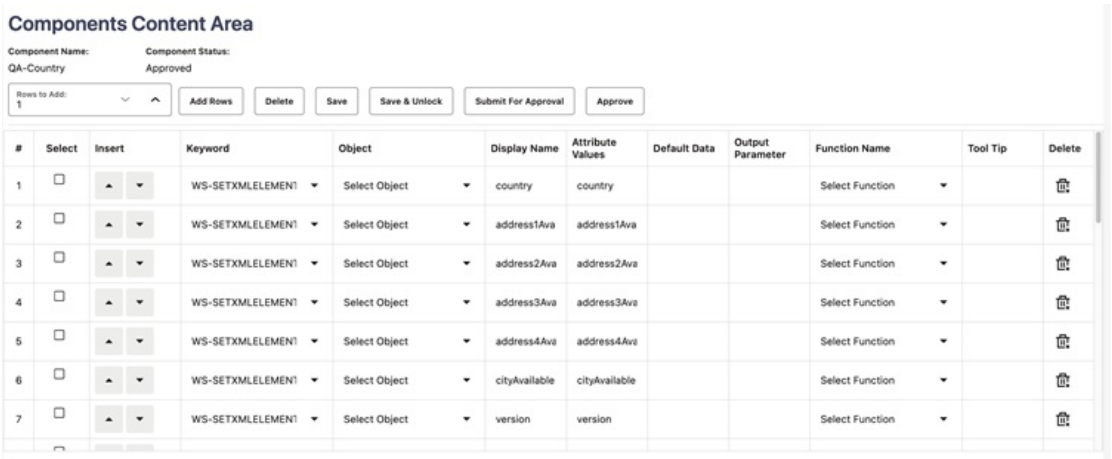
Copyright © 2026, Oracle and/or its affiliates.

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.

Figure 7-4 Component definition



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.

Copyright © 2026, Oracle and/or its affiliates.

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-Account component in Oracle Utilities Customer Cloud Service. This component maps to the C1-Account business object. It includes elements, such as:

```
<accountId/>
<billCycle/>
```

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 some of the test data can be 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 test 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 selected into the test data field by selecting from the drop down option available for each of the test data entry fields 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

Copyright © 2026, Oracle and/or its affiliates.

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.

Using Runtime Variables in Components

In some cases, few elements from the component run's response may have to be passed as inputs to another component's request XML. This can be achieved using the "Moving Data Between Components without Using Global Variables" feature. Or another option is to store the output of first component in the global variable by using the FUNCTIONCALL keyword along with the appropriate function in the provided function library.

An example would be the library rSVALIDATELIB and the function getElementValue. This function requires Xpath of the response element whose value is to be stored. It should be specified in the **Attribute Values** column. The global variable which holds this value in the script is defined in the **Output Parameter** column. This method of passing data between components allows for a single global variable to be used as input in multiple component's test data, where ever it is applicable.

file: prefix - csv file

Any test data value containing ".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 Creating Test Flows.

For a list of libraries and functions available in Oracle Utilities Testing Accelerator, refer to 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>
```


Copyright © 2026, Oracle and/or its affiliates.

```
</ContactDetails>
```

1. If building a custom component, you can 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”.

   Or, you can use the FUNCTIONCALL keyword and use appropriate functions from the base delivered function libraries, such as coreResponseUtillib library.
2. If trying to retrieve a value from the response XML for a component in a flow, in the post validations sections, use the FUNCTIONCALL keyword to call the appropriate function available in the coreResponseUtillib libraries.

For more information, refer to Function Library Reference.

Adding Validations

The different ways in which you can add validations are:

- Using the FUNCTIONCALL keyword in the component definition
  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 Function Library Reference.
- Using flow-level validations
  Validations can be added before and after a component step in a flow. The same flow can be reused with different or no validations before (pre-level validations) and after (post-level validations). The pre-validations can be used to determine if that component step needs to be run or skipped as part of the flow run, while the post validations can be used for validating the component step results.

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

Flow-Level Validations

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.

Copyright © 2026, Oracle and/or its affiliates.

> **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.

Refer to Adding Test Data in a Flow in Creating Test Flows for more information.

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.

Copyright © 2026, Oracle and/or its affiliates.

Figure 7-5 Nested XML list elements

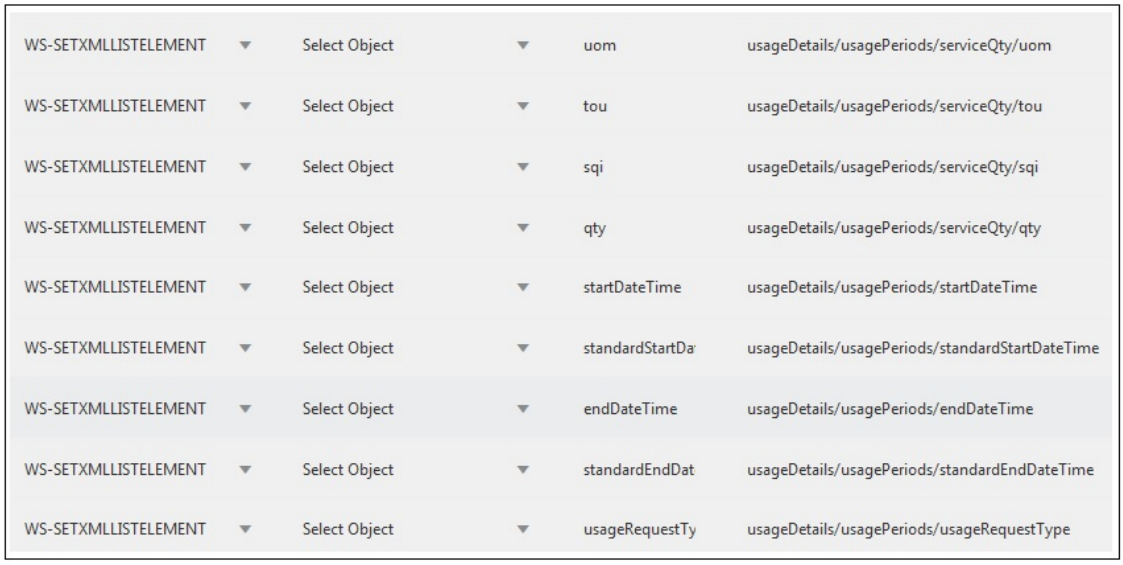


```
<usageDetails>
    <usagePeriods>
        <serviceQty>
            <seq>1</seq>
            <uom>TH</uom>
            <tou>ON</tou>
            <sqi>LOSSADJ</sqi>
            <qty>103.772922</qty>
        </serviceQty>
        <serviceQty>
            <seq>2</seq>
            <uom>TH</uom>
            <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>
```


Copyright © 2026, Oracle and/or its affiliates.

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.

Figure 7-6 XML list element mappings



> **Note**
>
> If any of the list nodes repeat (serviceQty occurrs thrice inside usagePeriods, which in turn occurrs twice in usageDetais), do not define the elements multiple times in the component definition. The number of occurrences can be controlled in the test data (as defined in the 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 List** button next to them under the **Action** column 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 List** next to the desired element under the **Action** column.

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

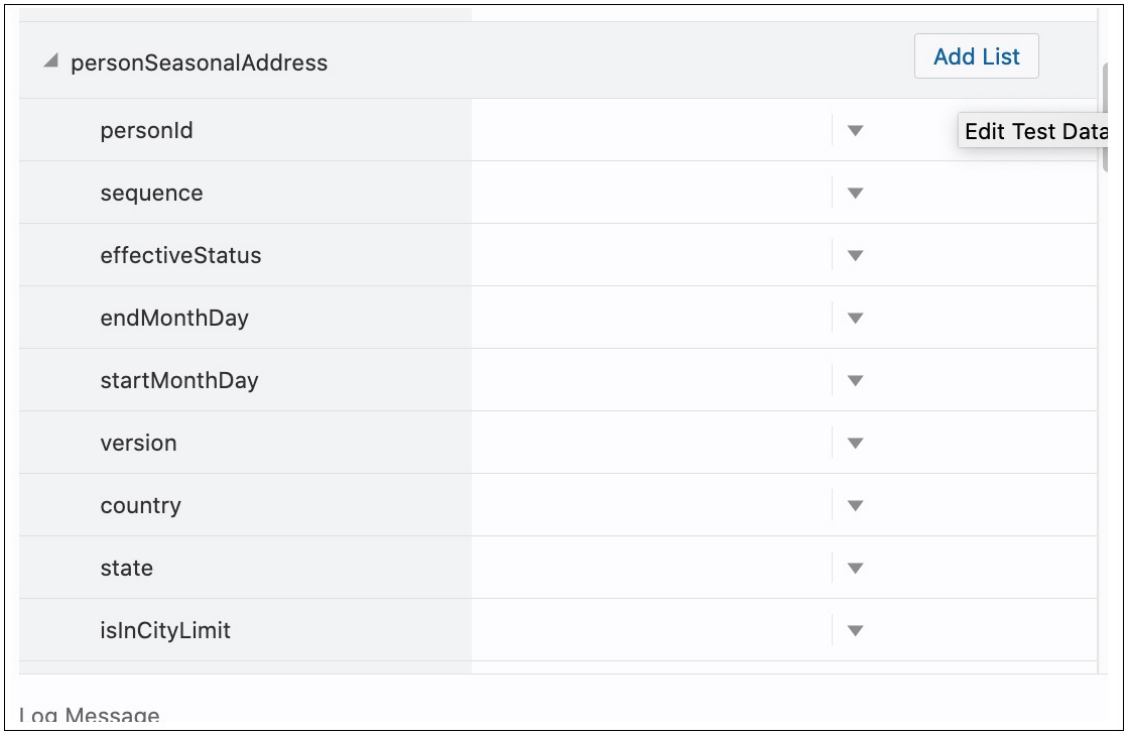
To view three serviceQty nodes in the first usagePeriods node and four in the second one:

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

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

Copyright © 2026, Oracle and/or its affiliates.

Figure 7-7 Adding List Entries to Component Test Data



Extending the Base Component Definition

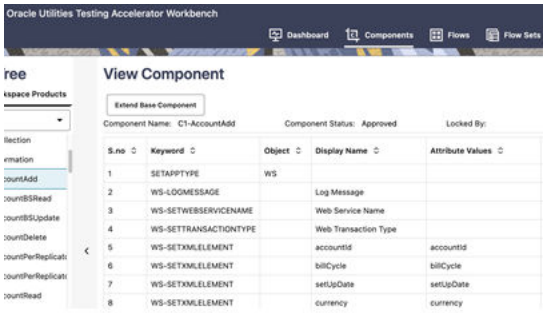
Base components that are delivered by the product in Oracle Utilities Testing Accelerator can be extended to add additional custom elements in the component definition. Custom elements can be added to the component definition through the component definition GUI. When a flow is built using this extended component, test data can be provided for both the base and extended schema elements through the same test data GUI in the **Flow Definition** screen. Extended elements in the component definition can be added or removed based on the requirement, across upgrades.

To add new custom elements to the component definition:

1. Login to the application.
2. Navigate to the **Components** menu.
3. On the left pane, navigate to the module where the new component needs to be added.
4. Right-click the component and select **View Component**.
5. In the **View Component** screen, click **Extend Base Component**.

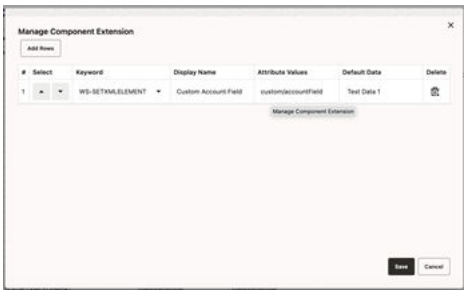
Copyright © 2026, Oracle and/or its affiliates.

Figure 7-8 Extend Base Component action



6. In the **Manage Component Extension** GUI, select the “WS-SETXMLELEMENT” or “WS-SETXMLLISTELEMENT” keywords depending on whether the custom element to be added is a list or an individual element.
7. Add one or more rows based on the requirement.
8. Provide the following information for each row:
   - **Display Name**: The description visible in the flow’s test data GUI against this custom element.
   - **Attribute Values**: The xpath of the element or list element in the Oracle Utilities Application Framework object’s schema.
   - **Default Data**: The data provided in this field is available as default data in the element’s **Test Data** field, in the flow’s **Test Data** GUI.

Figure 7-9 Component extension definition



9. Click **Save** to add the custom component lines to the base component.

To update or delete custom elements of the component definition:

1. Login to the application.
2. Navigate to the **Components** menu.
3. On the left pane, navigate to the module where the new component needs to be added.
4. Right-click the component and select **View Component**.
5. On the **View Component** screen, click **Extend Base Component**.
6. Update or delete the custom component lines.
7. Click **Save** to save the updated custom component definition.

Copyright © 2026, Oracle and/or its affiliates.

Creating Generic REST Web Service Components (Non OpenAPI Specification)

To create Generic 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 Web 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 Generic REST Service Component Definition
- Entering Test Data for a Generic REST Component

Creating a Generic 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.

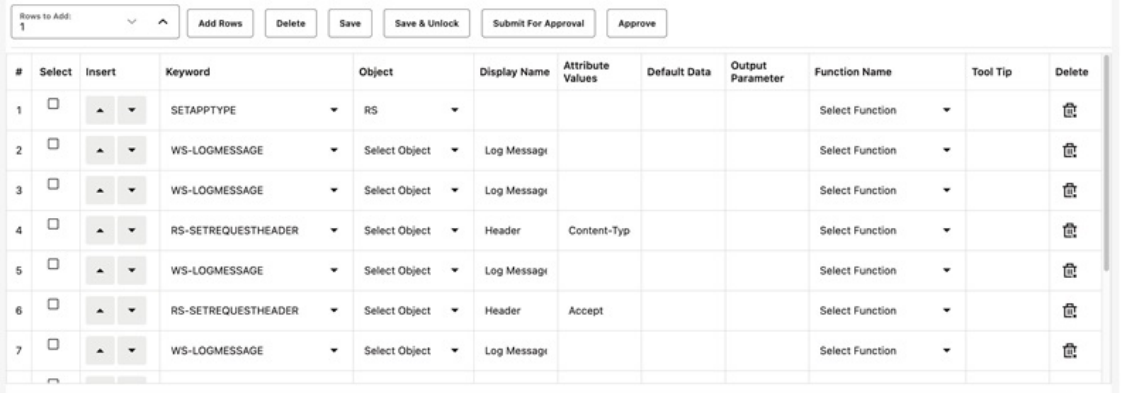
The following list describes each entity in a component definition:

- **Keyword**: The step to be performed.
  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**: Component definition
- **Attribute Values**: The web service XML tag name used as variable to store its value.
- **Default Data**: The default data used in the component definition.
- **Function Name**: The function name called from the library.
- **Output Parameters**: The output in the form of a variable.
  For more options, refer to Appendix D: Generating Re-runnable Test Data.
- **Tooltip**: The data presented as a tool tip during the flow creation.

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

Copyright © 2026, Oracle and/or its affiliates.

Figure 7-10 REST component definition



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 B: 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 generic rest 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 Generic REST Component

To enter test data for a REST component:

1. Navigate to the **Flows** menu.
2. On the left pane, click the **Edit** icon next to the flow name.
3. On the **Flow Definition** page, right-click the REST component and select **Edit Test Data**.

Copyright © 2026, Oracle and/or its affiliates.

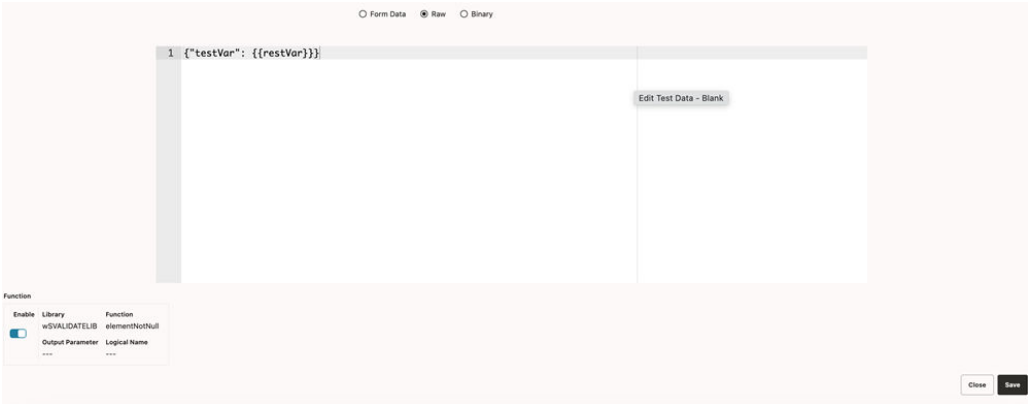
Figure 7-11 REST component test data



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

Figure 7-12 REST request body options



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. It is recommended to create a new module named “Custom Components” to hold any custom components.

Copyright © 2026, Oracle and/or its affiliates.

3. From the context menu, select **Paste Component**.
   If the component name already exists in the module, then a warning will be displayed followed with a prompt for providing a new name to the component.
4. Click **Save**. The component is copied successfully.

Copyright © 2026, Oracle and/or its affiliates.

8

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:

- Adding Products to Workspace Products
- Creating Flow Modules
- Creating Flows
- Support for Integration Flows
- Running Test Flows

Adding Products to Workspace Products

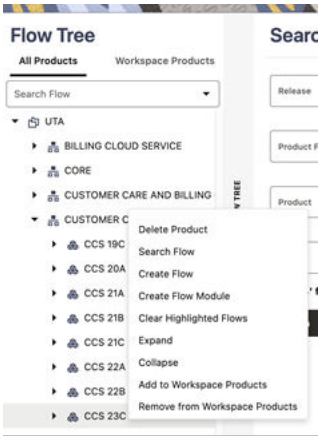
Oracle Utilities Testing Accelerator generally may have a number of product versions in the flow tree structure based on the number of upgrades being tested/managed. With the workspace products feature, you can focus on specific product versions that you are currently working on, so you do not need to navigate and search for the product version in the Oracle Utilities Testing Accelerator component or flow tree structure. You can bring specific product versions into focus by adding them to your workspace based on the product version that you need to work on. The workspace is specific to a user, so each user can add or remove different product versions in their own workspace.

To add a product to your workspace:

1. On the **All Products** tab, navigate to the product in the flow tree.
2. Right-click the product that needs to be added and select **More** > **Add to Workspace Products**.

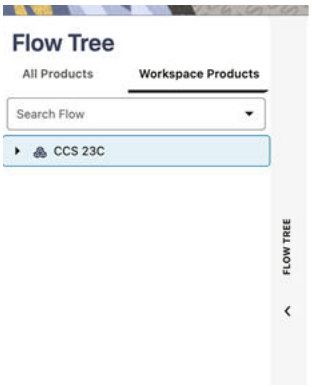
Copyright © 2026, Oracle and/or its affiliates.

Figure 8-1 Adding products to workspace products



3. To view the Workspace Products, click the **Workspace Products** tab in the **Flow Tree** zone.

Figure 8-2 Workspace Products tab in the Flow Tree



To remove a product from your workspace:

1. In the **Flow Tree** zone, navigate to the product either on the **All Products** tab or the **Workspace Products** tab.
2. Right-click the product that needs to be removed and select **More** > **Remove from Workspace Products** to remove a product from Workspace Products.

> **Note**
>
> The **Workspace Products** tab is common for to both components and flows.

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:

Copyright © 2026, Oracle and/or its affiliates.

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 **More** > **Move to Flow Module**.
3. Select the target flow module.
4. Click **Move**.

Creating Flows

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

This section includes the following:

- Creating Flows by Dragging-and-Dropping Components
- Adding Test Data in a Flow
- Moving Data Between Components without Using Global Variables
- Managing Flow Test Data Using Spreadsheets
- Annotating Components in a Flow
- Adding Documentation to a Flow
- Using Global Variables
- Using Container for Flow 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.

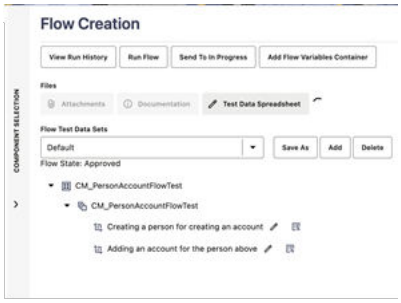
To create a flow:

1. Navigate to the product/module in the flow tree to create the flow.
2. Right-click the product/module and select **Create Flow**.

Copyright © 2026, Oracle and/or its affiliates.

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. To view or update the flow structure, click the edit icon next to the flow name in the flow tree.
   Alternatively, right-click the flow name and select **Create/Update Flow Structure**. The flow contains a default scenario with the same name as the flow name.
6. Expand the flow tree. The flow contains a default scenario with the same name as the flow name.
7. 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.
   By default, the **Approved Components** pane is collapsed if the **Flow Status** is anything other than “In Progress”. To expand it, click “>” on the **Component Selection** bar.

Figure 8-3 Collapsed Component Selection pane



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

8. Make sure to enter the test data at the component step level while defining a flow.

Adding Test Data in a Flow

To add data to a component in a flow:

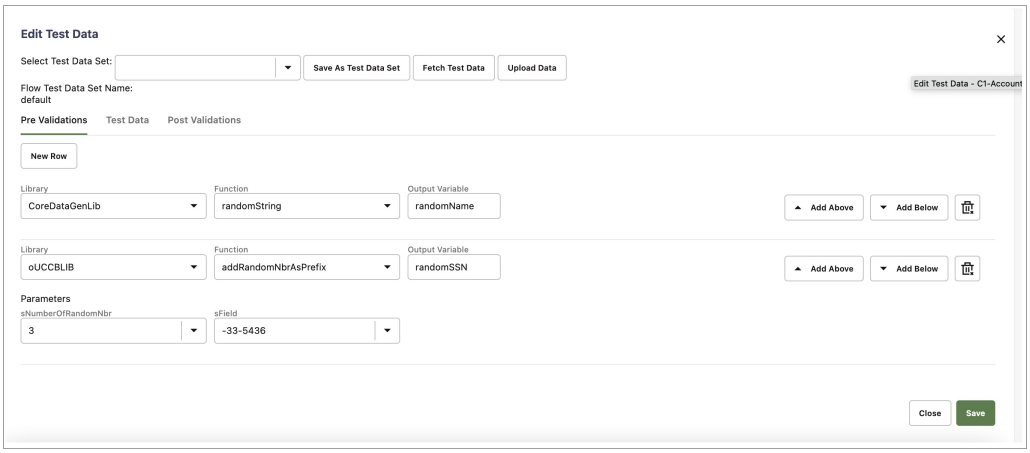
1. In the flow tree structure, click **Edit Test Data** icon next to the component. Alternatively, you can right-click the component and select **Edit Test Data**.
2. Enter the test data in the **Test Data** page. The **Webservice Test Data** page has 3 sections.
   - **Pre Validations**
     The **Pre Validations** section can be used for specifying functions that can generate randomized test data for the flow/component step. It is used to add functions in the

Copyright © 2026, Oracle and/or its affiliates.

components that may be specific to the flow being developed. Click **New Row** to add new rows.

Note: The new **Generate Test Data..** option in the **Test Data** drop-down list allows randomized test data to be generated based on certain user defined rules. These can be specified using an intuitive GUI. It is used for randomized generation of test data for a flow run. Usage of functions in the **Pre-validations** section for the same is supported, but is a legacy method of random data generation.

Figure 8-4 Pre-validation functions



The library in which the function exists can be selected in the **Library** field. Based on the library selected the function can be selected from the **Function** drop-down list. If the function outputs a value, provide the custom global variable name in the **Output Variable** field into which the function output is stored. This variable can be used as test data in the **Test Data** section or in subsequent pre validations or post validations sections. The function inputs can be specified against the parameter fields, based on the number of input parameters that 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 field's drop down under the **Global Variables** section, so they can clearly be distinguished from the global variables defined in the component definition.

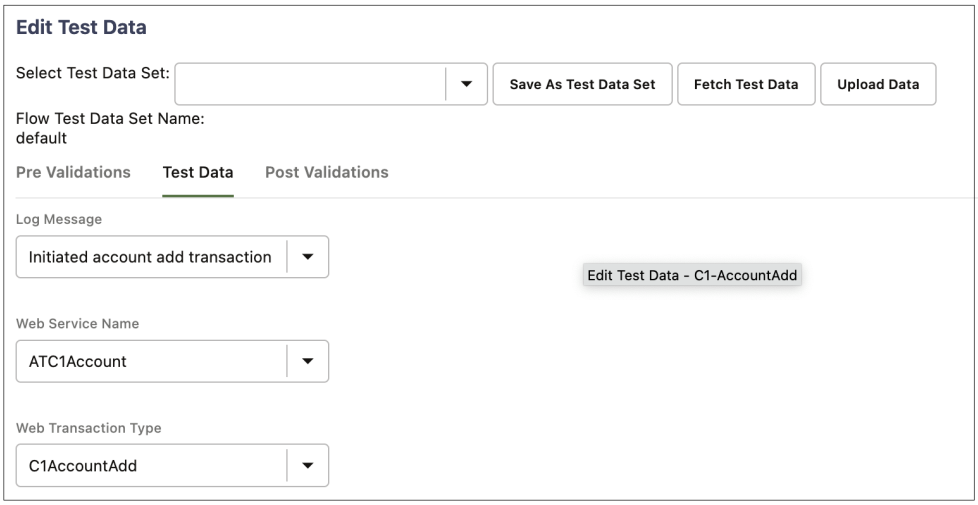
- **Test Data**
  The test data corresponding to each of the elements in the component can be specified in the **Test Data** GUI.
  - The **Web Service Name** and **Web Transaction Type** fields help define the web service end point to which the request needs to be posted to during the test execution. Most of the components have the web service name and transaction type specified as the default data in the component definition. You can select the corresponding values by clicking on the test data drop down corresponding to the web service name/transaction type fields and selecting the value under the **Default Data** section in the list of values.
    The **Web Service Details** section has a switch inside the **Integration Environment** sub section to enable/disable the field to provide integration environment details. This pertains to the environment identifier provided in the flow or user configuration set, for a given integration environment that is being tested.

    Refer to Support for Integration Flowsfor more details.

Copyright © 2026, Oracle and/or its affiliates.

– The **Log Message** field appears for most of the components that have this option enabled. This is free text field and any value entered in this field will be added to the flow run summary report. This helps to identify what a component step does in a flow, by looking at the summary report.

Figure 8-5 Web service settings for component test data



– Test data pertaining to a component line can be specified against that specific line in the test data **Value** field in the **Test Data** section. The test data field is an editable drop down field, so test data can either be selected from the drop-down list or can be keyed into the test data field. The test data field drop down provides 3 options to populate the test data:

* **Data From..**: This option allows the test data to be set from the web service response of any preceding component in the flow. It allows test data to be passed between components, without the use of global variables. Refer to the Moving Data Between Components without Using Global Variables for more information on this feature usage.
* **Generate Test Data..**: This option suppots randomized test data to be generated during a flow run, based on a set of rules. The generated test data is used as input to the field against which this option is specified.

To specify rules for the randomized test data generation:

a. Select **Generate Test Data..** option from the drop-down list.
b. In the **Generate Test Data** window, select the **Generator Type** from the following list:
    * **Random String Generator**: To generate random strings
    * **Random Number Generator**: To generate random numbers
    * **Data Generator**: To generate dates
c. Based on the selected generator type, appropriate fields and options are enabled.
d. Random String Generator supports 2 types of rule definitions.
    * If **Parameter Based Definition** is selected:
        i. Specify the minimum and maximum length of the random string to be generated.

Copyright © 2026, Oracle and/or its affiliates.

ii. Enable/disable upper case, lower case, and number switches to allow or disallow the character type in the generated string.

* If **Format Based Definition** is selected:

i. Specify the format of the string to be generated.

ii. 'U' for upper case character, 'L' for lower case character, and 'N' for number.

iii. Example: Specifying the format as UU43-LLLtest-NU generates BX43-hsgtest-7R. To use character U,L,N as part of the string, precede it with “\”.

e. Random Number Generator supports number generation between the specified minimum and maximum values. The number of decimals that should be part of the generated number can also be specified.

f. Date Generator is not a random generator, but generates the date based on specified conditions. It can be used to retrieve/generate the date when the flow is run. It can also used to add or subtract days, minutes, and hours from a specified date time or current time to generate a calculated date.
To generate a date, an input date has to be provided in the **Input Date** field, which can be Current Date (date of the flow run) or a variable containing a date or the date can be obtained from the output of a previous component using the **Data From..** option.

If a a global variable or **Data From ..** is selected, the **Input Date Format** must to be provided, which specifies the format of the date in the input variable.

* **Timezone**: Timezone of the input date field that defaults to the user’s timezone.

* **Operation**: Specifies if a duration needs to be added to or subtracted from the input date.

* **Day(s)/Hour(s)/Minute(s)**: Specifies the duration that needs to be added to or subtracted from the input date.

* **Output Date Format**: Specifies the format in which the output date should be generated.

> **Note**
>
> The date formats are pre-populated with 3 date time formats that are generally used in Oracle Utilities Application Framework. A custom format can be specified by entering the date time format. All Java date time formats are supported.

g. After entering the necessary inputs for Generate Test Data, click **Save**.

– **Default Data**: This section allows test data to be set from the default data specified in the component definition. For most of the UTA components, the default data is set only for the web service name and the transaction type fields. If the default data is not specified in the component definition for a given field, this option will not be displayed in the test data field drop down.

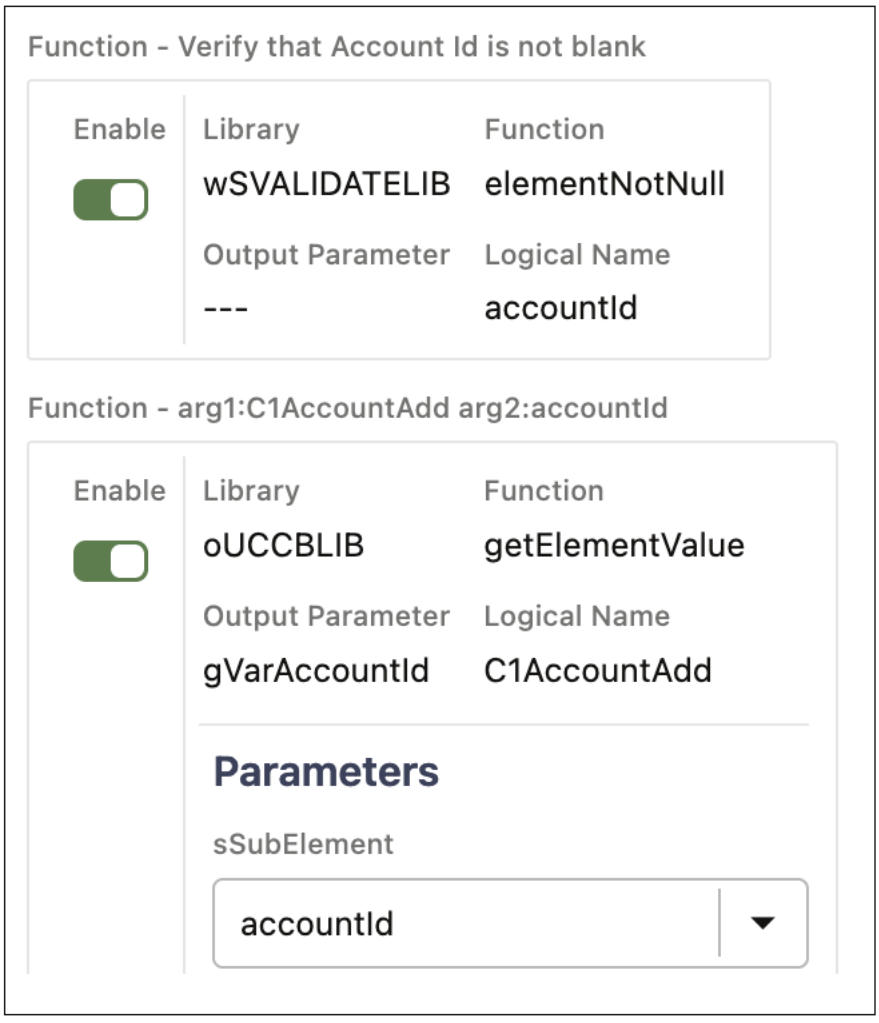
– **Global Variables**: This section allows test data to be set using the global variables defined in the preceding components in the same flow. The list of available global variables are displayed under this section. A global variable can be set as test data input for a given field, in which case, during the course of test flow run, the value

Copyright © 2026, Oracle and/or its affiliates.

populated into the global variable in the preceding component steps will be used as the test data.

- If an element is a repeatable group or list element, click **Add List** in the **Action** column to add multiple repetitions of the list elements. You can add a new instance of the list elements under the group, so another set of test data can be provided.
- The bottom part of the **Test Data** GUI shows the functions defined in the component. Enable or disable the validations/functions defined in the component by appropriately switching **Enable** in the first cell. If the switch is not enabled, during the course of the test run, the function/validation will not be triggered. The switch only appears for rows to which this feature is applicable.

Figure 8-6 Test Data Validation and Value Retrieval Functions



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””.

Copyright © 2026, Oracle and/or its affiliates.

- **Post Validations**
  The **Post Validations** section is used to add verification functions post the base validations. The validations determine if a particular flow step has passed or failed, during the course of the test flow run. Validation failure is considered as a flow step failure. Each of the component comes with a base set of validations and these can be disabled or enabled in the **Test Data** GUI using the switch 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.

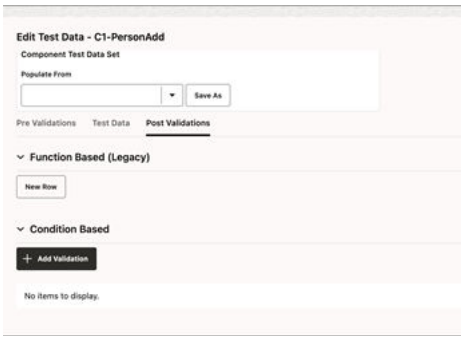
  The post validations section allows users to add any number of functions/validations to the component step in a flow. These will be specific to the component's instance in that flow. These will not apply to the component when used in other flows. Specification of functions in the **Post Validations** section follows the same pattern as the one specified in the **Pre-validations** section.

  The following types of validations can be provided in the **Post Validations** section:

  – **Function-Based Validations**
    Validations are added to the **Post Validations** section using the library functions that are similar to adding the functions in the **Pre-validations** section. The WSValidateLib function library provides the necessary functions to validate a response from the application being tested. The validation library functions process elements in the web service response to determine if the set conditions have been met. This determines if a component step in a flow has failed or passed.

    The function library based validations will continue to be supported along side the condition based validations. But, for ease of use and improved functionality, the condition based validations are recommended to be used.

    Figure 8-7 Function-based and condition-based post-validations

  – **Condition-Based Validations**
    With the condition-based flow validation feature, you can very quickly and easily define complex validations for a flow step using Oracle Utilities Testing Accelerator's advanced and intuitive user interface. This reduces the flow development times while increasing the ease of defining validations in the flow.

    To add a condition based validation:

    a. On the **Test Data** screen for a component in a flow, navigate to the **Post Validations** section.

    b. In the **Condition Based Validation** section, click **+ Add Validation**.

    c. On the **Add Validation** screen, provide appropriate information to create the validation.

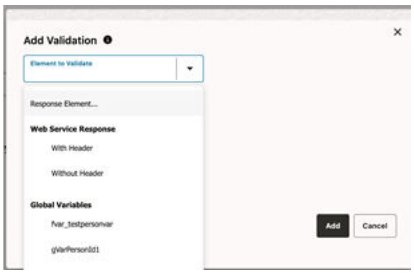
Copyright © 2026, Oracle and/or its affiliates.

To provide the details about the elements to validate:

i. Select the element/response/variable that needs to be validated.

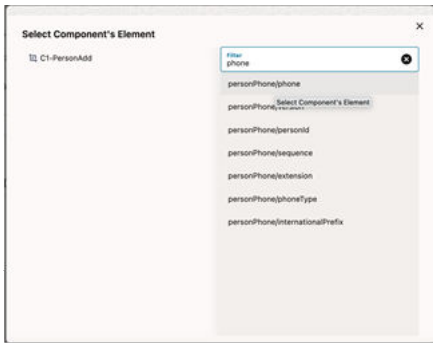
ii. From the **Response Element..** drop-down list, select the element to be validated. If the element is a group/list element, you can specify additional condition(s) to select specific values.
Example: If a Oracle Utilities Testing Accelerator flow reads a person object from Oracle Utilities Customer Cloud Service using the C1-PersonRead component and if the requirement is to validate that the office phone number in the person object is not null, then a new validation can be added. On the **Add Validation** screen, select **Response Element..** from the **Element to Validate** field.

Figure 8-8 Selecting an element to validate



On the new window, select the element to be validated. In this case, it is the personPhone/phone.

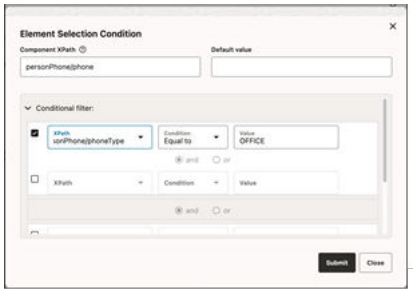
Figure 8-9 Selecting a component response element



After the group/list element is selected, add the condition. Since the office phone number needs to be validated, a condition to select the list that has the phoneType as OFFICE should be specified. Based on this condition, if there are multiple phone numbers for the person entity, only the phone number corresponding to the phoneType 'OFFICE' is validated.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-10 Element selection conditions

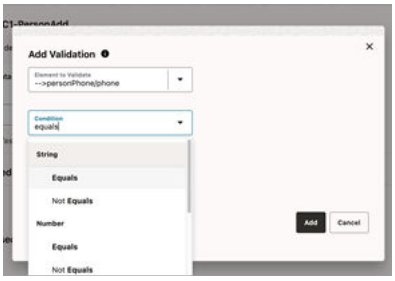


iii. On the **Element Selection Condition** screen, click **Submit**, and then click **Close**.
There is also another option to validate part of or the complete response of the component's web service request. On the **Add Validation** screen, click the **Element to Validate** field. In the drop-down list, scroll down to the **Web Service Response** section and select any of the **With Header** or **Without Header** options. This option allows validations involving verification of a substring in the response. As the name suggests, the 'with header' and 'without header' options allow validation of the response including and excluding the response header.

The third option on the **Add Validation** screen, **Global Variable** allows validation of values stored in the global variables of the flow.

The **Condition** field allows the specification of the condition to be applied to a validation, such as 'equals', 'not equals', etc.

Figure 8-11 Validation comparison conditions



Four different categories of validations are supported based on the type of data being validated:

* Validation conditions under the **String** section allow validating a string in the element/response/variables. These are essentially string comparisions. Note that the validations on the Web Service Response are typically string comparisions.
* Validation conditions under the **Number** section allow comparison of numbers using conditions such as less than, greater than, etc.
* Validation conditions under the **Date** section allow the comparison of date time values. If the conditions under the **Date** section are selected, the corresponding date time format needs to be specified against the **Element to Validate** and **Value to Compare** fields. A date format from the existing options can be selected or a Java supported date format can be explicitly specified in the date format field.

Copyright © 2026, Oracle and/or its affiliates.

* Validation conditions under the **Any** section allow the verification of existence or non-existence of elements in the response.
  The **Value to Compare** field specifies the value to be used for validation/ comparision based on the condition. The field supports specification free text/static values or selection of global variables or response values from previous component steps in the flow using the **Data From**.. option.

d. Click **Add** to add and save the validation.

e. The sequence of the condition based validations can be altered by dragging the dropping the rows holding the condition based validation.

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

Moving Data Between Components without Using Global Variables

Test data can be linked/moved from the response of one component to the input test data field of a subsequent or a later component without using global variables. You can directly select and map the component data fields so the corresponding values are mapped.

To invoke the GUI that supports this mapping feature, click the **Search** icon next to the input test data fields in the **Flow Test Data** page.

The following sections include steps to map the test data between components without variables.

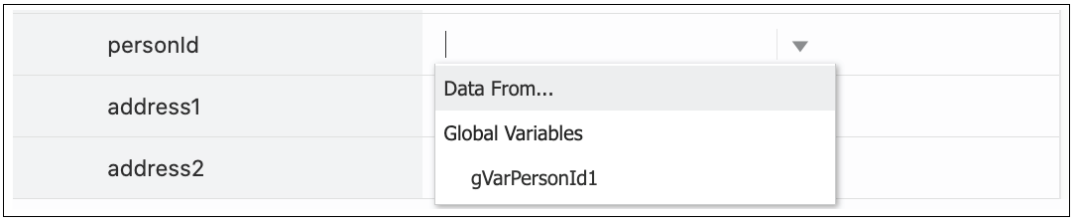
Example: Mapping personId from the response for C1-PersonAdd component to the personId field in C1-AccountAdd component.

To map to non-list elements in a response:

1. Navigate to the **Create/Update Flow Structure** for the test flow.
2. Navigate to the **Flow Test Data** page of the component into which test data should be linked to. Click the downward arrow corresponding to the test data field to show the drop-down list.
3. From the list, select **Data From..**.
   In the example, the test data page corresponds to C1-AccountAdd component in the flow, which has C1-PersonAdd component preceding the C1-AccountAdd component.

   The flow tree structure up to the preceding component of the current component is displayed.

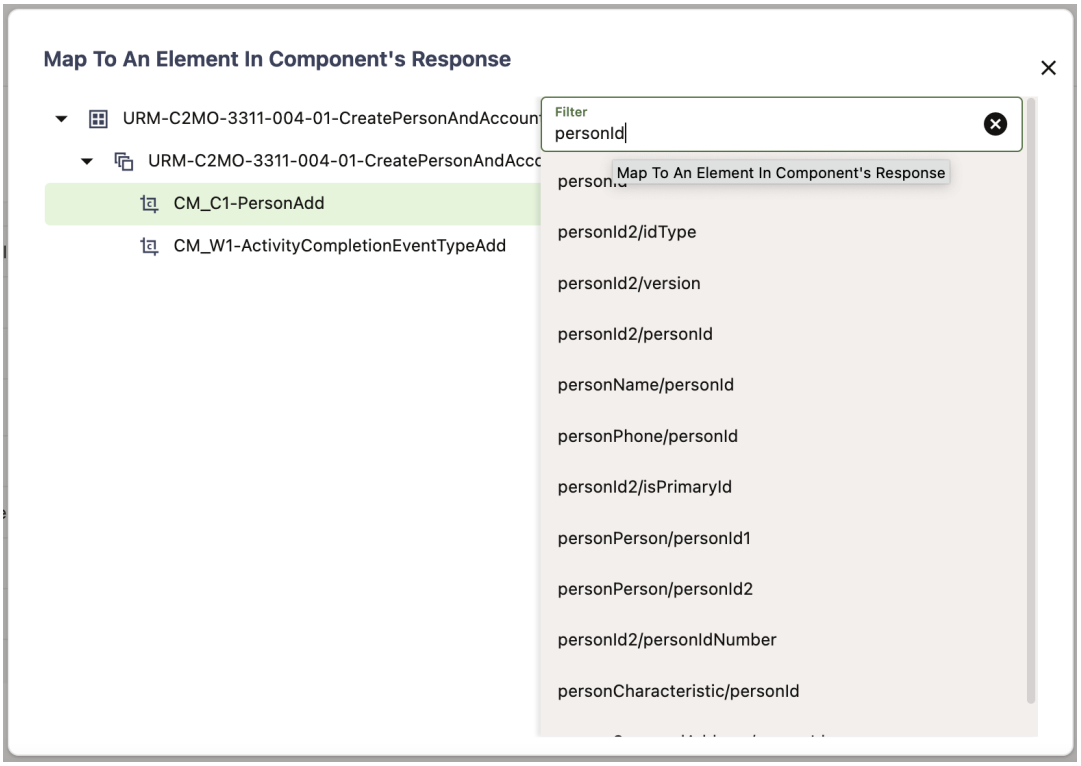
Figure 8-12 Test Data Value Source Options



4. Click the component from whose response the value should to be mapped. The corresponding elements in the component is displayed.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-13 Selecting a component response value



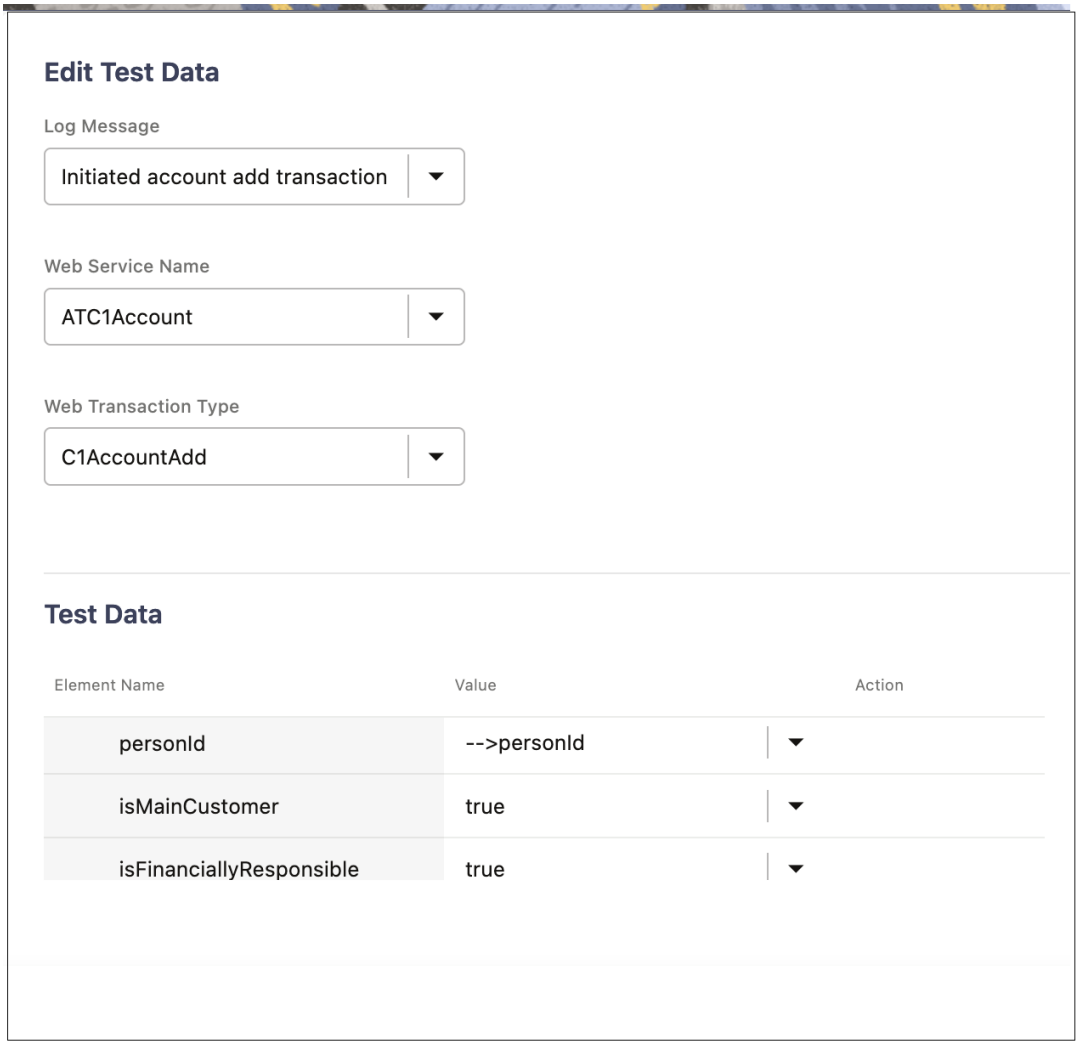
5. Select the element from the component's response whose value needs to be mapped to this field in the current component. The filter at the top of the xpath attributes can be used to quickly find the xpath that is needed.
In the case of the example, select the personId field from the C1-PersonAdd component.

   The test data field is populated with the selected element. "-->" is prefixed to the selected element name to differentiate it from the global variables and static test data.

   The following figure shows the selected personId field mapped between the components.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-14 Person ID Mapped from a Component Response



6. To view/update an existing mapping, click the **Test Data** drop-down list and select **Data From**. The **Map To An Element In Component's Response** screen highlights the existing mapping.

> **Note**
>
> - The mapping feature extracts the value from the web service response of the component used in the mapping and provides it as test data to the test data field to which it is mapped.
> - This feature can also be used in the **Pre-validations** and **Post-validations** sections to map a response value from the prior component as input to base or custom functions.

To map to list elements in a response:

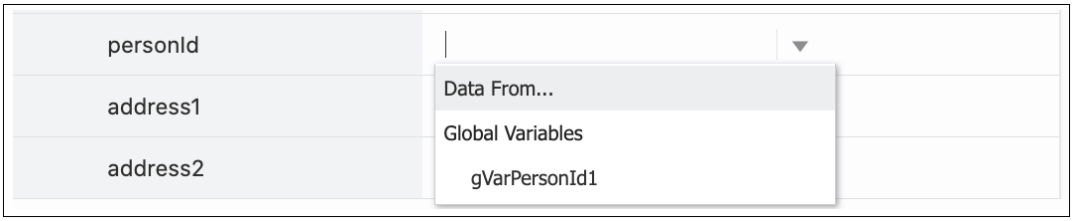
1. Navigate to the **Create/Update Flow Structure** for the test flow.

Copyright © 2026, Oracle and/or its affiliates.

2. Navigate to the **Flow Test Data** page of the component into which test data should be linked to. Click the downward arrow corresponding to the test data field to show the drop-down list.
3. From the list, select **Data From..**.
   In the example, the test data page corresponds to C1-AccountAdd component in the flow, which has C1-PersonAdd component preceding the C1-AccountAdd component.

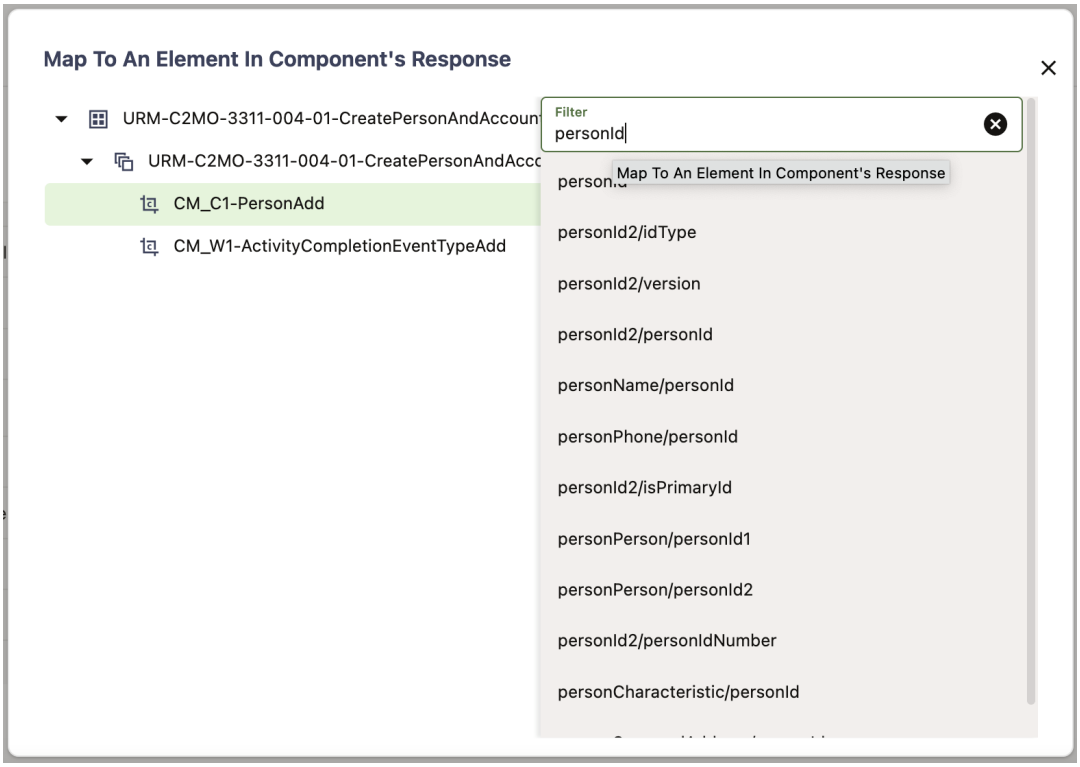
   The flow tree structure up to the preceding component of the current component is displayed.

Figure 8-15 Test Data Value Source Options



4. Click the component from whose response the value should to be mapped. The corresponding elements in the component is displayed.

Figure 8-16 Selecting a Component Response Value



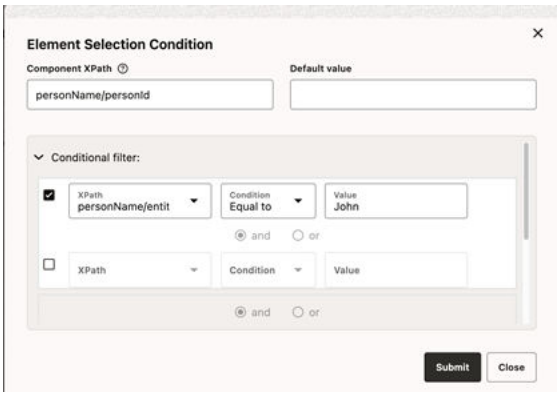
5. Select the element from the component's response whose value needs to be mapped to this field in the current component.
6. If a specific occurrence of the list element is already known, the xpath value can be updated to point to that specific occurrence of that list.

Copyright © 2026, Oracle and/or its affiliates.

Example: personName[2]/personId will map the personId from second occurrence of the personName list

7. If the specific occurrence is not known, conditions may be applied on one or more of the list elements to find the required value from the list.
In the above example: To find the personId corresponding to the person with the name “John”, the first condition can be enabled and the condition can be specified in the condition filter.

Figure 8-17 Selecting a list occurrence with XPath conditions



The list element on which the condition needs to be specified should be selected in the xpath field and a condition type can selected from the condition drop down. Based on the condition type, the value can be provided.

The supported conditions are:

- “Less than”, “Less than or equal to”, “Greater than”, “Greater than or equal to”: These are applicable only to value type Number.
- “Equal to”, “Not equal to”, “contains”: These are applicable to value type String.
- “Equal to” and “Not equal to” condition types as applicable to value type Number as well.
- “Starts with” and “Ends with” condition types are applicable to value type String.
- “Not null” and “Is null” is used to check if the value exists or not, in the response. In case **Not null** or **Is null** option is selected, the **Value** field should be left blank.
If more than one condition needs to be specified, the appropriate conditions may be added by enabling succeeding condition filters. The corresponding join type also needs to be selected for multiple filters. The support join types are “And” and “Or”.

  If a list/group that matches the conditions does not exist in the response, the value specified under the **Default value** gets populated in the corresponding test data field of the component. This can be left blank if the default value has to be populated.

8. After specifying all conditions, click **Submit** to save the conditions for mapping.
9. To review or update the conditions, click the **Test Data** drop-down list corresponding to this field and then select **Data From** option. The **Map to Xpath** window is displayed with the current conditions.

Managing Flow Test Data Using Spreadsheets

Copyright © 2026, Oracle and/or its affiliates.

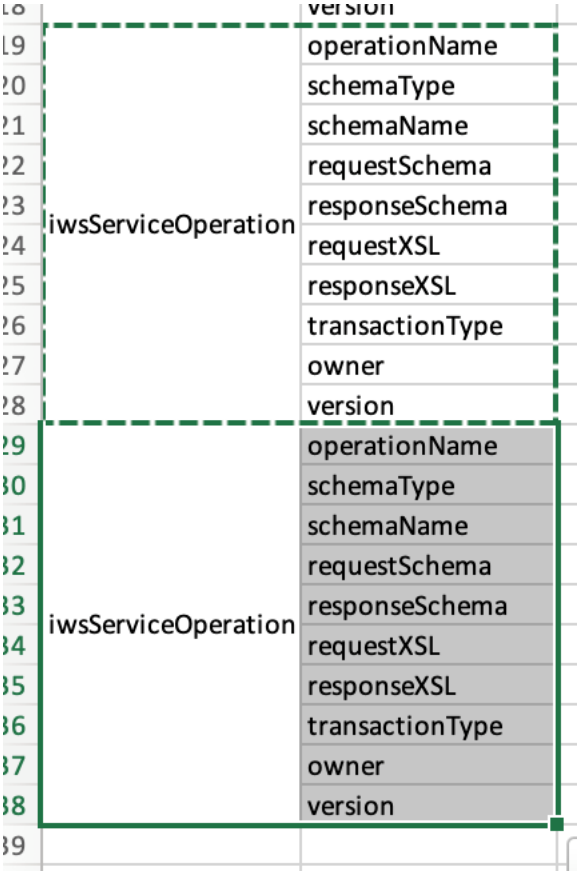
In addition to using the Oracle Utilities Testing Accelerator's workbench GUI for adding/updating test data in a flow, spreadsheets can also be used to add/update test data in the flows. The supported spreadsheet format is .xlsx, which provides versatile formatting for easier input of test data. Each flow has a download spreadsheet option that allows you to download a template corresponding to the flow definition. Each worksheet in the .xlsx spreadsheet holds the test data pertaining to an individual component in the flow. Test data can also be added/updated for test data sets in the flow. You can even create new test data sets in the spreadsheet. Once the spreadsheet is updated with appropriate test data, you can import the spreadsheet to update the test data in the flow from which it was downloaded.

To manage test data using spreadsheets:

1. Navigate to the **Create/Update Flow Structure** for the test flow.
2. Click **Test Data Spreadsheet** in the flow creation frame.
3. In the **Manage Test Data Through Spreadsheet** popup, click **Download**.
4. Download and save the spreadsheet to your laptop/desktop.
5. Open the spreadsheet and edit the test data:
    - The spreadsheet has multiple worksheets where each worksheet maps to a component in the flow definition.
    - The **Documentation** worksheet holds the high-level list of all the worksheets.
    - The worksheets are named based on the scenario number and the components sequence in the flow.
      Example: The prefix SC1_SEQ1 is used to denote scenario 1 and component sequence 1.
    - Test data corresponding to a component in a flow is held in each individual worksheet. The columns marked as element name, hold the names of each individual element in the component and the columns marked with the header test data sets holds the flow test data corresponding to the flow test data set.
    - Multiple values under group/list elements in the component definition can be added in the spreadsheet by duplicating the corresponding rows.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-18 Duplicated spreadsheet list rows



- The test data for elements in the component definition can be set against each of the rows, under the appropriate flow test data sets.
- New flow test data sets can be added in the spreadsheet. Deletion of flow test data sets through the spreadsheet is not allowed.
- After the test data is updated, the spreadsheet can be uploaded into the flow definition through the popup window displayed under step 3 in this process.

Please note the following:

- The sequence of the worksheets in the spreadsheet should not be altered.
- The flow variables defined in the flow are marked in the spreadsheet as “Global Variable”, in the test data. Similarly, the pointers used for mapping test data between components in Oracle Utilities Testing Accelerator GUI are also marked.
- Test data pertaining to subroutine flow calls cannot be managed through the spreadsheet. If the flow has a subroutine flow as a flow step, then the spreadsheet will not have test data or worksheet pertaining to the subroutine flow call step.

Annotating Components in a Flow

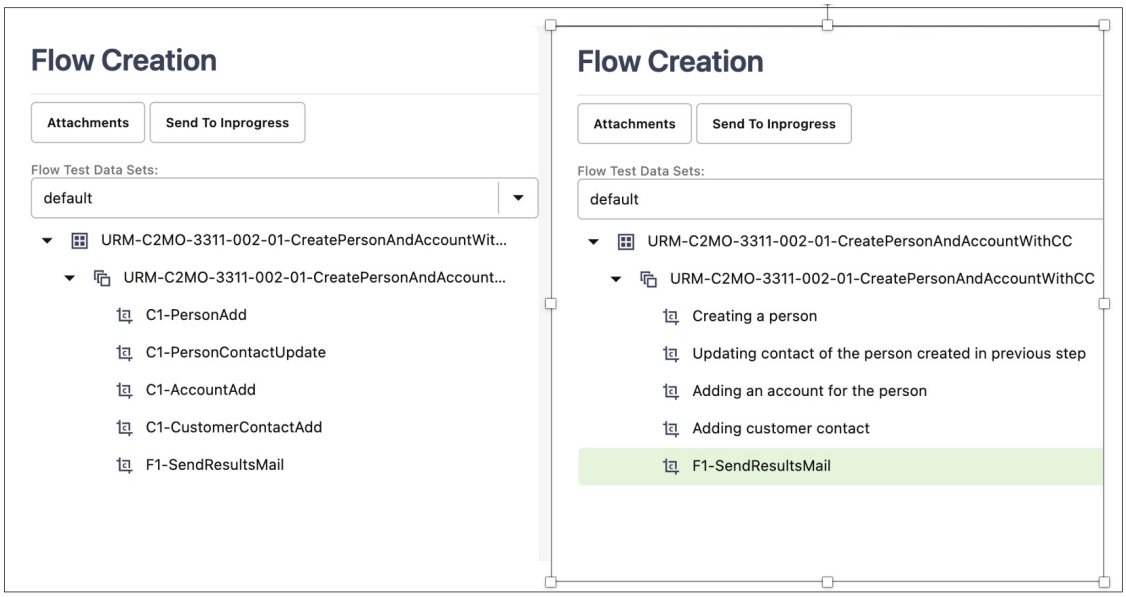
Annotations can be added for each component step to describe the purpose of each of the steps in an Oracle Utility Testing Accelerator test flow. This helps in understanding the functional aspects of the flow just by looking at the flow tree structure.

Copyright © 2026, Oracle and/or its affiliates.

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:

Figure 8-19 Flow Components Before and After Annotation



Adding Documentation to a Flow

Documentation is crucial to any technical artifact as it helps in preserving and sharing information/knowledge pertaining to that artifact. Oracle Utilities Testing Accelerator supports the ability to embed documentation pertaining to a flow within the flow definition.

This feature supports several popular documentation formats. The documentation of the flow is saved as a new type of attachment to the flow, allowing you to upload and download the documentation directly through the **Flow Definition** page.

The following document types are supported: .csv, .doc, .docx, .jpeg, .jpg, .pdf, .png, .txt, .xls, and .xlsx

This section focuses on the following:

- Adding Documents to a Flow
- Deleting Documents from a Flow
- Downloading Documents from a Flow

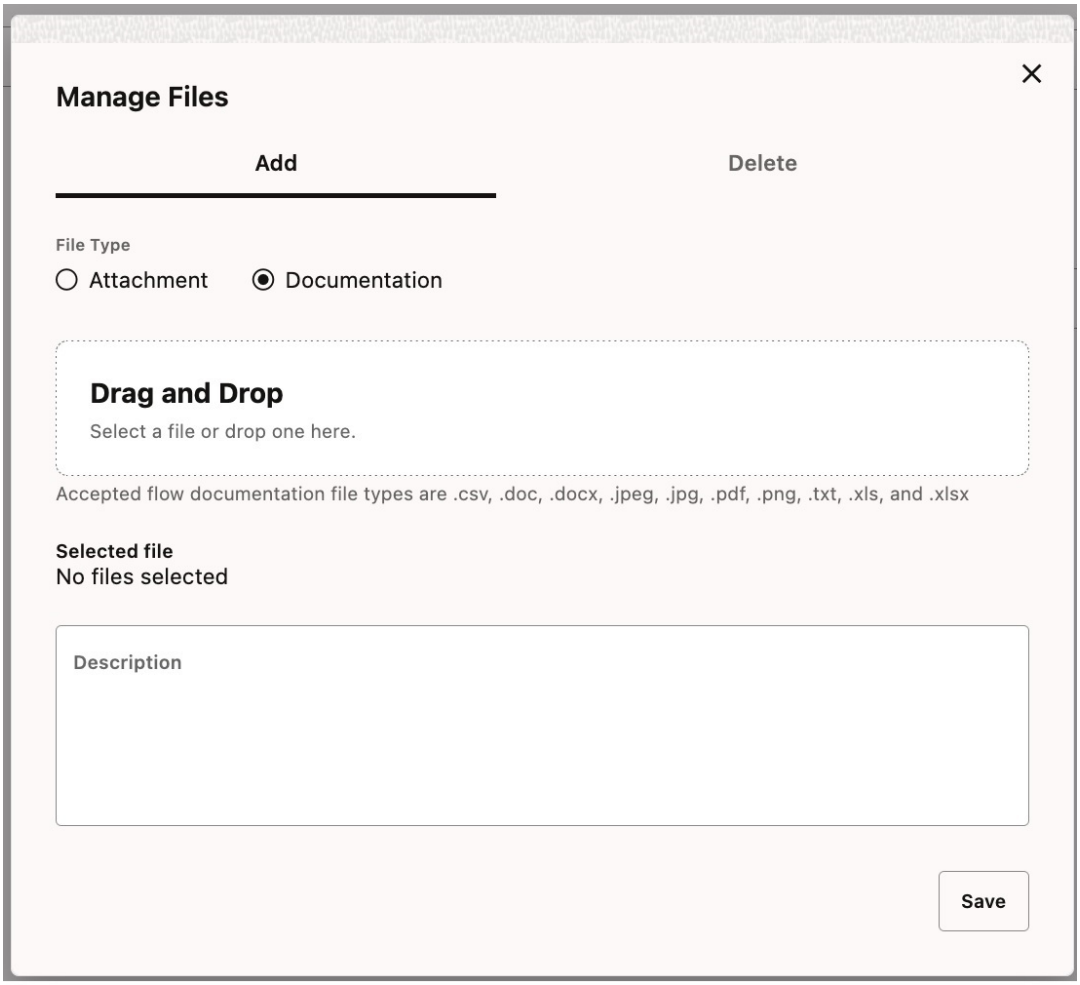
Adding Documents to a Flow

To add a document to a flow:

Copyright © 2026, Oracle and/or its affiliates.

1. Select and right-click the flow name in the Oracle Utilities Testing Accelerator flows tree structure.
2. Click **Create/Update Flow Structure**.
3. In the **Flow Creation** zone, click **Documentation**.
4. Click **Manage**.
5. Drag and drop the documentation file. Alternatively, click the field to add the file using file explorer. Make sure the **Documentation File Type** option is selected.
6. Add the file description in the **Description** field.

Figure 8-20 Adding documents to a flow



7. Click **Save**.

Deleting Documents from a Flow

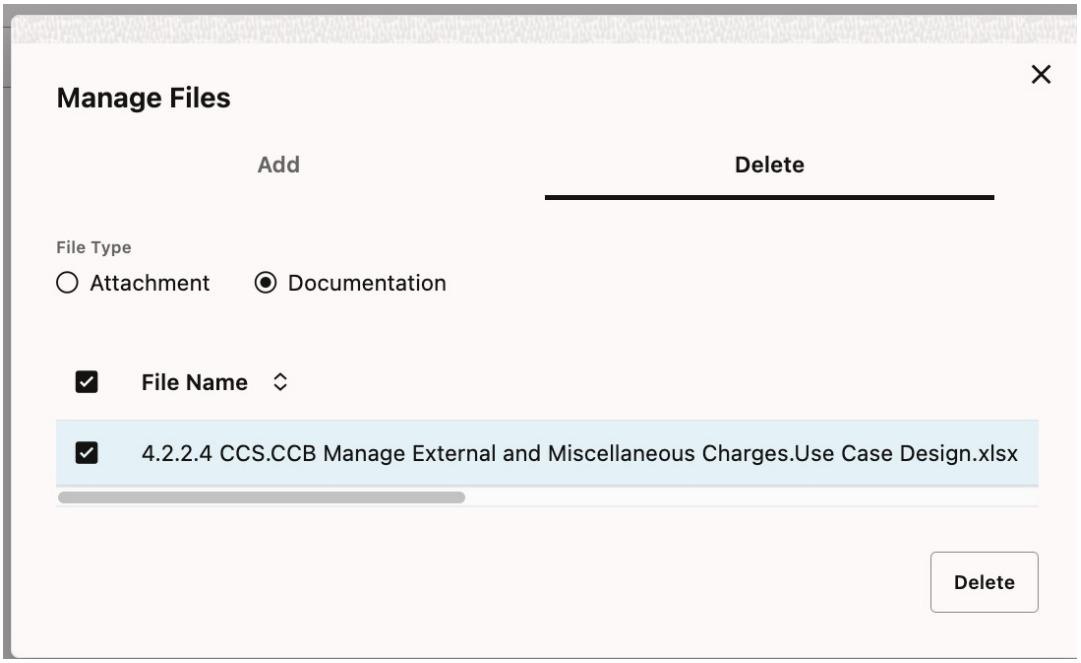
To delete a document from a flow:

1. Select and right-click the flow name in the Oracle Utilities Testing Accelerator flows tree structure.
2. Click **Create/Update Flow Structure**.

Copyright © 2026, Oracle and/or its affiliates.

3. In the **Flow Creation** zone, click **Documentation**.
4. Click **Manage**.
5. On the **Delete** tab, select the files to be deleted.
6. Click **Delete** and confirm the deletion.

Figure 8-21 Deleting documents from a flow



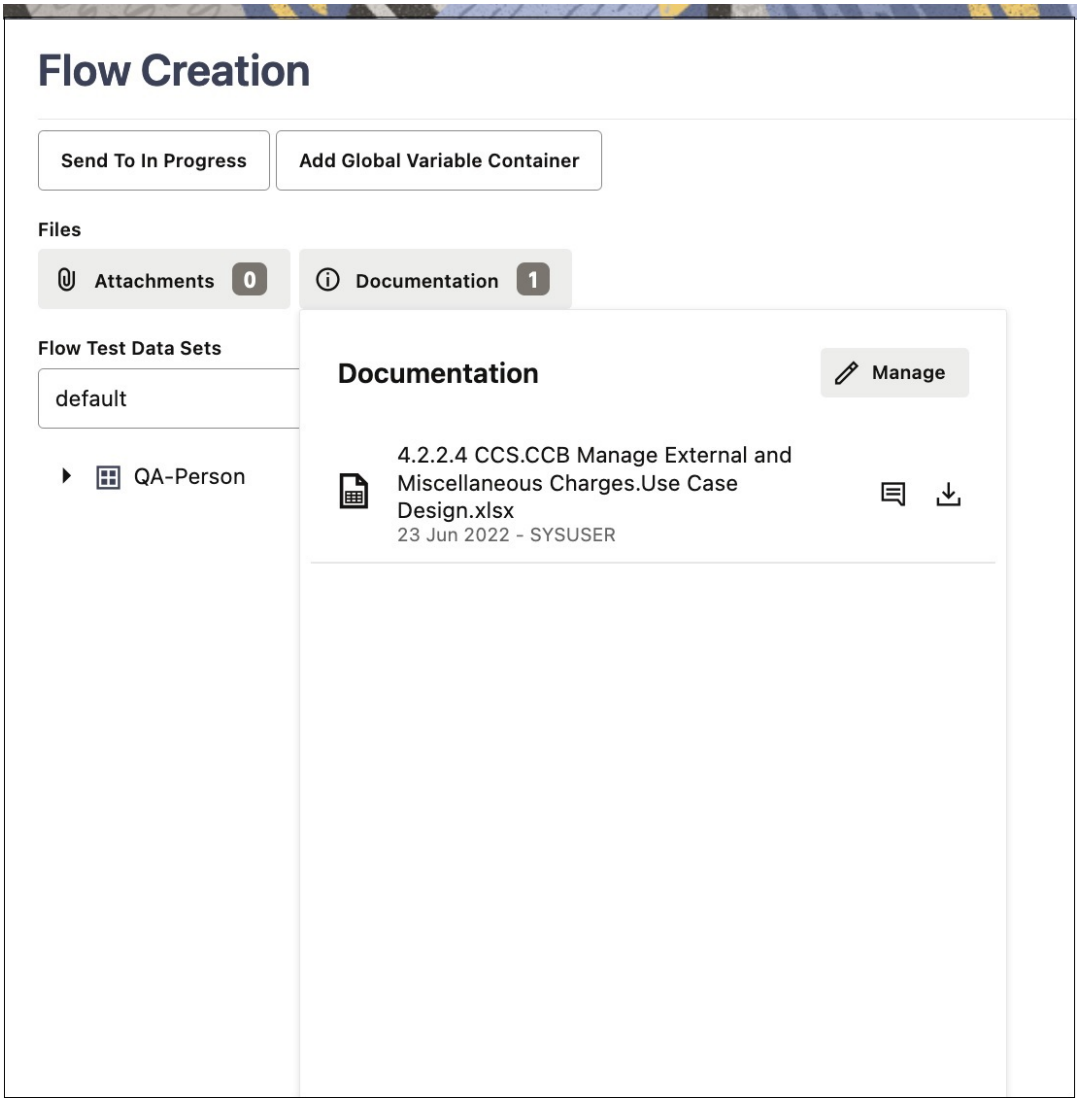
Downloading Documents from a Flow

To download a document from a flow:

1. Select and right-click the flow name in the Oracle Utilities Testing Accelerator flows tree structure.
2. Click **Create/Update Flow Structure**.
3. In the **Flow Creation** zone, click **Documentation**.
4. Click the **Description** icon next to the documentation file name to view the description of the file.
5. Click the **Download** icon next to the file to be downloaded.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-22 Downloading flow documentation



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.

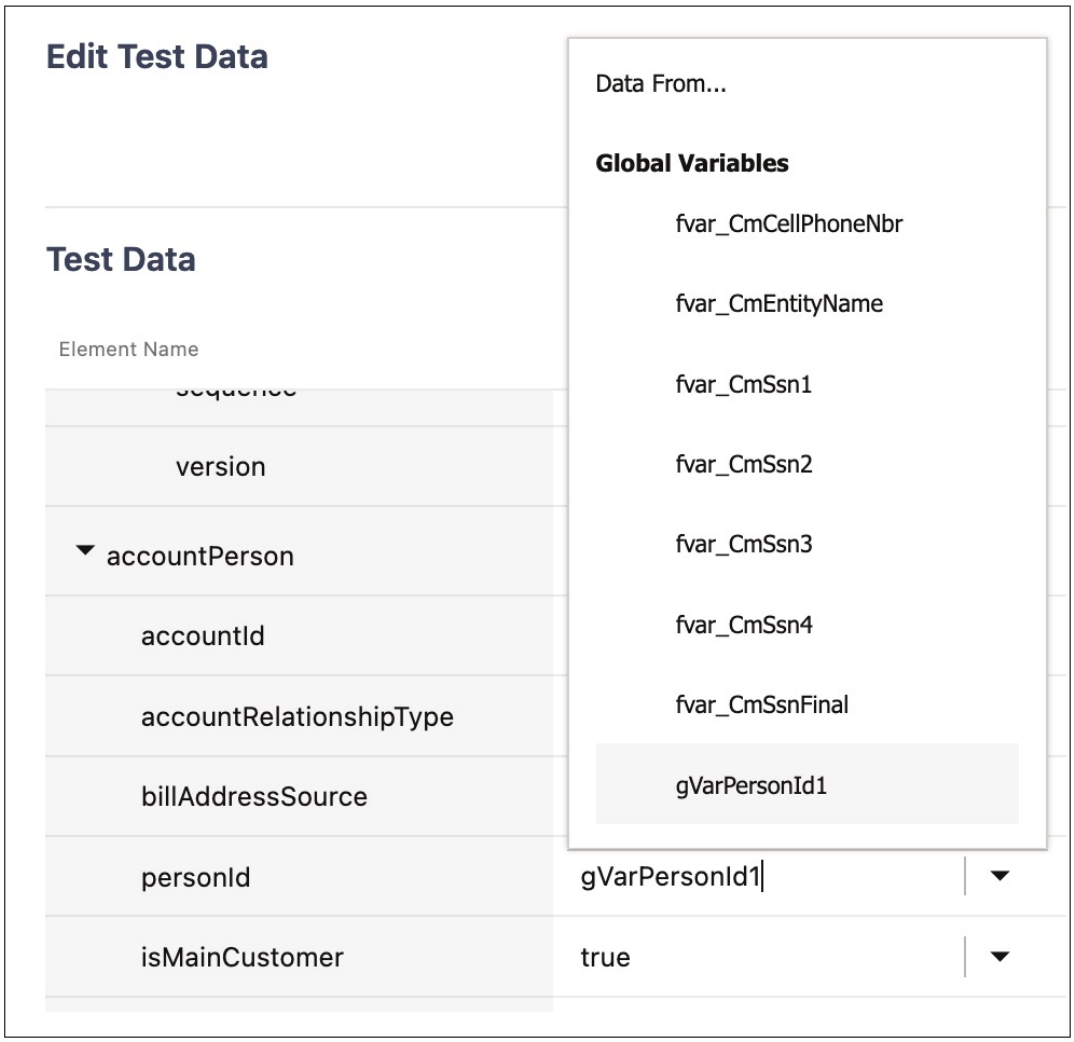
To add component references to a dependent component (C1-CustomerContactAdd):

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 downward arrow to open the **Test Data** drop-down list. All variables exposed by the preceding components are diaplayed in the **Global Variables** section.
   In this case, personId exposed by the C1- PersonAdd component.

Copyright © 2026, Oracle and/or its affiliates.

3. Select the personId variable from the drop down list and set it as test data against this element.

Figure 8-23 Selecting a global variable for test data



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.

Additionally, global variables are also created to hold output of functions used in pre-validations or post-validations sections. These global variables are defined in the **Output Variable** field and are to be set against only functions that return an output value. These function based global variables are automatically pre-fixed with “fVar_” and these can be used as input in the test data of subsequent steps in the flow.

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

Copyright © 2026, Oracle and/or its affiliates.

Using Container for Flow Variables

The flow variables whose values are likely to change between flow runs can be defined in a central container provided towards the beginning of the flow tree structure. This allows the Oracle Utilities Testing Accelerator users to quickly update the test data values in the flow definition before the flow run.

Container for flow variables in Oracle Utilities Testing Accelerator allows the definition of flow variables that can hold test data in a central container within a flow. These variables can be used in place of the actual test data in subsequent components in that flow, so if a need arises to update the test data within the flow, between flow runs or otherwise, you can quickly update the variable definition/value in the central container. This eliminates the need to search through the components to update the test data, in such a scenario. The flow variable container gets added as the first step in the flow definition. Defining/using this container in a flow is optional and would depend on the test flow design and requirements.

Adding Flow Variable Container to a Flow

To add a flow variable container to a flow:

1. Select and right-click the flow name in the Oracle Utilities Testing Accelerator flows tree structure.
2. Click **Create/Update Flow Structure**.
3. Click **Add Global Variable Container**.
4. Right-click the variable container in the flow tree structure and select **Edit Test Data**.
5. Define a flow variable name in the **Output Variable Name** field.
6. Define the variable value in the **Value** field.
7. Add more rows and enter the variables.
8. Click **Save** to save the updates.

These variables can be used as test data in the subsequent components in the flow.

Deleting Flow Variable Container from a Flow

To delete a flow variable container from a flow:

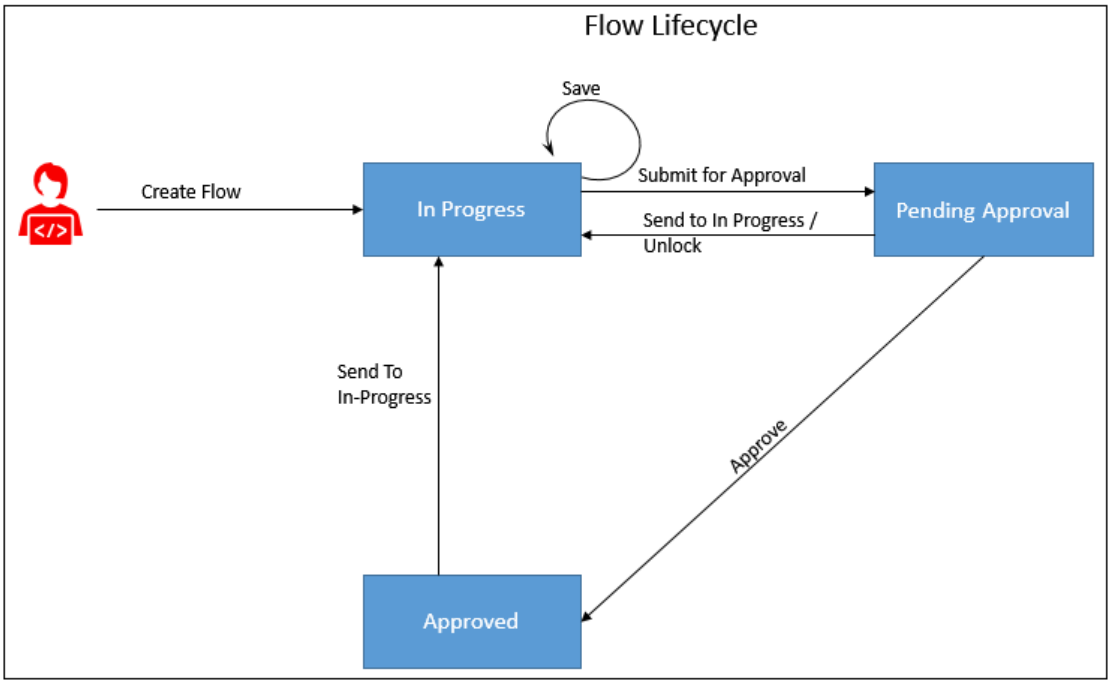
1. Select and right-click the flow name in the Oracle Utilities Testing Accelerator flows tree structure.
2. Click **Create/Update Flow Structure**.
3. Right-click the variable container in the flow tree structure and select **Delete Component**. Confirm the deletion.

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.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-24 Flow lifecycle states



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 (post 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.

Copyright © 2026, Oracle and/or its affiliates.

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

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**.

Copyright © 2026, Oracle and/or its affiliates.

Reorder the components in any of the following ways:

- By drag-and-drop method
- Moving the components to a desired location using menu:

5. Right-click the component to be moved and select **Move Component**.
   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.
6. 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, click the edit icon next to the flow name.
3. Expand the flow.
4. Right-click a component from which you want copy the test data and select **Copy Test Data**.
5. Navigate to the component in the flow.
6. Right-click the component where you want to paste the test data and select **Paste**.

Fetching Component Test Data from an Utilities Application

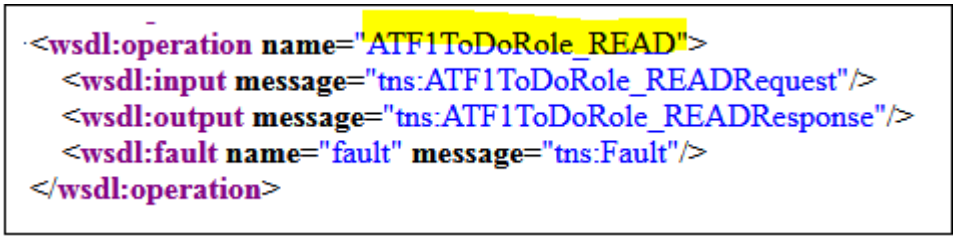
Instead of manually entering the test data for a component, you can fetch the test data from a Utilities application (such as Customer Care and Billing, Meter Data Management, etc.). Select the User, Flow configuration and provide the web service name and the transaction/operation name to access the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

To fetch the test data:

1. Navigate to the **Flows** tab.
2. Select and right-click the flow and then click **Create/Update Flow Structure**.
3. On the **Flow Definition** page, navigate to the component. Right-click and select **Edit Test Data**.
4. On the **Edit Test Data** page, click **Fetch Test Data**.
5. On the **Fetch Test Data** page, enter in the web service name from which the test data has to be retrieved, operation (typically READ operation) to invoke and necessary credentials and any required info (for example: to retrieve data related to ToDoRole).

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-25 ToDoRole Read Operation in WSDL



6. Enter the WSDL name and operation name. Select the user and flow configuration set. 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(pj0TFjXMczsoyzmQ8GuXPt2PSydO7VCbR2jhxtkUH0
6Fuz+zmChpGSCr241KggFC6FwgMg==)
```

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

1. Select the flow/user configuration file from the drop-down menu.
2. Enter the WSDL URL as shown below.

Figure 8-26 Web service name and operation settings



3. Provide the necessary key information to retrieve data (for example: in this case the ToDoRole name) and click **Fetch Test Data**.
4. After the data is retrieved from the target application, review/validate it. Click **Save and Close**.

Unit Testing a Component in a Flow

Copyright © 2026, Oracle and/or its affiliates.

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**.
   Alternatively, you can click the edit icon next to the component in the flow.
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, click the **Test Component** icon  next to the component name. Alternatively, you can right-click the component and select **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. Clicking **Send** does not save the data into the test data of the component 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**.

Copyright © 2026, Oracle and/or its affiliates.

- In the **Find Component Usage** interface, select the flows under which the component test data needs to be replaced. Select the checkbox next to the flow name(s) and click **Replace Test Data**.
- Click **Add Row** to add a row to choose the element of the component whose specified existing test data value needs to be replaced with a new value. To replace the test data of multiple elements of the component, add multiple rows that specify the xpath of the element whose test data value needs to be replaced.
- Set an existing element value to blank or enter test data for component element whose current test data value does not exist. Use #EMPTY as the value in appropriate field (**Existing Value/New Value**).
- Specify a particular occurrence of an element in a group element. Indicate the index of the element in the group. To replace the zip code of second address group element, specify similar to /user/address[2]/zipCode and specify the **Existing Value** and **New Value**.
- Use wildcard "%" in the **Existing Value** field to indicate replacing of any existing value that matches the pattern. Example: To replace a field value that contain anything that starts with a “Building” to “Apartment 123” specify the **Existing Value** as “Building%” and **New Value** as “Apartment 123”.

Flow Subroutines

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

- For a given flow test data set pertaining to the flow calling the subroutines, the test data set of the subroutine can be selected in the subroutine's test data GUI. Right-click the subroutine and select **Edit Test Data**.
- Only the variables defined in the default test data set of a subroutine flow can be used as input or output of the subroutine. This is to ensure standardized API for the subroutine.

Adding Subroutines to a Flow

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

1. Right-click the scenario/component in the flow.
2. Select **Add SubRoutine**.
3. Specify the **Release**, **Product Family**, and **Product** to filter the flows.
4. From the **Flows** drop-down list (search-able), select the flow to be included.
5. Click **Add** to add it to the current flow as a subroutine.

Copyright © 2026, Oracle and/or its affiliates.

> **Note**
>
> - A flow cannot be added to itself as a subroutine. Make sure that nested subroutines do not create a cyclic dependency.
> - A flow can only be added as a subroutine only after it's subroutine interface has been defined. Refer to the Defining Input-Output Parameters of a Subroutine section for information on how to define the subroutine interface.

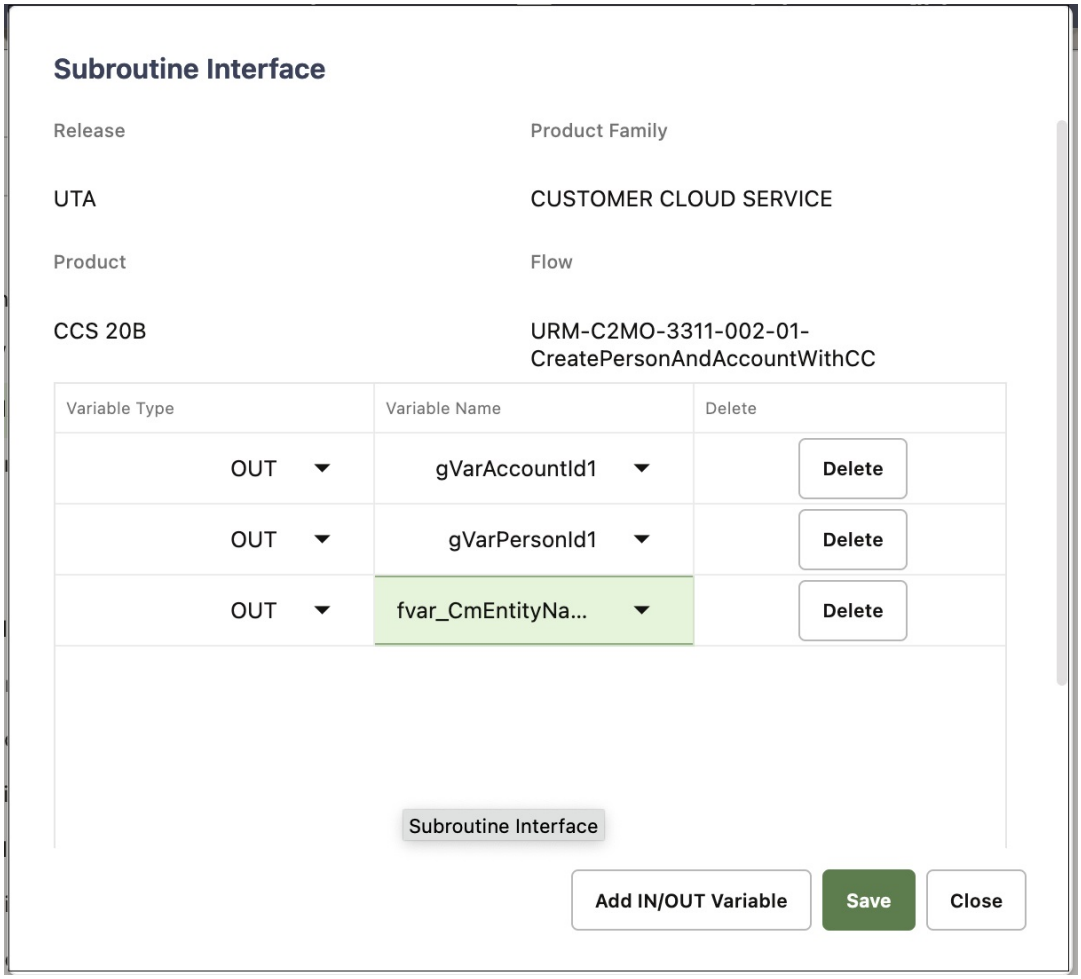
Defining Input-Output Parameters of a Subroutine

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 **More** > **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.
5. Click **Save**.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-27 Subroutine interface output variables



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.

6. 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. The **Test Data** page for subroutine has the following test data related sections:

      - Variable: To create new flow variables in the calling flow, so that the values exposed by the subroutine can be mapped to these flow variables and used in other components within the calling flow.
      - IN: Shows the inputs needed by the subroutine, as defined in the subroutine interface. Input values need to be provided appropriately for the subroutine to run.
      - OUT: Values exposed by the subroutine through its interface definition. In order to use the exposed values in the calling flow, these OUT values need to be mapped to the calling flow variables defined in the Variable section.

   c. Map the input/output variable of the subroutine to a variable in the calling flow. The variables defined inside the subroutine interface cannot be directly accessed by other

Copyright © 2026, Oracle and/or its affiliates.

components in the calling flow. The way to access the data exposed by the subroutine is to create variables in the calling flow and map the output from the subroutine to these flow variables.

Example: The subroutine might be exposing accountId as it is the variable value. 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.

Figure 8-28 Subroutine Test Data and Input Parameters

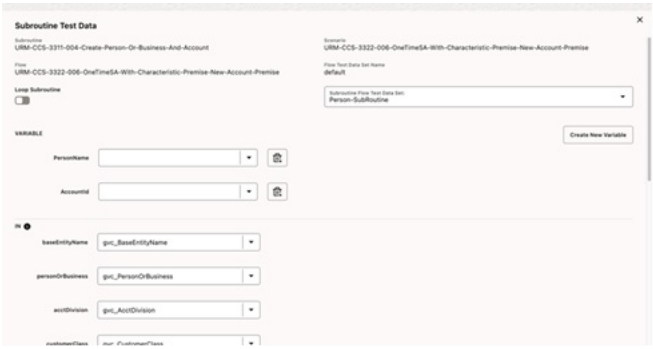


Figure 8-29 Subroutine Output Parameter Mappings



These figures show the **Edit Test Data** page for a subroutine that outputs a personName and accountId. New variables, personName and accountId are created and the outputs of the subroutine (gVarAccountId1 and gVarPersonId1) are mapped to the newly created variables.

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 flow. A subroutine flow can be run in a loop 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.

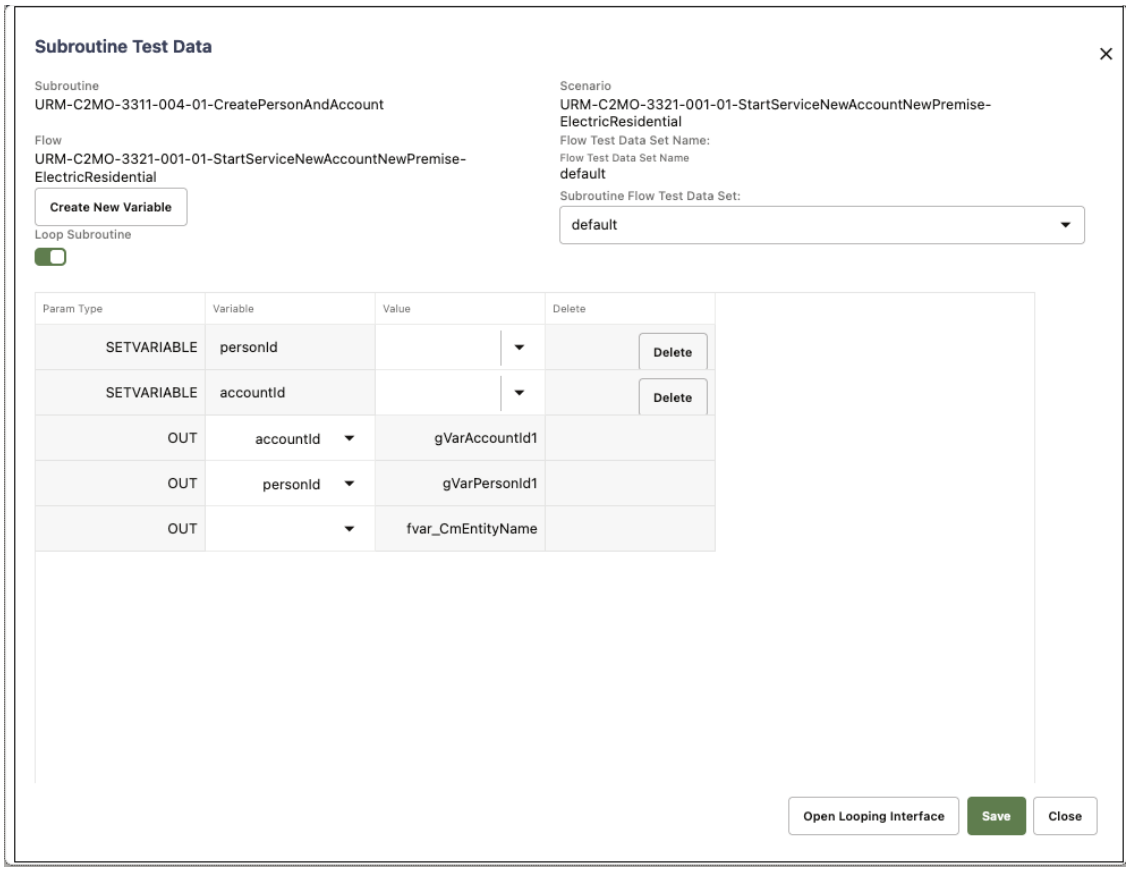
Copyright © 2026, Oracle and/or its affiliates.

> **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 subroutine 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. Right-click the subroutine flow and select **Edit Test Data**. Enable the **Loop** subroutine switch and click **Open Looping Interface** to provide the criteria for executing the subroutine in a loop.

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

Figure 8-30 Subroutine loop controls



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 10 times resulting in creation of 10 person entities in Oracle Utilities Customer Cloud Service.

Copyright © 2026, Oracle and/or its affiliates.

- **Incrementor Type**: Indicates if the loop incrementor would be a number or a date-time or list variable based, 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. The value of this incrementor sis available for usage in the subroutine flow, through the use of “incrementor” variable.
- **Initial Number/Initial Date-Time**: Based on the **Incrementor Type** selected, specify the starting number or the starting date-time to be used. This is not applicable for list variable based looping.
- **Increment Value**: Based on the **Incrementor Type** selected, specify the value by which the initial number/date should be incremented by (either a number or in days, hours, minutes and seconds). This is not applicable for list variable based looping.

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

Figure 8-31 Subroutine looping interface



- **Exit Condition**: The exit condition controls when the subroutine loop would end before the Maximum Number of Iterations. This is not applicable for list variable based looping. Specify the exit condition as follows:
  - Variable: Can be either based on a value of the incrementor variable or any OUT variable of the subroutine. An OUT variable of a subroutine can be used if the subroutine loop should be terminated after a particular value is available in the selected variable.
    Example: We may expect that the subroutine loop should terminate if a specific value is returned in the response element of the subroutine flow. This value can be stored in an OUT variable of the subroutine flow and can be used here.
  - **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.

Copyright © 2026, Oracle and/or its affiliates.

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 incrementer is a global variable that can 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 incrementer 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 COREDATETIMELIB or if necessary, a custom function may be created.

- List variable based looping: Subroutine looping can be used to iterate through a list of values stored in a variable.
  Example: A list of values retrieved from a component's response, such as accountIds retrieved using a search component. The setVariableFromResponseList function from CORERESPONSEUTILIB can be used to retrieve a list of values from the response into a variable.
- After the incrementor type is set as “Variable” in the Subroutine looping interface GUI, the select variable field will be enabled.
  - Select the variable containing the list of values that need to be iterated through, as part of the subroutine looping runs. Click **Save** to set the subroutine looping based on list values in a variable.
  - The subroutine iterations will continue until the list of values in the variable is exhausted or until it reaches the maximum number of iterations specified, which ever comes first.
  - For each iteration through the list of values, the value corresponding to the iteration from the list will be automatically stored in the variable “gvar_list_loopvar” that can be used as input to any component within the subroutine flow. This variable is automatically created as an IN variable for the subroutine flow as and when the looping is defined using the list variable option in the subroutine looping interface.
    Example: If the subroutine looping is defined using a list variable that holds a set of account IDs, the gvar_list_loopvar will hold a specific ID of the account that is being used/processed as part of a specific subroutine loop.

Conditional Bypass of Components in a Flow Run (Skip Component)

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.

Copyright © 2026, Oracle and/or its affiliates.

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 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.

Suspension/Pause and Conditional Resumption of Flow Run

In cases where a flow run needs to be suspended temporarily and resumed, subject to one or more conditions being met in the application being tested, this feature provides necessary flexibility to configure the flow steps. A simple example would be an integration test flow, where in the application/cloud service being tested sends out a message to a 3rd party application as part of the test scenario and the flow run needs to be in a wait state until the 3rd party

Copyright © 2026, Oracle and/or its affiliates.

application sends back an asynchronous message to the application/cloud servicer being tested. In such a case, the flow run needs to be suspended and the state of objects needs to be constantly monitored in the application being tested the flow run can continue, if the objects show that the asynchronous message has been received from the 3rd party application.

A new **Suspend** step can be added to the flow definition succeeding a component step, at which point the flow needs to be suspended until a predetermined condition in met. The predetermined condition is defined as a component/subroutine under the suspend-resume flow control structure.

The concept is simple, in that the flow will suspend the run and will resume execution if the component or subroutine step defined under the suspend-resume control structure pass.

The predetermined condition is deemed to be met if the component/subroutine step under the suspend-resume control structure passes. The validations in the component/subroutine step act as the condition verification steps.

The suspend step holds the time interval details, that specify the time/duration after which the flow run needs to check for the pre-determined condition, regularly. There is also a max time limit, after which the flow will be deemed as failed if the pre-determined condition is not met.

The following set up needs to be done to add the conditions for suspend-resume of flow run:

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 on the component step in the flow definition at which point the flow run needs to be suspended.
5. Select **Add Suspend Step**. A suspend step flow control structure is added to the flow definition and the component is moved under the structure. Subroutines can also be added under this feature as a condition.
6. Right-click the newly added suspend flow - resume control structure in the flow definition and select **Edit**.
7. In the pop-up window, provide the **Verification Interval** details that specify how often the flow run needs to check for the resume condition. This determines how often the flow run executes the component/subroutine under the suspend-resume control structure and checks for the validations to pass.
8. Provide the maximum time limit from the start of flow run suspension, until which the condition to resume needs to be verified, at the specified verification intervals. After this limit, the flow will be automatically failed if the condition to resume is not met.
9. Click **Save** to save the definition.

Copyright © 2026, Oracle and/or its affiliates.

> **Note**
>
> - If one or more components listed above a suspend step fail, and if the continue execution on failure flag in the flow configuration set or user configuration set is set to true, then the suspend step will be ignored and the flow run will progress to completion without suspending the flow run.
> - Multiple suspend-resume steps can be added to the same flow.
> - Subroutines can also be added under a suspend-resume flow control step.
> - When a flow containing a suspend-resume control step is run, a hour glass icon appears against the corresponding component for which the suspend-resume step has been added.

Also, the flow run status is set as “waiting”, from the time the flow run is suspended, until the time the resumption condition is met or the flow has failed. If the resumption condition is met, the flow run will proceed further and flow run status is updated appropriately.

Component Test Data Sets

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

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

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

Creating Reference Test Data for a Component

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

To create a test data set:

1. Login to Oracle Utilities Testing Accelerator.
2. On the **Flows** menu, navigate to the flow in which the component is used. Right-click and select **Create/Update Flow Structure** to open the **Flow Definition** page.
3. Navigate to the component for which the test data set needs to be created. Right-click the component and click **Edit Test Data**.
4. Click **Save As Test Data Set** to save the test data of the component. Specify the name of the test data set and click **Save**. Then, click **OK** to return to the **Edit Test Data** page.

Copyright © 2026, Oracle and/or its affiliates.

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

Figure 8-32 Saving a component test data set



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. Click the **edit** icon next to the flow name.
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.

Figure 8-33 Loading a component test data set



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.

Flow Test Data Sets

Copyright © 2026, Oracle and/or its affiliates.

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. The 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 the Iterative Flow Run section.

This section includes the following:

- Creating Flow Test Data Sets
- Loading Test Data from Flow Test Data Sets
- Deleting Flow Test Data Sets

Creating Flow Test Data Sets

To create a flow test data set:

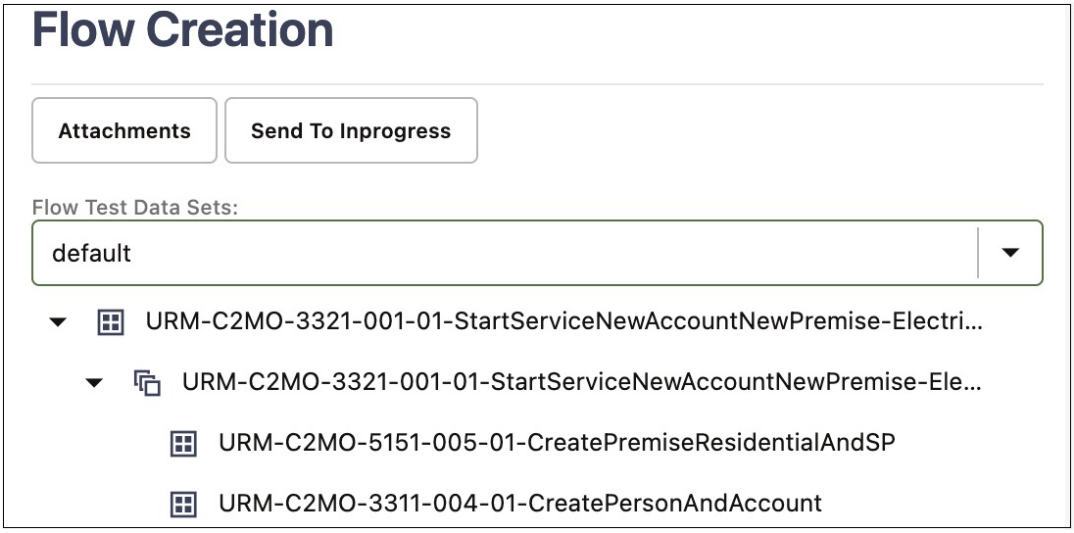
1. Login to Oracle Utilities Testing Accelerator.
2. On the **Flows** menu, navigate to the flow for which the test data set should be created.
3. Click the **edit** icon next to the flow name.
4. On the **Flow Definition** page, click **Add** under **Flow Test Data Sets**.
5. Specify the data set name and click **Add**.
6. Click **Save As** to save the test data of the flow.
7. Specify the name of the test data set and click **OK**.

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

8. If the flow definition includes a subroutine, select the test data set for the subroutine. Right-click the subroutine and select **Edit Test Data**. Select the test data set from the **Subroutine Flow Test Data Set** drop-down list.
9. To edit or add test data against a flow test data set, the corresponding flow test data set has to be selected on the flow structure definition pane.
10. Navigate to the **Edit Test Data** page of each component in the flow and update/add the test data.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-34 Flow test data set selection



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

Loading Test Data from Flow Test Data Sets

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

1. Login to Oracle Utilities Testing Accelerator.
2. On the **Flows** menu, navigate to the flow in which the component is used.
3. Click the **edit** icon next to the flow name.
4. On the **Flow Definition** page, select the test data set to be populated in the flow from the **Flow Test Data Sets** drop-down list.
   To edit or add test data against a flow test data set, the corresponding flow test data set has to be selected on the flow structure definition pane. Navigate to the **Edit Test Data** page of each component in the flow and update/add the test data.
5. If the flow definition includes a subroutine, select the test data set for the subroutine. Right-click the subroutine and select **Edit Test Data**. Select the test data set from the **Subroutine Flow Test Data Set** drop-down list.

Deleting Flow Test Data Sets

To delete the flow 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 name and select **More** > **Delete Flow Test Data Sets**.
4. Select the **Flow Test Data Set** to be deleted and confirm the deletion.

Support for Integration Flows

Support for integration flows allows users to create a single test flow that can interact with multiple applications. Individual components in the flow can be configured to post web service

Copyright © 2026, Oracle and/or its affiliates.

requests to different application URLs. The integration flow support includes support for creation, management and execution of hybrid test flows, where Oracle Utilities Testing Accelerator that is within Oracle Utilities Enterprise cloud service can have a test flow that interacts with that cloud service and one or more Oracle Utilities enterprise applications that may be at customer's premise/data center.

By default, Oracle Utilities Testing Accelerator automatically constructs the web service end point URL based on the web service name provided in the flow test data. This default URL executes test flows against the Oracle Utilities Enterprise cloud service that it is part of.

To configure a component in a flow to post a request to a different application, depending on the integration scenario, one of the below approaches may be used.

Hybrid Integration Scenario

In this integration scenario, Oracle Utilities Testing Accelerator is used to develop and run a test flow spanning an Oracle Utilities cloud service and an Oracle Utilities enterprise application that is hosted either on-premises or PaaS. The following configuration needs to be added to either the flow configuration set or user configuration set.

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

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

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

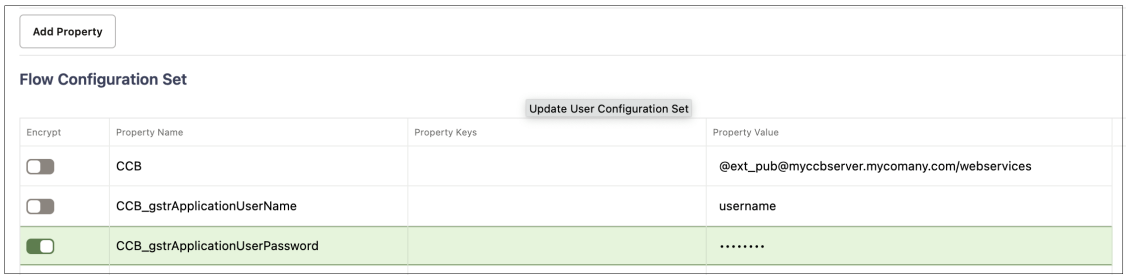
CCB property holds the external web service end point URL prefixed with “@ext_pub@”, up to but not including the web service name. If the web service end point URL for the WSDL of person object in CCB is https:myccbserver.mycompany.com/webservices/ATC1PersonAdd?WSDL, the CCB property should hold the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

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

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

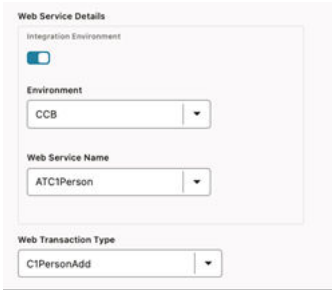
Copyright © 2026, Oracle and/or its affiliates.

Figure 8-35 Integration environment configuration



The integration flow in Oracle Utilities Testing Accelerator may contain a mix of components pertaining to the cloud service that it is part of and components pertaining to on-premise Oracle Utilities Enterprise applications. To get a component in flow to post a request to the on-premises Oracle Utilities Enterprise applications, the integration switch needs to be enabled to view the **Environment** field under the **Web Service Details** section in the test data GUI for the component in the flow. The 'Environment' keyword can be provided in the **Environment** field so that the requests from the component are directed to the URL specified using the environment keyword.

Figure 8-36 Integration web service details



In the example where Oracle Utilities Customer Care and Billing is the on-premises application and "CCB" is the keyword, the environment name for the C1-PersonAdd component's test data in the integration flow should be specified as "CCB" and the web service name should be specified as ATC1Person. 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".

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

Oracle Utilities Cloud Services Integration Scenario

In this integration scenario, Oracle Utilities Testing Accelerator is used to develop and run a test flow spanning two different Oracle Utilities cloud services, such as Oracle Utilities Customer Cloud Service (OUCCS) and Oracle Utilities Work and Asset Cloud Service (OUWACS).

In the above case, assuming that the Oracle Utilities Testing Accelerator that is part of Oracle Utilities Customer Cloud Service is being used for the flow development and execution, Oracle Utilities Work and Asset Cloud Service becomes the external application (in the context of

Copyright © 2026, Oracle and/or its affiliates.

Oracle Utilities Testing Accelerator integration testing) to which Oracle Utilities Testing Accelerator needs to post the integration test request to.

Before adding components from the external Oracle Utilities cloud service to the flow, import its product pack. For details, refer to the **Product Pack Import** section in Chapter 9.

In the flow and user configuration sets in Oracle Utilities Testing Accelerator, which is part of Oracle Utilities Customer Cloud Service, 3 properties pertaining to Oracle Utilities Work and Asset Cloud Service can be found, provided that the Oracle Utilities Work and Asset Cloud Service and Oracle Utilities Customer Cloud Service are part of the same tenancy. The parameters are listed as follows:

- wacs
- wacs_gStrUserName
- wacs_gStrPassword

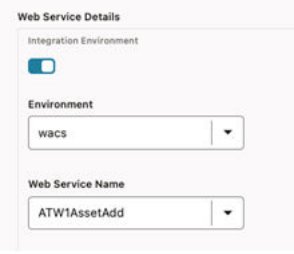
Depending on the Oracle Utilities enterprise cloud service available for the integration scenario, the properties corresponding to Oracle Utilities Meter Solution Cloud Service, Oracle Utilities Customer Care and Billing Cloud Service and others can be found in the flow and user configuration sets.

Enter the following values corresponding to each of the parameters:

- wacs: Prefix “$” to the parameter name and provide it as the value for that parameter. Example: $wacs
- wacs_gStrUserName: Provide the username for web service authentication to Oracle Utilities Work and Asset Cloud Service application.
- wacs_gStrPassword: Provide the password corresponding to the username for web service authentication to Oracle Utilities Work and Asset Cloud Service application.

In the above example, to post a component request (say W1-AssetAdd) from Oracle Utilities Testing Accelerator in Oracle Utilities Customer Cloud Service to Oracle Utilities Work and Asset Cloud Service, in the corresponding component's test data, the integration switch has to be enabled to view the **Environment** field under the **Web Service Details** section in the test data GUI for the component in the flow. The keyword ‘wacs’ (the environment to which the request needs to be sent to) should be specified in the **Environment** field. The web service name for W1-Asset component's test data in the integration flow should be specified as “ATW1AssetAdd”.

Figure 8-37 Integration web service environment



This ensures that the W1- AssetAdd component posts the request to Oracle Utilities Work and Asset Cloud Service whose configuration has been specified in the flow configuration set or user configuration set using the keyword prefix “wacs”.

Copyright © 2026, Oracle and/or its affiliates.

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

> **Note**
>
> For the Oracle Utilities Testing Accelerator integration flows to be able to communicate with the external environment on SaaS/PaaS/on-premises, the corresponding environment end points must be added to the cloud allow list. Please log a ticket with the Dev Ops team to allow listing of the URI.

Running Test Flows

This section focuses on executing a test flow.

- Running Test Flows
- Iterative Flow Run
- Stopping Flow Run on Validation Failure
- Stopping Flow Run Manually
- Viewing Flow Run Details
- Viewing Flow Run Failure Details
- Viewing Flow Run Summary Report
- Conversational Test Data Management
- Runtime Configuration for Flow Run

Running Test Flows

To run a test flow:

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 **Run Flow**.
   Alternatively, you can do any of the following:
   - Click the run icon ▷ next to the flow name.
   - click the broadcast icon next to the test flow name, and then click **Run Flow** on the **Flow Definition** pane.

   > **Note**
   >
   > The test flow can be run only if it is either in the “Pending Approval” or the “Approved” state.
3. Select the **Flow Configuration** and **User Configuration** to be used to run the test flow. These configurations hold the authentication information along with other configuration parameters required for the flow run. The flow configuration set is common for all the users of Oracle Utilities Testing Accelerator and is usually created by the admin, where as the

Copyright © 2026, Oracle and/or its affiliates.

user configuration set corresponds to the user who created the configuration set and is accessible only to the user who created it.

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 section.

5. On the **Flow Run** page, the run details are displayed.
At the top of the flow run tree, the flow run start time and duration are displayed, along with the live flow run status.

   The flow run has the following statuses:

   - Running: Denotes that the flow run is in-progress.
   - Passed: Displayed for all the flow runs that have successfully completed execution and all the validations in the flow have passed.
   - Failed: The flow run execution is complete, but the flow has failed due to one or more failed in the validations in the flow.
   - Stopped: Set on the flow and the flow run is stopped when user manually clicks the **Stop** button on the **Flow Run** screen.
   - Waiting: The flow run is in-progress, but is currently suspended waiting for a user defined condition in the flow, to be met, so the flow execution can continue. The condition to be met is checked for at regular intervals, as defined in the flow's suspend-resume definition. Refer to the Suspension/Pause and Conditional Resumption of Flow Run section for more information.
     The flow run 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.

     A toggle switch has been provided to view the request-response in different formats. If the **Advanced View** switch is enabled, the request-response details are displayed as label-value pairs, similar to the flow test data GUI, where in the label is the element name and the value is the data that was set in the request or received in the response for that element.

     With the **Advanced View** switch disabled, the display defaults to legacy XML based view where in the request and response are visible as XML documents.

     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.

     The request-response details are displayed as lable-value pairs, similar to the flow test data GUI, where in the label is the element name and the value is the data that was set in the request or received in the response, for that element.

     A toggle switch has been provided to switch to a legacy xml based view where in the request and response are visible as XML documents.

6. Click **View Logs** to view the logs of the run.

Iterative Flow Run

To run a test flow iteratively:

1. Login to Oracle Utilities Testing Accelerator.

Copyright © 2026, Oracle and/or its affiliates.

2. On the **Flows** menu, select the product to which the flow belongs.
3. Right-click the test flow and select **Run Flow**. Alternatively, click the run icon ▷ next to the flow name.

> **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, the application will use test data sets for each of the iterations. 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.

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.

Copyright © 2026, Oracle and/or its affiliates.

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 tree for the corresponding flow is displayed.
5. Click the info icon next to the failed component step in the flow run. The appropriate error is displayed along with a list of possible reasons leading to the error.

Viewing Flow Run Failure Details

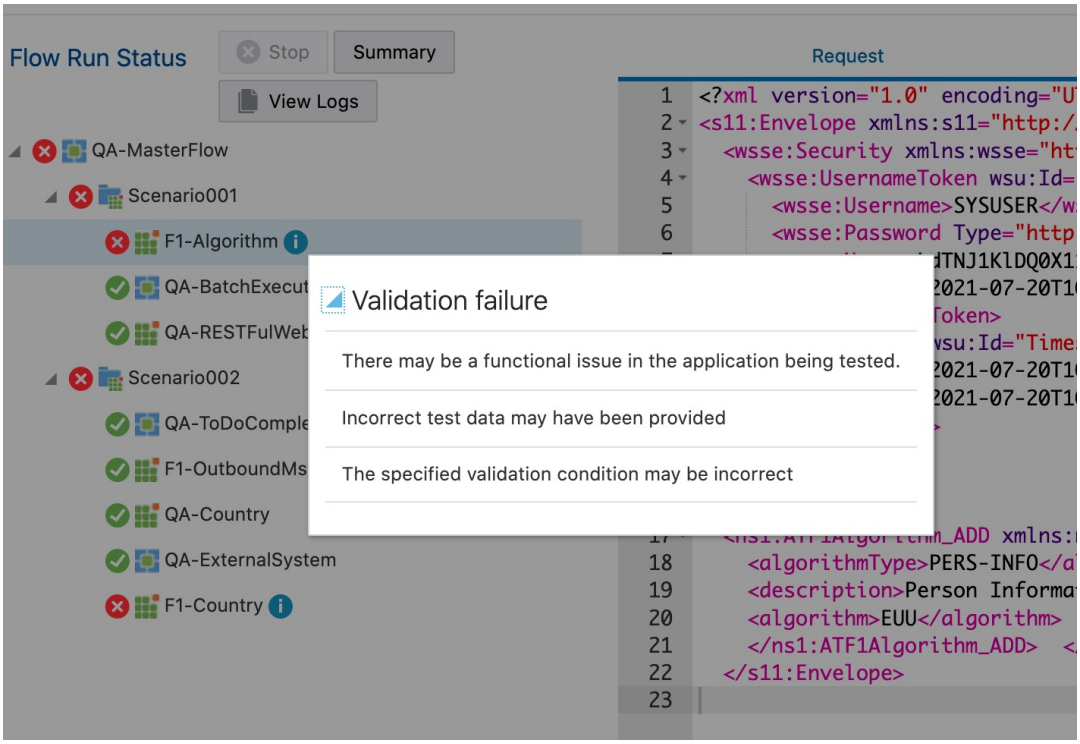
If a flow run fails, the possible reasons for failure can be viewed. Click the **Info** icon next to the failed component step in the **Flow Run Status** tree structure.

To view the failure details for a flow run:

1. Login to Oracle Utilities Testing Accelerator.
2. Navigate to the **Flow** menu.
3. On the left pane, navigate to the flow and right-click it to view the run summary. Click **View Run History**.
4. Click the flow run entries to view the respective details of that run.
5. Click the **Info** icon next to the failed component step in the flow run. The error is displayed along with a list of possible reasons leading to the error.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-38 Flow run validation failure details



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**.

Copyright © 2026, Oracle and/or its affiliates.

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

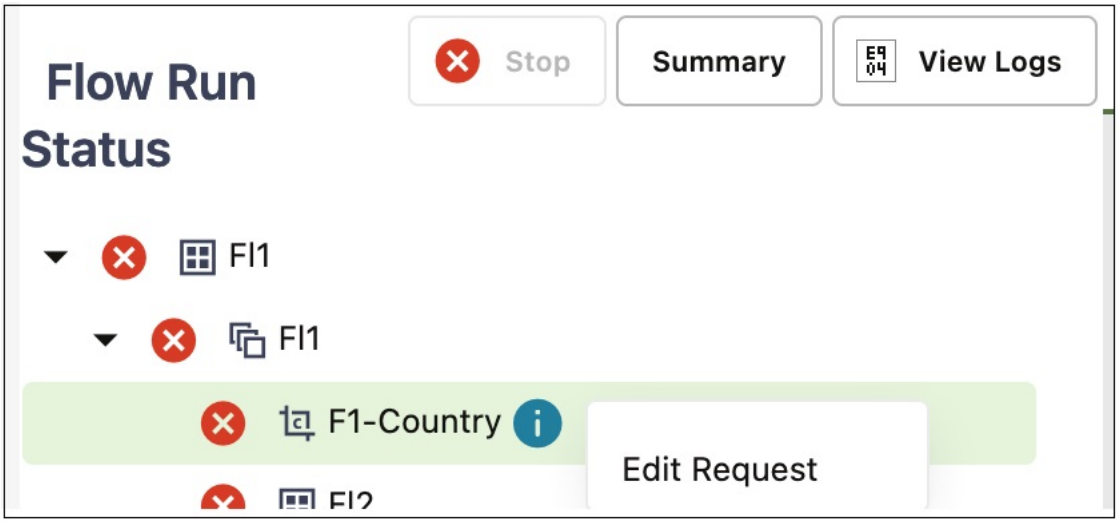
The **Edit Request** feature allows to:

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.

Figure 8-39 Edit Request Action in Flow Run Status



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 is provided with the following 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.

Copyright © 2026, Oracle and/or its affiliates.

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

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

Runtime Configuration for Flow Run

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.

> **Note**
>
> Configuration in the user configuration sets overrides configuration in the flow configuration if the same parameter is defined in both the configuration sets.

The properties under flow and user configuration sets are categorized to make it easier to identify specific properties applicable for your testing.

The following properties are available by default:

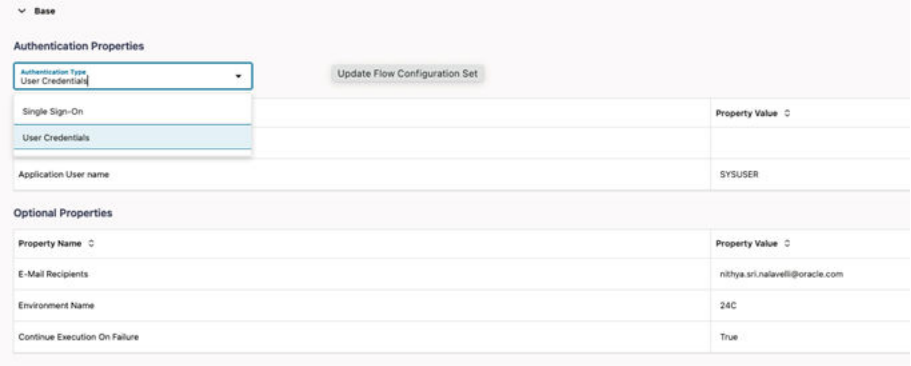
- **Base Category**: Lists the properties that are common for SOAP and REST based web services. This category is further divided into the following:
  - **Authentication Properties**
    - **Application URL**: The common web service's end point URL of the application that is being tested needs to be specified against this property.
      Note that for Oracle Utilities Testing Accelerator that is available as part of Oracle Utilities Enterprise cloud services, the corresponding environment's URL is predefined within Oracle Utilities Testing Accelerator. Users need not provide the Application URL for the same. However, users can provide the user name and password for authentication.
    - **Authentication Type**: Select **User Credentials** based authentication from the **Authentication Properties** drop-down list to provide the **Application User Name**

Copyright © 2026, Oracle and/or its affiliates.

and **Application User Password**. This option uses basic authentication for web service authentication with the application being tested.
If the **Authentication Type** is set as “Single Sign-On”, then the username and password need not be provided. The Single Sign-On federated authentication is used for web service authentication with the application being tested. Selecting the **Single Sign-On** option automatically removes the username and password from the view and sets the **SSO Federated Login** to “true”.

This parameter can be set to “Single Sign-On” only in Oracle Utilities Testing Accelerator that is available as part of Oracle Utilities Enterprise cloud services. For Oracle Utilities Testing Accelerator on-premises installations, this option cannot be used.

Figure 8-40 Flow run authentication properties



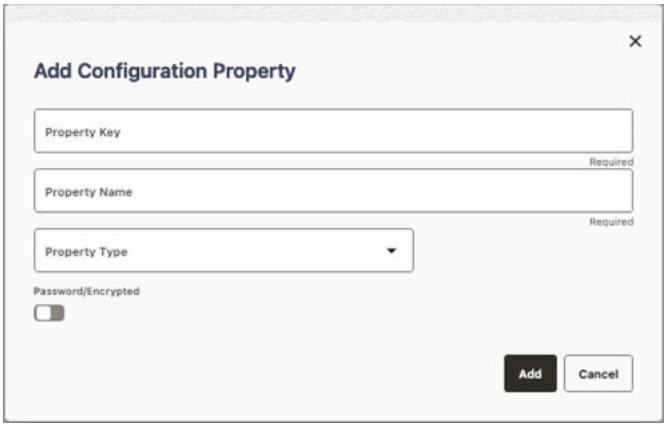
- **Application User Name**: The user name for authentication of the web services of the application being tested needs to be specified here, if the **Authentication Type** is “User Credentials”.
- **Application User Password**: The password corresponding to the above application user name for authentication of the web services of the application being tested needs to be specified here, if the **Authentication Type** is “User Credentials”.
- **SSO Federated Login**: In the flow configuration set or the user configuration set, the Single Sign On Federated Login parameter can be set to “true”, in which case, the web service request from Oracle Utilities Testing Accelerator to the corresponding Oracle Utilities cloud service will use the current user’s Single Sign On authentication, implicitly. This is automatically set to “true” if the **Authentication Type** is set to “Single Sign On”.
- Other Properties (Optional)
  - **Continue Execution on Failure**: Setting this flag to “true” will cause a flow run to execute all the components defined in the flow, even if one or more of the component steps fail. Setting this flag to “false” will ensure that the flow execution stops at the first component step that has failed, during a flow run.
  - **Environment Name**: A custom free text environment name can be provided to denote the environment against which the flows are/will be run. This name appears in the summary report so users reviewing the report know the environment the flow/flows were run against.
  - **Email Recipients**: The summary report is automatically emailed to the email addresses specified against this property. Multiple email addresses can be provided, separated by a “;”(semi-colon).

Copyright © 2026, Oracle and/or its affiliates.

- **REST Category**: Lists the properties that are specific to REST based web services.
  - isRest: Determines if the web service component used in the flow is REST or SOAP based component. Setting this to ‘true’ will ensure that the Oracle Utilities Testing Accelerator runtime triggers all component requests as REST requests during the flow run. Default is blank which is the same as ‘false’. This setting is applicable to components whose SETAPPTYPE is set to ‘WS’.
    Note that this setting does not apply to Generic REST components. The Generic REST components are always run as REST service based components.
  - hasOpenAPI: This setting will be applicable when IsRest is set to ‘true’. It is used to determine if the REST endpoint supports OpenAPI specifications. If the REST end point supports Open API, then this value needs to be set to ‘true’. Default is blank which is the same as ‘false’.
  - **Rest Authorization Type**: This property is applicable to REST based components and allows for adding authorization type in the header. Supported values: [basic|none]. Default is ‘none’.
  - **Rest Proxy**: In cases where a REST end point needs to be accessed through a proxy, this rest proxy can be set against this property.
  - **Rest Redirect**: Setting this property to ‘true’ will enable Oracle Utilities Testing Accelerator to access the any end points that are a result of redirection of the REST API request. Default value is ‘false’.
- **Custom Category**: Lists all custom properties that have been added to the configuration set.
  To create a custom property:
  1. Click **Add Property** in the **Update Flow Configuration Set** or **Update User Configuration Set** section.
  2. Enter the following details:
     - **Property Key**: An unique identifier for the property.
     - **Property Name**: A descriptive name for the property.
     - **Property Type**: Choose between “Text” and “Boolean” based on the type of property that is being added. Selecting “Boolean” as the property type will limit the user input in the flow and user configuration sets to just “true” or “false” that can be selected from a drop down.
     - **Password/Encrypted**: Enable this flag to specify if the custom property needs to be encrypted. Enabling this flag will automatically encrypt any user input against the property in the flow and user configuration sets.

Copyright © 2026, Oracle and/or its affiliates.

Figure 8-41 Configuration property settings



3. Click **Add** to create the configuration property.

The remainder of 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**.

> **Note**
>
> Important! It is recommended that sensitive information (such as passwords) be encrypted.

Copyright © 2026, Oracle and/or its affiliates.

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**.

> **Note**
>
> Important! It is recommended that sensitive information (such as passwords) be encrypted.

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.

> **Note**
>
> Important! It is required that sensitive information (such as passwords) be encrypted.

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**.

Copyright © 2026, Oracle and/or its affiliates.

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.

> **Note**
>
> Important! It is recommended that sensitive information (such as passwords) be encrypted.

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.

Copyright © 2026, Oracle and/or its affiliates.

9

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. Multiple flow sets can be run in parallel, whereas flows in a flow set will be run in the specified sequence. Unlike the flow subroutines, the flows in a flow set do not have a direct dependency on each other. The test data/outputs cannot be passed from one-flow to another, within Oracle Utilities Testing Accelerator.

This chapter focuses on flow sets including:

- Creating Flow Sets
- Adding Flows to a Flow Set
- Deleting Flows from a Flow Set
- 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:

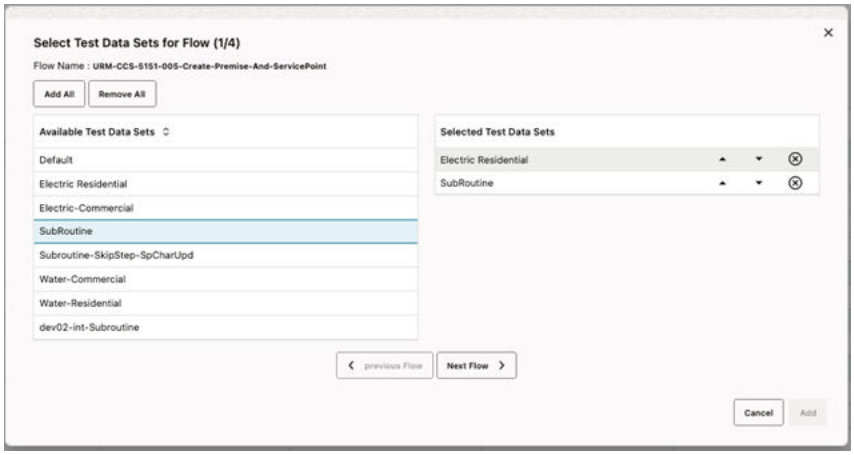
1. Login to the application.
2. Navigate to the **Flow Sets** menu.
3. On the left pane, click **Manage Flow Set**. The list of available flow sets is displayed.
4. Select the flow set name to which the flow(s) needs to be added.
5. Click **Add Flows**. You can search for one or more flows using the wildcard “%” to search for flows matching a name.

Copyright © 2026, Oracle and/or its affiliates.

Example: Search for all flows that contain the text “person” in their name by searching for string “%person%”.

6. Select the flows from the list and click **Add To Flow Set**.
7. On the **Select Test Data Sets for Flow** page, drag and drop the required flow test data sets from the **Available Test Data Set** section to the **Selected Test Data Sets** section. When the flow set is run, the selected flow will be run with all selected test data sets, one at a time, in the specified sequence.

Figure 9-1 Selecting Test Data Sets for a Flow



8. If more than one flow is added to the flow set, click **Next Flow** and select the required flow test data sets for that flow. Repeat this step for all the selected flows.
9. Click **Add**. The **Add** button is enabled after the test data sets are specified for all selected flows.
10. Click **Close** and then 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.

Copyright © 2026, Oracle and/or its affiliates.

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 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.

Copyright © 2026, Oracle and/or its affiliates.

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**.

Copyright © 2026, Oracle and/or its affiliates.

10

Creating Test Plans

This chapter focuses on test plans, including:

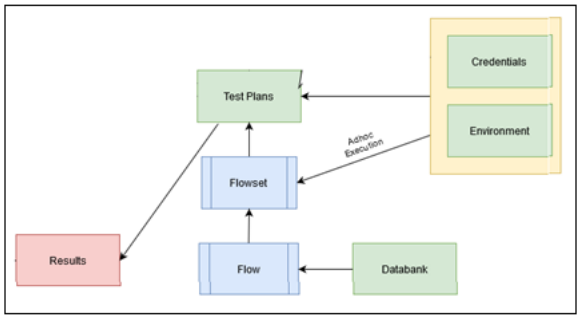
- About Test Plans
- Creating a Test Plan
- Adding and Removing Flow Sets in a Test Plan
- Managing Test Plan Lifecycle
- Running a Test Plan
- Viewing Test Plan Run Results

About Test Plans

A test plan is an object that brings all the test assets together to make the test automation process simpler and easy to use. Using a test plan, the test flows can be grouped to meet an objective (positive and negative tests/assertions) as part of testing. It can be defined using flow sets, which in turn hold all the requisite flows.

The following figure defines the test plan process.

Figure 10-1 Test plan process



Run the specified set of flows with the prescribed environment using the Credentials, On-demand.

Running the tests to meet a given objective may take multiple iterations.

Example:

- A test plan may be created to successfully generate bills for any given customer.
- The test plan may include some automated test flows that verify the objective.
- Users can run this test plan ‘n’ number of times (during sprints in agile), logging bugs/ issues during the each iterative run of the test plan.

Copyright © 2026, Oracle and/or its affiliates.

The Test Plan object encompasses the Flow Set object. One or more flow sets can be grouped together to form a single test plan. It can be executed as a whole, which provides consolidated run results of all the tests within the test plan (under the flow sets).

Creating a Test Plan

To create a test plan:

1. Login to the application.
2. Navigate to the **Test Planning** menu.
3. Click **New** under the **Test Plans** section.
4. Provide the **Test Plan Name** (a unique identifier for a test plan).
5. Provide a **Short Description** and a **Long Description** for the test plan.
6. Select the type of the test plan from the **Type** drop-down list. It helps to differentiate if the test plan pertains to new feature testing, sanity testing or other types of testing. You can also use it when searching for a test plan.
7. Click **Create**.

The test plan will be created and displayed in the **Test Plan** page. The new test plan will be in the “Planning” stage.

Adding and Removing Flow Sets in a Test Plan

To add a flow set to a test plan:

1. Login to the application.
2. Navigate to the **Test Planning** menu.
3. Search for the test plan using the search filters.
4. Select the test plan to be edited.
5. Click **Edit** next to the **Test Plan Definition**.
   Alternatively, click **Add Flow Set**.
6. In the **Edit Test Plan** screen, select the flow set to be added to the test plan and drag and drop it into the **Test Plan Definition** column.
   Make sure to select the flow set before it is dragged into the **Test Plan Definition** column.
   Use the **Flow Sets** filter to find one or more flow sets based on the flow set name.
7. Click **Save** to save the test plan definition.

To remove a flow set from a test plan:

1. Login to the application.
2. Navigate to the **Test Planning** menu.
3. Search for the test plan using the search filters.
4. Select the test plan to be edited.
5. Click **Edit** next to the **Test Plan Definition**.
6. Select the flow set and click **X** next to it in the **Test Plan Definition** column.
7. Click **Save** to save the test plan definition.

Copyright © 2026, Oracle and/or its affiliates.

Managing Test Plan Lifecycle

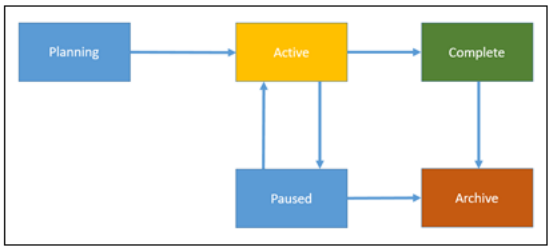
A test plan object supports various lifecycle states for easier and robust management of a test plan. Based on its lifecycle state, certain operations may or may not be allowed on the test plan.

The following lifecycle states are supported:

- Planning: This is the initial state of the test plan. All (newly created) test plans will be in the planning state. The test plan definition is allowed in this stage. Flow sets under the test plan can be added or removed when the test plan is in the planning state. The test plan cannot be run in this state, but can be deleted.
- Active: After the test plan planning is complete, to run it, it should be moved to the active state. It can be moved to active state from the planning state. An active test plan can be run. Flow sets cannot be edited/updated when the test plan is in this lifecycle state. The descriptions and type can be updated. The test plan can be moved to paused or complete state from the active state, but cannot be deleted in this state.
- Paused: If the test plan definition needs to be changed after moving it to the active state, pause it so that the test plan runs are not triggered during the update. In this state, the test plan can no longer be run. It can be edited or updated and the corresponding flow sets under the test plan can be added or removed. It can be moved to active or archive state, but cannot be deleted.
- Complete: The test plan in Complete state signifies that the runs have been completed and the objective has been met, but it still needs to be maintained in the application for tracking/reporting purposes. Updates to the test plan are no longer allowed and the runs are also not allowed. The only state allowed from this is the archived state. The test plan cannot be deleted.
- Archive: This is the end state for a test plan that allows a test plan to be deleted. The test plan updates and runs are not allowed in this state. It can be deleted.

The following figure shows the possible test plan lifecycle state transitions.

Figure 10-2 Test plan lifecycle states



To update a test plan lifecycle:

1. Login to the application.
2. Navigate to the **Test Planning** menu.
3. Search for the test plan using the search filters.
4. Select the test plan to be edited.

Copyright © 2026, Oracle and/or its affiliates.

5. Select the appropriate state from the drop-down list. The test plan will be moved to the selected lifecycle state.

Running a Test Plan

To run a test plan:

1. Login to the application.
2. Navigate to the **Test Planning** menu.
3. Search for the test plan using the search filters.
4. Select the test plan to be run (make sure that the test plan is in “Active” state).
5. Click **Run** in the **Test Plan Definition** section on the **Test Plan** page.
6. Select the **Flow Configuration Set** and the **User Configuration Set**. Click **Confirm**. The test plan run page is displayed showing the state of the flow set runs for the test plan.
7. To view detailed results of the flow set runs, click the corresponding flow set on the **Test Plan Results** page.

Viewing Test Plan Run Results

To view the run results of a test plan:

1. Login to the application.
2. Navigate to the **Test Planning** menu.
3. Search for the test plan using the search filters.
4. Select the test plan.
5. Click **Run History** in the **Test Plan Definition** section on the **Test Plan** page. The test plan run history page is displayed with the test plan run details.
6. Click the appropriate test plan run record. The **Test Plan Run** page displays the state of the flow set runs under the test plan.
7. To view the detailed results of the flow set runs, click the corresponding flow set on the **Test Plan Results** page.

Copyright © 2026, Oracle and/or its affiliates.

11

Development Accelerator Tools

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

- Component Export Tool
- Flow Export Tool
- Component/ Flow Import Tool
- Component Generation Tool

Component Export Tool

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

To export a component pack:

1. Login to the application.
2. Navigate to the **Tools** tab.
3. Click **Export Components** in the left pane.
4. Select the **Release**, **Portfolio**, **Product**, **Module**, **Component Name**, **Tags** (example: CM) and **Owner Flag** as required.
5. Click **Export**.
   A prompt appears on the screen to open or save the generated zip file “component.zip”.
6. Click **Save** to download the zip file.

The component has been exported as a .zip file.

Flow Export Tool

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

To export multiple flows at the same time:

1. Login to the application.
2. Navigate to the **Tools** tab.
3. Click **Export Flows** in the left pane.
4. Select the **Release**, **Product Family**, **Product**, **Module**, **Flow Name**, **Tags** (example: CM) and **Owner Flag** as required.
5. Click **Search**.
6. From the displayed results, select one or more flows.

Copyright © 2026, Oracle and/or its affiliates.

7. Click **Add to Export List**.
8. Repeat steps 4, 5, 6, and 7 as required and continue adding the flows to the **Export List**, as needed.
9. Click **Next**.
10. Click **Export**. There is an option to remove flows from the export list before clicking Export, if needed.
    A prompt appears on the screen to open or save the generated zip file “flow.zip”.
11. Click **Save** to download the zip file.

The flow has been exported as a .zip file.

Alternately, a single flow can be exported from Oracle Utilities Testing Accelerator:

1. Login to the application.
2. Navigate to the **Flows** tab.
3. Expand the flow tree and navigate to the flow that needs to be exported.
4. Right-click the flow name.
5. From the context menu, select **Export Flow**. A prompt appears on the screen to open or save the “flow.zip” generated zip file.
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 from another environment and to import product packs required for integration testing. The **Import** page has two tabs: **File Import** and **Product Pack Import**.

To import a component/flow file:

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.

Product Pack Import

Copyright © 2026, Oracle and/or its affiliates.

During an upgrade, Oracle Utilities Testing Accelerator automatically imports the product pack for the Oracle Utilities Enterprise cloud service that it is part of. You can import additional product packs for supported Oracle Utilities Cloud Services when they are needed for integration testing.

To import a product pack:

1. Login to the application.
2. Navigate to the **Tools** tab.
3. Click **Import** in the left pane.
4. Click the **Product Pack Import** tab.
5. Select a product from the **Product** list.
6. Select **Overwrite** to replace matching flows and components, or **Skip** to leave existing flows and components unchanged.
7. Click **Save**. A confirmation dialog displays the selected product and import option.
8. Click **Overwrite** or **Import**, as shown in the dialog, to confirm. Click **Cancel** to stop the import.

Expand **History** to view the import status and access its logs.

Component Generation Tool

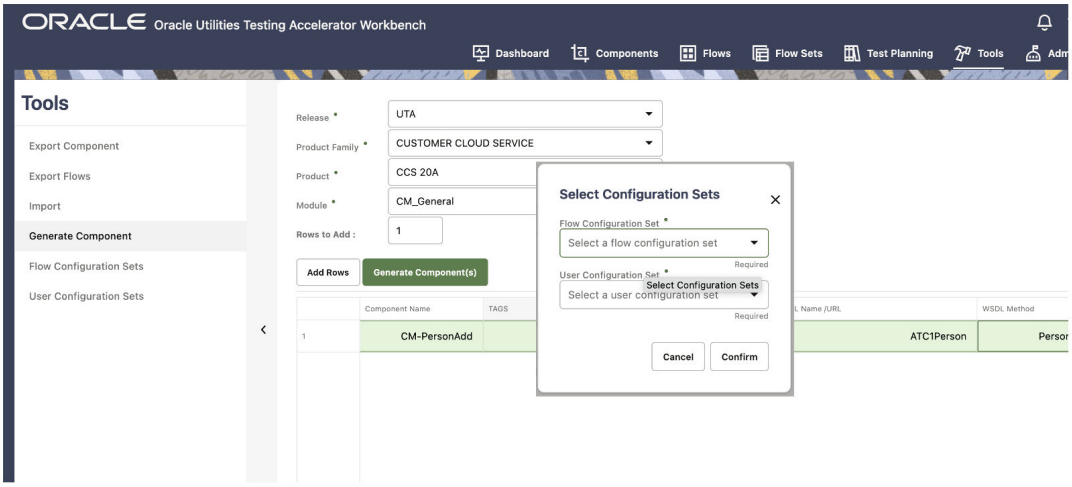
This feature is used to generate components from WSDL.

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.

Copyright © 2026, Oracle and/or its affiliates.

Figure 11-1 Configuration sets for component generation



> **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.

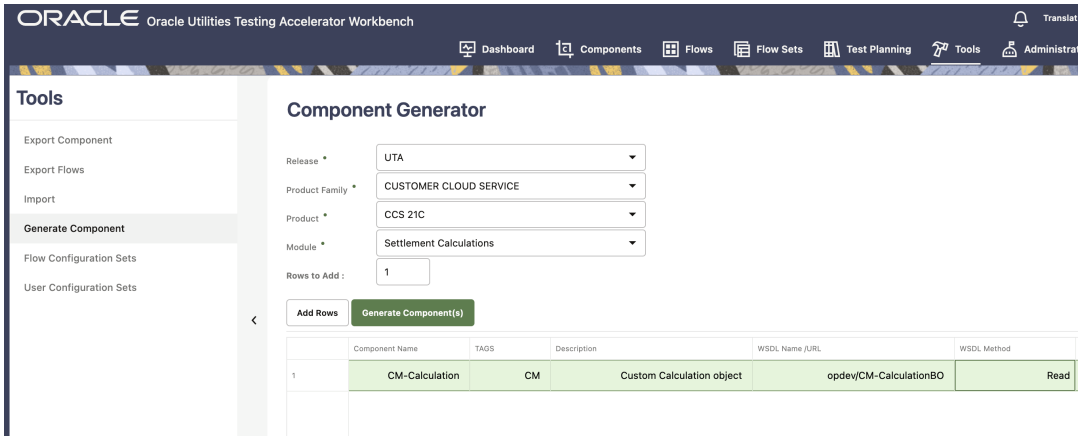
> **Note**
>
> Example: If an Oracle Utilities Meter Data Management component should be generated along with an Oracle Utilities Customer Cloud Service component, specify the three properties mentioned below either in flow or user configuration properties.

```
opdev=<MDM url>
opdev_gStrApplicationUserName=johnDoe
opdev_gStrApplicationUserPassword=enc(pj0TFjXMczsoyzmQ8GuXPt2PSydO7VCbR2jhx
tkUH06Fuz+zmChpGSCr241KggFC6FwgMg==)
```

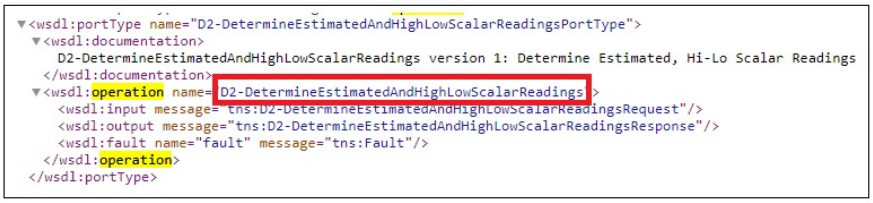
To generate a Oracle Utilities Meter Data Management component, enter the webservice name prefixed with “opdev/”.

Copyright © 2026, Oracle and/or its affiliates.

Figure 11-2 Component Generator web service settings



8. Upon successful component generation, a list of generated components and failed components is displayed.
9. To view components that were created new, navigate to the components GUI and expand the component tree structure.

> **Note**
>
> If the component generated is not visible in the tree structure, refresh the application's cache. Click the username on the top-right corner of the application and select **Clear Cache**. Refresh the web browser.
>
> > **Note**
> >
> > Tip: The **WSDL Method** column is an operation in WSDL. The following figure shows the name of operation in WSDL.
> >
> > **Figure 11-3 WSDL operation name**
> >
> > 
> >

Copyright © 2026, Oracle and/or its affiliates.

12

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
- CORERESPONSEUTILLIB
- COREDATETIMELIB
- COREDATAGENLIB
- COREVALIDATEVARIABLELIB
- COREVERIFYCONDITIONVARIABLELIB
- 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 **Output Variable Name** column, so it can be used across the flow.

```
setVariable (String valueToBeStored)
```


Copyright © 2026, Oracle and/or its affiliates.

```
Input Parameters:
valueToBeStored - value to be set/stored into a variable.

Return Type: String
```

getCurrentTimeInMilliSeconds

Gets the time in milliseconds and stores the value into a global variable specified in the **Output Variable Name** column.

Example:

```
getCurrentTimeInMilliSeconds ()

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

Randomstring

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

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

Example:

```
compare2Strings (String_stringA, String_stringB)

Input Parameters:
stringA - The first String to be compared
stringB - The second String to be compared to stringA

Return Type: String
```


Copyright © 2026, Oracle and/or its affiliates.

randomNumberUsingDateTime

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

Copyright © 2026, Oracle and/or its affiliates.

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 **Output Variable Name** column.

Example:

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

Input Parameters: 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 **Output 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
```


Copyright © 2026, Oracle and/or its affiliates.

```
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 **Output 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 **Output 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)
```


Copyright © 2026, Oracle and/or its affiliates.

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 **Output 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 **Output 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 **Output 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.

Copyright © 2026, Oracle and/or its affiliates.

Usage:

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

Example:

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

Input Parameters:
listVariableName: List of values separated by comma
index: the 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 **Output 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 **Output Variable Name** column.

Usage:

```
getCurrentMonth()

Input Parameters: none
Return Type: String
```


Copyright © 2026, Oracle and/or its affiliates.

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 **Output 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.

Usage:

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

Input Parameters:
xPath-  xpath of the element whose value needs to be checked.

Return Type: void
```


Copyright © 2026, Oracle and/or its affiliates.

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:

```
validateXpathOccurenceCount (String xpath,String expectedCount)
```

Example:

```
validateXpathOccurenceCount(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.

Copyright © 2026, Oracle and/or its affiliates.

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
```


Copyright © 2026, Oracle and/or its affiliates.

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.
```


Copyright © 2026, Oracle and/or its affiliates.

```
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 xpath,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)
```


Copyright © 2026, Oracle and/or its affiliates.

Example:

```
elementNotContains ("description", "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
```

CORERESPONSEUTILLIB

Use the RESPONSEUTILLIB 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.

Copyright © 2026, Oracle and/or its affiliates.

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

setResponseIntoVariable

This function stores the complete web service response body into a global variable specified against the **Output Variable Name** column. The output variable is used as input to the COREVALIDATEVARIABLELIB and COREVERIFYCONDITIONVARIABLELIB, which can be used to bypass or skip component as part of a run based on certain conditions.

Usage:

```
setResponseIntoVariable()

Input Parameters: NA
Return Type: String
```

setVariableFromResponseList

This function 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 **Output 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 **Output 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)
```


Copyright © 2026, Oracle and/or its affiliates.

```
Input Parameters:
String xPath, String xPath, String comparisionCondition
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>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:version>1</ouaf:version>
            </ouaf:batchJobExtraParameter>
            <ouaf:batchJobExtraParameter>
                <ouaf:batchJobId>82503583049027</ouaf:batchJobId>
```


Copyright © 2026, Oracle and/or its affiliates.

```
                <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.

Copyright © 2026, Oracle and/or its affiliates.

- 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 setVariableFromReponseListWithFilter 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 setVariableFromReponseListWithFilter 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 **Output 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, which is 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 comparisionCondition, String xPath, String
comparisionCondition

Return Type: String
```

Refer to the setVariableFromResponseListWithFilter function for details on valid comparison conditions and their usage.

getValueFromListWithIndex

This is a supporting util function for the setVariableFromReponseListWithFilter & setVariableFromReponseListWithFilters functions, which takes the a comma separated list of

Copyright © 2026, Oracle and/or its affiliates.

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 IndexNumber
Return Type: String
```

getGroupsInIntervalFromResponse

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

This 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. Further 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>F1OT</ouaf:submissionMethod>
            <ouaf:batchNumber>1</ouaf:batchNumber>
            <ouaf:batchRerunNumber>0</ouaf:batchRerunNumber>
            <ouaf:user>SYSUSER</ouaf:user>
```


Copyright © 2026, Oracle and/or its affiliates.

```
            <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.

Copyright © 2026, Oracle and/or its affiliates.

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 that will have to be retrieved and used in later components in the same flow, then this function can be used to retrieve the groups of XML elements and their values.

This 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. Further functions in this library can be used 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/>
```


Copyright © 2026, Oracle and/or its affiliates.

```
    <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: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>
```


Copyright © 2026, Oracle and/or its affiliates.

```
    </env:Body>
</env:Envelope>
```

To retrieve the first group of elements under the batchJobExtraParameter.

The function needs to be called 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 that will have to be retrieved and used in later components in the same flow, then this function can be used to retrieve the groups of XML elements and their values.

This 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 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.

Further functions in this library can be used 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 & 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 & values of repeating group
of elements for the specified number of occurrences.
```

Example:

Copyright © 2026, Oracle and/or its affiliates.

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: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>
```


Copyright © 2026, Oracle and/or its affiliates.

```
                <ouaf:batchParameterValue>K1BT</ouaf:batchParameterValue>
                <ouaf:version>1</ouaf:version>
            </ouaf:batchJobExtraParameter>
        </ouaf:ATF1BatchSubmission_READ>
    </env:Body>
</env:Envelope>
```

In order to retrieve the group of elements under the batchJobExtraParameter for first 2 occurrences of the group.

The function needs to be called 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 that will have to be retrieved and used in later components in the same flow, then this function can be used to retrieve the groups of XML elements and their values.

This function retrieves all the 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. Further functions in this library can be used 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 & values need
to be retrieved.
```


Copyright © 2026, Oracle and/or its affiliates.

```
Output Parameters: String - The XML string holding all the elements & 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>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: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>
```


Copyright © 2026, Oracle and/or its affiliates.

```
                <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.

The function needs to be called 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>
```


Copyright © 2026, Oracle and/or its affiliates.

getLastGroupFromResponse

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

This 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. Further functions in this library can be used 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 &
values need to be retrieved.

Output Parameters:
String - The XML string holding the elements & 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>
```


Copyright © 2026, Oracle and/or its affiliates.

```
                <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 last occurrence of the group.

The function needs to be called using 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

Copyright © 2026, Oracle and/or its affiliates.

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

This 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 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 &
values need to be retrieved.
        NumberOfLastNOccurrences - the 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>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>
```


Copyright © 2026, Oracle and/or its affiliates.

```
            <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 last 2 occurrences of the group.

The function needs to be called 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>
```


Copyright © 2026, Oracle and/or its affiliates.

```
            <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 that will have 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, then this function can be used to retrieve the groups of XML elements and their values.

This 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 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 &
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 & 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>F1OT</ouaf:submissionMethod>
            <ouaf:batchNumber>1</ouaf:batchNumber>
```


Copyright © 2026, Oracle and/or its affiliates.

```
            <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>
```


Copyright © 2026, Oracle and/or its affiliates.

To retrieve the group of elements under the batchJobExtraParameter parent element, whose sequence is greater than 20, the function needs to be called 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, then this function can be used 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 - The count of the number of occurrences of the specified group.
```

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.

getCurrentDatetimeWithGivenDateFormat

Copyright © 2026, Oracle and/or its affiliates.

Gets the current date and time in the specified format and stores it into a global variable specified in the **Output 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
```

getFormattedDateWithGivenDate

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

Usage:

```
getFormattedDateWithGivenDate(String sourceDatetime, 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 **Output 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 **Output Variable Name** column.

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

Copyright © 2026, Oracle and/or its affiliates.

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 **Output Variable Name** column.

Usage:

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

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

getCurrentTimeinMilliSeconds

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

Usage:

```
getCurrentDateTimeinMilliSeconds()

Input Parameters: None
Return Type: String
```

getEpochInGivenDateTimeFormat

Copyright © 2026, Oracle and/or its affiliates.

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 - Format of the date to be output by the function.
Return Type: String
```

getNthDayOfCurrentMonth

Calculates and returns the date for Nth day of the current month based on the input date format and day of the month.

Usage:

```
getNthDayOfCurrentMonth(String dFormat, String noOfDays)
getNthDayOfCurrentMonth("yy-MM-dd", "15")
```

Input Parameters:

```
dFormat - Format of the output date
noOfDays - day of the month for which the date has to be calculated
```

getNthDayOfCurrentYear

Calculates and returns the date for Nth day of the current year based on the input date format and day of the year.

Usage:

```
getNthDayOfCurrentMonth(String dFormat, String noOfDays)
getNthDayOfCurrentMonth("yy-MM-dd", "45")
```

Input Parameters:

```
dFormat - Format of the output date
noOfDays - day of the year for which the date has to be calculated
```

COREDATAGENLIB

Copyright © 2026, Oracle and/or its affiliates.

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 of lower case based on the input parameters and stores it into a global variable specified in the **Output 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 **Output Variable Name** column.

Usage:

```
randomString ()

Input Parameters: None
ReturnType: String
```

randomNumber

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

Usage:

```
randomNumber()

Input Parameters: None
ReturnType: String
```

COREVALIDATEVARIABLELIB

Copyright © 2026, Oracle and/or its affiliates.

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, the 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

Copyright © 2026, Oracle and/or its affiliates.

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 does not 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. Variable name should 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.
```


Copyright © 2026, Oracle and/or its affiliates.

```
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:
```


Copyright © 2026, Oracle and/or its affiliates.

```
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 functions 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
```


Copyright © 2026, Oracle and/or its affiliates.

elementValueLesserThanEqualTo

Verifies if the specified element value in the XML string stored in a variable is lesser than or equal the provided value. The functions 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 varibleName, 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 varibleName, String valueToCompare)
elementContains("batchName",gVarVariable, "F1-BILLING)
```


Copyright © 2026, Oracle and/or its affiliates.

```
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
```

COREVERIFYCONDITIONVARIABLELIB

This library has been designed to be used in conjunction with the skip component feature of Oracle Utilities Testing Accelerator. 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. The library validates the conditions on the response XML and outputs true or false value, which can be stored into an output global variable.

The functions in this library take a variable holding the response XML as one of the inputs and apply the validation condition on the XML in the variable.

Copyright © 2026, Oracle and/or its affiliates.

The response of a component step can be stored in to a global variable using the "setResponseIntoVariable" function in the CORERESPONSEUTILLIB library.

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 variable holding the response XML and 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 will return false, else it will return true.

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: String (true|false)
```

elementListNull

Verifies if all the elements with the specified xpath are null in the XML string stored in a variable. The variable holding the response XML and the xpath to be verified needs to be provided in the value columns in the pre/post validations sections. If the value is NOT null, this function will return 'false', else it will return 'true'.

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: String (true|false)

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. It returns 'false' if the count in the response does not match the specified number, else it returns 'true'.

Copyright © 2026, Oracle and/or its affiliates.

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: string (true|false)
```

elementNotNull

If the specified element in the XML string stored in a variable is null, this function returns ‘false’, else it returns ‘true’.

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: String (true|false)
```

elementIsNull

If the specified element in the XML string stored in a variable is null, the function returns ‘true’, else it returns ‘false’.

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: String (true|false)
```


Copyright © 2026, Oracle and/or its affiliates.

elementValueEquals

If the specified element value in the XML string stored in a variable is equal to the provided value, the function returns ‘true’, else it returns ‘false’.

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: String {true|false)
```

elementValueNotEquals

If the specified element value in the XML string stored in a variable is not equal to the provided value, the function returns ‘true’, else it returns ‘false’.

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: String (true|false)
```

elementValueGreaterThan

If the specified element value in the XML string stored in a variable is greater than the provided value, the function returns ‘true’, else it returns ‘false’.

Usage:

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

Input Parameters:
```


Copyright © 2026, Oracle and/or its affiliates.

```
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: String (true|false)
```

elementValueGreaterThanEqualTo

If the specified element value in the XML string stored in a variable is greater than or equal the provided value, the function returns ‘true’, else it returns ‘false’.

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: String (true|false)
```

elementValueLesserThan

If the specified element value in the XML string stored in a variable is less than the provided value, the function returns ‘true’, else it returns ‘false’.

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: String (true|false)
```


Copyright © 2026, Oracle and/or its affiliates.

elementValueLesserThanEqualTo

If the specified element value in the XML string stored in a variable is lesser than or equal the provided value, then the function returns false, else it returns true.

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: String (true|false)
```

elementContains

If the specified element contains the specified value in the XML string stored in a variable, the function returns ‘true’. Else, it returns ‘false’.

Usage:

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


Copyright © 2026, Oracle and/or its affiliates.

```
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

If the specified value or element is not available in the XML string held in the variable, the function returns true. Else, it returns false.

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: String (true|false)
```

variableContains

If the specified value or element is available in the XML string held in a variable, the function returns true. Else, it returns false.

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: String {true|false)
```

CORESTOREVALUES

Use this function library to store values into a variable.

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

setVariable

Copyright © 2026, Oracle and/or its affiliates.

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

Usage:

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

Input Parameters:
valueToStore - The value to be stored in a global variable.

Return Type: String
```

appendValueToList

This function appends a value to a list of values. The input to the function can either be a value or a variable. The output will be a list stored into a global variable. The function can be used as part of subroutine looping to store a list of values into the function, one from each iteration of the subroutine.

Example: Consider a subroutine that has a component that adds a new meter read into Oracle Utilities Meter Solution Cloud Service, for each loop that it runs for. Oracle Utilities Meter Solution Cloud Service,responds with the meterReadId for each of the meter read add transactions, which can be stored as a list of values i.e., the list of meterReadIds that are generated as part of subroutine looping. MeterReadId can be extracted from the response into a variable using coreresponseutilib functions and then this variable can be used as input to the appendValueToList function.

Users can also use the value of the incrementer from the subroutine looping data as the input.

Stores the list of values in a global variable specified in the **OP Variable Name** column.

Usage:

```
appendValueToList (String variable)
```

Input Parameters:

```
Variable: The value to be appended to the list
Return Type: String
```

getListValueUsingIndex

Fetches a particular value from the a list of values based on the provided index. This function complements the appendValueToList function in that it allows for extracting individual values from the list of values generated by the appendValueToList function. The input to the function is the index of the value that we need.

Stores the fetched value from the list into a global variable specified in the **OP Variable Name** column.

Copyright © 2026, Oracle and/or its affiliates.

Usage:

```
getListValueUsingIndex (String index)
```

Input Parameters:

```
index: Index of the value to be retrieved from the list
VariableName: Variable holding the list of values

Return Type: String
```

appendKeyValueToList

Creates a key based list of values, which users can later use to retrieve specific values. This is an extension of the appendValueToList function wherein the index has been replaced with a custom key.

The function appends a key and a value/variable to a map. The input to the function will be a key and a corresponding value/variable. The output will be a map stored into a global variable.

An example is a subroutine iteration that needs to create activities and complete them. After the completion of the subroutine loop run with 10 iterations that create 10 activities, subsequent steps in the parent flow will be able to get the ID of an activity that could not be completed or that remained in the started state.

Usage:

```
appendKeyValueToList (String key,String value)
```

Input Parameters:

```
key: The key for the corresponding value
value: The value to be added along with the key to the map
Return Type: map
```

getListValueUsingKey

Fetches a particular value from the key value pair map generated by appendKeyValueToList function based on the provided key. The input to the function is a key.

Stores the fetched value from the map into a global variable specified in the **OP Variable Name** column.

Usage:

```
getListValueUsingKey (String key)
```


Copyright © 2026, Oracle and/or its affiliates.

Input Parameters:

```
key: The key corresponding to a value that needs to be fetched from the map
Return Type: String
```

COREFILEOPS

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

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

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 **Output 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: The name of file in flow attachments
fileEncoding: The encoding of the file being read

Return Type: String
```

CORESTRINGOPS

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

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

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 **Output 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:
```


Copyright © 2026, Oracle and/or its affiliates.

```
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 **Output 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 **Output 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

Copyright © 2026, Oracle and/or its affiliates.

Gets the product of two input numbers specified in the value 1 & value 2 columns and stores the output into a global variable specified in the **Output 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 & value 2 columns and stores the output into a global variable specified in the **Output 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 of input numbers specified in the value 1 & value 2 columns and stores the content as byte array into a global variable specified in the **Output 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 & value 2 columns and stores the output into a global variable specified in the **Output Variable Name** column.

Copyright © 2026, Oracle and/or its affiliates.

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 & value 2 columns and stores the output into a global variable specified in the **Output 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 **Output 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 Testing Accelerator, such as pausing a flow run for a specified time, conditional constructs to exit polling of a specific component Inbound Web Services.

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

Copyright © 2026, Oracle and/or its affiliates.

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
```

customLog

This function can be used to add custom messages to the flow run summary report. This function Concatenates the input parameters and adds the final concatenated message to the flow execution summary report.

Usage:

```
customLog(String message1,String message2)

Input Parameters:
message1: the string that needs to be printed into to the summary report
message2: the suffix to the message1 that needs to be appended to the summary
report

Return Type: void
```

> **Note**
>
> If more than two strings need to be appended and printed to the output, use the appendStrings function from CORESTRINGOPS and store the output into a global variable and use the global variable as a parameter to customLog function. If only one message needs to be printed and there is no suffix to it, you can set #EMPTY in the 2nd input parameter field.

Copyright © 2026, Oracle and/or its affiliates.

13

Custom Libraries

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

Note the following:

- Only Java Script language can be used to develop new custom function libraries.
- Due to security constraints, only a few approved Java Script packages are allowed in custom libraries.
- Starting release 22A, Groovy script based libraries are not supported in Oracle Utilities Testing Accelerator. Custom libraries created using Groovy have to be manually migrated to Java Script.

The chapter includes the following:

- 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. Based on the library type selected, it can only be used for web services based components, REST based components, or all type of components.
5. Click **Create**.

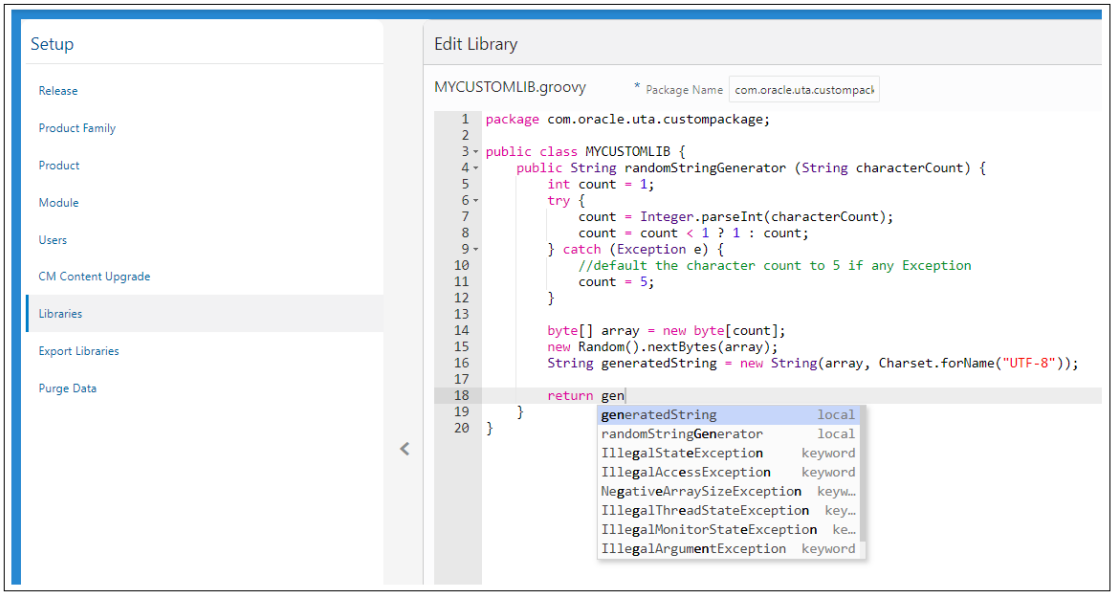
> **Note**
>
> This step only creates a header definition of the library, the actual code supporting/ implementing the definition is expected to have been already developed using an IDE like Eclipse or JavaScript consoles. See the example at the end of this section for more details.

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), comma separated comments for each of the input parameters, which gets displayed against the parameter in the test data screen and description of the function.

Copyright © 2026, Oracle and/or its affiliates.

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.
   - **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/code a Java Script library 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.
13. Click **Save**. This triggers the compilation of the custom library Java Script code and displays errors if any. Rectify the code and click **Save** again to verify and save the changes.
14. Click **Save** to overwrite any existing library with an updated version or a new library being created.

Figure 13-1 Groovy library editor



Example: Creating a Java Script 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.

Copyright © 2026, Oracle and/or its affiliates.

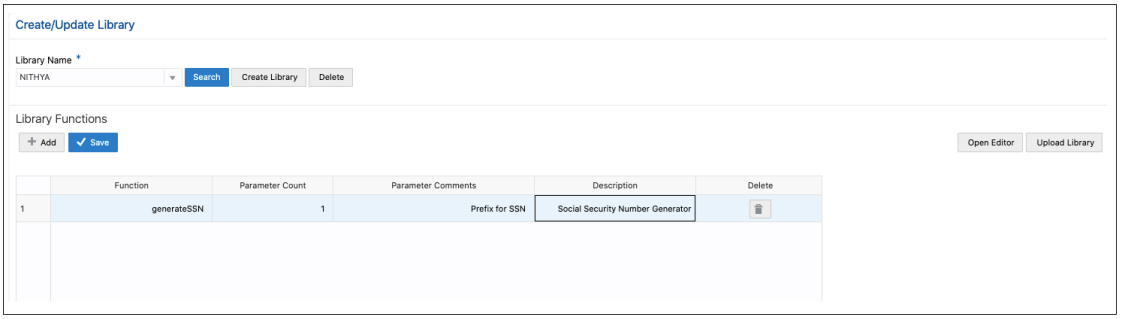
The script contents are as follows:

```
var ArrayList = Java.type('java.util.ArrayList');
var List = Java.type('java.util.List');
var Random = Java.type('java.util.Random');
var OUTSPCORELIB = Java.type('com.oracle.utilities.core.lib.OUTSPCORELIB');

function generateSSN(prefix){
        var random = new Random();
        var x = random.nextInt(900) + 100;
        var y = random.nextInt(90) + 10;
        var z = random.nextInt(9000) + 1000;
        var zz = x+"-"+y+"-"+z;
        print(prefix+zz);
        return prefix+zz;
}
```

Below is the function definition in Oracle Utilities Testing Accelerator:

Figure 13-2 Custom library function definition



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, click **Export**. An export file of the custom function library currently displayed is generated.

To export multiple custom libraries:

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

Copyright © 2026, Oracle and/or its affiliates.

2. Navigate to the **Administration** tab and click **Export Libraries** on the left pane.
3. Search for the library based on names and 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 is 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 can the library functions in any of the components/flows. The usage is similar to the usage of base function libraries.

Copyright © 2026, Oracle and/or its affiliates.

14

User Management

This chapter focuses on user settings that include user management, flushing cache, language and timezone selections. It focuses on the following:

- Selecting User Time Zone
- Selecting User Language
- Removing User
- Clearing Server Side Cache

Selecting User Time Zone

Oracle Utilities Testing Accelerator supports viewing the date time values on the application's workbench in multiple time zones.

To support a user time zone:

1. Login to Oracle Utilities Testing Accelerator.
2. Click the drop-down menu having your username on the top-right corner of the workbench and select **Settings**.
3. Select the time zone from the **Time Zone** drop-down menu.
4. Click **Save**.

Selecting User Language

Oracle Utilities Testing Accelerator supports viewing of the application's workbench in multiple languages.

1. Login to Oracle Utilities Testing Accelerator.
2. Click the drop-down menu having your username on the top-right corner of the workbench and select **Settings**.
3. Select language from the **Language** drop-down list.
4. Click **Save**.

> **Note**
>
> The Language options drop-down list shows the available languages only if your Oracle Utilities Testing Accelerator instance supports multiple languages. Else, the drop-down list will be empty.

Copyright © 2026, Oracle and/or its affiliates.

Removing User

This allows removal of all references to a specific inactive user from all objects in Oracle Utilities Testing Accelerator. A user who does not have requisite privileges in the identity management service to access Oracle Utilities Testing Accelerator is considered as an inactive user.

1. Login to Oracle Utilities Testing Accelerator.
2. On the **Administration** tab of the workbench, select **Users**.
3. Search for the inactive user and click the delete icon corresponding to the user record.
4. Select the **Confirmation** checkbox and click **Delete**.

> **Note**
>
> Users cannot be created or updated in Oracle Utilities Testing Accelerator. Active users cannot be removed from Oracle Utilities Testing Accelerator. All user management is through the identity management service. This feature is supports the removal of references of any non-existing users.

Clearing Server Side Cache

The Oracle Utilities Testing Accelerator metadata is cached on the server side to ensure faster load times of the application.

To flush the server side cache:

1. Login to Oracle Utilities Testing Accelerator.
2. Click the drop-down menu showing your user name on the top-right corner of the workbench.
3. Select **Clear Cache** from the drop-down menu.
4. Click **Yes** to confirm.

Copyright © 2026, Oracle and/or its affiliates.

A

Web Service Component Keywords

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

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

WS-SETWEBSERVICENAME

Sets the name of the application web service.

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

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

WS-SETXMLELEMENT

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

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

Usage Details	Value
Keyword	WS-SETXMLELEMENT

Copyright © 2026, Oracle and/or its affiliates.

Usage Details	Value
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

Copyright © 2026, Oracle and/or its affiliates.

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
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.

Copyright © 2026, Oracle and/or its affiliates.

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

Copyright © 2026, Oracle and/or its affiliates.

B

REST Component Keywords

This chapter provides the following REST component keywords:

- RS-SETREQUESTHEADER
- RS-SETENDPOINT
- RS-ARGUMENT
- RS-SETMETHOD
- RS-PROCESSRESTREQUEST

RS-SETREQUESTHEADER

Sets the header in the defined REST request.

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

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

RS-SETENDPOINT

Sets the endpoint for the REST request.

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

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

RS-ARGUMENT

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

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

Copyright © 2026, Oracle and/or its affiliates.

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

Copyright © 2026, Oracle and/or its affiliates.

C

Setting Up Inbound Web Services

The Oracle Utilities application-specific components are developed using the web services method, and these components need the Inbound Web Services to be defined in the application.

This chapter includes the following sections:

- Creating Inbound Web Services
- Importing Inbound Web Services
- Searching Inbound Web Services

Creating Inbound Web Services

To create an Inbound Web Service for any custom component to be created:

1. Login to the Oracle Utilities cloud service.
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. Enter the **Operation Name**.
7. Select the **Schema Type**, **Schema Name** and **Transaction Type**.
8. Click **Save**.

Importing Inbound Web Services

To import the delivered Inbound Web Services corresponding to the components into the Oracle Utilities cloud service:

1. Login to the Oracle Utilities Testing Accelerator.
2. Click the **Flows** tab on the top-right section.
3. Expand the flow tree structure corresponding to the Oracle Utilities cloud service version.
4. Right-click the **Pre-requisites** flow for the respective product version.
5. Select **Run Flow**.
6. Select the **Flow** and **User Configuration** sets and click **Run**.
7. The flow should successfully run to completion with status as ‘Passed’. The corresponding Inbound Web Services will be imported into Oracle Utilities cloud service.

Copyright © 2026, Oracle and/or its affiliates.

Searching Inbound Web Services

To search an Inbound Web Service in an Oracle Utilities cloud service:

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.

Copyright © 2026, Oracle and/or its affiliates.

D

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	vancara
		3?appl	applxtg
		6?AC	ACkdbvdl
		2?	nd
		?	ufdbn
A specified number of random upper case characters need to be appended to the given test data.	<int>U?data	4U?van	vanCARA
		3u?appl	applXTG
		6U?AC	ACKDBVDL
		2U?	ND
		U?	UFDBN
A specified number of random lower case characters need to be prefixed to the given test data.	<int>B?data	4B?van	caravan
		3b?appl	xtgappl
		6B?AC	kdbvdlAC
		2B?	nd
		B?	ufdbn
A specified number of random upper case characters need to be prefixed to the given test data.	<int>BU?data	4BU?van	CARAvan
		3bu?appl	XTGappl
		6Bu?AC	KDBVDLAC
		2BU?	ND
		BU?	UFDBN
A specified number of random numbers need to prefixed to the given test data	<int> d?data	d?ABCD	ABCD32940
		2d?ABCD	ABCD43
A specified number of random numbers need to suffixed to the given test data	<int> bd?data	bd?ABCD	32940ABCD
		4bd?ABCD	1534ABCD

Copyright © 2026, Oracle and/or its affiliates.

E

Configuring Authentication for Web Service Requests

Based on the version of the Oracle Utilities application (and the Oracle Utilities Application Framework), the web service requests are expected to include additional information apart from the user credentials. In order to support this, two new properties have been introduced in the flow and user configuration sets 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 user or flow configuration sets contain 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
```


Copyright © 2026, Oracle and/or its affiliates.

```
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
```


Copyright © 2026, Oracle and/or its affiliates.

F

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
- Next Generation REST APIs
- Legacy REST APIs
- 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 a base64 encoded username and password as one of the parameters on the request header for authorization.

You can obtain the base64 encoded username and password using the following command on Linux:

```
echo -n '<username>:<password>' | base64
```

Where username is the username that needs to be encoded and password is the password corresponding to the username that needs to be encoded.

Next Generation REST APIs

The next generation of REST APIs to invoke the Oracle Utilities Testing Accelerator flow executions and retrieve results supports Open API specifications.

The authentication mechanism is the same for the next generation REST APIs as those used for the older REST APIs. For more information, refer to the Prerequisites section. REST APIs can be invoked remotely using commands (such as curl) depending on the application that supports the commands.

To access Open API specifications for next generation REST APIs:

Copyright © 2026, Oracle and/or its affiliates.

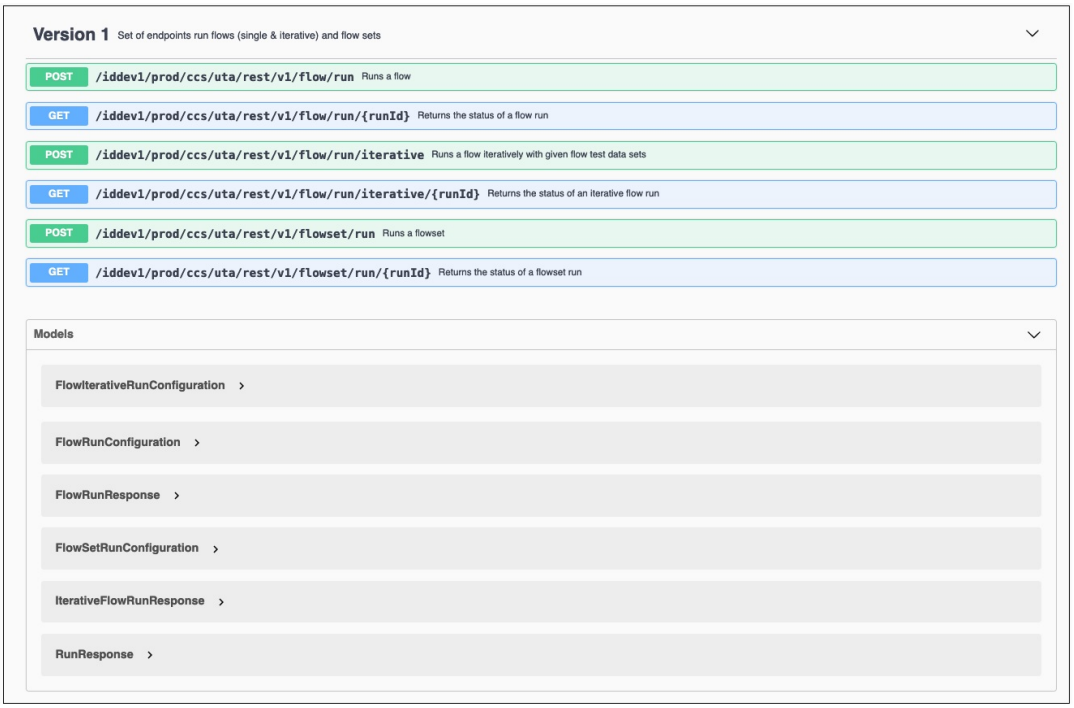
1. Open the following URL in a supported web browser:

   ```
   <UTA URL>/swagger-ui/
   ```

   Example: If the Oracle Utilities Testing Accelerator URL is https://someoraclecloudserver.com/prod/ccs/uta/, the corresponding swagger UI URL will be https://someoraclecloudserver.com/prod/ccs/uta/swagger-ui/.

2. The REST API specifications are documented on the page. The following figure shows Swagger UI with Open API specification for UTA REST APIs.

Figure F-1 REST API operations in Swagger UI



Example curl command for running a UTA flow:

```
curl -k --request POST 'https:// someoraclecloudserver:8086/prod/ccs/uta/
rest/v1/flow/run' \
--header 'Authorization: Basic U1lTVVNFUjpXZWxjb21lMTIzNDU2' \
--header 'Content-Type: application/json' \
--data '{
  "configuration": "SCHED_54",
  "flow": "QA-ToDoComplete",
  "flowTestDataSet": "default",
  "identity": "SCHED_54",
  "portfolio": "CORE",
  "product": "CORE",
  "release": "UTA"
}'
```

The list of available APIs and the corresponding field definitions can be found in Oracle Utilities Testing Accelerator's API specification.

Copyright © 2026, Oracle and/or its affiliates.

Legacy REST APIs

It is recommended to use the next generation REST APIs described in the Next Generation REST APIs section to invoke Oracle Utilities Testing Accelerator flow runs and retrieving results using REST APIs.

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>/…../ uta/rest/run/flow' -H
'authorization:Basic <Base64 encoded username & password>' -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

Copyright © 2026, Oracle and/or its affiliates.

- <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/dev01/ccs/uta/rest/execute/flow -H
'authorization:  Basic bXJpbmFsOk9yYWNsZTEyMw==' -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:Basic <Base64
encoded username & password>'
```

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:  Basic
bXJpbmFsOk9yYWNsZTEyMw=='
```


Copyright © 2026, Oracle and/or its affiliates.

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>/…../ uta/rest/run/flow' -H
'authorization:Basic <Base64 encoded username & password>' -H 'cache-control:
no-cache' -H 'content-type: application/json' -d '{ "executionType" :
"flowSet",
"flowSet" : "<flowSet>", "configuration":"<configuration>",
"identity": "<user configuration>"}'
```

Parameters

- <hostname>: Oracle Utilities Testing Accelerator application host name
- <port>: Oracle Utilities Testing Accelerator 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:  Basic bXJpbmFsOk9yYWNsZTEyMw==' -H 'content-type:
application/json' -d
'{"executionType": "flowSet",  "flowSet" : "MyBillingTestSuite",
"configuration":"myCCBConfig", "identity": "myCCBConfig"  }'
```


Copyright © 2026, Oracle and/or its affiliates.

Once a flow set run is triggered, the service responds with a response as follows:

```
[{"flowSetExecutionId":"502","flowSetName":"MyBillingTestSuite","status":"Runn
ing"}]
```

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:Basic <Base64 encoded username & password>'
```

Parameters

- <hostname>: Oracle Utilities Testing Accelerator Application host name
- <port>: Oracle Utilities Testing Accelerator 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:  Basic bXJpbmFsOk9yYWNsZTEyMw=='
```

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.

Copyright © 2026, Oracle and/or its affiliates.

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: Basic <Base64 encoded username & password>'
```

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://<hostname>:<port>/rest/analytics/user/admin/
flowExecutionAnalytics/from/01-JAN-20/to/15-JAN-20 -H 'authorization: basic
62593bbd-b057-4f38-8a3e-5915d4ab559f'
```

Once a flow set run is triggered, the service responds with a response as follows.

```
{"totalFlowsRanByUser":50,"totalFlowsRanByUserSuccess":48,"totalFlowsRanByUser
Failure":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
```


Copyright © 2026, Oracle and/or its affiliates.

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: Basic <Base64 encoded username & password>'
```

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:
Basic bXJpbmFsOk9yYWNsZTEyMw=='
```

Once a flow set run is triggered, the service responds with a response as follows:

```
{"totalFlowSetsRanByUser":30,"totalFlowSetsRanByUserSuccess":29,"totalFlowSets
RanByUserFailure":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: Basic <Base64 encoded username & password>' \
-H 'authorization:Basic <Base64 encoded username & password>' \
```


Copyright © 2026, Oracle and/or its affiliates.

```
  -d '["<flow execution id>"]'
```

Parameters

- <hostname>: Oracle Utilities Testing Accelerator application host name
- <port>: Oracle Utilities Testing Accelerator 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://<hostname>:<port>/rest/summary/report?type=flow&flowName=x' \
  -H 'accept: application/xml' \
 -H 'authorization:  Basic bXJpbmFsOk9yYWNsZTEyMw==' \
  -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:Basic <Base64 encoded username & password>' \
  -H 'content-type: application/json' \
  -d '["<flow execution id>"]'
```

Parameters

- <hostname>: Oracle Utilities Testing Accelerator application host name
- <port>: Oracle Utilities Testing Accelerator application port
- < token>: Authentication token from the steps mentioned in pre-requisites above

Copyright © 2026, Oracle and/or its affiliates.

- <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://<hostname>:<port>/rest/summary/report?type=flowset&flowSetName=x' \
  -H 'accept: application/xml' \
  -H 'authorization: Basic <Base64 encoded username & password>' \
  -H 'content-type: application/json' \
  -d '["2507"]'
```

The server responds with a HTML summary file of the flow set as response.

Copyright © 2026, Oracle and/or its affiliates.