Oracle Functional Testing Advanced Pack for Oracle Utilities

Reference Guide for Oracle Utilities Mobile Workforce Management/ Oracle Real-Time Scheduler (v2.3.0)

Release 5.0.1

E74664-02

November 2016

ORACLE

Oracle Functional Testing Advanced Pack for Oracle Utilities Reference Guide for Oracle Utilities Mobile Workforce Management/ Oracle Real-Time Scheduler (v2.3.0), Release 5.0.1

E74664-02

Contents

Preface

This guide describes the Oracle Utilities Mobile Workforce Management (MWM) v2.3.0 and Oracle Real-Time Scheduler (ORS) v2.3.0 components and the function libraries used to create those components in Oracle Functional Testing Advanced Pack for Oracle Utilities (OFTAPOU) v5.0.1. These components are used to build test flows in Oracle Flow Builder (OFB).

This preface includes the following sections:

Audience

This guide is intended for QA/Test Engineers and Automation Developers to understand the various components and libraries available for them to automate the business test flows for Oracle Utilities Mobile Workforce Management using Oracle Functional Testing Advanced Pack for Oracle Utilities (OFTAPOU) for Oracle Utilities Mobile Workforce Management and Oracle Real-Time Scheduler.

Related Documents

For more information, see the following documents:

- *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 User's Guide*

See also:

- Oracle Utilities Mobile Workforce Management Documentation Library
- Oracle Real-Time Scheduler Documentation Library

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

Component Reference

This chapter lists the Oracle Utilities Mobile Workforce Management/Oracle Real-Time Scheduler starter components available to create flows in Oracle Flow Builder for testing the Oracle Utilities Mobile Workforce Management/ Oracle Real-Time Scheduler application.

The chapter includes the following sections:

Overview

Oracle Functional Testing Advanced Pack for Oracle Utilities for Oracle Utilities Mobile Workforce Management/ Oracle Real-Time Scheduler is a test starter pack built on top of Oracle Functional Testing Advanced Pack for Oracle Utilities that generates test automation scripts using Oracle Flow Builder.

Oracle Functional Testing Advanced Pack for Oracle Utilities for Oracle Utilities Mobile Workforce Management/ Oracle Real-Time Scheduler contains out-of-the-box product-specific components used to build new test flows in Oracle Flow Builder to test the Oracle Utilities Mobile Workforce Management/ Oracle Real-Time Scheduler applications. These out-of-the-box components correspond to specific business entities, such as business objects, service scripts, or business services used for interfacing with the application. These components can be used as available or can be extended. Create new components to be used to create flows. This starter pack also contains a set of function libraries that can be used for creating custom components.

> **Note:** See Chapter 2: Function Library Reference for detailed information about using these function libraries.

Consider this pack to be a starter kit which can be expanded and built upon. A few sample flows are included as examples.

> **Note:** See the *Oracle Functional Testing Advanced Pack for Oracle Utilities User's Guide* for information about creating components and flows.

Components

This section lists the starter components available in Oracle Utilities Mobile Workforce Management.

Pre-requisites: The Inbound Web Service service using the respective business object should be available in the application.

Additional Notes: Failure while creating, reading, or updating the component is logged in the test execution report, thus facilitating debugging/analysis of the problem.

The components are categorized under the following functional areas:

Administration Data

Component	Description
M1-ActivityType	Used to create, read, or update M1 Activity Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ActivityType business object.
M1-AdditionalUserInfo	Used to maintain additional user information as part of the miscellaneous section on the user portal via a Web service. After creation, the CRUD operations can be performed through these components against the M1- AdditionalUserInfo business object.
M1-AlertType	Used to create, read, or update an M1 Alert Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-AlertType business object.
M1-AppointmentBookingGroup	Used to create, read, or update an M1 Appointment Booking Group via a Web service. After creation, the CRUD operations can be performed through these components against the M1-AppointmentBookingGroup business object.

Component	Description
M1-BreakTaskType	Used to create, read, or update an M1 Break Task Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-BreakTaskType business object.
M1-CapacityType	Used to create, read, or update an M1 Capacity Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CapacityType business object.
M1-ComplexActivityType	Used to create, read, or update an M1 Complex Activity Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ComplexActivityType business object.
M1-CrewHierarchy	Used to create, read, or update an M1-CrewHierarchy via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CrewHierarchy business object.
M1-ContractorType	Used to create, read, or update an M1 Contractor Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ContractorType business object.
M1-CrewShiftType	Used to create, read, or update an M1 Crew Shift Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CrewShiftType business object.
M1-CrewType	Used to create, read, or update an M1 Crew Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CrewType business object.
M1-Deployment	Used to activate or deactivate a deployment via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Deployment business object.

Component	Description
M1-DeploymentPart	Used to create, read, or update a deployment part via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DeploymentPart business object.
M1-DeploymentType	Used to create, read, or update a deployment type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DeploymentType business object.
M1-DeploymentTypeRestfulSvc	Used to create and describe the structure and rules applicable to deployment type definitions supporting the JSON based mobile platform. After creation, the CRUD operations can be performed through these components against the M1- DeploymentTypeRestfulSvc business object.
M1-Depot	Used to create, read, or update a depot via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Depot business object.
M1-DepotProfile	Used to create, read, or update a depot profile via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DepotProfile business object.
M1-DepotTaskType	Used to create, read, or update a depot task type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DepotTaskType business object.
M1-DispatchArea	Used to create, read, or update an M1 Dispatch Area via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DispatchArea business object.
M1-DispatcherType	Used to create, read, or update an M1 Dispatcher Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DispatcherType business object.

Component	Description
M1-Equipment	Used to create, read, or update an M1 Equipment via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Equipment business object.
M1-GeographicArea	Used to create, read, or update an M1 Geographic Area via a Web service. After creation, the CRUD operations can be performed through these components against the M1-GeographicArea business object.
M1-GlobalConfig	Used to create, read, or update M1 Global Configurations via a Web service. After creation, the CRUD operations can be performed through these components against the M1-GlobalConfigurations business object.
M1-KPI	Used to create, read, or update an M1 KPI via a Web service. After creation, the CRUD operations can be performed through these components against the M1-KPI business object.
M1-Location	Used to create, read, or update an M1 Location via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Location business object.
M1-MobileDeviceTerminal	Used to create, read, or update an M1 Mobile Device Terminal via a Web service. After creation, the CRUD operations can be performed through these components against the M1-MobileDeviceTerminal business object.
M1-MobileWorkerType	Used to create, read, or update an M1 Mobile Worker Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-MobileWorkerType business object.
M1-MobileDeviceTerminalType	Used to create, read, or update an M1 MobileDeviceTerminalType via a Web service. After creation, the CRUD operations can be performed through these components against the M1-MobileDeviceTerminalType business object.

Component	Description
M1-NPTTaskType	Used to create, read, or update an M1 Non Prd Task Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-NonPrdTaskType business object.
M1-POUTaskType	Used to create, read, or update an M1 POU Task Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-POUTaskType business object.
M1-POUType	Used to create, read, or update an M1 POU Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-POUType business object.
M1-PriorityProfile	Used to create, read, or update an M1 Priority Profile via a Web service. After creation, the CRUD operations can be performed through these components against the M1-PriorityProfile business object.
M1-ProcedureType	Used to create, read, or update an M1 Procedure Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ProcedureType business object.
M1-RemarkType	Used to create, read, or update an M1 Remark Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-RemarkType business object.
M1-ResTimesheetType	Used to create, read, or update an M1 Resource TimesheetType. via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Resource TimesheetType business object.
M1-Scheduler	Used to create, read, or update an M1 Scheduler via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Scheduler business object.

Component	Description
M1-SchedulerArea	Used to create, read, or update an M1 Scheduler Area via a Web service. After creation, the CRUD operations can be performed through these components against the M1-SchedulerArea business object.
M1-SchedulerConfig	Used to create, read, or update an M1 Scheduler Config via a Web service. After creation, the CRUD operations can be performed through these components against the M1-SchedulerConfig business object.
M1-ServiceArea	Used to create, read, or update an M1 Service Area via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ServiceArea business object.
M1-ServiceAreaHierarchy	Used to create, read, or update an M1 Service Area Hierarchy via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ServiceAreaHierarchy business object.
M1-ServiceClass	Used to create, read, or update an M1 Service Class via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ServiceClass business object.
M1-ShiftCostProfile	Used to create, read, or update an M1 Shift Cost Profile via a Web service. The CRUD operations can be performed through these components against the M1-ShiftCostProfile business object.
M1-SpeedProfileGeoArea	Used to create, read, or update an M1-SpeedProfileGeoArea via a Web service. The CRUD operations can be performed through these components against the M1-SpeedProfileGeoArea business object.
M1-Skill	Used to create, read, or update an M1 Skill via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Skill business object.

Component	Description
M1-StatusReason	Used to create, read, or update an M1 Status Reason via a Web service. After creation, the CRUD operations can be performed through these components against the M1-StatusReason business object.
M1-WorkCalendar	Used to create, read, or update an M1-WorkCalendar via a Web service. After creation, the CRUD operations can be performed through these components against the M1-WorkCalendar business object.
M1-VehicleType	Used to create, read, or update an M1 Vehicle Type via a Web service. After creation, the CRUD operations can be performed through these components against the M1-VehicleType business object.
M1-WorkProfile	Used to create, read, or update an M1 Work Profile via a Web service. After creation, the CRUD operations can be performed through these components against the M1-WorkProfile business object.
M1-WorkerAttribute	Used to create, read, or update an M1 WorkerAttribute via a Web service. After creation, the CRUD operations can be performed through these components against the M1-WorkerAttribute business object.
M1_BatchExecuteBS	Used to create, read, or update an M1 BatchExecuteBS via a Web service. After creation, the CRUD operations can be performed through these components against the M1-BatchExecuteBS business object.

Connected MCP Processing

Component	Description
ConnectedMCP_CompleteActivity	Performs the actions of a field worker for completing an M1 Activity using Connected MCP. **Note:** To be called only after calling the ConnectedMCP_StartActivity component.

Component	Description
ConnectedMCP_CompleteBreak	Performs the actions of a field worker for completing an M1 Break Task using Connected MCP. **Note:** To be called only after calling the ConnectedMCP_StartBreak component.
ConnectedMCP_CompleteNPT	Performs the actions of a field worker for completing an M1 Non Productive Task using Connected MCP. **Note:** To be called only after calling the ConnectedMCP_StartNPT component.
ConnectedMCP_CompletePOU	Performs the actions of a field worker for completing an M1 POU task using Connected MCP. **Note:** To be called only after calling the ConnectedMCP_StartPOU component.
ConnectedMCP_CompleteShift	Performs the actions of a field worker for completing a shift using Connected MCP. **Note:** To be called only after a shift is in 'start' status.
ConnectedMCP_DispatchedTask	Performs the actions of a field worker to check if a task has been dispatched using Connected MCP. Then, click it.
ConnectedMCP_EnrouteActivity	Performs the actions of a field worker to move the status of a task to 'En-route' using Connected MCP. Then, click it. **Note**: To be called only after calling the ConnectedMCP_ DispatchedTask component.
ConnectedMCP_EnrouteNPT	Performs the actions of a field worker to move the status of an NPT task to 'En-route' using Connected MCP. Then, click it. **Note**: To be called only after calling the ConnectedMCP_ DispatchedTask component.
ConnectedMCP_EnroutePOU	Performs the actions of a field worker to move the status of an NPT task to 'En-route' using Connected MCP. Then, click it. **Note**: To be called only after calling the ConnectedMCP_ DispatchedTask component.
ConnectedMCP_StartActivity	Performs the actions of a field worker to move the status of an activity to 'Started' using Connected MCP. Then, click it. **Note**: To be called only after calling the ConnectedMCP_ EnrouteActivity component.

Component	Description
ConnectedMCP_StartBreak	Performs the actions of a field worker to move the status of a break task to 'Started' using Connected MCP. Then, click it. **Note**: To be called only after calling the ConnectedMCP_ DispatchedTask component.
ConnectedMCP_StartCrewShift	Performs the actions of a field worker to start a crew shift using Connected MCP. Then, click it.
ConnectedMCP_StartNPT	Performs the actions of a field worker to move the status of an activity to 'Started' using Connected MCP. Then, click it. **Note**: To be called only after calling the ConnectedMCP_ EnrouteNPT component.
ConnectedMCP_StartPOU	Performs the actions of a field worker to move the status of a POU to 'Started' using Connected MCP. Then, click it. **Note**: To be called only after calling the ConnectedMCP_ EnroutePOU component.

Contractor Portal

Component	Description
M1-Capacity	Used to create, read, or update an M1 Capacity via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Capacity business object.
M1-CapacityTemplate	Used to create, read, or update an M1 Capacity Template via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CapacityTemplate business object.
M1-CapacityWeeklyTemplate	Used to create, read, or update an M1-CapacityWeeklyTemplate via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CapacityWeeklyTemplate business object.
M1-Company	Used to create, read, or update an M1 Company via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Company business object.

Component	Description
M1-Contractor	Used to create, read, or update an M1 Contractor via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Contractor business object.
M1-ContractorEligibility	Used to create, read, or update an M1 Contractor Eligibility via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ContractorEligibility business object.

Resource Management

Component	Description
M1-BreakTask	Used to create, read, or update an M1 Break Task via a Web service. After creation, the CRUD operations can be performed through these components against the M1-BreakTask business object.
M1-CommonShiftWeekly Template	Used to create, read, or update an M1 Common Shift Weekly Template via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CommonShiftWeeklyTemplate business object.
M1-CrewShftCloseOpenAllocate	Used to close or open a shift using M1-CrShfClAl via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CrShfClAl service script.
M1-CrewShift	Used to create, read, or update an M1 Crew Shift via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CrewShift business object.
M1-CrewShiftTemplate	Used to create, read, or update an M1 Crew Shift Template via a Web service. After creation, the CRUD operations can be performed through these components against the M1-CrewShiftTemplate business object.

Component	Description
M1-DeactivateMobileWorker	Used to deactivate an active mobile worker via a Web service. After creation, the mobile worker can be deactivated. The business object used for this component is M1-MobileWorker.
M1-DeactivateVehicle	Used to deactivate an active vehicle via a Web service. After creation, the vehicle can be deactivated. The business object used for this component is M1-Vehicle.
M1-DeactivateCrew	Used to deactivate an active crew via a Web service. After creation, crew can be deactivated. The business object used for this component is M1-Crew.
M1-DeleteResourceLeavePOU	Used for deleting M1 Resource Leave POU
M1-DepotRelatedShift	Used to create, read, or update an M1 Depot Related Shift via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DepotRelatedShift business object.
M1-DepotSinglePersonCrew	Used to create, read, or update an M1 Depot Single Person Crew via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DepotSinglePersonCrew business object.
M1-Dispatcher	Used to create, read, or update an M1 Dispatcher via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Dispatcher business object.
M1-DispatcherShift	Used to create, read, or update an M1 Dispatcher Shift via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DispatcherShift business object.
M1-DispatcherShiftLite	Used to retrieve dispatcher shift information when we only need base elements. After creation, the CRUD operations can be performed through these components against the M1-DispatcherShiftLite business object.
M1-ForceLogOffCrewShift	Used to force logging off an MDT crew shift

Component	Description
M1-GetShiftCapabilities	Used to retrieve the capabilities of a crew shift
M1-HazardArea	Used to create, read, or update an M1 Hazard Area via a Web service. After creation, the CRUD operations can be performed through these components against the M1-HazardArea business object.
M1-MailingList	Used to create, read, or update an M1 Mailing List via a Web service. After creation, the CRUD operations can be performed through these components against the M1-MailingList business object.
M1-MainMail	Used to create M1 MainMail
M1-MobileWorker	Used to create, read, or update an M1 Mobile Worker via a Web service. After creation, the CRUD operations can be performed through these components against the M1-MobileWorker business object.
M1-NonProductiveTask	Used to create, read, or update an M1 NonPrdTask via a Web service. After creation, the CRUD operations can be performed through these components against the M1-NonPrdTask business object.
M1-POUTask	Used to create, read, or update an M1 POU Task via a Web service. After creation, the CRUD operations can be performed through these components against the M1-POUTask business object.
M1-RealPOU	Used to create, read, or update an M1 Real POU via a Web service. After creation, the CRUD operations can be performed through these components against the M1-RealPOU business object.
M1-ResourceTimesheet	Used to create, read, or update an M1 Res Time Sheet via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ResTimesheet business object.

Component	Description
M1-Resourceleave	Used to create, read, or update an M1 Resource Leave via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ResourceLeave business object.
M1-ResourceLite	Used to retrieve dispatcher information when we only need base elements, however it also includes other elements which may only be relevant for other Resource business objects (Crews, Mobile Workers, Vehicles, etc). After creation, the CRUD operations can be performed through these components against the M1-ResourceLite business object.
M1-ShiftWeeklyTemplate	Used to create, read, or update an M1 Shift Weekly Template via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ShiftWeeklyTemplate business object.
M1-SimpleProcedure	Used to create, read, or update an M1 Simple Procedure via a Web service. After creation, the CRUD operations can be performed through these components against the M1-SimpleProcedure business object.
M1-SinglePersonCrew	Used to create, read, or update an M1 Single Person Crew via a Web service. After creation, the CRUD operations can be performed through these components against the M1-SinglePersonCrew business object.
M1-SubscriptionWeeklyTemplate	Used to create, read, or update an M1 Subscription Weekly Template via a Web service. After creation, the CRUD operations can be performed through these components against the M1-SubscriptionWeeklyTemp business object.
M1-TemplateMail	Used to create, read, or update an M1 Template Mail via a Web service. After creation, the CRUD operations can be performed through these components against the M1-TemplateMail business object.

Component	Description
M1-TemplatePOU	Used to create, read, or update an M1 Template POU via a Web service. After creation, the CRUD operations can be performed through these components against the M1-TemplatePOU business object.
M1-Vehicle	Used to create, read, or update an M1 Vehicle via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Vehicle business object.
M1-VehicleLeaveArea	Used to create, read, or update an M1 Vehicle Leave Area via a Web service. After creation, the CRUD operations can be performed through these components against the M1-VehicleLeaveArea business object.
M1-getPOUTaskIdForPOUId	Used to retrieve the POU task ID (the task scheduled to a crew shift) for a given real POU ID.
M1-getdeactivateMWId	Used to retrieve the mobile worker ID for the given user whose BO status is 'Active'.
M1-getdeactivateVehicleId	Used to retrieve the vehicle ID for the given user whose BO status is 'Active'.
M1-getdeactivateCrewId	Used to retrieve the crew ID for the given user whose BO status is 'Active'.

Scheduling

Component	Description
M1-AllocateActivityToAShift	Used to manually allocate an activity to a crew shift.
M1-AppointmentBookingRequest	Used to trigger an M1 Appointment Booking Request.
M1-CheckIfActivityIsScheduled	Verifies if an activity is scheduled to a crew shift.
M1-DailySpeedTemplate	Used to create, read, or update an M1 Daily Speed Template via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DailySpeedTemplate business object.
M1-FixToCrew	Used to fix an activity to be scheduled to a particular crew
M1-SchedulerRegistryRead	Used to retrieve the scheduler registry details for an entity

Component	Description
M1-SpeedProfileTemplate	Used to create, read, or update an M1 Speed Profile Template via a Web service. After creation, the CRUD operations can be performed through these components against the M1-SpeedProfileTemplate business object.
M1-TaskScheduleDetails	Used for reading an M1 Task Schedule entity.
M1-UnAssignActivity	Used to unassign an activity from the crew shift.
M1-WaitForTime	Used to wait for certain amount of time. The unit of measure is minutes.

Service Management

Component	Description
M1-Activity	Used to create, read, or update an M1 Activity via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Activity business object.
M1-Assignment	Used to create, read, or update an M1 Assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M1-Assignment business object.
M1-ComplexActivity	Used to create, read, or update an M1 Complex Activity via a Web service. After creation, the CRUD operations can be performed through these components against the M1-ComplexActivity business object.
M1-DeferActivity	Used to defer an activity
M1-DepotRelatedActivity	Used to create, read, or update an M1 Depot Related Activity via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DepotRelatedActivity business object.
M1-DepotTask	Used to create, read, or update an M1 Depot Task via a Web service. After creation, the CRUD operations can be performed through these components against the M1-DepotTask business object.
M1-GetAssignmentIdForTaskId	Used to retrieve the assignment ID of a dispatched task

Component	Description
M1-GetDepotTaskId	Used to retrieve the depot task ID associated to a given depot related or a pre-requisite activity.
M1-GetDeploymentId	Used to retrieve the deployment ID for the given deployment type.
M1-OverideTimeWindow	Used to create, read, or update an M1 Overide Time Window via a Web service. After creation, the CRUD operations can be performed through these components against the M1-OverideTimeWindow business object.
M1-ActivityCommitByCont	Used to commit the work done by the contractors via a Web service.
M1-ActivityCompleteByCont	Used to complete the work done by Contractors via Web service.

M2 Activities

Component	Description
M2-InstallMeterActivity	Used to create, read, or update an M2 Install Meter Activity via a Web service. After creation, the CRUD operations can be performed through these components against the M1-InstallMeterActivity business object.
M2-InstallMeterAssignment	Used to create, read, or update an M2 Install Meter Assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2-InstallMeterAssignment business object.
M2-BasicItemActivity	Used to create, read, or update an M2 Basic Item Activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2-BasicItemActivity business object.
M2-BasicItemAssignment	Used to create, read, or update a basic item assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2-BasicItemAssignment business object.

Component	Description
M2- ConnectSPActivity	Used to create, read, or update an M2 Connect Service Point Activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2- ConnectSPActivity business object.
M2-ConnectSPAssignment	Used to create, read, or update a connect service point assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2-ConnectSPAAssigngnment business object.
M2-CutNonPayItemActivity	Used to create, read, or update an M2 Cut For Non Payment Item Activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2- CutNonPayItemActivity business object.
M2-CutNonPayItemAssignment	Used to create, read, or update a cut for non payment item assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2-CutNonPayItemAssignment business object.
M2-DisconnectMeterActivity	Used to create, read, or update a disconnect meter activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2- DisconnectMeterActivity business object.
M2-DisconnectMeterAssignment	Used to create, read, or update a disconnect meter assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2-DisconnectMeterAssignment business object.
M2-DisconnectSPActivity	Used to create, read, or update a disconnect service point activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2-DisconnectSPActivity business object.
M2-DisconnectSPAssignment	Used to create, read, or update a disconnect service point assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2- DisconnectSPAssignment business object.

Component	Description
M2-ExchangeItemActivity	Used to create, read, or update an exchange item activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2- ExchangeItemActivity business object.
M2-ExchangeItemAssignment	Used to create, read, or update an exchange item assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2- ExchangeItemAssignment business object.
M2-InstallItemActivity	Used to create, read, or update an install item activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2- InstallItemActivity business object.
M2-InstallItemAssignment	Used to create, read, or update an install item assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2-InstallItemAssignment business object.
M2-MeterReadActivity	Used to create, read, or update a meter read activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2- MeterReadActivity business object.
M2-MeterReadAssignment	Used to create, read, or update a meter read assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2-MeterReadAssignment business object.
M2-RemoveItemActivity	Used to create, read, or update a remove item activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2- RemoveItemActivity business object.
M2-RemoveItemAssignment	Used to create, read, or update a remove item assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2-RemoveItemAssignment business object.

Component	Description
M2-RemoveMeterActivity	Used to create, read, or update a remove meter activity via a Web service. After creation, the CRUD operations can be performed through these components against the M2- RemoveMeterActivity business object.
M2-RemoveMeterAssignment	Used to create, read, or update a remove meter assignment via a Web service. After creation, the CRUD operations can be performed through these components against the M2-RemoveMeterAssignment business object.
M2-MaintUtilityActByHost	Used to create an M2 Activity from an external system

Hybrid MCP Processing

Component	Description
HybridMCP_StartShift	Performs the actions of a field worker to start a crew shift using HybridMCP. Then, clicks it.
HybridMCP_TaskDispatched	Verifies whether task (activity, break, NPT, or POU) is dispatched or not. Then, clicks it from the Task List screen.
HybridMCP_EnrouteActivity	Performs the actions of a field worker to move the status of a task to 'En-route' using HybridMCP. Then, clicks it. **Note**: To be called only after calling the HybridMCP _DispatchedTask component.
HybridMCP_StartActivity	Performs the actions of a field worker to move the status of an activity to 'Started' using HybridMCP. Then, clicks it. **Note**: To be called only after calling the HybridMCP _EnrouteActivity component.
HybridMCP_CompleteActivity	Performs the actions of a field worker for completing an M1 Activity using HybridMCP. **Note**: To be called only after calling the HybridMCP _StartActivity component.
HybridMCP_StartBreak	Performs the actions of a field worker to move the status of a break task to 'Started' using HybridMCP. Then, clicks it. **Note**: To be called only after calling the HybridMCP _DispatchedTask component.

Component	Description
HybridMCP_CompleteBreak	Performs the actions of a field worker for completing an M1 Break Task using HybridMCP. **Note**: To be called only after calling the HybridMCP _StartBreak component.
HybridMCP_EnrouteNPT	Performs the actions of a field worker to move the status of an NPT task to 'En-route' using HybridMCP. Then, clicks it. **Note**: To be called only after calling the HybridMCP _DispatchedTask component.
HybridMCP_StartNPT	Performs the actions of a field worker to move the status of an activity to 'Started' using HybridMCP. Then, clicks it. **Note**: To be called only after calling the HybridMCP _EnrouteNPT component.
HybridMCP_CompleteNPT	Performs the actions of a field worker for completing an M1 Non Productive Task using HybridMCP. **Note**: To be called only after calling the HybridMCP_StartNPT component.
HybridMCP_EnroutePOU	Performs the actions of a field worker to move the status of an NPT task to 'En-route' using HybridMCP. Then, clicks it. **Note**: To be called only after calling the HybridMCP_DispatchedTask component.
HybridMCP_StartPOU	Performs the actions of a field worker to move the status of a POU to 'Started' using HybridMCP. Then, clicks it. **Note**: To be called only after calling the HybridMCP_EnroutePOU component.
HybridMCP_CompletePOU	Performs the actions of a field worker for completing an M1 POU task using ConnectedMCP. **Note**: To be called only after calling the HybridMCP_StartPOU component.
HybridMCP_CompleteShift	Performs the actions of a field worker for completing a shift using HybridMCP. **Note**: To be called only after a shift is in 'start' status.
HybridMCP_getCurrentTime	Returns the current date and time. It is designed for entering the current date and time value in the MDT Tag while logging into the HybridMCP Shift.

Component	Description
Webdriver_AndroidLogin	Performs the crew shift login on an Android emulator or android device. **Note**: To be called only after configuring an Android device or emulator. More details can be found on Sample flows Section for Android flow.
Webdriver_completeBreak	Performs the actions of a field worker for completing an M1 Break Task using Web Driver on an Android / iOS emulator or device. **Note**: To be called only after calling the Webdriver_startBreak component.
Webdriver_completeM1Activity	Performs the actions of a field worker for completing an M1 Activity using Web Driver on an Android / iOS emulator or device. **Note**: To be called only after calling the Webdriver_startM1Activity component.
Webdriver_completeM2Activity	Performs the actions of a field worker for completing an M2 Activity using Web Driver on an Android / iOS emulator or device. **Note**: To be called only after calling the Webdriver_startM2Activity component.
Webdriver_completeNPT	Performs the actions of a field worker for completing an Non Productive Task using Web Driver on an Android / iOS emulator or device. **Note**: To be called only after calling the Webdriver_startNPTcomponent.
Webdriver_completePOU	Performs the actions of a field worker for completing an M1 POU task using Web Driver on an Android / iOS emulator or device. **Note**: To be called only after calling the Webdriver_completePOU component.
Webdriver_endShift	Performs the actions of a field worker for completing a shift using Web Driver on an Android / iOS emulator or device.. **Note**: To be called only after a shift is in 'start' status.
Webdriver_gotoTaskHome	Performs the actions of a field worker for navigating to the task list home screen using Web Driver on an Android / iOS emulator or device.

Component	Description
Webdriver_iOSLogin	Performs the crew shift login on an iOS emulator. **Note**: To be called only after configuring an iOS emulator. More details can be found on Sample flows Section for iOS flow.
Webdriver_selectDeployment	Performs the actions of a field worker for selecting the active deployment before the crew shift starts, using Web Driver on an Android / iOS emulator or device.
Webdriver_startBreak	Performs the actions of a field worker to move the status of a break task to 'Started' using Web Driver on an Android / iOS emulator or device. **Note**: To be called only after calling the Webdriver_waitForTaskDispatch component.
Webdriver_startM1Activity	Performs the actions of a field worker to move the status of an M1activity to 'Started' using Web Driver on an Android/ iOS emulator or device. **Note**: To be called only after calling the Webdriver_startM1Activity component.
Webdriver_startM2Activity	Performs the actions of a field worker to move the status of an M2activity to 'Started' using Web Driver on an Android / iOS emulator or device. **Note**: To be called only after calling the Webdriver_startM2Activity component.
Webdriver_startNPT	Performs the actions of a field worker to move the status of the NPT to 'Started' using Web Driver on an Android/ iOS emulator or device. **Note**: To be called only after calling the Webdriver_waitForTaskDispatch component.
Webdriver_startPOU	Performs the actions of a field worker to move the status of the POU to 'Started' using Web Driver on an Android/ iOS emulator or device. **Note**: To be called only after calling the Webdriver_waitForTaskDispatch component.
Webdriver_startShift	Performs the actions of a field worker to move the status of the Crew shift to 'Started' using Web Driver on an Android/ iOS emulator or device.

Component	Description
Webdriver_waitForTaskDispatch	Performs the actions of a field worker to verify all the assigned tasks are dispatched in the MDT using Web Driver on an Android/ iOS emulator or device.

Chapter 2

Function Library Reference

This chapter lists the Oracle Utilities Mobile Workforce Management function libraries and functions available to create components and flows in Oracle Flow Builder for testing the Oracle Utilities Mobile Workforce Management application.

OUMWMLIB

The OUMWMLIB library comprises functions that work on the connected MCP browser UI. These functions mimic the actions of a field worker who updates the task/shift information into the application using the connected MCP.

The library is made up of a collection of actions, such as clicking various buttons, entering text into fields, navigating through the connected MCP UI, etc., used for creating the required functions. These functions are, in turn, used for creating the connected MCP components.

The functions use the object repository "MWM_ConnectedMCP_REPOSITORY.properties" that is in the "etc" directory of the Oracle Functional Testing Advanced Pack for Oracle Utilities repository.

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

getTaskScheduledShiftId

Retrieves the Crew Shift ID to which a given task is scheduled.

This function checks continuously for the given time until it finds the crew shift that the activity is scheduled to. If the activity is not scheduled within the given time, it returns a blank string.

Example:

```
getTaskScheduledShiftId(String <timeInMinutes>, String <activityId>)

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

getPOUTaskIDForPOUId

Retrieves the period of unavailability Task ID for a given real-time period of unavailability ID.

This function checks continuously for the given time to retrieve the POU Task ID. If the POU Task ID is not created within the time, it returns a blank string.

Example:

```
getPOUTaskIDForPOUId(String <timeInMinutes>, String <realPOUId>)

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

getAssignmentIDForTaskId

Retrieves the Assignment ID for a given Activity ID. The assignment will be created when the activity is in "Assignment In Progress" state.

This function checks for the Assignment ID for a given Activity ID until the given time. If an assignment is not created within the time, it returns a blank string.

Example:

```
getAssignmentIDForTaskId(String <timeInMinutes>, String <activityId>)

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

getAssignmentCurrentStatus

Retrieves the current status of an assignment.

This function returns a blank string if the assignment does not exist for the given ID.

Example:

```
getAssignmentCurrentStatus (String <assignmentId>)

Input Parameters: String
Return Type: String
```

verifyIfAssignmentIsDispatched

Verifies if an assignment has been dispatched or not.

This function checks until the provided time to assert if the assignment has been dispatched. If the assignment is dispatched, it returns "Y". Else, returns "N".

Example:

```
verifyIfAssignmentIsDispatched (String <timeInMinutes>, String
<assignmentId>)

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

checkActivityScheduledState

Verifies if an activity has been scheduled to any shift.

This function checks for the activity to be scheduled to a shift for a given time. If the activity is scheduled with in the given time, it returns "Y", else returns "N".

Example:

```
checkActivityScehduledState (String <timeInMinutes>, String
<activityId>)

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

getProcedureId

Retrieves the Procedure ID for a given procedure name and Crew Shift ID.

This function checks for a procedure to be created for the procedure name and Crew Shift ID within the given time. If the procedure ID is not found within the given time, it returns a blank string. Else, returns the procedure ID.

Example:

```
getProcedureId(String <procedureName>,String <crewShiftId>,String
<timeInMinutes>)

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

getTimesheetIdFromShiftId

Retrieves the Timesheet ID for a given Crew Shift ID.

This function checks for a time sheet to be created for the Crew Shift ID within the given time. If Timesheet ID is not found within the given time, it returns a blank string, else returns the Timesheet ID.

Example:

```
getTimesheetIdFromShiftId(String <crewShiftId>, String
<timeInMinutes>)

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

getVisitIds

Retrieves the Visit IDs (comma separated) for a given Complex Activity ID.

If the Visit ID is not found, it returns a blank string. Else, it returns the Visit IDs.

Example:

```
getVisitId(String <complexActivityId>)

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

getDeploymentId

Retrieves the Deployment ID for a given Deployment Type. This function checks for the latest deployment generated for the given Deployment Type.

If the Deployment ID is not found, it returns a null. Else, it returns the Deployment ID.

Example:

```
getDeploymentId (String < deploymentType>)

Input Parameters: String
Return Type: String
```

deactivateCrew

Retrieves the Crew ID for the given Crew Name. This function checks for active Crew ID with the given Crew Name.

If the Crew ID is not found, it returns an empty string. Else, it returns the Crew ID.

Example:

```
deactivateCrew (String < crewName>)

Input Parameters: String
Return Type: String
```

deactivateMobileWorkerId

Retrieves the Mobile ID for the given User ID. This function checks for active mobile worker with the given User ID.

If the Mobile Worker ID is not found, it returns an empty string. Else, it returns the Mobile Worker ID.

Example:

```
deactivateMobileWorkerId (String < User_ID>)

Input Parameters: String
Return Type: String
```

deactivateVehicle

Retrieves the Vehicle ID for the given Vehicle Tag. This function checks for active Vehicle with the given Vehicle Tag.

If the Vehicle ID is not found, it returns an empty string. Else, it returns the Vehicle ID.

Example:

```
deactivateVehicle (String < vehicleTag>)

Input Parameters: String
Return Type: String
```

OUMWMCONNECTEDMCPLIB

The OUMWMCONNECTEDMCPLIB library contains functions that have been/can be used for creating Oracle Utilities Mobile Workforce Management components used to create the Oracle Utilities Mobile Workforce Management flows.

The library includes the Oracle Utilities Mobile Workforce Management specific functions that use the underlying framework of core functions. These functions interface with the database and read information, such as scheduling information of tasks, etc.

The library uses the UI element's xpath values specified in the MWM_ConnectedMCP_REPOSITORY.properties file in the **etc** directory of the repository.

> **Note:** The private functions in this library are used to implement the public functions. The private functions cannot be used directly in the Oracle Flow Builder scripts.

invokeApp

Launches the Web browser, navigates to the URL provided, and then checks for the object in the Web page.

If the object is found, the function returns "Y", else returns "N".

Example:

```
loginToMDT(String <URL>,String <objectToCheck>)

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

loginToMDT

Internally calls the invokeApp function with the MCP URL that is connected.

This function inputs the User ID and password, and logs into the MCP application that is connected. It returns "Y" on the successful login, else returns "N".

If the URL parameter is specified as "NA", the connected MCP URL is formulated using the application URL specified in the configuration.properties file by appending "/mobility/Login.jsp" to that URL. If the connected MCP URL is specified to this function, then it is used instead.

Example:

```
loginToMDT(String <username>, String <password>, String
<connectedMCPURL>)

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

downloadDeployment

Internally calls the loginToMDT function and waits for the deployment to download.

This function verifies if the **Start Shift** page is displayed within a minute. If the page is displayed, it returns "Y", else it returns "N".

If the URL parameter is specified as "NA", the connected MCP URL is formulated using the application URL specified in the configuration.properties file by appending "/mobility/Login.jsp" to that URL. If the connected MCP URL is specified to this function, then it is used instead.

Example:

```
downloadDeployment(String <username>, String <password>, String
<connectedMCPURL>)

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

startShift

Internally calls the downloadDeployment function and waits for the crew shift to start.

This function verifies if the Open Task List page is displayed. If the page is displayed, it returns "Y", else it returns "N". The loginError should be set as "N" for the crew shift to start successfully.

The function also verifies the negative scenarios where an error message is expected to be displayed during the shift start. The error message verification can be triggered by setting the loginError flag as "Y" as the input. If the loginError is set as "Y", the function checks for an error message to be displayed, returns the error message, and then logs off the crew shift. If no error message is retrieved, it then returns a blank string.

If the URL parameter is specified as "NA", the connected MCP URL is formulated using the application URL specified in the configuration.properties file by appending "/mobility/Login.jsp" to that URL. If the connected MCP URL is specified to this function, then it is used instead.

Example:

```
startShift(String <username>, String <password>, String
<connectedMCPURL>, String <loginError>)

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

verifyTaskDispatched

Verifies if a task has been dispatched for a given Task ID and then clicks on the task to open the UI.

This function returns a "Y" if the task has been dispatched. It checks for a task to be dispatched to the connected MCP within five minutes. If it is not dispatched, it returns a "N".

Example:

```
verifyTaskDispatched(String <tasked>)

Input Parameters: String
Return Type: String
```

enrouteActivity

Performs the actions of a field worker and moves the task to "Enroute" status using the connected MCP.

This function returns "Y" on success, else returns "N". To assign an activity, it has to be called after calling the verifyTaskDispatched function.

Example:

```
enrouteActivity ()

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

startActivityFromEnroute

Performs the actions of a field worker and moves the task to "Started" status using the connected MCP.

This function returns "Y" on success, else returns "N". To assign an activity, it has to be called after calling the enrouteActivity function.

Example:

```
startActivityFromEnroute ()

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

completeActivity

Performs the actions of a field worker and moves the task to "Completed" status using the connected MCP.

This function returns "Y" on success, else returns "N". To assign an activity, it has to be called after calling the startActivityFromEnroute function.

Example:

```
completeActivity ()

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

startBreak

Performs the actions of a field worker and moves the break task status to "Started" using the connected MCP.

This function returns "Y" on success, else returns "N". To assign a break task, it has to be called after calling the verifyTaskDispatched function.

Example:

```
startBreak ()

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

completeBreak

Performs the actions of a field worker and moves the break task status to "Completed" using the connected MCP.

This function returns "Y" on success, else returns "N". To assign a break task, it has to be called after calling the startBreak function.

Example:

```
completedBreak ()

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

enrouteNPT

Performs the actions of a field worker and moves the non productive task (NPT) status to "Enroute" using the connected MCP.

This function returns "Y" on success, else returns "N". To assign a non productive task, it has to be called after calling the verifyTaskDispatched function.

Example:

```
enrouteNPT ()

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

startNPT

Performs the actions of a field worker and moves the non productive task (NPT) status to "Started" using the connected MCP.

This function returns "Y" on success, else returns "N". To assign a non productive task, it has to be called after calling the enrouteNPT function.

Example:

```
startNPT ()

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

enroutePOU

Performs the actions of a field worker and moves the period of unavailability (POU) task status to "Enroute" using the connected MCP.

This function returns "Y" on success, else returns "N". To assign a POU task, it has to be called after calling the verifyTaskDispatched function.

Example:

```
enroutePOU ()

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

startPOU

Performs the actions of a field worker and moves the period of unavailability (POU) task status to "Started" using the connected MCP.

This function returns "Y" on success, else returns "N". To assign a POU task, it has to be called after calling the enroutePOU function.

Example:

```
startPOU ()

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

completePOU

Performs the actions of a field worker and moves the period of unavailability (POU) task status to "Completed" using the connected MCP.

This function returns "Y" on success, else returns "N". To assign a POU task, it has to be called after calling the startPOU function.

Example:

```
completePOU()

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

endShift

Completes/ends the crew shift by performing the actions of a field worker to complete the current day's crew shift using the connected MCP.

This function also closes the connected MCP application. It returns "Y" on success, else returns "N".

Example:

```
endShift()

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

OUMWMMCPLIB

The OUMWMMCPLIB library contains functions that have been/can be used for creating Oracle Utilities Mobile Workforce Management components.These components are later used to create the Oracle Utilities Mobile Workforce Management flows.

The library includes the Oracle Utilities Mobile Workforce Management specific functions that use the underlying framework of core functions. These functions interface with the database and read information, such as scheduling information of tasks, etc.

The library uses the UI elements' xpath values specified in the MCP.properties file added as an asset to this functional library.

The library includes three new functions (hybridMCP_startShift, hybridMCP_verifyTaskDispatched, hybridMCP_endShift) to support the HybridMCP MDT flow.

> **Note**: The private functions in this library are used to implement the public functions. The private functions cannot be used directly in the Oracle Flow Builder scripts.

startShift

Internally calls the downloadDeployment function and waits for the crew shift to start. This Function invokes the application in Chrome browser using the URL specified in config.properties file. The URL is provided as a value to the variable gStrNewMCPURL in config.properties file.

The function fetches this URL and invokes the application, and then logs in to the application fetching user credentials from config.properties file. On successful login, it checks for the available deployments. It selects the required deployment and waits for it to be downloaded.

Once the deployment is downloaded, it starts the crew shift and waits for the tasks to get dispatched in The **Open Tasks List** screen. If in case, the user tries to launch the application for an already started shift, it directly logs into the application and waits for the activities to get dispatched in the Open Task List screen.(No start shift step is performed as it is referring to already started shift.)

The function also verifies the negative scenarios where an error message is expected to be displayed when user login fails for any of the reasons, such as invalid user credentials, etc.

Example:

```
startShift(String <username>, String <password>, String HardwareId,
String <deploymentType>)

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

verifyTaskDispatched

Verifies if a task has been dispatched for a given Task ID for a specified amount of time maxTimeToCheck, and then clicks on the task in UI to perform actions on it.

This function returns a "Y" if the task has been dispatched. It checks for a task to be dispatched to the new MCP browser application within specified amount of time in minutes. If it is not dispatched, it returns a "N".

Example:

```
verifyTaskDispatched(String <taskId>,String  maxTimeToCheck)

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

enrouteActivity

Performs the actions of a field worker and moves the task to "Enroute" status using the Next Gen MCP_ EnrouteActivity component.

This function returns "Y" on success, else returns "N". To assign an activity, it has to be called after calling the verifyTaskDispatched function.

Example:

```
enrouteActivity ()

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

startActivityFromEnroute

Performs the actions of a field worker and moves the task to "Started" status using the NextGenMCP_StartActivity component.

This function returns "Y" on success, else returns "N". To assign an activity, it has to be called after calling the enrouteActivity function.

Example:

```
startActivityFromEnroute ()

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

completeActivity

Performs the actions of a field worker and moves the task to "Completed" status using the Next Gen MCP_ CompleteActivity component

This function returns "Y" on success, else returns "N". To assign an activity, it has to be called after calling the startActivityFromEnroute function.

Example:

```
completeActivity ()

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

startBreak

Performs the actions of a field worker and moves the break task status to "Started" using the Next Gen MCP_ StartBreak component

This function returns "Y" on success, else returns "N". To assign a break task, it has to be called after calling the verifyTaskDispatched function.

Example:

```
startBreak ()

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

completeBreak

Performs the actions of a field worker and moves the break task status to "Completed" using the Next Gen MCP_CompleteBreak component

This function returns "Y" on success, else returns "N". To assign a break task, it has to be called after calling the startBreak function.

Example:

```
completedBreak ()

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

enrouteNPT

Performs the actions of a field worker and moves the non productive task (NPT) status to "Enroute" using the Next Gen MCP._EnrouteNPT component.

This function returns "Y" on success, else returns "N". To assign a non productive task, it has to be called after calling the verifyTaskDispatched function.

Example:

```
enrouteNPT ()

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

startNPT

Performs the actions of a field worker and moves the non productive task (NPT) status to "Started" using the Next Gen MCP_StartNPT Component

This function returns "Y" on success, else returns "N". To assign a non productive task, it has to be called after calling the enrouteNPT function.

Example:

```
startNPT ()

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

completeNPT

Performs the actions of a field worker and moves the period of unavailability (POU) task status to "Completed" using the Next Gen MCP._CompleteNPT component.

This function returns "Y" on success, else returns "N". To assign a POU task, it has to be called after calling the startPOU function.

Example:

```
completeNPT()

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

enroutePOU

Performs the actions of a field worker and moves the period of unavailability (POU) task status to "Enroute" using the Next Gen MCP._EnroutePOU component.

This function returns "Y" on success, else returns "N". To assign a POU task, it has to be called after calling the verifyTaskDispatched function.

Example:

```
enroutePOU ()

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

startPOU

Performs the actions of a field worker and moves the period of unavailability (POU) task status to "Started" using the Next Gen MCP_StartPOU component.

This function returns "Y" on success, else returns "N". To assign a POU task, it has to be called after calling the enroutePOU function.

Example:

```
startPOU ()

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

completePOU

Performs the actions of a field worker and moves the period of unavailability (POU) task status to "Completed" using the Next Gen MCP_CompletePOU component.

This function returns "Y" on success, else returns "N". To assign a POU task, it has to be called after calling the startPOU function.

Example:

```
completePOU()

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

endShift

Completes/ends the crew shift by performing the actions of a field worker to complete the current day's crew shift using the Next Gen MCP_ CompleteShift component.

This function also closes the Next Gen MCP application. It returns "Y" on success, else returns "N".

Example:

```
endShift()

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

CheckforAnActivityIncompletionScreen

Navigates to the **Activity Completion** screen and verifies if the activity with specified Activity ID is in completed state for a time specified as maxTimetoCheck in minutes.

If found, it returns "true", else "false".

Example:

```
CheckforAnActivityInCompletionScreen(String taskID,String
maxTimetoCheck)

Input Parameters: <String,String>
Return Type: boolean
```

moveToOpenTasksAfterTaskComplete

Performs actions such as navigating back to open the task list screen from the **Activity Completion** screen. It returns TRUE on successful navigation to open the task list screen. Else, it returns false.

Example:

```
moveToOpenTasksAfterTaskComplete ()

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

hybridMCP_startShift

Works similar to the startShift function. This function is designed to handle the Hybrid MCP browser, where it additionally enters the MDT tag. It internally calls the downloadDeployment function and waits for the crew shift to start. It invokes the application in the Google Chrome browser using the URL specified in config.properties file. The URL is provided as a value to the variable gStrNewMCPURL in the config.properties file.

This function fetches the URL and invokes the application, and then logs in to the application fetching user credentials from the config.properties file. On successful login, it checks for the available deployments. It selects the required deployment and waits for it to be downloaded.

After the deployment is downloaded, it starts the crew shift and waits for the tasks to get dispatched in **The Open Tasks List** screen. If in case, the user tries to launch the application for an already started shift, it directly logs into the application and waits for the activities to get

dispatched in the **Open Task List** screen.(No start shift step is performed as it is referring to already started shift.)

The function also verifies the negative scenarios where an error message is expected to be displayed when user login fails for any of the reasons, such as invalid user credentials.

Example:

```
startShift(String <username>, String <password>, String HardwareId,
String <MDTTag> String <deploymentType>)

Input Parameters: String, String, String, String, String.
Return Type: String
```

hybridMCP_verifyTaskDispatched

This function works similar to the verifyTaskDispatched function, except that it is designed to handle the HybridMCP browser. It verifies if a task has been dispatched for a given Task ID for a specified amount of time maxTimeToCheck, and then clicks on the task in the UI to perform actions on it.

This function returns a "Y" if the task has been dispatched. It checks for a task to be dispatched to the new MCP browser application within the specified time (in minutes). If it is not dispatched, it returns a "N".

Example:

```
verifyTaskDispatched(String <taskId>,StringmaxTimeToCheck)

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

hybridMCP_endShift

This function works similar to the verifyTaskDispatched function, except that it is designed to handle HybridMCP browser. It completes/ends the crew shift by performing the actions of a field worker to complete the current day's crew shift using the HybridMCP_ CompleteShift component.

This function also closes the HybridMCP application. It returns "Y" on success, else returns "N".

Example:

```
endShift()

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

OUMWMHYBRIDLIB

The OUMWMHYBRIDLIB library includes functions that are/ can be used to create Oracle Utilities Mobile Workforce Management Mobile web driver related components. These components are later used to create the Oracle Utilities Mobile Workforce Management flows.

The library includes the Oracle Utilities Mobile Workforce Management web driver specific functions that use mobile specific activities.

> **Note**: The private functions in this library are used to implement the public functions. The private functions cannot be used directly in the Oracle Flow Builder scripts.

AndroidMCP_startRemoteDriver

This function is used with the Android-MDT Flow. It is an initial step required to connect the android mobile device/ emulator that is running with the Appium server using the following capabilities:

- MWM Application Package Name
- MWM Application Activity Name
- Android OS Version(gStrAndroidOSVersion)
- Android Device Name(gStrAndroidDeviceName)
- Android Device UID(gStrAndroidDeviceUID)
- Android MWM Build Path(gStrAndroidBuildPath)
- Android Node URL(gStrAndroidMCPRemoteNodeURL)

Using these capabilities, the Appium server chooses the particular Android device/emulator for automation. The emulator should be running so that it automatically installs the MWM-debug build, located in the defined build-path folder. Finally, it automatically launches the Oracle Utilities Mobile Workforce Management application.

> **Note**: All variables mentioned with the capabilities should be defined in configuration.properties file.

Example:

```
AndroidMCP_startRemoteDriver()

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

IosMCP_startRemoteDriver

This method is used with the iOS-MDT Flow. This is the initial step required to connect the iOS mobile Simulator running with the Appium server (MAC) using the following capabilities:

- iOS X-Code Version(gStrIOSVersion)
- iOS Simulator Name(gStrIOSDeviceName)
- iOS Device UID(gStrIOSDeviceUID): <OPTIONAL>
- iOS Build Path(gStrIosBuildPath)
- iOS Node URL(gStrIosMCPRemoteNodeURL)

Using these capabilities, the Appium server chooses the particular iOS Simulator for automation. The Simulator automatically invokes; it automatically installs the MWM-debug build located in the defined build-path folder. Finally, it automatically launches the Oracle Utilities Mobile Workforce Management application.

> **Note**: All variables mentioned with the capabilities should be defined in configuration.properties file.

Example:

```
IosMCP_startRemoteDriver()

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

MobileMCP_switchToDebugContextNodeURL

After the Oracle Utilities Mobile Workforce Management application is successfully launched, this method handles the Application Context-Mode and switches the context from Native-View to Debug-View (Web-View).

The method is used to handle the automation on OUML code-level, as both Android and IOS applications have same OUML code. So it is easier to create and maintain the same functional flow for both application types (Android / iOS).

Note: This method is common for both MDT flows.

Example:

```
MobileMCP_switchToDebugContextNodeURL()

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

MobileMCP_selectTerms

When the Oracle Utilities Mobile Workforce Management application is launched, the Terms and Conditions need to be accepted. This method just presses the **Accept** button.

Note: This method is common for both MDT flows.

Example:

```
MobileMCP_selectTerms()

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

MobileMCP_applicationUrl

This function inputs the application URL with the username and password. It takes the following variables from the "configuration.properties" file:

- gStrMobileDeploymentURL
- gStrApplicationUserName
- gStrApplicationUserPassword

After providing input to these three fields, the function clicks the **Download** button that triggers the download app bundle process.

Example:

```
MobileMCP_applicationUrl()

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

MobileMCP_loginPage

This function performs the login action by entering the MDT-Tag, application username, and password, and then clicking the **Login** button.

Example:

```
MobileMCP_loginPage()

Input Parameters: <String>
Return Type: String
```

HybridMCP_selectDeployment

After successful login, the **Select Deployment** page appears that requires a human intervention to select a proper deployment from the available list. This method simulates this action by selecting a valid REST-DEPLOYMENT.

Example:

```
HybridMCP_selectDeployment()

Input Parameters: <String>
Return Type: String
```

HybridMCP_startShift

After selecting deployment, this method starts the crew shift in MDT by clicking **Start**.

Example:

```
HybridMCP_startShift()

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

HybridMCP_waitForAllTaskDispatch

This method enables wait on the **Task - Home** page till all tasks are dispatched in the MDT. Pass "Number of Tasks" as an argument and also the max time to wait for all tasks to dispatch. If it crosses the max time, the test fails.

Example:

```
HybridMCP_waitForAllTaskDispatch()

Input-Parameters:<String, String>
Return Type: String
```

HybridMCP_gotoTaskHome

After the completion of all tasks, go to the **Task - Home** page to see the remaining task list. This method simulates this action. It opens **Menu** and clicks **Home** from the list.

Example:

```
HybridMCP_gotoTaskHome()

Input Parameters:<String>
Return Type: String
```

HybridMCP_startBreak

On the **Task - Home** page, this method finds the break-task by the Task ID, and then selects the particular task. On that task action page, it clicks **Start**.

Example:

```
HybridMCP_startBreak()

Input Parameters:<String,SString>
Return Type: String
```

HybridMCP_completeBreak

After starting break-task, this method completes it by clicking **Complete**.

Example:

```
HybridMCP_completeBreak()

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

HybridMCP_startNonProductiveTask

On the **Task - Home** page, this method finds the NPT-task by the Task ID, and then selects the particular task. When on the task action page, it clicks **Start**.

Example:

```
HybridMCP_startNonProductiveTask()

Input Parameters:<String, String>
Return Type: String
```

HybridMCP_completeNonProductiveTask

After starting break-task, this method completes the NPT-task by clicking **Complete**.

Example:

```
HybridMCP_completeNonProductiveTask()

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

HybridMCP_startPOU

On the **Task-Home** page, this method finds the POU-task by the Task ID and then selects the particular task. When on the task action page, it clicks **Start**.

Example:

```
HybridMCP_startPOU()

Input Parameters:<String, String>
Return Type: String
```

HybridMCP_completePOU

After starting break-task, this method completes that POU-Task by clicking **Complete**.

Example:

```
HybridMCP_completePOU()

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

HybridMCP_startM1Activity

On the **Task - Home** page, this method finds the M1-task by the Task ID and then selects the particular task. When on the task action page, it clicks **Start**.

Example:

```
HybridMCP_startM1Activity()

Input Parameters:<String, String>
Return Type: String
```

HybridMCP_completeM1Activity

After starting break-task, this method completes that M1-task by clicking **Complete**.

Example:

```
HybridMCP_completeM1Activity()

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

HybridMCP_startM2ConnectSPActivity

On the **Task - Home** page, this method finds the M2-task by the Task ID and then selects the particular task. When on that task action page, it clicks **Enroute** and **Start**.

Example:

```
HybridMCP_startM2ConnectSPActivity()

Input Parameters:<String, String>
Return Type: String
```

HybridMCP_completeM2ConnectSPActivity

After starting break-task, this method completes that M2-task by clicking **Complete**.

Example:

```
HybridMCP_completeM2ConnectSPActivity()

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

HybridMCP_endShift

This method completes/ends the crew shift by performing the actions of a field worker to complete the current day's crew shift. This function also closes the Next Gen MCP application. It returns “Y” on success, else returns “N”.

Example:

```
HybridMCP_endShift()

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

HybridMCP_closeRemoteDriver

This method ends or closes the connection with the Remote IOS/ Android MDT device. It is usually used at the end of an automation test.

Example:

```
HybridMCP_closeRemoteDriver()

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

Chapter 3

Sample Work Flows

This chapter describes the Oracle Utilities Mobile Workforce Management sample flows that illustrate common use cases for Oracle Functional Testing Advanced Pack for Oracle Utilities. It also explains the procedure to execute these sample flows.

The chapter includes the following sections:

Sample Flows

The sample flows delivered as part of Oracle Functional Testing Advanced Pack for Oracle Utilities for Oracle Utilities Mobile Workforce Management demonstrate how flows can be created for Web services based testing and for a combination of Web services and UI based testing using the same framework.

These flows are designed to run using the demo data, there by giving the user, the ability to deploy Oracle Functional Testing Advanced Pack for Oracle Utilities for Oracle Utilities Mobile Workforce Management and execute the sanity flows immediately. The flows perform a part of the basic sanity testing required to certify that the Oracle Utilities Mobile Workforce Management environment has been setup appropriately.

This section includes the following sample work flows:

Non-MDT Flow

The Non-MDT flow comprises the creation and completion life cycle of an M1 activity - break, non-productive task (NPT), and real period of unavailability (POU), along with the crew shift.

The following table lists the tasks that are created and their respective components in Oracle Utilities Mobile Workforce Management:

Task	MWM Component
Activity	M1-Activity
Break	M1-Break
Non-productive Task	M1-NonProductiveTask
Real Period of Availability	M1-RealPOU

Activity, real POU, break, and non-productive tasks are scheduled to a planned crew shift that has the matching location, required capability, and scheduling time window. The dispatcher logs in the planned crew shift and starts the shift. Once the crew shift starts, activity, non-productive task, real POU, and break are dispatched to the crew shift. The dispatcher starts, completes the activity, real POU, break, and non -productive task.

The M1-Assignment component is used to assign a transition (to enroute --> start --> completing an activity). Similarly, M1-BreakTask, M1-NonProductiveTask, and M1-POUTask components are used to start, complete the break, non-productive tasks, and POU respectively. The dispatcher completes the crew shift after completing the dispatched tasks.

MDT Flow

The MDT flow includes the creation and completion of an activity, break, non-productive task (NPT), and real period of unavailability (POU) using connected MCP.

Task	MWM Component
Activity	M1-Activity
Break	M1-Break
Non-productive Task	M1-NonProductiveTask
Real Period of Availability	M1-RealPOU

Activity, real POU, break, and non-productive tasks are scheduled to a planned crew shift that has the matching location, required capability, and scheduling time window.

The MDT flow is as follows:

1. The dispatcher launches the connected MCP.
2. The crew logs in with the valid credentials and starts the crew shift.
3. The activity, break task, and non -productive tasks are dispatched to the crew shift.
4. Use the following components applicable for each task:

Task	MWM Component
Start the crew shift	ConnectedMCP_StartCrewShift
Verify if the tasks are dispatched	ConnectedMCP_DispatchedTask

Task	MWM Component
Enroute status	ConnectedMCP_EnrouteActivity
Start activity	ConnectedMCP_StartActivity
Complete activity	ConnectedMCP_CompleteActivity

5. The crew shift is completed after completing all the dispatched tasks.

M2 Non-MDT Flow

The M2-Non-MDT flow includes the creation and completion of M-Activity (Install Meter Activity, for example).

Create an Install Meter Activity using the **M2-InstallMeterActivity** component.

The M2-Activity is scheduled to a planned crew shift that has the matching location, required capability, and the scheduling time window.

The M2 Non-MDT flow is as follows:

1. The dispatcher logs in the planned crew shift.
2. The dispatcher starts and completes the M2-Activity.

   The M2-InstallMeterAssignment component is used for M2-Activity assignment transition.
3. The dispatcher completes the crew shift after completing the M2-Activity (Install Meter Activity).

MDT Flow Using NextGen MCP

The MDT flow includes the creation and completion of an activity, break, non-productive task (NPT), and real period of unavailability (POU) using NextGen MCP.

Task	MWM Component
Activity	M1-Activity
Break	M1-Break
Non-productive Task	M1-NonProductiveTask
Real Period of Availability	M1-RealPOU

Activity, real POU, break, and non-productive tasks are scheduled to a planned crew shift that has the matching location, required capability, and scheduling time window.

The MDT flow is as follows:

1. The dispatcher launches the NextGen MCP (Google Chrome Browser).
2. The crew logs in with valid credentials and starts the crew shift.
3. The activity, break task, and non -productive tasks are dispatched to the crew shift.
4. Use the following components applicable for each task:

Task	MWM Component
Start the crew shift	NextGenMCP_StartCrewShift
Verify if the tasks are dispatched	NextGenMCP_DispatchedTask

Task	MWM Component
Enroute status	NextGenMCP_EnrouteActivity
Start activity	NextGenMCP_StartActivity

5. The crew shift is complete after completing all the dispatched tasks.

> **Note**: The MDT Flow using NextGen MCP can be run/played back in OpenScript 12.5 version. OpenScript v12.5 currently does not support recording on Google Chrome at its current level. But, this is an expected feature in OpenScript v12.6 (which would soon be available).

MDT Flow Using HybridMCP

> **Note**: While using secured (https) connection to access the HybridMCP Web application for the first time, "Your connection is not private" message is displayed. Accept and proceed further to access the application.

The HybridMCP MDT flow includes the creation and completion of an activity, break, non-productive task (NPT), and real period of unavailability (POU) using HybridMCP.

Task	MWM Component
Activity	M1-Activity
Break	M1-Break
Non-productive Task	M1-NonProductiveTask
Real Period of Availability	M1-RealPOU

Activity, real POU, break, and non-productive tasks are scheduled to a planned crew shift that has the matching location, required capability, and scheduling time window.

The MDT flow is as follows:

1. The dispatcher launches the HybridMCP (Google Chrome browser).
2. The crew logs in with valid credentials and starts the crew shift.
3. The activity, break task, and non -productive tasks are dispatched to the crew shift.
4. Use the following components applicable for each task:

Task	MWM Component
Start the crew shift	HybridMCP_StartShift
Verify if the tasks are dispatched	HybridMCP_TaskDispatched
Enroute status	HybridMCP_EnrouteActivity
Complete activity	HybridMCP_CompleteActivity
Complete the crew shift	HybridMCP_CompleteShift

5. The crew shift is complete after completing all the dispatched tasks.

> **Note**: The MDT Flow using HybridMCP can be run/played back in OpenScript 12.5 version. OpenScript v12.5 currently does not support recording on Google Chrome at its current level. But, this is an expected feature in OpenScript v12.6 (which would soon be available).

MWM_Cloud_Environment_Sanity

Note: This Cloud Environment Sanity flow is not certified and is only for demo purposes. Also, to use this flow, the Framework patch # 24941499 should be applied on Framework v4.3.0.1 (Oracle Utilities Mobile Workforce Management v2.3.0 uses this).

The MWM_Cloud_Environment_Sanity flow sets up the necessary gold data, its pre-requisites, and post gold data configuration, for any Oracle Utilities Mobile Workforce Management Cloud Initial Install environment. It also executes one sample Non-MDT Sanity test flow on that environment to ensure the environment is ready for use.

The flow includes the scenarios mentioned below:

Scenario	Task
Pre-requisites	Inserts the pre-required data, such as Time zone, Country, and other mandatory installation options into the Cloud application, before inserting the gold data.
Gold Data	Inserts all the required gold data setup into the application.
Post Gold Data Configuration	Inserts the post-gold data setup, such as Scheduler configuration and Scheduler setup.
Sanity	Performs the Non-MDT sanity to ensure the Cloud application is ready to use.

Note: This Oracle Utilities Mobile Workforce Management Cloud Environment Sanity Flow can be run/played back in OpenScript 12.5 version. OpenScript v12.5 currently does not support recording on Google Chrome at its current level. But, this is an expected feature in OpenScript v12.6 (which would soon be available).

M2 Android-MDT Sanity Flow

The M2 Android-MDT Sanity flow runs only on the Oracle Utilities Mobile Workforce Management Android apk file that is in debug build mode. Place your Oracle Utilities Mobile Workforce Management Android apk file in any of your local directory.

Note: To run this flow, download the required five JAR files. The table below lists the versions and links to download.

File Name with JAR Version	Download Link
selenium-server-standalone-2.53.0.jar	http://selenium-release.storage.googleapis.com/2.53/selenium-server-standalone-2.53.0.jar
selenium-java-2.53.0.jar	http://selenium-release.storage.googleapis.com/2.53/selenium-java-2.53.0.zip
commons-validator-1.4.0.jar	http://central.maven.org/maven2/commons-validator/commons-validator/1.4.0/commons-validator-1.4.0.jar
commons-lang3-3.4.jar	http://central.maven.org/maven2/org/apache/commons/commons-lang3/3.4/commons-lang3-3.4.jar

File Name with JAR Version	Download Link
java-client-4.1.1.jar	http://central.maven.org/maven2/io/appium/java-client/4.1.1/java-client-4.1.1.jar

To execute the Android MDT Sanity flow, follow these steps:

1. Download the JAR files and place them in the local directory.

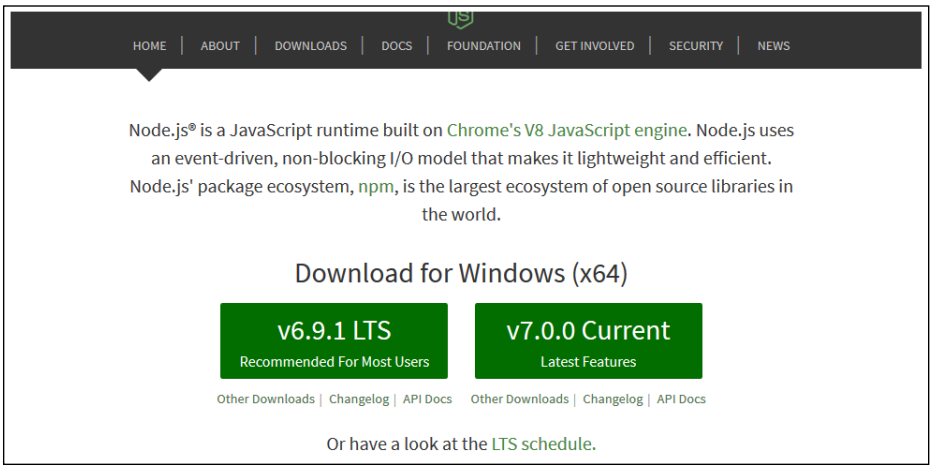
   <Open Script repository location>\outsp-function-libs\MWM\OUMWMHYBRIDLIB

2. Install Node.js server which is a freeware and does not require a license.

   a. Navigate to the below URL.

      https://nodejs.org/en/

   b. Download the latest version (v7.0.0 is the latest as on date).



   c. Launch the Setup Wizard and complete the installation.

   d. Verify that Node.js is installed properly. Run the following command at the Command prompt.

      ```
      node -v
      ```

      This command shows the Node.js version currently installed.

   e. Verify that Node Package Manager (NPM) is properly configured. Run the following command at the Command prompt.

      ```
      npm -v
      ```

      This command shows the NPM version currently configured.

3. Set the environment variable for proxy.

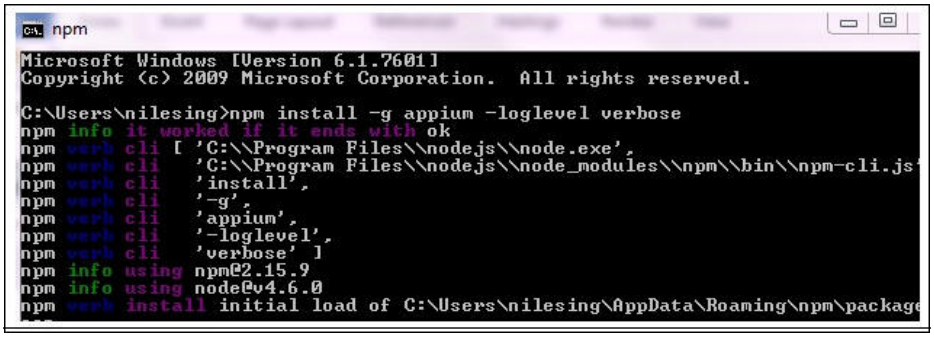
   This step is required if you are connected behind the corporate proxy. In order to set the proxy variable for Node.js, "proxy-IP" and "PORT" values are needed. Please contact your respective IT-team for this information.

Run the following commands at the Command prompt:

```
set HTTP_PROXY=http://<<server>>:<<port>>
set HTTPS_PROXY=http:// <<server>>:<<port>>
```

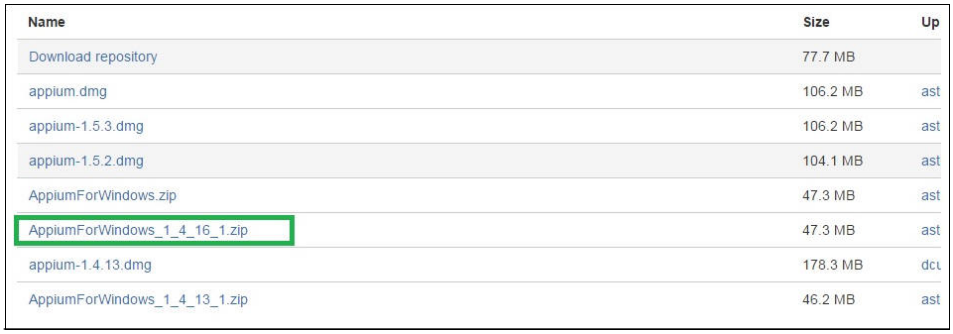
4. Install the latest Appium server. Run the command below at the Command prompt:

   ```
   npm install -g appium -loglevel verbose
   ```



   Alternatively, you can install Appium GUI from the following URL. Download the latest version and extract the ZIP file in local directory.

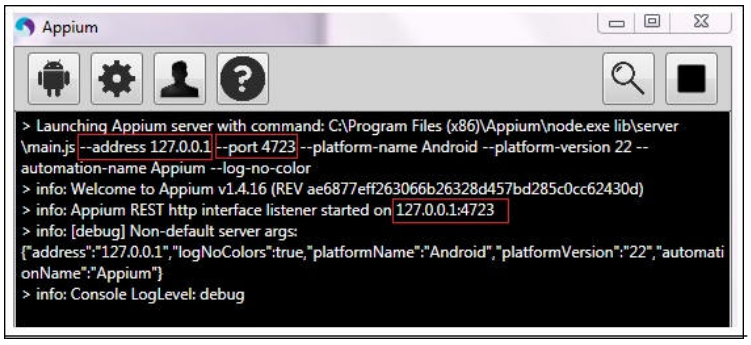
   https://bitbucket.org/appium/appium.app/downloads/



5. Start the Appium server.

   - **NPM command line**: Run the `appium` at the Command prompt.
   - **Appium GUI**: Launch `appium.exe` file and click **Start**.

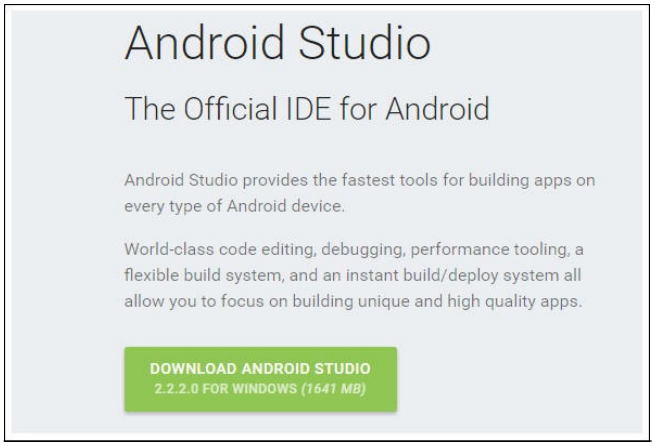
   The appium server starts running at the default port.W



For more information on Appium server, see https://www.npmjs.com/package/appium.

6. Download and install the latest Android Software Development Kit (SDK) from https://developer.android.com/studio/index.html.

   Since the latest Android SDK does not come as a standalone installer, download and install Android Studio.

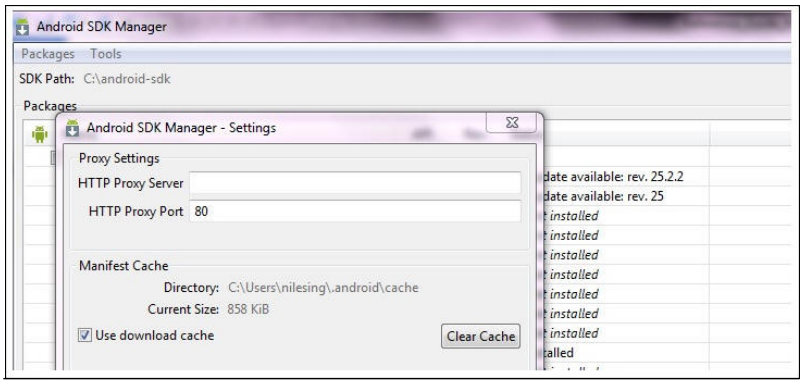


   After download, do a quick installation. After successful installation, verify the ANDROID_HOME environment variable by running the command at the Command prompt. The output will be the location path where the android SDK resides.

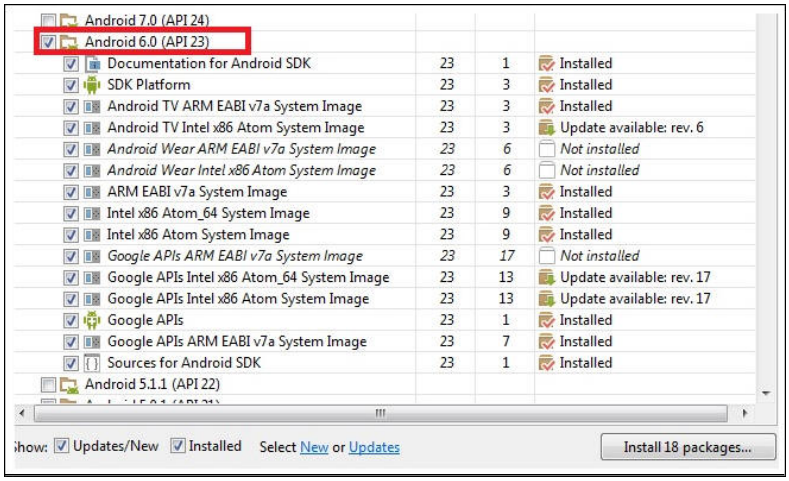
```
echo %ANDROID_HOME%
```

7. Ensure that the latest Android API (5.0 or above) is installed. Use ANDROID_HOME/SDK Manager.exe to verify the same.

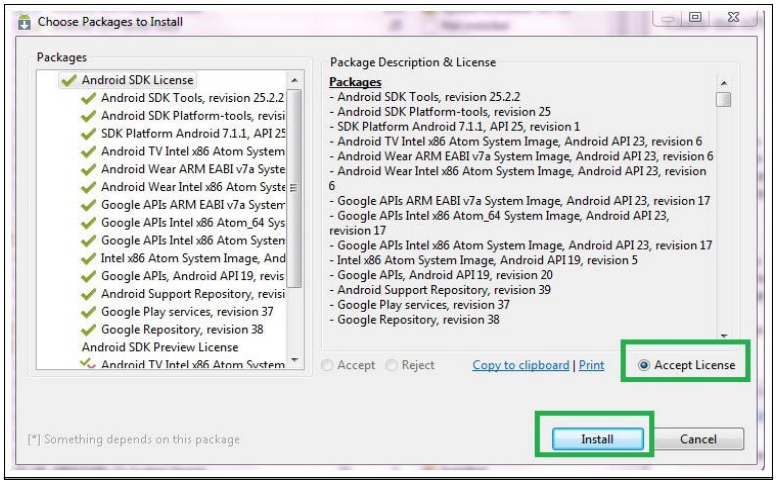
   SDK Manager.exe automatic fetches the latest Android images and related API dependencies. If it does not, it may be due to the proxy firewall. Configure the proxy settings in **Tools** > **Options** and provide the proxy IP and port.



   Select the API version to be installed. Finally, click **Install (N) Packages** to proceed with installation.



Accept the **License Agreement** and click **Install**.



At this point, all the necessary packages are successfully installed. Proceed with executing the flow.

8. To execute the Andriod-MDT Sanity Flow on an Android Emulator, add the virtual device in Android SDK as follows:

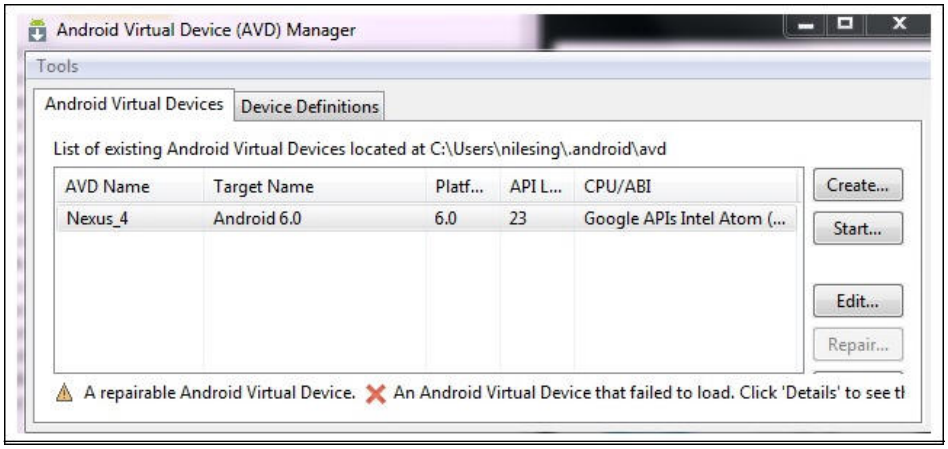
   a. Run ANDROID_HOME/AVD Manager.exe. The **Android Virtual Device (AVD) Manager** opens.

   b. On the **Android Virtual Devices** tab, click **Create** and enter appropriate details. The list below shows sample details.

      - AVD Name: Nexus_4
      - Device: Nexus 4 (4.7", 768 x 1280: xhdpi)
      - Target: Android (5.0 or above)
      - Keyboard: Enable
      - Skin: No Skin
      - Front Camera: None
      - Back Camera: None

- Ram: 1200, VM Heap: 64
- Internal Storage: 200 MB
- SD card Size: 1000 MB
- Use Host GPU: Enable

c. Click **OK**. A new AVD profile with name "Nexus_4" is created.



d. Select **Nexus_4** and click **Start**. The emulator runs in the GUI mode.

e. At the Command prompt, run the `adb devices` command. You can see the emulator type and UID.

```
C:\Users\nilesing>adb devices
List of devices attached
emulator-5554   device
```

f. Enter the emulator name and UID in the **configuration.properties** file. The file is in the OATS_REPOSITORY/etc folder.

```
29 gStrAndroidDeviceName=emulator
30 gStrAndroidDeviceUID=emulator-5554
```

See the Setting Up Sample Flows section to enter details in the configuration.properties file.

For more information on managing your Android virtual devices, go to https://developer.android.com/studio/run/managing-avds.html.

9. Follow the steps listed out in the Running the Android-MDT Flow section to complete executing the flow.

To execute the Android-MDT flow on a real Android device that is connected, follow these steps:

1. Follow the instructions below to allow the debugging mode:
   a. On the Android device, navigate to **Settings** > **About Device**.
   b. Tap the **Build Number** button about 7 times. Developer Mode should now be unlocked.
   c. Navigate to **Settings** > **Developer Options** > **USB Debugging**.

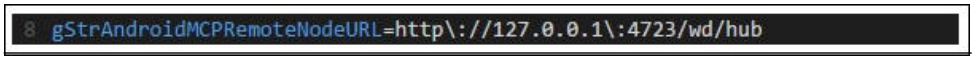
For more information on setting the debug mode, go to http://www.howtogeek.com/129728/how-to-access-the-developer-options-menu-and-enable-usb-debugging-on-android-4.2/.

2. Run the `adb devices` command at the Command prompt. You can see the real device UID.

3. Enter the device name and UID in the **configuration.properties** file. The file is in the OATS_REPOSITORY/etc folder.

   See the Setting Up Sample Flows section to enter details in the configuration.properties file.

4. Enter the Appium node URL connection string and Appium build path in the **configuration.properties** file.



5. Set the IP and Port details based on the running Appium service. This can be found from the Appium console logs.
6. Follow the steps listed out in the Running the Android-MDT Flow section to complete executing the flow.

Running the Android-MDT Flow

The Android-MDT flow includes the creation and completion of an activity, break, non-productive task (NPT), and real period of unavailability (POU) using Web Driver.

Task	MWM Component
Activity	M1-Activity
Break	M1-Break
Non-productive Task	M1-NonProductiveTask
Real Period of Availability	M1-RealPOU

Activity, real POU, break, and non-productive tasks are scheduled to a planned crew shift that has the matching location, required capability, and scheduling time window.

The Android-MDT flow is as follows:

1. The dispatcher launches the Android Emulator or a real Android device.
2. The crew logs in with valid credentials and starts the crew shift.
3. The activity, break task, and non-productive tasks are dispatched to the crew shift.
4. Use the following components applicable for each task.

Task	MWM Component
Login into Crew Shift on Android device	Webdriver_AndroidLogin
Start the crew shift	Webdriver_startShift

Task	MWM Component
Verify if the tasks are dispatched	Webdriver_waitForTaskDispatch
Enroute status	Webdriver_startM1Activity
Complete activity	Webdriver_completeM1Activity
Complete the crew shift	Webdriver_endShift

5. The crew shift is complete after completing all the dispatched tasks.

M2 iOS-MDT Sanity Flow

The M2 iOS-MDT Sanity flow can only be run on an iOS system. As of now, this flow can be run only on an iOS emulator and cannot be run on iOS real devices. It runs only on the MWM iOS ipa file which is in debug build mode. At the time of execution, place the iOS ipa file in a local directory.

Note: To run this flow, download the required five JAR files. The table below lists the versions and links to download.

File Name with JAR Version	Download Link
selenium-server-standalone-2.53.0.jar	http://selenium-release.storage.googleapis.com/2.53/selenium-server-standalone-2.53.0.jar
selenium-java-2.53.0.jar	http://selenium-release.storage.googleapis.com/2.53/selenium-java-2.53.0.zip
commons-validator-1.4.0.jar	http://central.maven.org/maven2/commons-validator/commons-validator/1.4.0/commons-validator-1.4.0.jar
commons-lang3-3.4.jar	http://central.maven.org/maven2/org/apache/commons/commons-lang3/3.4/commons-lang3-3.4.jar
java-client-4.1.1.jar	http://central.maven.org/maven2/io/appium/java-client/4.1.1/java-client-4.1.1.jar

To execute the iOS-MDT Sanity flow, follow these steps:

1. Download the JAR files and place them in the local directory.

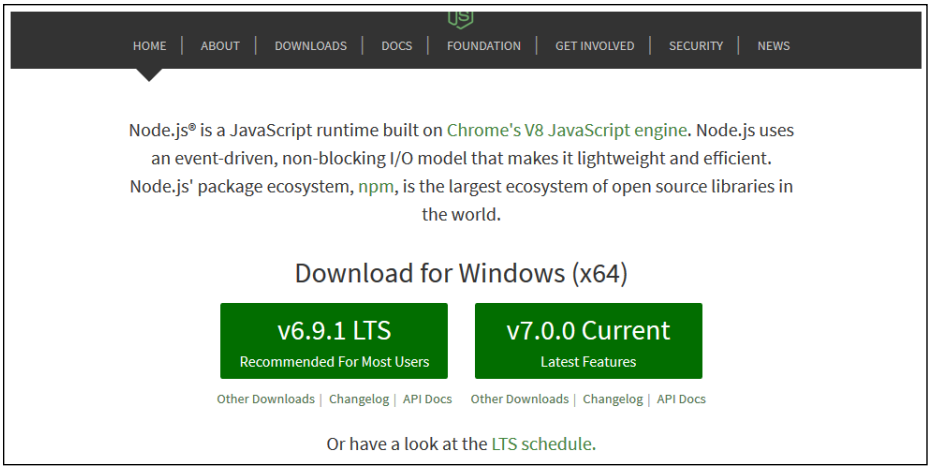
   <Open Script repository location>\outsp-function-libs\MWM\OUMWMHYBRIDLIB

2. Install Node.js server which is a freeware and does not require a license.

   a. Navigate to the below URL.

      https://nodejs.org/en/

b. Download the latest version (v7.0.0 is the latest as on date).



c. Launch the Setup Wizard and complete the installation.

d. Verify that Node.js is installed properly. Run the following command at the Command prompt.

```
node -v
```

This command shows the Node.js version currently installed.

e. Verify that Node Package Manager (NPM) is properly configured. Run the following command at the Command prompt.

```
npm -v
```

This command shows the NPM version currently configured.

3. Set the environment variable for proxy.

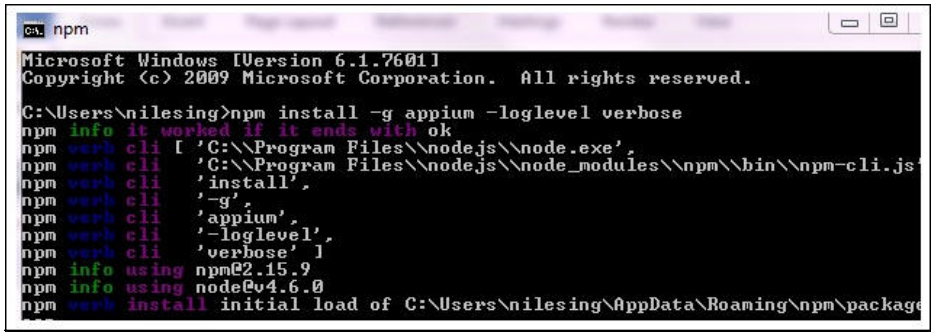
This step is required if you are connected behind the corporate proxy. In order to set the proxy variable for Node.js, "proxy-IP" and "PORT" values are needed. Please contact your respective IT-team for this information.

Run the following commands at the Command prompt:

```
set HTTP_PROXY=http://<<server>>:<<port>>
set HTTPS_PROXY=http:// <<server>>:<<port>>
```

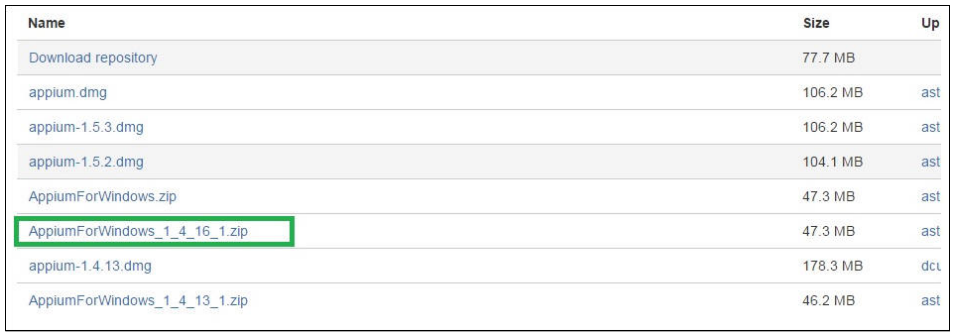
4. Install the latest Appium server. Run the command below at the Command prompt:

   ```
   npm install -g appium -loglevel verbose
   ```



   Alternatively, you can install Appium GUI from the following URL. Download the latest version and extract the ZIP file in local directory.

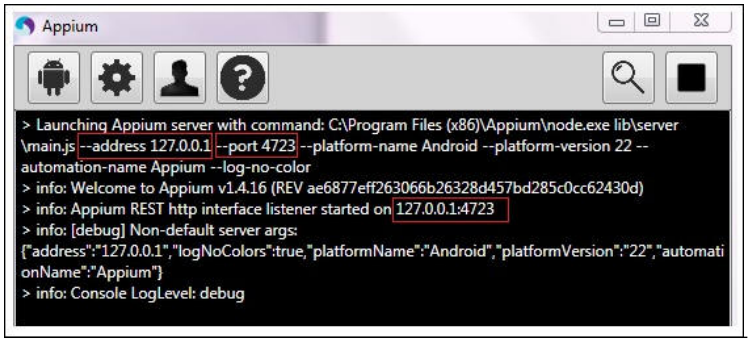
   https://bitbucket.org/appium/appium.app/downloads/



5. Start the Appium server.

   - NPM command line: Run the `appium` at the Command Prompt.
   - Appium GUI: Launch `appium.exe` file and click **Start**.

   The appium server starts running at the default port.W



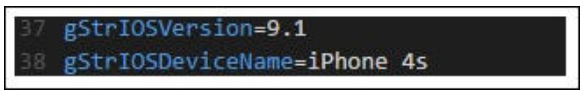
   For more information on Appium server, see https://www.npmjs.com/package/appium.

6. Install X-code on your iOS system if your machine does not have it. X-Code is available with your iOS installation CD.

7. Get the UID, name, and version of the simulator. Enter the following command using Terminal.

   ```
   instruments -s devices
   ```

8. Enter the UID, name, and version in the **configuration.properties** file.



9. Enter the Appium node URL connection string and Appium build path in the **configuration.properties** file.

10. Set the IP and Port details based on the running Appium service. This can be found from the Appium console logs.

The iOS-MDT flow includes the creation and completion of an activity, break, non- productive task (NPT), and real period of unavailability (POU) using Web Driver.

Task	MWM Component
Activity	M1-Activity
Break	M1-Break
Non-productive Task	M1-NonProductiveTask
Real Period of Availability	M1-RealPOU

Activity, real POU, break, and non-productive tasks are scheduled to a planned crew shift that has the matching location, required capability, and scheduling time window.

The iOS-MDT flow is as follows:

1. The dispatcher launches the iOS Emulator.
2. The crew logs in with valid credentials and starts the crew shift.
3. The activity, break task, and non-productive tasks are dispatched to the crew shift.
4. Use the following components applicable for each task.

Task	MWM Component
Login into Crew Shift on Android device	Webdriver_AndroidLogin
Start the crew shift	Webdriver_startShift
Verify if the tasks are dispatched	Webdriver_waitForTaskDispatch
Enroute status	Webdriver_startM1Activity
Complete activity	Webdriver_completeM1Activity
Complete the crew shift	Webdriver_endShift

5. The crew shift is complete after completing all the dispatched tasks.

Executing Sample Flows

This section describes the procedure to setup sample flows and execute them.

Pre-requisites

To execute the sample flow, ensure the following pre-requisites are met:

- Oracle Utilities Mobile Workforce Management v2.3.0/Oracle Real-Time Scheduler v2.3.0 is up and running with the demo data pack.
- OpenScript is installed in the local machine. See the *Oracle Functional Testing Advanced Pack for Oracle Utilities Installation and Administration Guide* for the version details.
- Oracle Functional Testing Advanced Pack for Oracle Utilities is installed and repository/ directory is setup in the local machine appropriately. See the *Oracle Functional Testing Advanced Pack for Oracle Utilities Installation and Administration Guide* for more details.
- **Important**: Browser Playback settings of OpenScript must be set explicitly to either Mozilla Firefox or Chrome Browser for MDT flow and only "Chrome Browser" for NextGen MCP and HybridMCP flows.

Setting Up Sample Flows

To setup a sample flow, follow these steps:

1. Login to Oracle Utilities Mobile Workforce Management/Oracle Real-Time Scheduler.
2. Import the Inbound Web services into the Oracle Utilities Mobile Workforce Management/ Oracle Real-Time Scheduler application where the scenarios need to be executed.

   See the **Inbound Web Services** section in *Oracle Functional Testing Advanced Pack for Oracle Utilities User's Guide* for steps to import the Inbound Web services.
3. Navigate to **Admin > B > Bundle Import > Add**.
4. Enter the **External Reference**, **Detailed Description**, and **Bundle Details** from the IWS Bundle Export Dump.
5. Click **Save**, and then click **Apply bundle**.
6. Launch OpenScript in the local machine and perform the following steps:
   a. Navigate to **View > OpenScript Preferences**.
   b. In the left tree, select **OpenScript**. In the sub tree, select **Playback**, and then select **Error Recovery**.
   c. Click **SetAll** and select **Report Error and Continue**.
   d. Click **Apply**, and then click **Close**.
7. Configure the **configuration.properties** file as follows:
   a. Provide the application and NextGen MCP URLs for the parameter:

   ```
   gStrNewMCPURL=http\://<%serverName%>\:<%portNumber%>/ouaf/
   mobility/www/index.html
   ```

   b. Provide the additional path required for Inbound Web service URL:

   ```
   gStrApplicationXAIServerPath=/<%webservices/
   gStrApplicationURL%>/<%AppendThisToAbove gStrApplicationURL%>/
   ```

c. Provide an environment name for display in the results email:

```
gStrEnvironmentName= <%testEnvironmentName%>
```

d. Provide the application login user ID:

```
gStrApplicationUserName= <%UserNAme%>
```

e. Provide the application login password:

```
gStrApplicationUserPassword= <%password%>
```

f. Provide the SMTP email server and e-mail ID:

```
gStrSMTP_HOST_NAME=<%SMTP ServerName%>
gStrSMTP_PORT=<%PortNumber%>
gStrTO_EMAIL_RECIPIENTS=<%e-mail Id%>
```

g. Provide the application database details as below:

```
gStrApplicationDBConnectionString =<%jdbc Connectionstring%>
eg: jdbc\:oracle\:thin\:@<%DBserverName%>\:<%port%>\:<%DBSID%>
gStrApplicationDBUsername=<%DBUserID%>
gStrApplicationDBPassword=<%DBPassword%>
```

h. Provide the full directory path of Oracle Application Testing Suite repository directories in the local machine.

```
gStrOutputFilePath=<%LogFilePath%>
Example: C:\\OATSOUMWM_DEMO\\OUTSP\\Logs\\
gStrXSDFiles=<%XSD Folder path%>
Example: C:\\OATSOUMWM _DEMO\\OUTSP\\Logs\\
```

i. Provide the Mobile Android device details:

Note: Below is an example for mentioning the version, name, and UID. Run the '`adb devices`' command for identifying the respective Android device details.

```
gStrAndroidDeviceName=emulator
gStrAndroidDeviceUID=emulator-5554
gStrAndroidDeviceName=Nexus_4
```

j. Provide the Mobile iOS device details

Note: Below is an example for mentioning the version, name, and UID. Run the '`instruments -s devices`' command for identifying the respective iOS device details.

```
gStrIOSVersion=9.1
gStrIOSDeviceName=iPhone 4s
gStrIOSDeviceUID=33ad17f071d36d191d0a6e6b102b8ed3e7783c58
```

k. Provide the Appium node URL details.

```
gStrAndroidMCPRemoteNodeURL=http\://<<Appium
Server>>\:<<port>>/wd/hub
gStrIosMCPRemoteNodeURL= http\://<<Appium Server>>\:<<port>>/
wd/hub
```

l. Provide the Android/iOS build paths.

```
gStrAndroidBuildPath=<<your local directory
>>\\Oracle_MWM_Debug.apk
```

```
gStrIosBuildPath=<<your local directory
>>\\Oracle_MWM_Debug.ipa
```

8. Create two folders (MWM and Core) in the outsp-function-libs folder in the Oracle Application Testing Suite repository directory. Copy the function libraries to the respective folders.
   - MWM
     - OUMWMCONNECTEDMCPLIB
     - OUMWMLIB
     - OUMWMMCPLIB
     - OUMWMHYBRIDLIB
   - Core
     - OUTSPCORELIB
     - WSCOMMONLIB
     - WSVALIDATELIB
9. Copy all the .jar files provided in the installer into the **genericJars** folder in the Oracle Application Testing Suite repository directory.

Appendix A

Inbound Web Services

The Oracle Utilities Mobile Workforce Management components are developed using Web services method, and these components require Inbound Web Services to be defined in the application.

For instructions to create, import, or search an Inbound Web Service, see the **Setting Up Inbound Web Services** appendix in *Oracle Functional Testing Advanced Pack for Oracle Utilities User's Guide*.

List of Inbound Web Services

The list of Inbound Web Services provided to use with the delivered components and flows is as follows:

- ATM1Activity
- ATM1Alert
- ATM1AllocateActivityToAShift
- ATM1AppointmentBookingRequest
- ATM1Assignment
- ATM1AdditionalUserInfo
- ATM1BreakTask
- ATM1CommonShiftWeeklyTemplate
- ATM1Company
- ATM1ComplexActivity
- ATM1Contractor
- ATM1ContractorEligibility
- ATM1Crew
- ATM1CrewShift
- ATM1CrewShiftCloseOpenAllocate
- ATM1CrewShiftTemplate
- ATM1DailySpeedTemplate
- ATM1DeferActivity
- ATM1DepotRelatedActivity

- ATM1DepotRelatedShift
- ATM1DepotSinglePersonCrew
- ATM1DepotTask
- ATM1Dispatcher
- ATM1DispatcherShift
- ATM1FixToCrew
- ATM1ForceLogOffCrewShift
- ATM1GetShiftCapabilities
- ATM1HazardArea
- ATM1MailingList
- ATM1MainMail
- ATM1MobileWorker
- ATM1NonPrdTask
- ATM1OverrideTimeWindow
- ATM1POUTask
- ATM1RealPOU
- ATM1ResTimesheet
- ATM1ResourceLeave
- ATM1SchedulerRegistryRead
- ATM1ShiftWeeklyTemplate
- ATM1SimpleProcedure
- ATM1SinglePersonCrew
- ATM1SpeedProfileTemplate
- ATM1SubscriptionWeeklyTemplate
- ATM1TaskScheduleDetails
- ATM1TemplateMail
- ATM1TemplatePOU
- ATM1UnassignActivity
- ATM1Vehicle
- ATM1VehicleLeaveArea
- ATM2InstallMeterActivity
- ATM2InstallMeterAssignment
- ATM2ConnectSPActivity
- ATM2ConnectSPAssignment
- ATM2CutNonPayItemActivity
- ATM2CutNonPayItemAssignment
- ATM2DisconnectMeterActivity
- ATM2DisconnectMeterAssignment

- ATM2ExchangeItemActivity
- ATM2ExchangeItemAssignment
- ATM2InstallMeterActivity
- ATM2InstallMeterAssignment
- ATM2MeterReadActivity
- ATM2MeterReadAssignment
- ATM2RemoveMeterActivity
- ATM2RemoveMeterAssignment
- ATM2BasicItemActivity
- ATM2BasicItemAssignment
- ATM2RemoveItemActivity
- ATM2RemoveItemAssignment
- ATM1ActivityType
- ATM1AlertType
- ATM1BreakTaskType
- ATM1Capacity
- ATM1CapacityTemplate
- ATM1CapacityType
- ATM1CapacityWeeklyTemplate
- ATM1ComplexActivityType
- ATM1ContractorType
- ATM1CrewHierarchy
- ATM1CrewShiftType
- ATM1CrewType
- ATM1Deployment
- ATM1DeploymentPart
- ATM1DeploymentType
- AT_M1DeploymentTypeRestfulSvc
- ATM1Depot
- ATM1DepotProfile
- ATM1DepotTaskType
- ATM1DispatchArea
- ATM1DispatcherType
- ATM1DispatcherShiftLite
- ATM1Equipment
- ATM1GeographicArea
- ATM1GlobalConfigurations
- ATM1KPI

- ATM1Location
- ATM1MobileDeviceTerminal
- ATM1MobileDeviceTerminalType
- ATM1MobileWorkerType
- ATM1NonPrdTaskType
- ATM1POUTaskType
- ATM1POUType
- ATM1PriorityProfile
- ATM1ProcedureType
- ATM1RemarkType
- ATM1ResTimesheetType
- AT_M1ResourceLite
- ATM1Scheduler
- ATM1SchedulerArea
- ATM1SchedulerConfiguration
- ATM1ServiceArea
- ATM1ServiceAreaHierarchy
- ATM1ServiceClass
- ATM1ShiftCostProfile
- ATM1Skill
- ATM1StatusReason
- ATM1VehicleType
- ATM1WorkCalendar
- ATM1WorkProfile
- ATM2MaintainUtilityActByHost
- ATM1RetrievePOUTaskIdsOfPOU
- ATM1RetrieveMWIdFromUserId
- ATM1RetrieveDeploymntId
- ATM1RetrieveAssgnmtsOfTask
- ATM1FetchActiveResourceIDByType
- ATM1CheckIfActScheduled