Oracle Functional Testing Advanced Pack for Oracle Utilities

User's Guide

Release 5.0.0

E67845-01

October 2015

ORACLE

Oracle Functional Testing Advanced Pack for Oracle Utilities User's Guide, Release 5.0.0

E67845-01

Contents

Preface

Welcome to the Oracle Functional Testing Advanced Pack for Oracle Utilities (OFTAPOU) User's Guide. This guide explains how to use Oracle Functional Testing Advanced Pack for Oracle Utilities to automate the business test flows for testing the Oracle Utilities' applications.

Oracle Functional Testing Advanced Pack for Oracle Utilities is a licensed product and requires Oracle Functional Tester (OFT).

This preface includes the following:

Audience

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

Prerequisite Knowledge

This guide does require an understanding of software testing concepts. The users must be familiar with Oracle Flow Builder.

> **Note**: *Oracle Flow Builder User's Guide* can be downloaded from Oracle Technology Network (http://www.oracle.com/technetwork/oem/downloads/index-084446.html).

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

Related Documents

For more information, see the following documents in the Oracle Functional Testing Advanced Pack for Oracle Utilities documentation set:

- *Oracle Functional Testing Advanced Pack for Oracle Utilities Release Notes*
- *Oracle Functional Testing Advanced Pack for Oracle Utilities Installation and Administration Guide*

- *Oracle Functional Testing Advanced Pack for Oracle Utilities Reference Guide for Oracle Utilities Application Framework*
- *Oracle Functional Testing Advanced Pack for Oracle Utilities Reference Guide for Oracle Utilities Mobile Workforce Management and Oracle Real-Time Scheduler*
- *Oracle Functional Testing Advanced Pack for Oracle Utilities Reference Guide for Oracle Utilities Customer Care and Billing*
- *Oracle Functional Testing Advanced Pack for Oracle Utilities Reference Guide for Oracle Utilities Work and Asset Management*

See also:

- Oracle Application Testing Suite Documentation Library
- Oracle Functional Testing OpenScript Documentation Library

Notational Conventions

The following text conventions are used in this document:

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

Chapter 1

Overview

This chapter introduces the Oracle Functional Testing Advanced Pack for Oracle Utilities application and provides an overview of the application architecture.

Introduction

Oracle Functional Testing Advanced Pack for Oracle Utilities (OFTAPOU) comprises test automation accelerators for automated testing of the Oracle Utilities applications. It is a framework based on Oracle Functional Tester (OFT) for creating the Web services automation scripts.

Oracle Functional Testing Advanced Pack for Oracle Utilities enables users to create the automation scripts using keywords or metadata, and without using any programming language. This saves the test automation development effort and avoid programming the scripts using OpenScript Workbench.

> **Note**: Oracle Functional Tester and OpenScript are part of Oracle Application Testing Suite. See the *Oracle Functional Tester User Guide* for more information.

The accelerators contain out-of-the-box delivered test components that can be used to build test flows for the Oracle Utilities applications. Users can extend the delivered components or create a new component to build their customized test flows. Utilities' application specific sample test flows are provided in the respective reference guides.

This introduction includes the following sections:

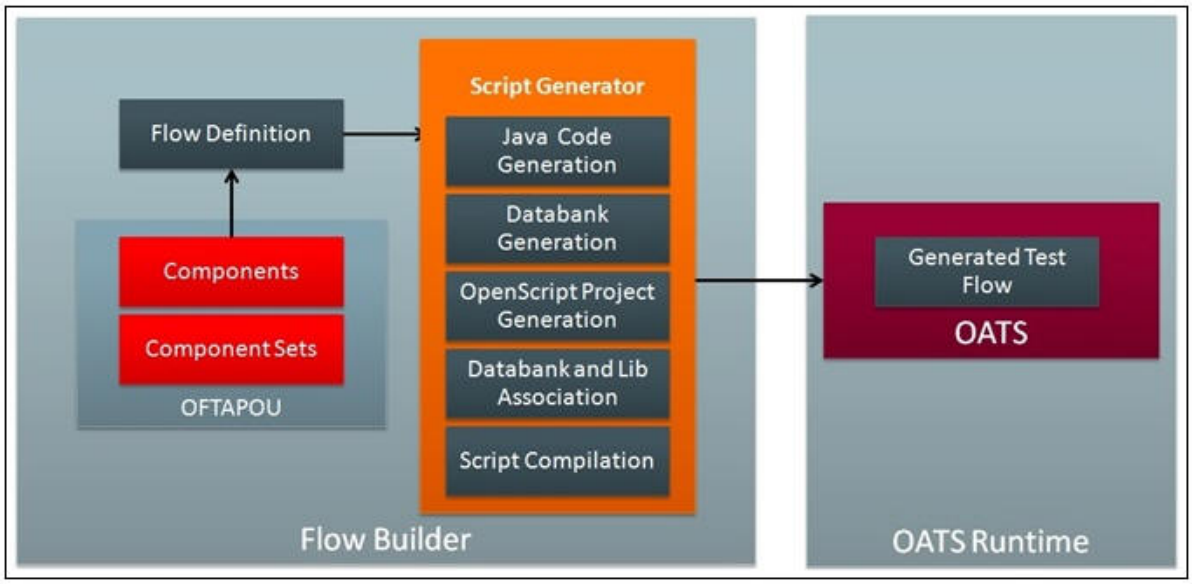
Terminology

This section lists the different terms used in the document.

Term	Description
Oracle Flow Builder (OFB)	Helps to build and maintain components and flows for automated testing. This tool is part of Oracle Application Testing Suite. **Note**: See the *Oracle Flow Builder User's Guide* for more information.
Keyword	Pre-defined set of words used to define a specific step in a test case.
Component Line	Represents a step in a test case defined by a keyword, and associates values and parameters.
Component	Reusable automated test or part of a test. A component is the building block of an automated test flow, comprising one or more component lines.
Component Set	Set of reusable group of components arranged in a pre-determined order. A component set is used to perform a set of repeatable tasks. Component sets, along with components, are used to define a flow.
Flow	Automated test A flow comprises one or more components and/or component sets that are called in a pre-determined sequence.
Databank	Container of test data used by an automated test flow. The databank is defined using comma separated values (.csv) in a text file.

Application Architecture

Below is a high-level architecture diagram for Oracle Functional Testing Advanced Pack for Oracle Utilities.



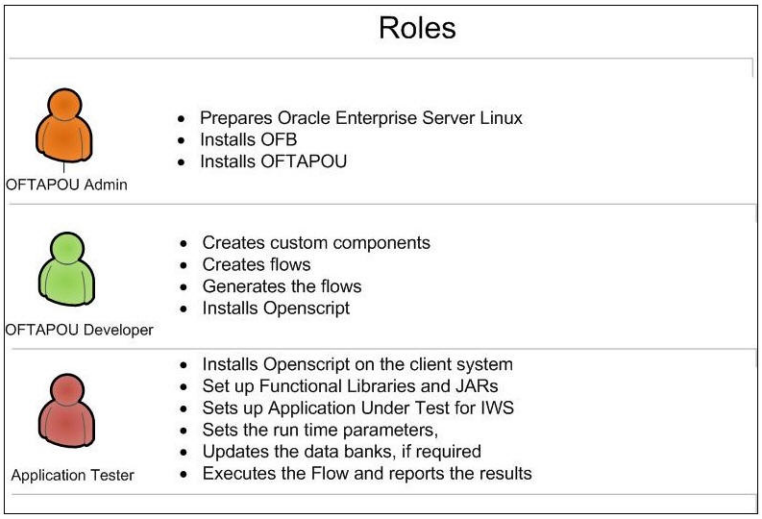
High-Level Architecture Diagram

Components and component sets are defined using metadata in Oracle Flow Builder. Using these components a flow can be assembled and then generated. The generated script, then, can be used and executed using Oracle Functional Tester.

For more information about Oracle Functional Testing Advanced Pack for Oracle Utilities see the *Oracle Functional Testing Advanced Pack for Oracle Utilities Installation and Administration Guide.*

Understanding the Roles

The following diagram shows various roles and the setup tasks performed by each of the roles.



Roles and Tasks

Pointers for Getting Started

This section provides the references to get started with the Oracle Functional Testing Advanced Pack for Oracle Utilities application.

Installing Oracle Functional Testing Advanced Pack for Oracle Utilities

See *Oracle Functional Testing Advanced Pack for Oracle Utilities Installation and Administration Guide* for detailed installation instructions.

Administrative Setup

See *Oracle Functional Testing Advanced Pack for Oracle Utilities Installation and Administration Guide* for detailed administrative setup instructions.

Developing Test Automation

See Chapter 3: Developing Metadata Driven Web Service Based Test Automation for instructions on how to develop metadata driven Web service based automation tests using Oracle Functional Testing Advanced Pack for Oracle Utilities.

Product Components Reference

See the Oracle Utilities product specific component reference guide for more information.

Also, see the *Oracle Functional Testing Advanced Pack for Oracle Utilities Release Notes* for the products included in this Oracle Functional Testing Advanced Pack for Oracle Utilities release.

Chapter 2

Developing Metadata Driven Web Service Based Test Automation

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

This chapter is intended primarily for Automation Developers and Testers. It describes the procedures to create components and component sets, and to create and execute test work flows.

Metadata Driven Automation Development Methodology

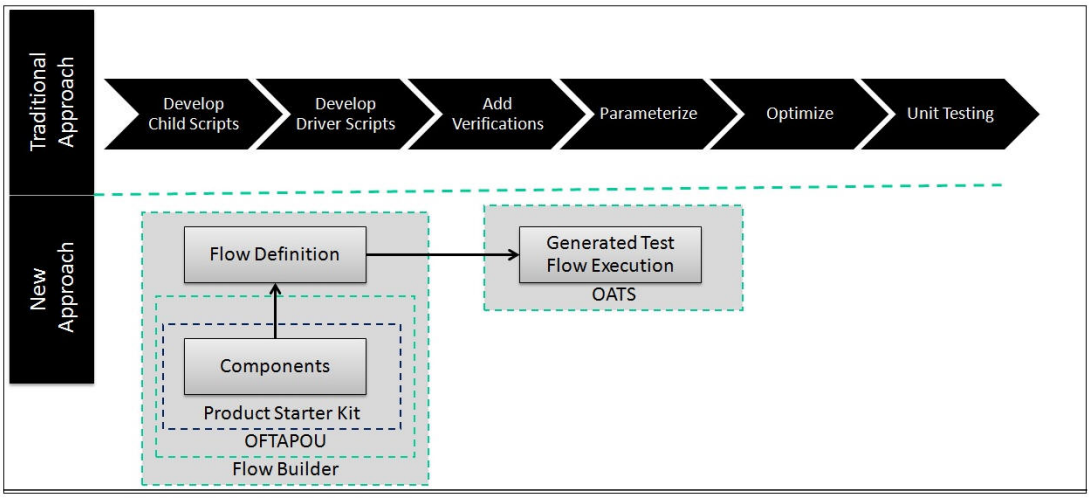
This section describes the metadata-driven automation development methodology that enables a test automation engineer to create automation scripts for an Oracle Utilities application.

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

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

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

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



Development Methodology

For the Oracle Utilities applications built on Oracle Utilities Application Framework, Web service based automated testing is proven to be more robust, maintainable, and faster to develop and execute. Hence, Oracle Functional Testing Advanced Pack for Oracle Utilities comprises Web services based components that enable creation of test flows and executing the same. UI based automation components can also be created using Oracle Flow Builder. See the *Oracle Flow Builder User's Guide* for more information.

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

Planning

To plan an automated test flow, identify the business test flow to be automated and the components required for the flow. If necessary, create additional components or extend the delivered components. See the Extending Components section for details on how to extend the components.

Design and Development

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

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

1. Create/extend the required components that are identified in planning phase.
2. Create a test flow in Oracle Flow Builder that maps to the identified business test flow in the application.

   See the Creating Test Flows section for details on how to create a test flow. See the **Sample Work Flows** chapter in the respective product specific reference guides for delivered sample flows to understand the flow creation.
3. Drag and drop the required components into the flow.

4. Add the test data for the flow.

   The test data can be modified at the runtime using the standard OpenScript databanks. See the Test Data Management section for more details.

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

Test Execution

To execute the automated test flow, execute the script in OpenScript Workbench.

To use another data set to execute the script, change the databanks in the generated scripts project, and then execute the script. See the Executing Test Flows section for more details.

The components and test flows developed using this approach are stored and version controlled in the Oracle Flow Builder database. It takes care of the challenges in configuration management of automated tests. This methodology and framework works only with Oracle Functional Tester.

Setting Up Automation Development Environment

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

- Step 1: Setting Up the OFTAPOU Server
- Step 2: Setting Up Workstations for Development
- Step 3: Setting Up Application Under Test
- Step 4: Setting Up Workstations for Testing

Step 1: Setting Up the OFTAPOU Server

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

- Installing Oracle Flow Builder
- Installing Oracle Functional Testing Advanced Pack for Oracle Utilities
- Administrative Tasks

Installing Oracle Flow Builder

Ensure that Oracle Flow Builder is installed before installing Oracle Functional Testing Advanced Pack for Oracle Utilities. See *Oracle Flow Builder User's Guide* for instructions to install Oracle Flow Builder.

> **Note**: The database connection pool size has to be (recommended) set to 50 or higher, using the WebLogic console.
>
> Below is the pseudo format of the URL to access the WebLogic console:
>
> http://<OFB_HOST>:<OFB_ADMIN_PORT>/console

Installing Oracle Functional Testing Advanced Pack for Oracle Utilities

Oracle Functional Testing Advanced Pack for Oracle Utilities has to be installed on a client workstation. See the Installing on Client Admin Workstation section for installation instructions.

> **Note:** Oracle Functional Testing Advanced Pack for Oracle Utilities need not be installed on the user workstations. Users only need browser access to Oracle Flow Builder for the component and flow development.

Administrative Tasks

See the *Oracle Flow Builder User's Guide* for the Oracle Flow Builder administrative tasks (such as stopping and starting the application instance).

Step 2: Setting Up Workstations for Development

This section provides the steps to set up the Oracle Functional Testing Advanced Pack for Oracle Utilities developer workstations. The tasks include:

- Installing OpenScript
- Creating Test Execution Folder Structure
- Downloading and Setting up Jars

Installing OpenScript

Ensure OpenScript is installed on each user workstation where automation execution is performed or where component and flow development is intended to be performed. See the System Requirements section for the certified OpenScript version details.

Creating Test Execution Folder Structure

See the Creating Folder Structure for Generated Scripts section to create the folder structure for placing the necessary test artifacts.

Downloading and Setting up Jars

See the Creating Folder Structure for Generated Scripts section to download and set up the necessary .jar files.

Step 3: Setting Up Application Under Test

See the respective Oracle Utilities' application specific installation guide for the setup details.

Ensure that Oracle Functional Testing Advanced Pack for Oracle Utilities related metadata exists in this application instance. For more details, see the **Post-Installation Tasks** section in *Oracle Functional Testing Advanced Pack for Oracle Utilities Installation and Administration Guide.*

Step 4: Setting Up Workstations for Testing

This section provides the steps to set up the Oracle Functional Testing Advanced Pack for Oracle Utilities workstations for test environment. The tasks include:

- Installing OpenScript
- Creating a Test Execution Folder Structure
- Using the configuration.properties file
- Downloading and Setting up Jars

Installing OpenScript

Install OpenScript on each test workstation where automation execution has to be performed. See the System Requirements section for the certified OpenScript version details.

Creating a Test Execution Folder Structure

See the Creating Folder Structure for Generated Scripts section to create the folder structure for placing the necessary test artifacts.

Using the configuration.properties file

Update the configuration.properties file to suit the test execution product setup requirements. See the Configuring the Runtime Properties section to understand the properties to be configured.

Downloading and Setting up Jars

See the Creating Folder Structure for Generated Scripts section to download and set up the necessary .jar files.

Creating Web Service Based Components

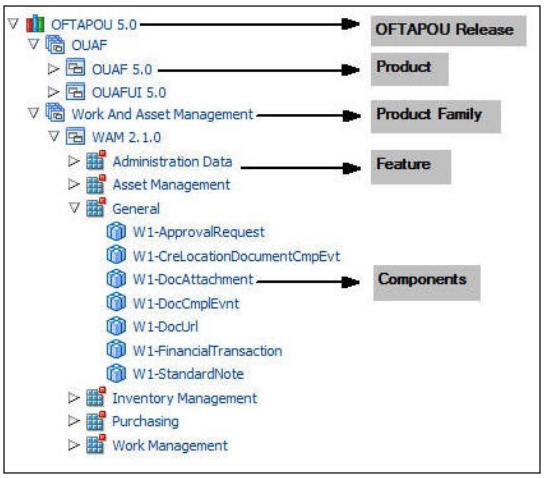
This section describes the component hierarchy in Oracle Functional Testing Advanced Pack for Oracle Utilities. The hierarchy is organized as follows:

> Oracle Functional Testing Advanced Pack for Oracle Utilities Release > Product Family > Product Release > Features > Components.

The section also explains the steps to create a Web service based component. It includes:

- Creating Components
- Keywords, Definitions, and the Usage
- Handling the List Elements
- Setting Up OperationName for a Web Service
- Using Runtime Variables in Components
- Resolving the Repeating Elements in Response XML
- Adding Validations
- Logging and Reporting
- Extending Components
- Using Function Libraries

The figure below shows the high-level component structure.



Component Structure

Creating Components

Ensure the component is created under the accurate hierarchy level.

To create a component, follow these steps:

1. Navigate to the component tree where the component has to be created.
2. Right-click the feature in the component tree.

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

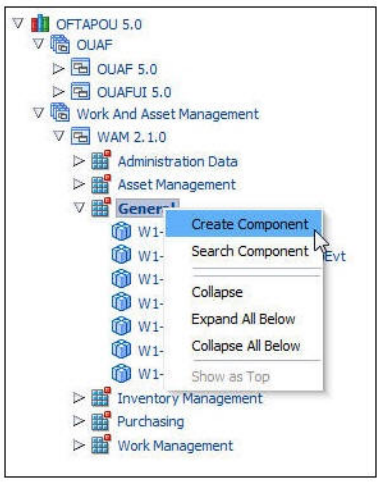
   For example: To create a "CM-MobileWorker" component under the "Resource Management" feature for the product release ORS 2.2.0.3.0:

   a. Navigate to **OFTAPOU 5.0** > **ORS / MWM Product Family** > **ORS 2.2.0.3.0** > **Resource Management**.

   b. Right-click the **Resource Management** feature.

   c. 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 Functional Testing Advanced Pack for Oracle Utilities release.



Creating Component

3. Enter **CM-MobileWorker** in the **Component** field to name the component.
4. Select **Web Service** in the **CompType** drop-down list.
5. Enter a description in the **Description** field.
6. Click **Attach Code** to add the metadata. The **Component** window is displayed.
7. Create component lines. See the Creating a Component Line section for more information about creating component lines.
8. Click **Save & Unlock** to save and create the component.

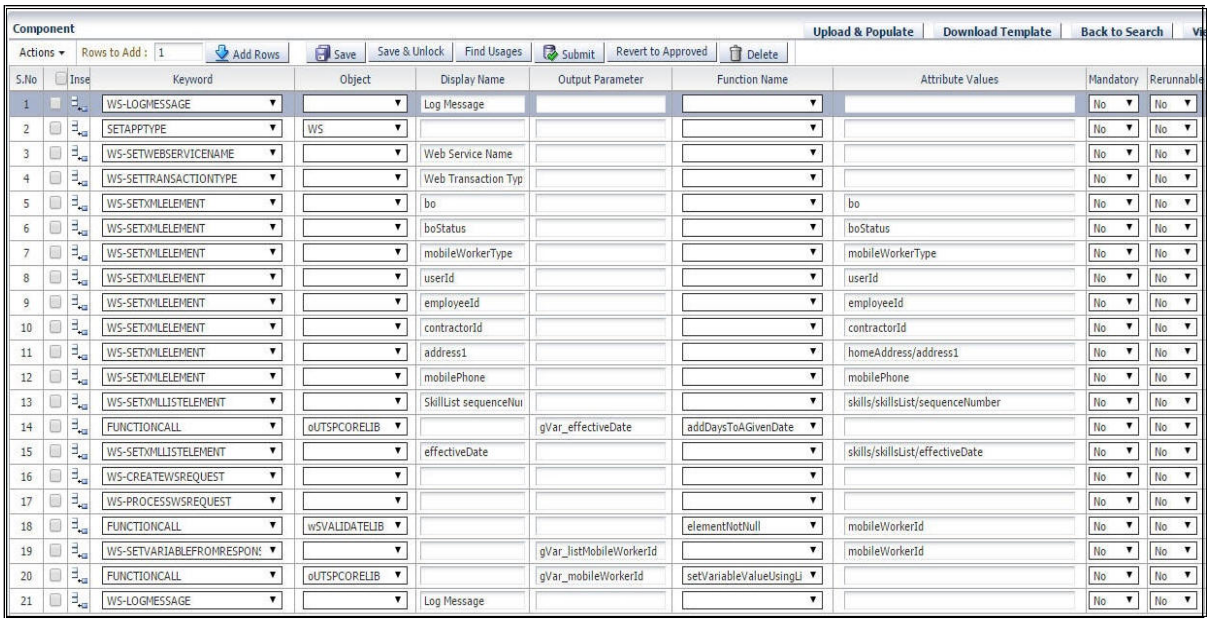
Creating a Component Line

A component consists of several component lines. Each component line comprises a keyword, display name, attribute value, library, function, output parameters, rerunnable flag, mandatory flag.

The list below describes each entity in a component line. See the *Oracle Flow Builder User's Guide* for more details.

- **Keyword**: Specifies the step to be performed. Example: WS-SETVARIABLEFROM RESPONSE, WS-VALIDATE, etc
- **Object**: Specifies the Oracle Functional Testing Advanced Pack for Oracle Utilities function library name from where the function is called.
- **Display Name**: Indicates the component line description.
- **Attribute Values**: Specifies the Web service XML tag name used as variable to store its value.
- **Default Data**: Stores the default data used in the component line.
- **Function Name**: Stores the function name called from the library.
- **Output Parameters**: Stores the output in the form of a variable.
- **Rerunnable**: Specifies "Yes" to create unique data for rerunnable. Rerunnable appends a random variable to the test data.
- **Mandatory**: Specifies "Yes" for all the mandatory fields.
- **Tooltip**: Presents the data as a tool tip during the flow creation.

The figure below shows the **Component** page with the available component lines.



Component Page

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

See the Keywords, Definitions, and the Usage section for a list of keywords used to define the Web service based components. See the *Oracle Flow Builder User's Guide* for a complete list of generic keywords.

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

1. Select SETAPPTYPE in the **Keyword** drop-down list to define the application type. Then, select "WS" in the **Object** drop-down list to denote that it is a Web services based component.
2. Select the WS-SETWEBSERVICENAME keyword to define the Web service name.
3. Select the WS-SETTRANSACTIONTYPE keyword to define the transaction type of the Web service call.

   Note: The final script of a component is Web service call to create, update, and delete.
4. Select the WS-LOGMESSAGE keyword to log comments in component definition. This helps in debugging the script code for that component.
5. Select the WS-SETXMLELEMENT keyword to set the value into a specific element of request xml.

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

   ```
   <mobileWorkerType />
   <contractorId />
   ```
6. Select the WS-SETXMLLISTELEMENT keyword to set a value into the list element tags. The list element is 'skills'.
7. Click **Save**.

Keywords, Definitions, and the Usage

This section provides the description, definition (use case), and the usage details (object, display name, attribute values, default data, function name, and output parameters) for each of the keywords used to define Web service based components.

WS-SETWEBSERVICENAME

Sets the name of the application Web service.

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

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

WS-SETXMLELEMENT

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

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

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

WS-SETXMLLISTELEMENT

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

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

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

WS-SETVARIABLE

Sets a value to a global variable.

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

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

WS-SETVARIABLEFROMRESPONSE

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

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

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

WS-SETTRANSACTIONTYPE

Sets a value for the transaction type.

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

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

WS-LOGMESSAGE

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

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

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

WS-CREATEWSREQUEST

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

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

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

WS-PROCESSWSREQUEST

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

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

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

Handling the List Elements

Use the WS-SETXMLLISTELEMENT keyword to define a list element of request XML. The XML schema for the CM-MobileWorker component has the 'Skills' list element. The element "skills/skillsList" has multiple occurrences in the 'skills' element.

The WS-SETXMLLISTELEMENT keyword allows you to enter data for such elements while entering the test data.

```
<skills type="group">
<skillsList type="list" mapChild="M1_RESRC_CAP">
<mobileWorkerId suppress="true" mapField="RESRC_ID"
dataType="string"/>   <sequenceNumber suppress="true"
dataType="number" mapField="SEQNO"/>
<skill mdField="M1_SKILL" fkRef="M1-SKILL" mapField="CAP_TYPE_CD"
dataType="string"/>  <effectiveDate dataType="date"
mapField="EFF_DT"/>
<expirationDate dataType="date" mapField="M1_EXP_DT"/>
</skillsList>
</skills>
```

Working with Multi-lists

In few requests, a particular list block is repeated. If a schema contains multi list, break the component into parts - _part1, _part2, _part3, etc.

The _part1 of the component includes root/parent elements followed by _part2 with the second level parents, and _part3 with the third level parents, and so on. The parents are matched to the child lists using fkRef and pkRef elements, added in the component definition. The _part1 pkRef value should correspond to fkRef of _part2.

In the main (_part1) component, the fkRef is blank (with no test data). The fkRef and pkRef should be integers starting with 1 (data to be provided in flow test data). The _part2 component can be dragged multiple times (after _part1 component) to add as many list repetitions as required.

Below is a sample schema:

```
<exceptionInfo type="list" mapXML="USG_DATA_AREA">
   <sequence mdField="SEQ_NUM" dataType="number" isPrimeKey="true"/>
   <messageCategory mdField="MESSAGE_CAT_NBR" dataType="number"/>
   <messageNumber mdField="MESSAGE_NBR" dataType="number"/>
   <comments mdField="COMMENTS"/>
   <messageParameters type="list">
      <parameterSequence mdField="PARM_SEQ" dataType="number"
isPrimeKey="true"/>
      <messageParameterValue mdField="MSG_PARM_VAL"/>
   </messageParameters>
</exceptionInfo>
```

In the sample above, Part I component elements are as shown.

```
<exceptionInfo type="list" mapXML="USG_DATA_AREA">
   <sequence mdField="SEQ_NUM" dataType="number" isPrimeKey="true"/>
   <messageCategory mdField="MESSAGE_CAT_NBR" dataType="number"/>
   <messageNumber mdField="MESSAGE_NBR" dataType="number"/>
   <comments mdField="COMMENTS"/>
   <messageParameters type="list">
```

- The Part 1 component should have a Web service name specified.
- It should have fkref and pkref specified using the WS-SETXMLLISTELEMENT keyword.

- Do not pass test data for fkRef and pkRef values. They denote the number of times the list is repeated.

  For example: If the list is repeated thrice, then in the test data, value 1=1, value2=2, and value3=3.

- All the first level (parent level) elements should be declared in the Part 1 component.

In the sample above, Part II component elements are as shown:

```
      <parameterSequence mdField="PARM_SEQ" dataType="number"
isPrimeKey="true"/>
      <messageParameterValue mdField="MSG_PARM_VAL"/>
```

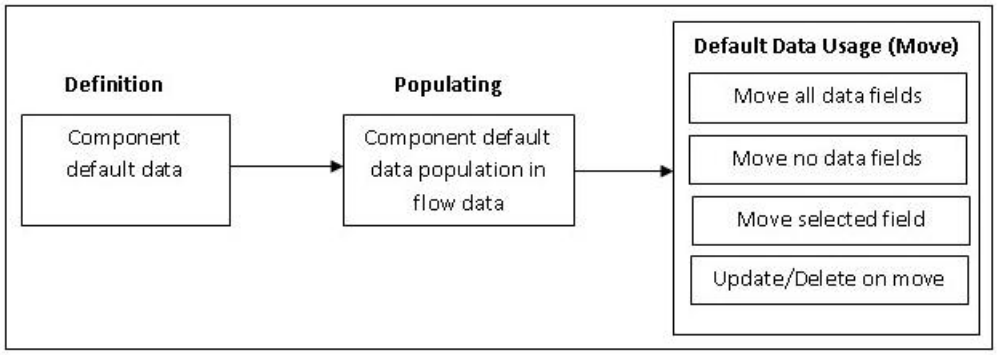
- For part 2 components, do not specify the Web service name.
- Part 2 components consists of multi-list elements that repeat.
- Park 2 components should include fkRef and pkRef values that map parent to the child list elements.

Defining Default Data at Component Level

In Oracle Functional Testing Advanced Pack for Oracle Utilities, test data can be maintained at the component level for quick and easy use at the flow level.

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

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



Data Flow

Setting Up OperationName for a Web Service

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

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

Using Runtime Variables in Components

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

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

See the Creating Test Flows section for procedure about how a dependent component reads such global variables

Resolving the Repeating Elements in Response XML

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

Consider the response as follows:

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

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

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

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

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

See Chapter 3: Function Library Reference for more details.

Adding Validations

To validate the response, use the FUNCTIONCALL keyword to validate the content, in particular, the Xpath of response XML. Select the function library wSVALIDATELIB from the **Object** drop-down list and the function to be called from the **Function Name** drop-down list.

See Chapter 3: Function Library Reference for a complete reference of the Validation function library.

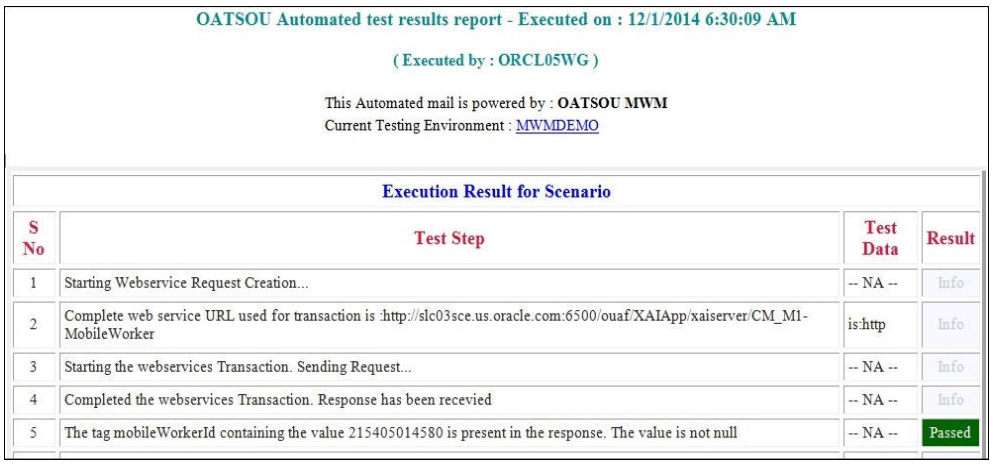
Logging and Reporting

Apart from OpenScript, Oracle Functional Testing Advanced Pack for Oracle Utilities provides the following types of logging and reporting:

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

2. **Email report in HTML format**: The test execution email provides brief information about the overall test execution. It comprises of the following:
    - Test step
    - Test data
    - Result (Pass/Fail)

The figure below is a snippet of a generated email.



OATSOU Automated test results report - Executed on : 12/1/2014 6:30:09 AM

(Executed by : ORCL05WG)

This Automated mail is powered by : **OATSOU MWM**
Current Testing Environment : MWMDEMO

Execution Result for Scenario

S No	Test Step	Test Data	Result
1	Starting Webservice Request Creation...	-- NA --	Info
2	Complete web service URL used for transaction is :http://slc03sce.us.oracle.com:6500/ouaf/XAIApp/xaiserver/CM_M1-MobileWorker	is:http	Info
3	Starting the webservices Transaction. Sending Request...	-- NA --	Info
4	Completed the webservices Transaction. Response has been recevied	-- NA --	Info
5	The tag mobileWorkerId containing the value 215405014580 is present in the response. The value is not null	-- NA --	Passed

Logging and Reporting Result

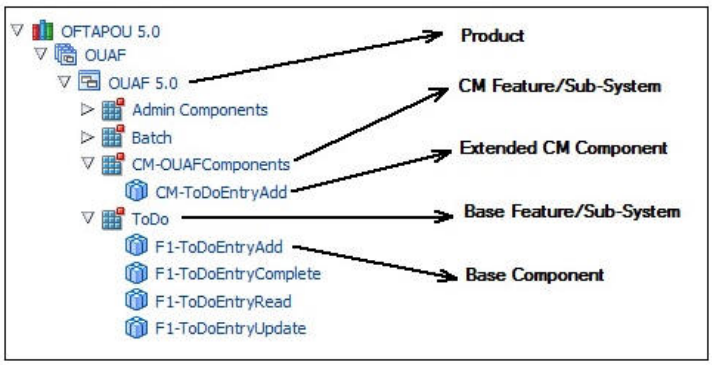
Extending Components

The components delivered can be customized; however, modifying the existing components is not supported. Any attempt to do so does not allow other components to work with the modified component.

A component can be extended by making its copy or clone 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. Create a subsystem (Feature) to store extended components.
2. Right-click a base component and select **Copy Component**.
3. Select and right-click on the subsystem (Feature) created in Step 1, and then select **Paste Component** from the component context menu.
4. Enter a new name (that is different from the component being cloned). The name should always be prefixed with 'CM'.
5. Add the **'CM'** tag in the **Tag** field, and then click **Save**.
6. Right-click the cloned component and select **Attach Code** from the context menu to modify it.
7. Follow the steps listed out in the Creating Components section to proceed with modifications.

The figure below shows how the Oracle Utilities Application Framework (OUAF) component F1-ToDoEntryAdd is extended.



Extending Components

Using Function Libraries

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

Function libraries shipped with Oracle Flow Builder and Oracle Functional Testing Advanced Pack for Oracle Utilities 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. See the Test Data Management section for more details.

See Chapter 3: Function Library Reference for a list of libraries and functions available in Oracle Functional Testing Advanced Pack for Oracle Utilities.

Creating Component Sets

A component set is a group of components arranged in a pre-determined order. It can be used to perform a set of tasks that are repeatable.

For example: The M1-CrewShift component in Oracle Utilities Mobile Workforce Management is actually a component set. To create an M1-CrewShift component in a flow, ensure M1-MobileWorker, M1-Vehicle, and M1-MultiPersonCrew are available. Then, create the dependant M1-CrewShift component. To avoid dragging and dropping these four components to a flow, they can be grouped in a sequence to form a component set. It can later be used to create flows.

To create a component set, follow these steps:

1. Navigate to the component set tree to locate the feature under which the component set should be created.
2. Right-click the feature and select **Create Component Set**. The **Component Set** page is displayed.
3. Enter a unique component name **CM-CrewShift** and add a **Description**.
4. Click **Create Structure**.
5. Drag and drop the components M1-MobileWorker, M1-Vehicle, M1-MultiPersonCrew, M1-CrewShift to the component set structure root.
6. Click **Unlock** to remove the lock on the component set.

Creating Test Flows

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

This section explains the steps to create a flow. It also includes:

- Creating Scenarios
- Using Global Variables
- Test Data Management
- Adding the Email Capabilities to Flows

To create a flow, identify the components required to create the flow.

> **Note**: The components delivered with Oracle Functional Testing Advanced Pack for Oracle Utilities have to be extended or new components have to be created.

To create a flow, follow these steps:

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

   For example: The "Assign and Complete a CrewShift" flow includes the following components:

   - M1-MobileWorker
   - M1-Vehicle
   - M1-MultiPersonCrew
   - M1-CrewShift (to create a crew shift)
   - M1-Activity (to create an activity)
   - M1-CrewShift (to plan a crew shift)
   - M1-CheckIfActivityIsScheduled (to check if the activity has been scheduled)
   - M1-CrewShift (to start the crew shift)
   - M1-GetAssignmentIdForTaskId (to get the assignment ID for the activity)
   - M1-Assignment (to enroute to the assignment)
   - M1-Assignment (to start the assignment)
   - M1-Assignment (to complete the assignment)
   - M1-CrewShift (to complete the crew shift)

2. Right-click the product "ORS 2.2.0.1.0" under "ORS / MWM Product Family" and select **Create Flow**.
3. In the **Create Flow** pane, enter the **Flow Name**, **Flow Type**, and **Description**.
4. Click **Create Structure** in the **Create Flow** pane. This creates a new flow and a new scenario is added to the flow tree.
5. Expand the flow tree and select the scenario to add the components.
6. Drag and drop the components from the **Select Component** pane to the **Flow Creation** pane. See the Creating Scenarios section for information about adding scenarios to a flow.
7. The test data needs to be entered at the component level while defining a flow and before the flow is assembled. To add data for M1-MobileWorker component, right click it and select **Enter Test Data**. Similarly, data can be added for the remaining components.

8. Enter the test data in the **Enter Component Test Data** page.
9. Click **Save & Close** to return to the **Flow Creation** page.
10. Click **Assemble**.
    This completes the procedure to define a test flow in Oracle Flow Builder.

Creating Scenarios

See the **Adding Scenarios to Flow** section in the *Oracle Flow Builder User's Guide* to understand how to create and use scenarios in the flow.

Using Global Variables

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

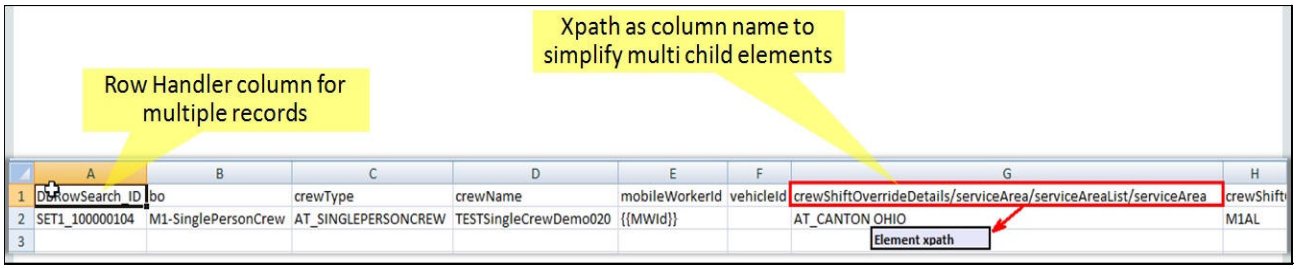
The component M1-CrewShift is a dependant component and during runtime needs The IDs of the M1-MobileWorker, M1-Vehicle, and M1 MultiPersonCrew components in addition to the other data.

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

1. To add the MobileWorker ID, select **gVar_mobileWorkerId1** from the **Value1** drop-down list against the **resourceAllocationList/resourceId** display name.
2. To add VehicleId, select **gVar_vehicleId1** from the **Value2** drop-down list against the **resourceAllocationList/resourceId** display name.
3. To add MultiPersonCrewId, select **gVar_CrewId** in the **Value 1** drop-down list against the **crewId** display name.

Test Data Management

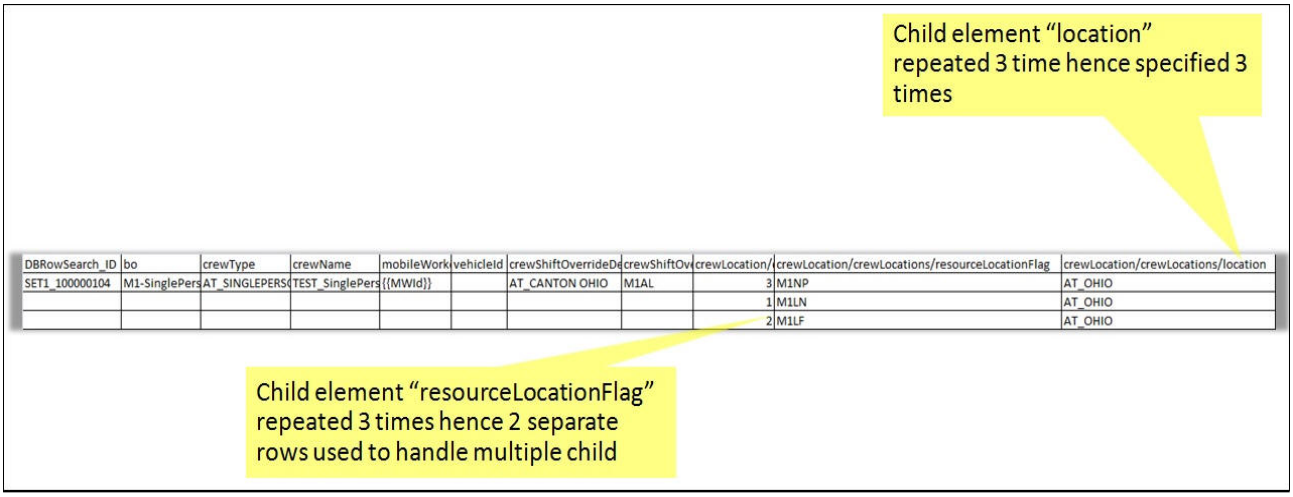
The test data can be mentioned in the flow meta data. The Oracle Flow Builder code generator generates OpenScript databanks (CSV files) for each component. Number and names of the columns in the generated databanks are based on the test data provided. The databanks can be updated with new data before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



Sample CSV File

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

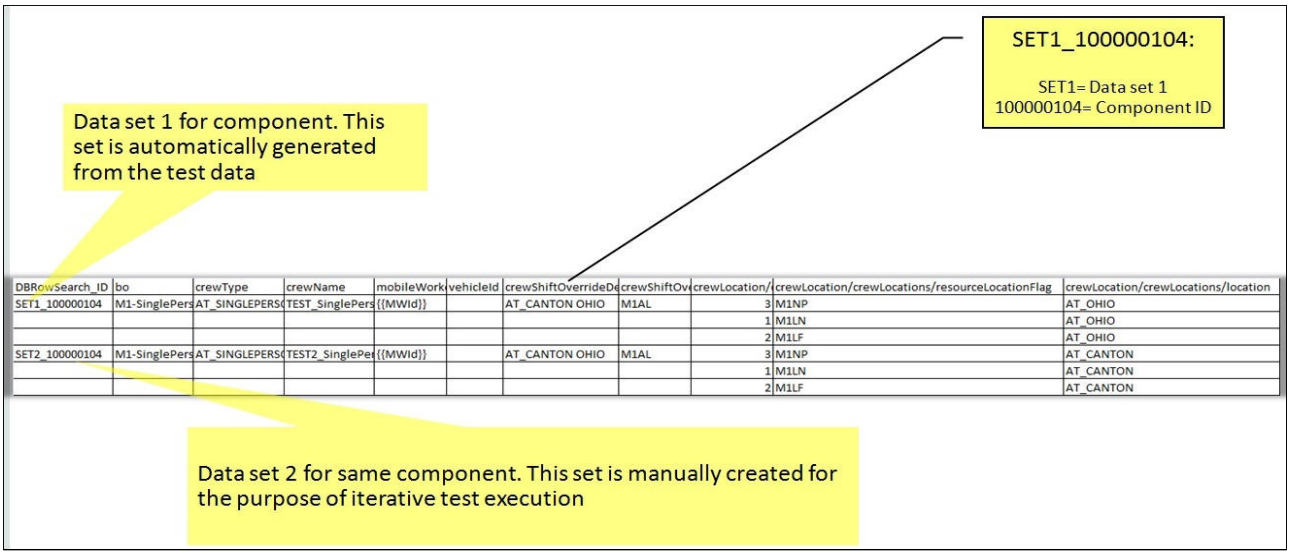
Case 1: The component is called just once in the flow and it has repeating list elements (such as location). The figure below shows the CSV generated for the component.



DBRowSearch_ID	bo	crewType	crewName	mobileWork	vehicleId	crewShiftOverrideDe	crewShiftOv	crewLocation/	crewLocation/crewLocations/resourceLocationFlag	crewLocation/crewLocations/location
SET1_100000104	M1-SinglePers	AT_SINGLEPERS	TEST_SinglePers	{{MWId}}		AT_CANTON OHIO	M1AL	3	M1NP	AT_OHIO
								1	M1LN	AT_OHIO
								2	M1LF	AT_OHIO

Case 1: CSV for Component

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



DBRowSearch_ID	bo	crewType	crewName	mobileWork	vehicleId	crewShiftOverrideDe	crewShiftOv	crewLocation/	crewLocation/crewLocations/resourceLocationFlag	crewLocation/crewLocations/location
SET1_100000104	M1-SinglePers	AT_SINGLEPERS	TEST_SinglePers	{{MWId}}		AT_CANTON OHIO	M1AL	3	M1NP	AT_OHIO
								1	M1LN	AT_OHIO
								2	M1LF	AT_OHIO
SET2_100000104	M1-SinglePers	AT_SINGLEPERS	TEST2_SinglePer	{{MWId}}		AT_CANTON OHIO	M1AL	3	M1NP	AT_CANTON
								1	M1LN	AT_CANTON
								2	M1LF	AT_CANTON

Case 1: CSV After Adding Second Data Set

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

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

Case 2: CSV for Component

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

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

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

Case 2: CSV After Adding Second Data Set

Adding the Email Capabilities to Flows

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

Usage Details	Value
Keyword	FUNCTIONCALL
Object	wSCOMMONLIB
Function Name	generateAndSendReport

The email configuration properties file is in the server at:

```
install directory] /data/function-libraries/function-libs/OUTSP/
configuration.properties
```

Update the values mentioned below to configure the email:

```
#Email Details
gStrSMTP_HOST_NAME=internal-mail-router.oracle.com
gStrSMTP_PORT=25
gStrTO_EMAIL_RECIPIENTS=<user_email_id>
```

Support for HTTPS Web Services

While connecting to the edge applications that use the https protocol, before executing the scripts through Oracle Application Testing Suite, the security certificate should be saved on the system from where the Oracle Application Testing Suite test cases are being executed, and then register the certificate in the Java security certificates repository.

To import the security key store into Java key store follow these steps:

1. Enter the URL (https) of the application in the browser (Internet Explorer).
2. Click the **Continue to this Website (not recommended)** link on the **Security certificate** page.
3. Click **Certificate error** in the address bar.
4. Click the **View certificates** link on the **Certificate Invalid** pop-up window.
5. On the **Details** tab, click **Copy to File**.
6. Click **Browse** and select the file you want to export. Click **Next**.
7. Review the settings and click **Finish**.

8. Login to the machine where this certificate has to be imported into the Java key store, and then open the command prompt.
9. If the Java path is not set in the environment variables, navigate to the Java/jdk/bin directory and execute the following command:

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

   Note: The Java key store path in the machine where OpenScript is installed in the INSTALL_DIR\OracleOATS directory is INSTALL_DIR\OracleATS\jdk\jre\lib\security\cacerts.
10. Enter "changeit" as the **Password**.
11. Enter **Yes** to import the certificate.

    The property file attributes for https requests are as below:

    ```
    ##Handling Https  WSDL - Java key Store
    gStrJavaKeyStorePath=INSTALL_DIR\\OracleATS\\jdk\\jre\\lib\\securi
ty
    gStrJavaKeyStorePwd=changeit
    ```
12. The setup is ready to process the https requests.

Support for Integration Flows

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

To perform complete end-end tests, add the url as an attribute in the properties file as shown below:

configuration.properties

```
# Integration Environments
gStrMWMApplication=https\://slc09ute.us.oracle.com\:9590/ouaf/XAIApp/
xaiserver
gStrCCBAppliation=https\://slc05ats.us.oracle.com\:8501/ouaf/XAIApp/
xaiserver
gStrMDMAppliation=https\://slc08ftq.us.oracle.com\:5251/ouaf/XAIApp/
xaiserver
```

Below is the example where the Oracle Utilities Customer Care and Billing service AT-C1Premise is called:

Test data:

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

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

Executing Test Flows

This section explains the steps to execute a test flow.

Generating OpenScript Projects

To generate an OpenScript project, follow these steps:

1. On the flow tree page, right-click a flow and select **Generate OFT Scripts**.
2. Enter the local folder details where the OpenScript project has to be generated.

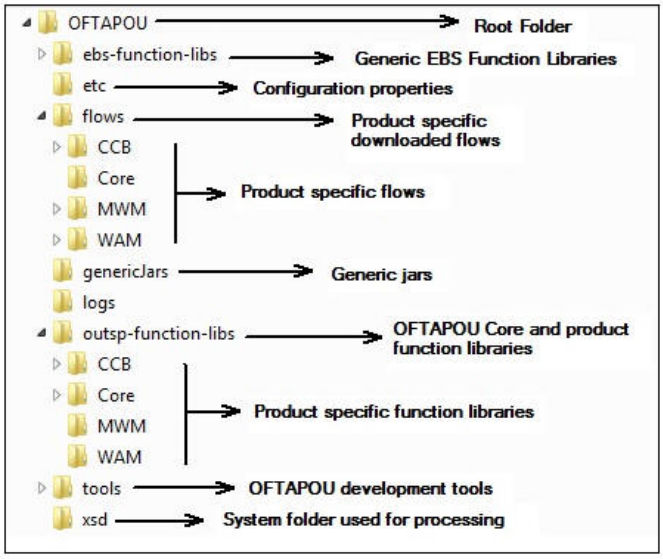
The OpenScript project generated from the test flow has the following structure:

- **Master Driver** folder - This is an OpenScript project containing the script.java file which is the master driver script invoking all the scenario scripts in the scenario folder.
- **Scenario** folder - This is an OpenScript project. There will be one OpenScript project for each scenario defined in the flow. Each scenario's script will be invoked from the Master Driver script.
- **Components** - Components will be the OpenScript instructions called from the scenario scripts. Each component is enclosed within a step group in OpenScript.

Creating Folder Structure for the Generated Scripts

After generating and downloading the OpenScript project, create the following folder structure to organize the generated test scripts. This folder structure needs to be created on every workstation where the automation scripts are executed.

Below is the sample folder structure to organize the test scripts generated.



Folder Structure for Generated Scripts

- **ebs-function-libs**

  Stores the generic function library. Oracle Functional Testing Advanced Pack for Oracle Utilities needs three generic function libraries, namely GENLIB, WEBLABELLIB, and WEBTABLEOBJ.

These function libraries should be copied from the Oracle Flow Builder server. Below is the server location from where above libraries can be copied.

```
<OFB_INSTALL_DIR>/data/function-libraries/ebs-function-libs
```

- **outsp-function-libs**

  Stores the Oracle Functional Testing Advanced Pack for Oracle Utilities related function libraries.

  Extract the product specific function libraries by following the steps specified in the **Extracting OFTAPOU Packages** section in *Oracle Functional Testing Advanced Pack for Oracle Utilities Installation and Administration Guide*.

  Copy all the directories under <OFTAPOU_HOME>\OatsouLibrary\modules\oatsou-function-libs\directory to <OpenScript_INSTALL_DIR>/outsp-function-libs/.

  > **Note**: For details about the <OFTAPOU_HOME> location, see the **Extracting OFTAPOU Packages** section in *Oracle Functional Testing Advanced Pack for Oracle Utilities Installation and Administration Guide*.

- **genericJars**

  For executing the generated Web service automation code, the following third party jars are required. These jars need to be copied to the genericJars folder.

Jar Name	Jar Version	Download URL
soa-model-core	1.4.1.4	http://www.membrane-soa.org/downloads/
groovy	2.0.4	http://groovy.codehaus.org/Download
groovy-xml		http://groovy.codehaus.org/Download
sjsxp	1.0.2	https://java.net/projects/sjsxp/downloads/
asm	4.0	http://asm.ow2.org/download/index.html
httpclient	4.2.2	http://archive.apache.org/dist/httpcomponents/httpclient/
httpcore		http://archive.apache.org/dist/httpcomponents/httpcore/
Jdom	1.1.1	http://www.jdom.org/dist/binary/archive/

The jars mentioned above can be copied from the Oracle Functional Testing Advanced Pack for Oracle Utilities server place to the following location:

```
<OFB_INSTALL_DIR>/data/function-libraries/genericjars
```

- **xsd** - Stores the run time generated XSDs required for processing the Web Services request.
- **Etc** - Consists of the property file for providing runtime and email configuration details, such as test environment name, application user and password, email ID, etc.
- **Logs** - Stores the runtime generated test execution logs that can be later used for debugging.
- **flows** - Stores the downloaded flows, product wise. The applications need to create their application specific folder in the "**flows**" folder to download and store the flows.

  The figure mentioned above shows the flows arranged application wise.

- **tools** - This folder.

Configuring the Runtime Properties

The **configuration.properties** file is located in the **etc** folder. It is used to store the runtime test execution parameters, such as application URL, application access information, email configuration, etc.

To use the email functionality for receiving the test execution report, provide the values mentioned below in the configuration.properties file (etc\configuration.properties file).

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

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

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

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

```
gStrApplicationUserName =
gStrApplicationUserPassword =
```

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

```
gStrApplicationDBConnectionString =
gStrApplicationDBUsername =
gStrApplicationDBPassword =
```

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

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

See the Support for Integration Flows section for properties related to integration environment.

Chapter 3

Function Library Reference

This chapter lists the Oracle Functional Testing Advanced Pack for Oracle Utilities function libraries and functions available to create components and flows in Oracle Flow Builder for testing Oracle Functional Testing Advanced Pack for Oracle Utilities.

The chapter explains the following libraries:

- OUTSPCORELIB
- WSVALIDATELIB
- WSCOMMONLIB

OUTSPCORELIB

Use the OUTSPCORELIB function library to develop the component code and flows for Web services and general applications. The library includes functions with date and time processing and string processing capabilities, as well as database and file operations.

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

runBatchFile

Executes a existing batch file.

Example:

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

Input Parameters: String
Return Type: void
```

killBatchFile

Kills the batch process in execution.

Example:

```
killBatchFile ("cmd.exe")

Input Parameters: String
Return Type: void
```

getCurrentTimeInMilliSeconds

Gets the time in milliseconds.

Example:

```
getCurrentTimeInMilliSeconds ()

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

rand

Gets the random number for the given range.

Example:

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

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

randomStringWithGivenRange

Gets the random string for the given range.

Example:

```
randomStringWithGivenRange(int lo, int hi)

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

Randomstring

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

Example:

```
randomstring (lowerLim, higherLim)

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

compare2Strings

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

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

Example:

```
compare2Strings (String_A, String_B)

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

randomNumberUsingDateTime

Gets the random string with date and time in it.

Example:

```
randomNumberUsingDateTime()

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

getCurrentDateTimeWithGivenDateFormat

Gets the current date and time in the specified format.

Example:

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

Input Parameters: dFormat
Return Type: String
```

getDateDiffInSecsWithGivenDateFormat()

Gets the difference in the date.

Example:

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

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

getAdjustedTimeWithGivenDateTime

Gets the adjusted time with the given date and time.

Example:

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

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

getAdjustedTimeWithCurrentDateTime

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

Example:

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

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

getAdjustedTimeWithGivenDateAndTime

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

Example:

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

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

addDaysToCurrentDateWithGivenFormat

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

Example:

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

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

serverDate

Gets the server date.

Example:

```
serverDate()

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

executeSQLQry

Executes SQL and returns the record set.

Example:

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

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

executeSQLQryWithGivenDBDetails

Executes SQL and returns the record set.

Example:

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

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

serverTime

Gets the server time.

Example:

```
serverTime()

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

waitForTime

Waits for the specified time.

Example:

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

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

verifyLastBatchRun

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

Example:

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

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

getCurrentOffsetTime

Gets the current offset time.

Example:

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

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

addDaysToAGivenDate

Adds days to the provided date.

Example:

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

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

randomNumber

Gets the random number.

Example:

```
randomNumber()

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

createFile

Creates file in the specified path.

Example:

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

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

getWaitConditionState

Waits for the specified time.

Example:

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

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

compare2Files

Compares two files.

Example:

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

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

copyFile

Copies files from source to destination.

Example:

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

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

deleteFile

Deletes a file.

Example:

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

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

executeSQLQryUpdate

Executes SQL for the update query.

Example:

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

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

getDistinctObjects

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

Example:

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

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

setVariableValueUsingListIndex

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

Example:

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

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

closeConnections

Closes the database connection opened for database verification.

Example:

```
closeConnection()

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

executeSQLQryForSingleRecord

Executes the SQL query for a single record.

Example:

```
executeSQLQrySingleRecord(String Query,String recId)

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

appendStrings

Appends strings provided in the parameters.

Example:

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

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

getCurrentMonth

Gets the current month.

Example:

```
getCurrentMonth()

Input Parameters: none
Return Type: String
```

WSVALIDATELIB

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

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

elementNotNull

Verifies if the specified element in response is null.

Example:

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

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

elementIsNull

Verifies if the specified element in response is not null.

Example:

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

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

elementValueEquals

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

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

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

elementValueNotEquals

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

Example:

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

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

elementValueGreaterThan

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

Example:

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

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

elementValueGreaterThanEqualTo

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

Example:

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

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

elementValueLesserThan

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

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

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

elementValueLesserThanEqualTo

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

Example:

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

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

elementContains

Verifies if the specified element is available in the response.

Example:

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

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

elementNotContains

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

Example:

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

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

reponseNotContains

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

Example:

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

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

responseContains

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

```
responseContains(String value)
responseContains("Exception")

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

WSCOMMONLIB

Use the WSCOMMONLIB function library to perform common operations in the Oracle Functional Testing Advanced Pack for Oracle Utilities Web services testing, such as composing request, sending request, composing email summary, converting it to HTML format, sending an email, and parsing WSDL.

> **Note**: This library does not have any component development functions other than the generateAndSendReport function that provides result reporting and email capabilities to the user. See the Logging and Reporting section in Chapter 2: Developing Metadata Driven Web Service Based Test Automation for more details.

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

generateAndSendReport

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

```
generateAndSendReport ()

Input Parameters: NA
Return Type: NA
```

Chapter 4

Development Accelerator Tools

This chapter provides information about the development accelerator tools.

Component Generation Tool

Each of the Oracle Functional Testing Advanced Pack for Oracle Utilities (OFTAPOU) components correspond to a business entity in the application being tested. That business entity is used to create an Inbound Web Service using Business Service/Service Script or Business Object.

All Oracle Flow Builder components have to be manually created by extracting the xpath of each of the schema elements in the XSD for a business component/Inbound Web Service. The extraction of xpath from an XSD is tedious given the sheer number of components that we have for each of the application. The maximum effort/time for automation using Oracle Flow Builder goes into the creation of the components.

The automated Component Generation Tool (CGT) reads the WSDL URL and the component name supplied in a user friendly format (excel sheet), and generates the respective Excel sheets for component (one for each WSDL URL). For each WSDL URL that is read by the tool, a request is sent to the URL for retrieving the WSDL XSD. Once the tool retrieves the WSDL XSD from the application, the schema xpath of the XSD is developed and inserted in to an output component template Excel. The template with details is uploaded to create new components.

This section includes information about the following:

Creating Folder Structure

To create the pre-requisite folder structure, follow these steps:

1. Create the Inbound Web Services for all Business Objects (BO), Business Service (BS), and Service Script (SS) necessary for automation in the environment.
2. Create the following folder structure in local drive (Example: D:)
    a. **ebs-function-libs**: Includes library files used to develop flows.
    b. **etc**: Includes the configuration.properties file where all the environment details are listed.
    c. **flows**: Contains the flows generated through Oracle Flow Builder.

d. **generic jars**: Includes jar files needed for execution.

e. **logs**: Includes the logs placed during the flow execution.

f. **outsp-function-libs**: Contains the Core function libraries and function libraries related to Oracle Utilities Meter Data Management.

g. **tools**: Includes the Component Generator and Component Validator to generate and validate components.

h. **xsd**: Includes the generated xml files.

The necessary folder structure is in place now.

Creating Components Using Component Generation Tool

To create a component using the Component Generation Tool, follow these steps:

1. Login to the Oracle Utilities Application Framework (OUAF) application.
2. Identify the **WSDL URL** for the target Business Objects/ Business Services for the specific edge application.
3. Enter the WSDL URL and the component details in the input sheet in Component Generation Tool.

   The component names should follow the naming convention as follows:

   <OwnerFlag>-<entityName>

   For example:

   C1-Person, F1-Batch
4. In the OpenScript application, create a "OATS" repository and point it to the Oracle Application Testing Suite folder.
5. Ensure the values in the **etc > configuration.properties** file are valid.
6. Navigate to **OATS > tools > Scripts** and select **ComponentExcelGenerator**.
7. Execute this tool in OpenScript to generate the component Excel. An output Excel is generated and saved in the **OATS > tools > Output_Excels** folder.

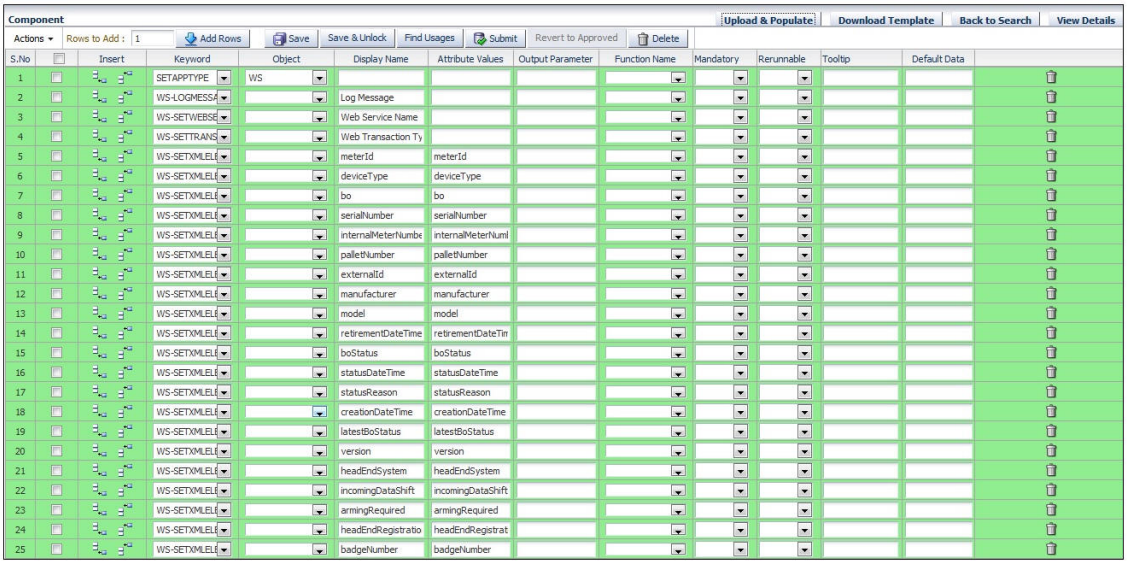
   Below is a sample Excel file for the D1-SmartMeter component. The Excel can be updated with new elements as required.

Keyword	Object	Display Name	Attribute value	Output Parameter	Function Name	Tool Tip	Mandatory	Rerunnable	Default Data
SETAPPTYPE	WS								
WS-LOGMESSAGE		Log Message							
WS-SETWEBSERVICENAME		Web Service Name							
WS-SETTRANSACTIONTYPE		Web Transaction Type							
WS-SETXMLELEMENT		meterId	meterId						
WS-SETXMLELEMENT		deviceType	deviceType						
WS-SETXMLELEMENT		bo	bo						
WS-SETXMLELEMENT		serialNumber	serialNumber						
WS-SETXMLELEMENT		internalMeterNumb	internalMeterNumber						
WS-SETXMLELEMENT		palletNumber	palletNumber						
WS-SETXMLELEMENT		externalId	externalId						
WS-SETXMLELEMENT		manufacturer	manufacturer						
WS-SETXMLELEMENT		model	model						
WS-SETXMLELEMENT		retirementDateTim	retirementDateTime						
WS-SETXMLELEMENT		boStatus	boStatus						
WS-SETXMLELEMENT		statusDateTime	statusDateTime						
WS-SETXMLELEMENT		statusReason	statusReason						
WS-SETXMLELEMENT		creationDateTime	creationDateTime						
WS-SETXMLELEMENT		latestBoStatus	latestBoStatus						
WS-SETXMLELEMENT		version	version						
WS-SETXMLELEMENT		headEndSystem	headEndSystem						
WS-SETXMLELEMENT		incomingDataShift	incomingDataShift						
WS-SETXMLELEMENT		armingRequired	armingRequired						
WS-SETXMLELEMENT		headEndRegistratio	headEndRegistration						
WS-SETXMLELEMENT		badgeNumber	badgeNumber						
WS-SETXMLELEMENT		specification	specification						
WS-SETXMLELEMENT		configuration	configuration						
WS-SETXMLELEMENT		assetId	assetId						
WS-CREATEWSREQUEST									
WS-PROCESSWSREQUEST									
WS-LOGMESSAGE		Log Message							

Output Excel

8. Import the component into Oracle Flow Builder as follows:
   a. Login to the Oracle Functional Testing Advanced Pack for Oracle Utilities environment.
   b. Click the **Components** link on the top-right corner. It brings up the **Component Tree** in the left pane.
   c. Navigate to the folder where the component: has to be created.
   d. Right-click folder and select **Create Component**.
   e. Enter the required values, and then click **Attach Code**.
   f. On the **Component** window, click **Upload & Populate**.
   g. Click **Browse** and navigate to the **OATS > tools > Output_Excels** folder to select the component Excel generated in step 7. Then, click **Open**.
   h. Click **Start** to upload and populate the component.

   The component Excel is imported into Oracle Flow Builder and a component is created.



Component Excel

9. After import, the component will be listed out in the component tree in Oracle Flowe Builder.

Component Validator

Errors in the component definition are hard to identify. Further, regular application upgrades can cause the components to be outdated. Identifying such components is rather difficult. Also, it is very important that the component definition is pristine and clear.

Component Validator tool automatically validates a given list of components in Oracle Flow Builder against their corresponding entities in the edge application. This saves time and effort during the component upgrades. Also, code generation and execution issues will be minimized if the component definition is accurate.

Component Validator parses through each of the component lines and highlights the incorrect component lines. Further, it identifies the missing schema elements in the component definition with respect to the edge application's business entity's schema. All the findings are displayed as a report for the user to act on.

This section includes information about the following:

Creating Folder Structure

To create the pre-requisite folder structure, follow these steps:

1. Install the latest OpenScript version (12.5.0.2 Build 537).
2. Create the following folder structure in local drive (Example: D:)
   a. **ebs-function-libs**: Includes library files used to develop flows.
   b. **etc**: Includes the configuration.properties file where all the environment details are listed.
   c. **flows**: Contains the flows generated through Oracle Flow Builder.
   d. **generic jars**: Includes jar files needed for execution.
   e. **logs**: Includes the logs placed during the flow execution.
   f. **outsp-function-libs**: Contains the Core function libraries and function libraries related to Oracle Utilities Meter Data Management.
   g. **tools**: Includes the Component Generator and Component Validator to generate and validate components.
   h. **xsd**: Includes the generated xml files.
3. Login to the environment being used.
4. Create the Inbound Web Services for all Business Objects (BO), Business Service (BS), and Service Script (SS) necessary for automation in the environment.

Executing the Component Validator Script

To execute the Component Validator script, follow these steps:

1. Navigate to OATS > tools > Input_Excels and open Validator_Input_File.csv.
2. Enter the necessary details.

   The Component_Validator shows two databanks attached:

   - **OFB_Environment_Details.csv** - Includes the Oracle Flow Builder environment details where existing components are validated. The structure of the databank is as below:

     DBUsername - Oracle Flow Builder database user name

     DBPassword - Oracle Flow Builder database user password

     DBServer - Oracle Flow Builder server name

     DBPort - Database port number

     DBSID - Database SID

   - **Validator_Input_File.csv** - Includes the URLs of the Web services for which validations need to be done. Below is the structure of the databank:

     WSDLURL - URL of the WSDL file to be validated. There can be multiple entries for WSDL URLs to validate multiple components at a time.

CompName - The component name created in Oracle Functional Testing Advanced Pack for Oracle Utilities for the specified Web service.

ReleaseName - The release name in which a component is created and validated.

3. Navigate to **OATS > tools > Scripts** and then execute **Component_Validator** in OpenScript.
4. The logs can be found in **OATS > tools > logs** as **ComponentValidatorResults.csv**.

   Below is a sample log file providing issues found during the validation.

1			
2	Component Name	WSDLURL.	
3			
4	M1-MobileWorker	http://slc03sce.us.oracle.com:6760/ouaf/XAIApp/xaiserver/CM_M1-MobileWorker.	
5			Retrieving the schema from application.
6			xpath: mainSection is missing from component schema - Failed
7			xpath: mobileWorkerId is missing from component schema - Failed
8			xpath: boStatusDateTime is missing from component schema - Failed
9			xpath: contactInfoSection is missing from component schema - Failed
10			xpath: homeAddress/postal is missing from component schema - Failed
11			xpath: workPhone is missing from component schema - Failed
12			xpath: skillsSection is missing from component schema - Failed
13			xpath: skills/skillsList/mobileWorkerId is missing from component schema - Failed
14			xpath: baseServiceAreasSection is missing from component schema - Failed
15			xpath: baseServiceAreas/baseServiceAreasList/mobileWorkerId is missing from component schema - Failed
16			xpath: version is missing from component schema - Failed
17			All the xpaths in the OFB component schema are present in the WSDL schema - Passed
18			Starting verification of component line definitions for OFB component.
19			Keyword SETAPPTYPE not set in the component as the first line. - Failed
20			Component's validation has been completed with the listed syntactical errors - Failed

Sample Log File

Analyzing the Component Validator Results

To analyze Component Validator log file, follow these steps:

1. Navigate to the logs folder available in the folder structure.
2. Open **ComponentValidatorResults.csv** file where the logs are generated.

   The first column displays the component that is validated and the second column shows the WSDL URL against which the component has been validated.
3. Check the third column for detailed comments:

   a. The log statement displays missing elements from the component schema. The component schema can be corrected by adding the missing elements.

   b. Basic validation for the all keywords is done. For example: The following log statement is displayed when the application type is not defined.

   ```
   "Keyword SETAPPTYPE not set in the component as the first line.
   - Fail"
   ```

Appendix

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

To create an Inbound Web Service, follow these steps:

1. Login to the Oracle Utilities application.
2. Navigate to **Admin > I > +Inbound Web Service**.
3. On the **Inbound Web Service** page, do the following:
    a. Enter the **Inbound Web Service Name**.
    b. Enter the **Description** and the **Detailed Description**.
    c. Select the appropriate **trace,debug.active,post error** option from the drop down.
    d. Select the **Annotation**.
    e. Enter the **Operation Name**, and then select the **Schema Type**, **Schema Name**, and **Transaction Type**.
    f. Click **Save**.

Importing Inbound Web Services

To import an Inbound Web Service into the Oracle Utilities application, follow these steps:

> **Note**: Ensure the exported Inbound Web Services are available in the local machine.

1. Login to the Oracle Utilities application.
2. Click **Admin > B > Bundle Import**.
3. On the **Inbound Web Service Import** page, enter the reference and detailed description.
4. Copy paste the bundle details from the Inbound Web Services bundle.

5. Click **Apply bundle**. The "Imported Successfully" message appears in the **Message text** column.

Searching Inbound Web Services

To search an Inbound Web Service in an Oracle Utilities application, follow these steps:

1. Login to the Oracle Utilities application.
2. Navigate to **Admin > I > Inbound Web Service**.
3. On the **Inbound Web Service Search** page, do the following:
   a. Enter the name of the required Web Service in the **Name** field.

      Alternatively, enter the description in the **Description** field.

   b. Click **Refresh**.

      The Web Service, if found, is retrieved and displayed.