Oracle Real-Time Scheduler Hybrid Mobile Application Implementation and Development Guide

Release 2.3.0.0

E72834-01

March 2016

ORACLE®

Oracle Real-Time Scheduler Hybrid Mobile Application Implementation and Development Guide, Release 2.3.0.0

E72834-01

Chapter 5
Customization and Extension Methodology 5-1

Preface

Audience

The target audience of this guide is implementers and system administrators responsible for implementation and deployment of mobile applications.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit: http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info

or

http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

Installation, Configuration, and Release Notes

- *Oracle Real-Time Scheduler Release Notes*
- *Oracle Real-Time Scheduler Quick Install Guide*
- *Oracle Real-Time Scheduler Server Application Installation Guide*
- *Oracle Real-Time Scheduler DBA Guide*
- *Oracle Real-Time Scheduler Hybrid Mobile Application Installation and Deployment Guide*

User Guides

- *Oracle Real-Time Scheduler Server Application User's Guide*
- *Oracle Real-Time Scheduler Mobile Application User's Guide (Java-based)*
- *Oracle Real-Time Scheduler Hybrid Mobile Application User's Guide*
- *Oracle Real-Time Scheduler Hybrid Mobile Contractor Application User's Guide*

Implementation and Development

- *Oracle Real-Time Scheduler Hybrid Mobile Application Implementation and Development Guide*

Map Editor Installation and User Guides

- *Oracle Real-Time Scheduler Map Editor User's Guide*

- *Oracle Real-Time Scheduler Map Editor Installation Guide*

Supplemental Documents

- *Oracle Real-Time Scheduler Server Administration Guide*

- *Oracle Real-Time Scheduler Security Guide*

Chapter 1

Overview

This guide provides development and configuration details for the Oracle Real-Time Scheduler Mobile Application including Oracle Utilities Mobile Library, APIs, development environment setup, customization and extension methodology.

This section provides a general overview and information about the mobile application components and architecture.

Architecture

Oracle Real-Time Scheduler simplifies and optimizes the scheduling, dispatching, and tracking of mobile service crews and field activities.
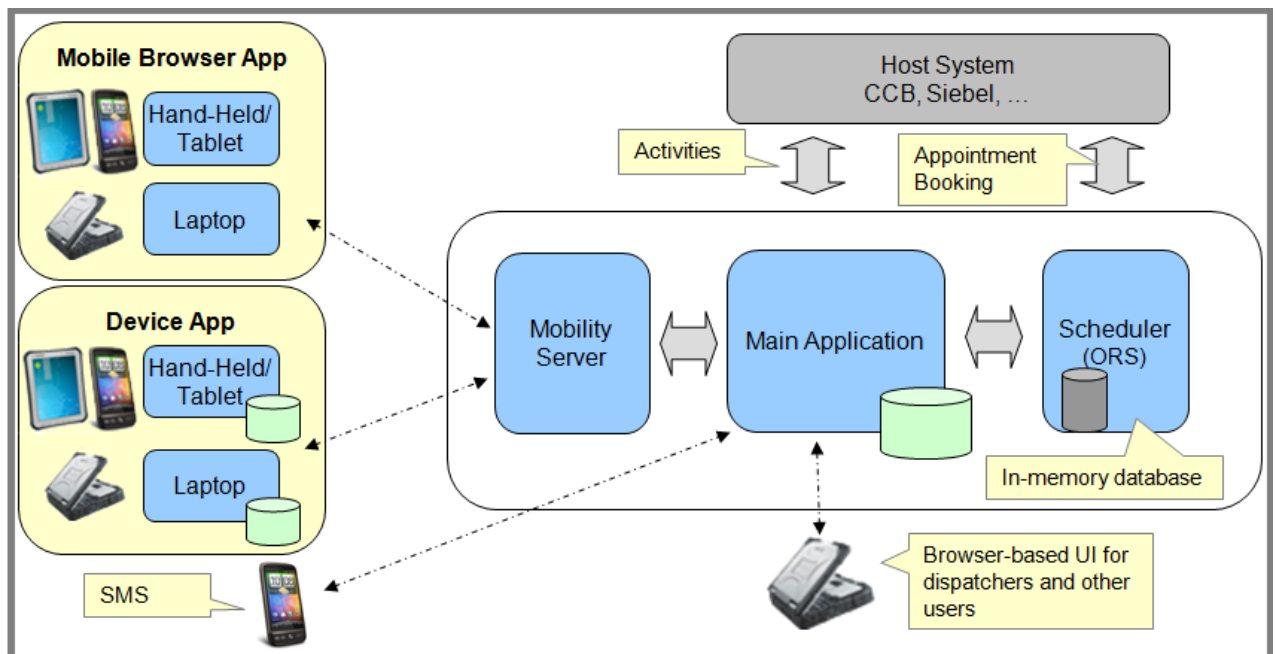
The mobile application consists of the Oracle Utilities Mobile library and application layers responsible for specific business functionality. It uses HTML5 and JavaScript to implement business logic, render the user interface and interact with mobile device services. Web services facilitate communication between the mobile application and the application server.

Oracle Utilities Mobile Library (OUML)

The Oracle Real-Time Scheduler Mobile Application is based on the Oracle Utilities Mobile Library (OUML) optimized to work with Oracle Utilities Application Framework (OUAF) based services, configurations and metadata. The Oracle Utilities Mobile Library provides a foundation layer and APIs for application development including offline storage, encryption, communication, logging, configuration, UI rendering/navigation, customization, deployment and so on. The Oracle Utilities Mobile Library makes use of third party libraries that are either bundled with the application or listed as pre-requisites.

Please reference Chapter 3: Oracle Utilities Mobile Library for more information on working with the Oracle Utilities Mobile Library.
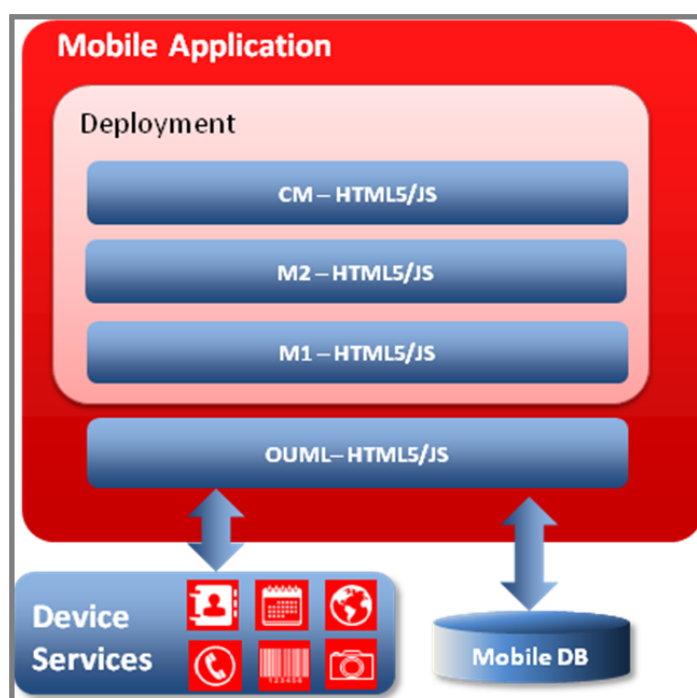
Deployment Models

The mobile application can be packaged and deployed in the format native to one of the supported runtime platforms. Alternately, it can be packaged as a web application and deployed to an application server to be accessed on the mobile device via a web browser.

Please note that certain device specific features are not available when the application is deployed as a web application and accessed via web browser.

The following table lists the features supported by application mode.

Feature	Compiled	Browser Based
GPS	✓	✓
Capture Picture	✓	X
Capture Signature	✓	X
Capture Sound	X	X
Barcode Scanning/Reading	✓	X
Download Attachments from MDT (All File Types)	✓	X

Feature	Compiled	Browser Based
Upload Attachment from MDT to Server (Only Captured Picture and Signature)	✓	X
Maps	✓	✓
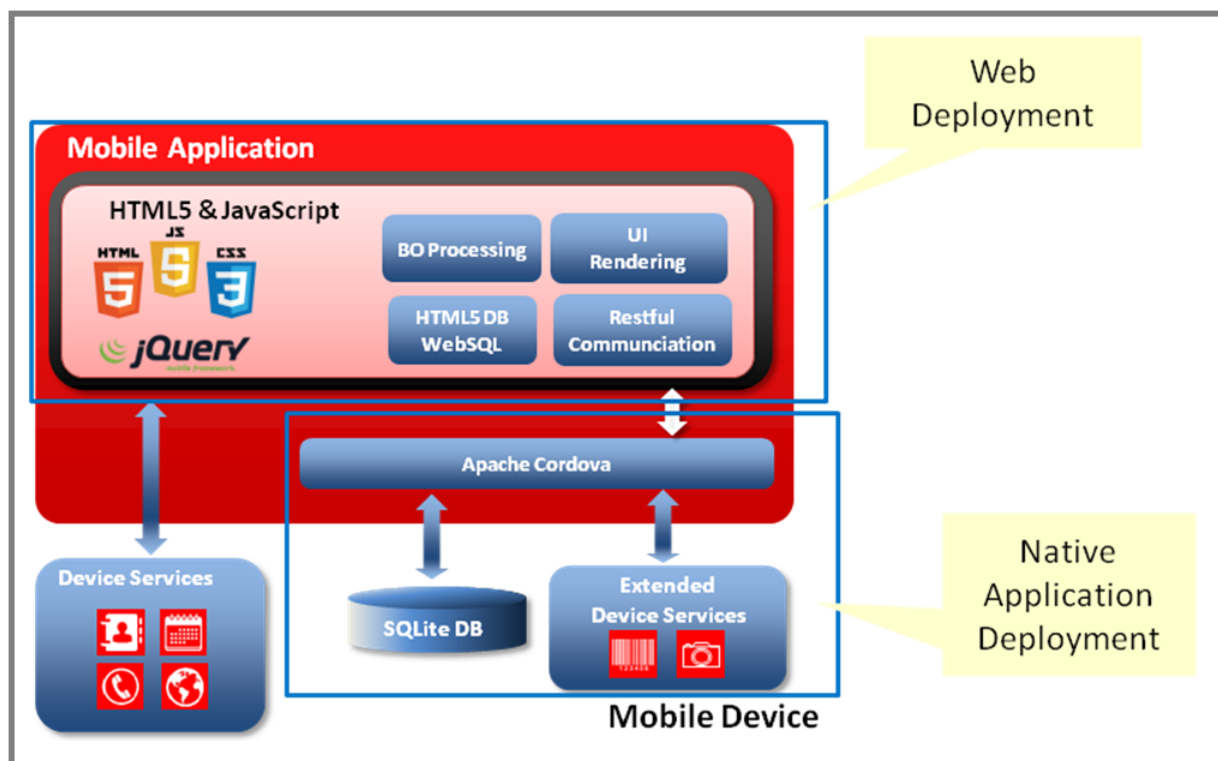
Inbound and Outbound Communication

Inbound and Outbound communication between ORS mobile and server applications is based on RESTFul services and JSON payload. In situations where device is offline at the time of making outbound HTTP request communication modules of ORS Mobile application ensure that delivery of the message when device is back online and communication with server is reestablished. To simplify debugging and implementation activities in a development environment, guaranteed delivery and asynchronous messaging can be turned on or off on the **Settings** page within the mobile application.

Please reference Mobile Application APIs for more details on communication between mobile and server applications.

Chapter 2

Development Environment Setup

This section provides information needed to setup the development environment for the mobile application. Implementers can use this environment to add new features and test their code locally or on devices using the steps provided in this section.

Prior to setting up the development environment, you must have completed general server side configuration. Please reference the *Oracle Real-Time Scheduler Server Administration Guide* for information.

Installing Prerequisite Software

Please reference the chapter on installing prerequisite software in the *Oracle Real-Time Scheduler Mobile Application Installation and Deployment Guide* for information.

Source Code

Required libraries and source code for development and customization in the local environment can be copied from the <PRODUCT_HOME> directory of the shared build environment that is created as part of the initial install. Please refer to the *Oracle Real-Time Scheduler Mobile Application Installation and Deployment Guide* for information on the initial install.

The <PRODUCT_HOME>/source/www folder in the shared build environment contains the source files which could be linked to a version control system to enable code contributions from multiple local development environments.

The www directory needs to be copied over or linked to the local Apache Cordova project. This project can be used to locally build native applications.

Apache Cordova Project

The same Apache Cordova project can be used to create native applications for different mobile operating systems.

Complete the following steps to create an Apache Cordova project:

1. Install Cordova.
   Please reference the installation instructions delivered with the Cordova product. The section on "The Command-Line Interface" includes steps to install the CLI tool information about Cordova project commands.

2. Create the Cordova project using the create command.
   Example: cordova create <Directory name> <Application namespace> <Application name>

   This creates the Cordova project folder on the machine.

The <PRODUCT_HOME>/source/www directory has to be copied or linked to the www directory for the Cordova project. Any changes made in the www directory for the Cordova project will be reflected in the generated native application.

3. Add different mobile platforms to the Cordova project. Examples:
```
Cordova platform add ios
Cordova platform add android
```

Plugin Installation

Cordova includes a set of "core plugins" which are used by the mobile application to access native device features such as the file system, camera, geolocation and so on. Aside from using the Cordova core plugins, implementers can also develop their own plugins or use other available plugins. These plugins are described in the Apache Cordova documentation.

Use the CLI tool to installing/uninstalling plugins. This is done by using the "plugin add" command:

Example:

```
cordova plugin add <path to plugin>
```

Note: Please reference the Oracle Real-Time Scheduler *Installation and Deployment Guide* in the "Plugin Configurations" section for the list of required plugins for the mobile application including the actual paths for the plugins with the release versions being used. The following section provides an overview of the plugin functions.

Device Plugins

This section provides a high level description of the device plugins used with the Oracle Real-Time Scheduler.

Device - The Cordova Device plugin defines a global device object, which describes the device's hardware and software.

Camera - The Cordova Camera plugin provides an API for taking pictures and for choosing images from the system's image library.

File - The Cordova File plugin implements a File API allowing read/write access to files residing on the device.

Geolocation - The Cordova Geolocation plugin provides information about the device's location, such as latitude and longitude. Common sources of location information include Global Positioning System (GPS) and location inferred from network signals such as IP address, RFID, WiFi and Bluetooth MAC addresses, and GSM/CDMA cell IDs.

InAppBrowser - The Cordova InAppBrowser plugin provides a web browser view that displays when calling window.open().

Network Information - The Cordova InAppBrowser plugin provides an implementation of an old version of the Network Information API. It provides information about the device's cellular and wifi connection, and whether the device has an internet connection.

Barcode Scanner - This is an external Barcode scanner plugin for Cordova which is optional and can be used with the application. The plugin provides implementation for scanning barcodes and provides the type and the barcode for a scanned item.

SQLite - This is an external SQLite plugin for Cordova which is optional and can be used with the application. The plugin provides implementation for using SQLite Database on the device. The plugin uses the same API as the HTML5 WEBSQL database.

Background Mode - This is an external background mode plugin for Cordova which is required only for iOS. This plugin prevents the application on iOS from going to sleep while in the background

Signature Capture - This is an external signature capture plugin for jQuery. This is a Javascript only plugin and does not require installation using the Cordova add plugin command. The plugin file needs to be included in the www/libs/jSignature folder. It provides a JavaScript widget for simplifying the creation of a signature capture field in a browser window, allowing a user to draw a signature using mouse, pen or finger.

Encryption

The encryption plugin is only used on devices running Android platform to:

- Store passwords encrypted on devices
- Store transaction data encrypted on devices (BO, Inbound, Outbound records)

For iOS, encryption is handled with native device encryption. If the Oracle Real-Time Scheduler Mobile Application is deployed as a web application and is being accessed on the device via web browser, the encryption module is not used, but rather, transaction data is stored in plain text format in offline database.

The encryption module is implemented entirely on the device side and there is no associated server side counterpart.

Transaction data generated by the application is securely stored in a non readable encrypted format accessible only to the authenticated user. User credentials are securely stored in private storage of the application in encrypted format for offline authentication. A Symmetric Key for Encryption is generated on the server. This key changes every time a new user session is started.

Encryption features can be enabled or disabled per specific mobile device. They can also be enabled system-wide via Feature Configuration by setting the Encryption value as "Default".

Local Testing

The HTML5 code added to the www directory can be tested locally using a Google Chrome browser. For device-specific features, such as a camera and/or barcode scanner, the testing must be done using native applications.

Use these steps to test the application in non-production mode:

1. If not already installed, install the Google Chrome desktop browser.

2. Create a shortcut to the executable on your desktop.

3. Right-click the shortcut and choose Properties, then append the following to the Target property:

   ```
   --user-data-dir="C:/Chrome dev session" --disable-web-security
   ```

   to disable cross-domain JavaScript security.

4. Start Chrome via the shortcut and load the mobile application (location: www/index.html in your local system).

5. If the login page does not appear or does not work correctly, reopen the Chrome shortcut properties, correct the path specified in the `user-data-dir` parameter (to specify the Chrome location), then reopen Chrome and retry the login.

Alternatively (instead of modifying the shortcut Target property), you can open a command window and enter:

```
chrome.exe --user-data-dir="C:/Chrome dev session" --disable-web-security
```

As with the shortcut, if the login page is not displayed properly, correct the path specified in the `user-data-dir` parameter in the command and rerun it.

Building and Deploying the Mobile Application

This section provides information on how to deploy the Oracle Real-Time Scheduler Mobile Application on various device types. Please refer to the *Oracle Real-Time Scheduler Mobile Application Installation and Deployment Guide* in the chapter titled "Deploying the Mobile Applications" for steps on deploying the mobile client application as a web application.

iOS Deployment

This section provides information on deploying the native application to an iOS emulator or device. Please reference the **Apache Cordova** documentation section titled "Platform Guides" under "iOS Platform Guide" for more information.

Deployment from a Command Line

Using Cordova CLI commands the native application (IPA) can be deployed to a iOS emulator or device.

1. Use the Build command to create the iOS Xcode project, e.g., `cordova build ios`

   The Xcode project is created under <Cordova_Project>/projects/ios directory as <Application_name>.xcodeproj

2. Open the xcodeproj file.
   The Xcode IDE file opens.

3. Click **Run** to deploy and run the application to an iOS simulator or device.
   The application will get deployed on a configured iOS emulator where the application can be tested.

   An iOS developer account is required to run to deploy the application on an iOS device.

Deployment Using an IPA File

Using Cordova commands, the native application (IPA) can be created and deployed to an iOS device.

1. Use the Build command to create the iOS Xcode project, e.g., `cordova build ios`

   The Xcode project is created under <Cordova_Project>/projects/ios directory as <Application_name>.xcodeproj

2. Open the xcodeproj file.
   The Xcode IDE file opens.

3. Select the Product > Archive option from the Xcode IDE to generate the IPA file.

4. Sign the IPA file.

5. Create the IPA file as an Ad-hoc or Enterprise deployment.

6. Copy the <Application_Name>.ipa file to the **iTune**s → **Automatically Add to iTunes** folder.

7. Connect your iOS device to the machine and open **iTunes**.

8. Under your device on the **Apps** tab the <Application_Name> will be displayed and can be installed by clicking the **Install** button then **Apply**.
This installs the app on the device.

Android Deployment

This section provides information on deploying the native application to an Android emulator or device. Please reference the **Apache Cordova** documentation section titled "Platform Guides" under "Android Platform Guide" for more information.

Deployment from a Command Line

Using Cordova CLI commands the native application (APK) can be deployed to a running Android emulator or device

- Use the Emulate command to deploy the native application to an emulator, e.g., `cordova emulate android`

    The application will get deployed on a configured Android emulator where it can be tested.

- Use the Run command to deploy the native application to a device, e.g., `cordova run android`

    - Make sure USB debugging is enabled on the device.

    - Use a mini USB cable to plug the device to your system.

Deployment using an APK File

Using Cordova commands, the native application (APK) can be created and deployed to an Android device

Use the Build command to compile and build the native applications, e.g., `cordova build android`

- The application (APK) file is created under the in the Cordova project under the folder <Cordova project folder>\platforms\android\ant-build

- The file created will be <application_name>-debug.apk and can be installed on Android devices for testing.

- To install the APK file on Android device make sure the option to install apps from Unknown sources is checked in your device.

- Browse to the APK location using File Manager and open the APK which prompt option to install the APK.
The application will get installed on the device and can be tested by running it on the device.

Chapter 3

Oracle Utilities Mobile Library

This section describes the key modules and APIs that are available for implementing new user interface pages and application features.

Device Communication

This section provides information on communication between the server application and mobile devices. Although we have two categories of messages, inbound and outbound, they are both transferred via HTTP requests initiated by device.

Device Inbound Messages

Configuration

MDT type uses the ASYNC_INTERVAL (seconds) property to configure the interval at which a REST service(M1-SyncData) will be invoked by client. Inbound message is processed by a script which is specified on incoming message (SCRIPT column). Inbound scripts should be located in scripts/ inbound folder and script name to filename should be mapped to the inboundMsgFiles property in config.js.

Note: One JS file can have more than one inbound scripts.

Message Storage

Messages received are stored in F1_INBOUND intermediate table on device DB. Please reference the Database Schema section for more information.

Inbound Message Event API

Once a message is downloaded and saved to intermediate table it is handed over to inbound processing script. This processing script should be implemented as follows:

Processing Script Code Structure

```
ouml.Inbound["M1-MCPDpAsgn"] = (function (ouml){

function process(msgEvent) {


}

return {
      process: process
   };
})(ouml);
```

M1-MCPDPAsgn is an example script code. This should be replaced with your actual script name. This script needs to implement a process method that is required to be exposed as public method of this module.

The Oracle Utilities Mobile Library invokes a process method with an event object with the following structure.

- **msgEvent.message** – inbound message in JSON format (format of this message is as defined on server, specific to a script)

- **msgEvent.error(ouml.ClientError)** – this method should be called in case an error occurs in processing this message. An instance of ouml.Error should be passed to it. This error message is saved to F1_INBOUND table's error column.

- **msgEvent.complete(transaction)** – this method should be called on successful message processing. Transaction used, if any, should be passed to this method and same will be used by the Oracle Utilities Mobile Library to update the F1_INBOUND'S PROC column. If no transaction is passed then a new transaction is created.

Message Acknowledgements

On successful download and save to the intermediate table, a message delivery acknowledgement is sent back with very next REST service call. This only indicates the delivery part, not the processing. On successful message processing another acknowledgment is sent with a flag to indicate whether or not the processing was successful. If during message processing an error occurs, the same error is also sent back to the server.

The Input to the REST service contains following payload:

```
{
"msgId": msg_id column value from F1_INBOUND,
"isProc": true/flase (true when PROC column value is Y)
"errorData": {error object}, error column value}
}
```

Device Outbound Messages

An outbound message is essentially a RESTful service invocation initiated by client which delivers a message (JSON payload) to that service on server. There are two types of outbound RESTful invocations modes from client:

Online Mode

A service invocation where response from the service is required to proceed further with the business flow. For this type of outbound call, client has to be connected to network as if device is offline we cannot proceed further.:

Online Mode API	Parameters	Description
ouml.AJAX.invokeService	service – service to be executed args – {onSuccess: <callback>, onFailure: <callback>, method: <GET or POST>, contentType: <content type header>, headers: {<all headers passed as is to ajax call>} } payload - JSON Data	Invokes a service immediately (device has to be connected to network) and returns the results via an asynchronous callback. This API adds mandatory headers required for authentication and connecting to server.

Offline Mode

A message is posted to a service however the actual call to service would be made only when the device is connected. Such outbound messages (service calls) are delivered to server as and when the device is connected and client business flow is not dependent on response from server. However it is ensured that no message will be lost and it will be delivered to server eventually. Client ensures that message sent via this outbound module are stored in offline storage and delivered in same sequence they were posted. Application crash or network connectivity should not result into any message loss.

Offline Mode API	Parameters	Description
ouml.OutboundWorker.queueOutbound	args – { transaction:<tx object>, onSuccess: <callback>, onFailure: <callback>, input: { service: <service name>, payload: <JSON data>} }	Message posted via this API will be saved to F1_OUTBOUND table using passed transaction else a new transaction will be used. Transaction object will be returned via success callback so that same can be used to execute the next transaction in case of multiple commits. Whenever device comes online the payload will be delivered to the specified service.

Server Communication

This section describes the outbound and inbound messaging used by the system.

Server Outbound Messages

This section refers to messages that are outbound from the server inbound to the mobile device.

Outbound messages are maintained through the M1-MessageToDevice business object. The different states that the outbound message can transition to are defined and managed by the business object's lifecycle.

For data synchronization the device sends:

- Device ID
- A list of acknowledgements. Includes Remote Message ID, PROC_SW Y/N (whether it's been processed yet), and optional error details.

The device receives:

- A list of new messages to be processed.
  This includes Remote Message ID, business object, message name, payload, priority.
- Ordered by priority (with high-priority messages first) and then in Created Date-Time order.
- Number of messages is based on bucket size.

The following "rules" apply for client applications that process outbound messages:

- Valid MDT_ID
  Defined in the server application.

- Number of messages received is based on bucket size defined on the MDT Type. There may be more or less messages than what is received, however the bucket size limits the number of messages received at one time. The system continues to send the messages in batches until the queue is empty.

- If the device does not acknowledge receipt of the message, the same message will be sent again.
  It is possible to set "callback" settings to cancel messages so that they aren't continually sent.

- If the device does acknowledge receipt, the message must be processed.
  Messages should be processed in order, high-priority first.

- Error details are provided in the outbound message.

If the caller wants to work with the output message IDs from M1-InvokeRSIScript or M1-GetRSIIdsByContext , it could be an RSI_ID (30 chars) OR a REMOTE_MSG_ID (14 chars).

All Callback and Error scripts, ditto. The existing element <rsiMessageId> may be 30 chars or 14 chars.

Callbacks are done only for messages that have not been delivered. If a message is delivered but never processed, it will remain in Queued status forever, unless some other process handles it. You may want a Monitor for that.

Output RSI ID for Various Business Services, Service Scripts, etc.

- Business Services: Invoke Remote Script (M1-InvokeRSIScript) and Get Remote Script Invocation By Context (M1-GetRSIIdsByContext)

- Output message IDs can now be either an RSI ID (30 characters long) or a Remote Message ID (14 characters long).

- Callback and Error Scripts

- Existing schema message IDs can now be either an RSI ID (30 characters long) or a Remote Message ID (14 characters long).

Callback Logic

Callbacks are done only for messages that have not been delivered. If a message is delivered but never processed, it will remain in Queued status forever, unless some other process handles it.

Call back is configured, in seconds, under **Master Configuration** > **Global Configuration**, field: **Remote Script Call Back Seconds**.

This indicates the number of seconds that should pass (from the message's creation date time) before the callback is executed. This works when the Remote Message monitor batch process triggers the remote message's monitor algorithm (which executes callback scripts when applicable).

Remote Message Batch Monitor

The remote message monitor, a timed monitor batch process, can be set to monitor the rules associated with the current state of messages that go between the server application and mobile devices. It is recommended that you set this monitor to run very frequently such as every 5 minutes so that processed messages can be transitioned to a non-queued state (to improve performance on queries for unprocessed messages).

Server Inbound Messages

This section refers to messages that are outbound from the mobile device and inbound to the server.

As described in Client side outbound messages section above, these messages to server are delivered by invoking specific services as per the given context or business logic, e.g., Get Shift, update shift, update task, etc.

Guaranteed Delivery

A special kind of inbound messaging called **Guaranteed Delivery** ensures that messages from a device are stored in the application database first, and then processed afterwards. This ensures that even though the message cannot be processed immediately because of other factors, the message is at least guaranteed to be delivered to the server.

Guaranteed Delivery Algorithm

The remote message guaranteed delivery algorithm, M1-REMMSG-GD, processes guaranteed delivery requests through remote message creation (through the business object M1-CrewMessage) and state transition. Your implementation must configure the base algorithm on Installation Options/ Guaranteed Delivery. This is configured by navigating to **Installation Options** > **Algorithms**, System Event: **Guaranteed Delivery**.

Remote Message

The **Remote Message** table uses a Device Message ID field that stores a unique ID sent from the mobile device to distinguish inbound messages sent from the server application. This field is later used by the Guaranteed Delivery (M1-REMMSG-GD) algorithm to verify whether or not an inbound message already exists in the Remote Message table before creating a new record (to avoid duplicate entries for inbound messages).

Logging

System logs are sometimes needed to diagnose how the server application is communicating with devices, investigate errors, or for other troubleshooting or informational purposes.

Mobile log files can be accessed in the MDT portal under the **Log** tab.

Changing Log Settings from a Device

Device users can change log settings from the Oracle Real-Time Scheduler **Mobile Application Settings** page. This includes turning logging on or off, as allowed by the user's permissions, setting the log level, and setting appenders.

Log Appenders

The logging module supports the following types of appenders to display logging messages:

- **Console Appender** (CONSOLE): Writes log messages on the web console.

- **File Appender** (FILE): Writes log messages in a local file on the client. The log files in the client are then sent to the server when requested.

- **Remote Appender** (AJAX): Sends log messages (json/xml/text) to the server with an asynchronous HTTP request.

- **Popup Window Appender** (POPUP): Opens a new window/sub window in the browser and writes log messages in real time.

Users can enable more than one appenders at the same time to write logs from setting page of application

Log Message Format

Log entries use the following format.

[Unique Prefix] - Date Time Log-Level Log Message (Origin Module Line Number)

Log API

The Oracle Utilities Mobile Library Logging module exposes the APIs required by your implementation to facilitate system logging. Any application module that requires logging uses this

module with the single log instance maintained for the complete application. Logs get the appropriate instance from ouml.JSLogger and use the exposed API.

For example to log an info message your implementation would use:

- ouml.JSLogger.info('Your message ');

- Extra public APIs exposed by this object (not part of the Oracle Utilities Mobile Library or parent business object)

  - **mdtdebug(message)**: The module that needs to log a **framework level debug** message calls this method.
    Passes the log message arguments to the methods.

  - **debug(message)**: The module that needs to log a **debug** message calls this method.
    Passes the log message arguments to the methods.

  - **info(message)**: The module that needs to log an **info** message calls this method.
    Passes the log message arguments to the methods.

  - **warn(message)**: The module that needs to log a **warn** message calls this method.
    Passes the log message arguments to the methods.

  - **error(message)**: The module that needs to log an **error** message calls this method.
    Passes the log message arguments to the methods.

  - **perf(message)**: The module that needs to log a **perf** message calls this method.
    Passes the log message arguments to the methods

  - **fatal(message)**: The module that needs to log a **fatal** message calls this method.
    Passes the log message arguments to the methods

  - **setLevel(level)**: These methods set the logging level of the logger instance that the application has acquired initially. The level that is to be set should be within the set of levels supported by Logger. Else default logging level will be used

  - **syncLogFile()** : This method synchronizes the log files to the server.

Offline Database

This section provides information regarding client side offline database tables and APIs available to interact with the offline database.

A WebSQL database is used for local data storage if the application is opened in a web browser. If the application is installed as a native app on a device and "sqliteDB" property is set (in config.js) then the SQLite DB on the device is used. The database is initialized with an initial size of 5MB.

Database Schema

API

getHandle - Returns the DB handle object. This returns a singleton instance of an object that should be used for any DB transactions.

Tables

At application launch, the tables indicated below are created in the browser database or in SQLite if they do not already exist. You can reference this schema and browse the database during development or debugging.

F1_BIZOBJ

This table stores both deployment and transaction data for all business objects. GEN_COL1 to GEN_COL10 can be used to store specific fields that can be used to query the business object.

Offline Field	Description
BO_KEY	combination of business objects PK1-PK5 (pk1^pk2^pk3^pk4^pk5)
BO_CODE	Business object code
MO_CODE	Maintenance object code
DATA	JSON data for a business object
TYPE	Type of data (DEPLOYMENT or TRANSACTION).
DATE_UPDATED	Timestamp(local) when the data was modified.
VERSION	Version of the record.
GEN_COL1 - GEN_COL10	Generic columns for storing business object attributes used in search and application logic.

F1-Inbound

This table supports inbound messages.

Offline Field	Description
MSG_ID	Unique message ID for the inbound message.
PAYLOAD	JSON Data received in a message.
SCRIPT	Script code (message processing script).
PRIORITY	Priority of the message.
ACK_REQUESTED	Flag to indicate whether acknowledgement is requested.
ACK	Flag to indicate whether acknowledgement was returned.
PROC	Flag to indicate whether the message is processed.
PROC_ACK	Flag to indicate whether processing acknowledgement was sent.
ERROR	Error message received during processing, if any.

F1-Outbound

This business object supports outbound messages.

Offline Field	Description
ID	Unique message ID of the outbound message.
SERVICE	Service name.
PAYLOAD	Service input payload.

Config

Each application layer has its own config.js file where a new property can be added. A property defined in the lower layer can also be overwritten by defining a new property with the same name.

Some of the properties that are of type array cannot be overridden completely but values from each layer are merged. Please reference the description of each property.

API -ouml.Config

- **apps** - an array of app owner codes (owner code) (e.g. ["M1","CM"])

- **restServerURL** - OUAF REST API URL

- **mobileAppURL** - Mobile app URL

- **DEFAULT_MDT_URL** - DEFAULT MDT URL

- **DEFAULT_DEPLOYMENT_ID** - DEFAULT_DEPLOYMENT_ID,

- **mainMenu** - menu items that should be available on every page menu

- **applicationFolder** - A folder name used to store the files on local device filesystem

- **URLMapping** - Mapping of URLs. This mapping can be used to override default UI pages

- **initScript**- Initial script that gets executed after successful login, this script decides the application home page.

- **getConfig** - Returns the value of a property (the property available in topmost app)

- **boFiles** – list all the files required by a business object. If the only file that a business object requires is same file as the name of business object and is available in scripts/bo folder then no need to include that. In case of CM config, files are assumed to be present in scripts/bo folder.

- **getBOFiles** - Returns the JS file names required by a business object. This API is internally used by ouml.Loader.loadBO so implementers will not have to ever use this. This property returns the value of boFiles variable after merging it from all layers.

- **pageFiles** – list all the files required by a UI page (business object or non-business object). If the page id (div having data-role =page) is same as file name then no need to include that file. provide absolute path starting from product folder (e.g. cm/taskList.js)

- **getPageFiles** - Returns the JS file names required by a Page(bo pages too) UI. This API is internally used by ouml.Loader.loadPage API so implementers will not have to ever use this. This property returns the value of pageFiles variable after merging it from all layers.

- **commonJSfiles** – List all JS files that should be loaded on successful login. This property is used by login module and it loads all the files defined at different layer, after merging it from all layers. Common JS files like plugins.js or common.js which hosts common APIs not specific to a business object or a Page should be declared in this property. File should contain the path starting from product folder name (e.g. m1/scripts/plugins/plugins.js)

- **inboundMsgFiles** – List mapping between an inbound script and corresponding file containing the script. File MUST be present in scripts/inbound folder. CM can override base .

- **getInboundMsgFiles** - Returns the inbound message handler file names for a given script code.This API reads the value from inboundMsgFiles variable in config.js of each app layer and returns the files from appropriate layer. This API is internally used by the Oracle Utilities Mobile Library and implementers will not have to use this.

- **capabilitiesMapping** – define a mapping between a capability type (defined on server) and corresponding script to be executed on client for a given capability. These scripts should be defined in common.js (e.g. cm/ui/common.js) or some JS file that is loaded via commonJSFiles so that whenever a capability request (e.g. scan barcode) is made this file should be already loaded.

- **oracleMapProperties**- Used to configure the Oracle MapViewer properties. The **serverConfig** property is very important. This is the name of the Feature Configuration created for the Oracle MapViewer on the server. The client gets all the MapViewer information like URL, Datasource, Tile Layer etc using the Feature configuration. Besides the serverConfig the images for activities can be changed here. The style for the information window which pops up on clicking an activity marker can also be modified here.

- **sqliteDB** – Set to false by default out of the box. If set to true the SQLite plugin is used to create a SQLite database on the client devices instead of using the HTML5 WEBSQL database. This flag can be set to true only if the SQLite plugin is installed for the Cordova project used to build the native application.

Encryption APIs

For android devices, encryption is provided by a cordova plugin. However, instead of using cordova plugin APIs directly, you should use the APIs indicated in the table below in the ouml.Crypto module.

The Oracle Utilities Mobile Library uses these APIs internally to store data to the F1_BIZOBJ table if the encryption is enabled for devices. Please reference Chapter 2: Encryption for more information.

These APIs return the original input as is if the encryption is not enabled for this device. Callers of the APIs can check the output in success callback to confirm if the data was indeed encrypted (or decrypted).

API	Parameters	Description
ouml.Crypto.encrypt	**args =** {onSuccess: <success callback>, onFailure:<failure callback>, input: {data: <text string or an array of text strings>}, encryptionKey : <optional, key to be used>}	Encrypted input data will be returned via success callback as {output: <encrypted text>, encrypted: <true|false>}. If the input was an array then output will be an array e.g. {output: []} with each array element corresponding to input array element. Encryption key is not required unless you have to use a different encryption key than what is configured on server. Encrypted property is set to false if no encryption was done in case of iOS device or encryption not enabled for this devce.
ouml.Crypto.decrypt	**args =** {onSuccess: <success callback>, onFailure:<failure callback>, input: {data: <text string or an array of text strings>}, **encryptionKey** : <optional, key to be used>}	Decrypted input data will be returned via success callback as {output: <decrypted text>, decrypted: <true|false>}. If the input was an array then output will be an array e.g. {output: []} with each array element corresponding to input array element. Encryption key is not required unless you have to use a different decryption key than what is configured on server. Decrypted property is set to false if no decryption was done in case of iOS device or encryption not enabled for this devce.

Cordova Encryption Plugin APIs

The plugin call takes the following parameters:

1. **success**: Function name of the function to be called on successful execution of the plugin. This function is called with a string parameter depending upon the value of the action parameter.

2. **failure**: Function name of the function to be called on execution failure of the plugin. This function is also called with a string parameter containing the error message of the error which occurred while executing the plugin leading to failure.

3. "**Crypto**": The plugin identifier.

4. **action**: The action parameter passed to the plugin. This includes one of the following values:

   a. **encrypt**
   For this action, the plugin will return the encrypted string of the input text on success. The encryption key will be passed along with the input text as parameter to the plugin in json format.

   b. **decrypt**
   For this action, the plugin will return the decrypted string of the input encrypted text on success. The encryption key will be passed along with the input text as parameter to the plugin in json format

   c. **hash**
   For this action, the plugin will return the hashed value of the input string on success.

5. **json**: The input parameter to plugin in json format. It will contain the input string to be encrypted along with the symmetric encryption key to be used for encryption.

Process Details

1. Users log in to the system in online mode. The user credentials are stored in persistent storage using the hashed value obtained from custom Cordova plugin for offline authentication.

2. After login the following device options are fetched from server in online mode and are stored in local storage:

   a. MDT_ENCRYPTION_FLAG

   b. MDT_ENCRYPTION_KEY

   If the user logs in offline mode then the last stored values of these device options are used in the application.

   If transaction data exists on the device then the new values obtained from the server for these device options are not overwritten in the local storage. Thus the MDT_ENCRYPTION_FLAG and MDT_ENCRYPTION_KEY device options values on the device cannot be changed after transaction data is generated on the device.

3. Using the MDT_ENCRYPTION_FLAG device option the encryption module can be turned on ('M1ON') or off ('M1OF) for a particular device using the MDT portal page.

4. If MDT_ENCRYPTION_FLAG set to 'M1ON' then the transaction data generated on the hybrid client is stored in encrypted format in local storage and its decrypted after reading from local storage to get the original form before use. If MDT_ENCRYPTION_FLAG is 'M1OF' then all transaction data on device is stored in readable text format.

   If the value is set to is M1DF (default), then the value is fetched as per the Master Global Configuration.

5. The MDT_ENCRYPTION_KEY is stored in local storage in encrypted format. It is encrypted using the base64 encoding value of username:password as encryption key and using the same encryption algorithm which is used to encrypt transaction data.

Deployment data is not encrypted on the device as it is not transactional data.

Deployment

The application consists of code and metadata:

- Code is installed (for native apps) or deployed (for webapp) as an application.

- Metadata that is required for the application to work properly, is downloaded on a successful logon in JSON format and stored in the offline database. Deployment metadata is stored in F1_BIZOBJ table with DEPLOYMENT as value in "type" column. The Oracle Utilities Mobile Library provides various APIs to access deployment metadata in simple format Data consist of various Oracle Utilities Application Framework objects including:

    - Labels

    - Lookups and Extended Lookups

    - Messages

    - Business Object Lifecycle

    - Business Objects Data (non transactional objects)

These objects can be configured on the server. Please reference the *Oracle Real-Time Scheduler Server Administration Guide* for more details.

API (module - ouml.Metadata)

API	Parameters	Description
getLabel	Label/field Id	Returns the label description (should be used instead of hardcoding text strings on UI). Check ViewModel wrapper API for usage on HTML pages.
getLookup	Lookup ID	Returns an array of items containing lookup value and description in the format [{lookupValue: "", description:""},] Check ViewModel wrapper API for usage on HTML pages.
getLookupDesc	Lookup, lookupValue	Returns the description for a specific lookup value of a lookup. Check ViewModel wrapper API for usage on HTML pages.
getExtLookup	Extended Lookup BO Name	Returns an array containing lookup value and description in the format [{lookupValue: "", description:""},] Check ViewModel wrapper API for usage on HTML pages.

API	Parameters	Description
getExtLookupDesc	Extended lookup BO Name, lookup Value	Returns the description for a specific lookup value of a extended lookup BO Check ViewModel wrapper API for usage on HTML pages.
getMessage	Message category, message Id	Returns the message. (To get formatted message with parameters, use ouml.ClientError API)
getNextBOStates	Bo name, bo status	Returns a list of next valid states which the business object can transition to from a given state. It returns an array of objects with this format {boNextStatusLabel, boStatus, role}
getStatusReasons	Bo Name, status	Returns an array containing the status reasons valid for a given state. Format of the output is [{description: "", selectability:, "", statusReasonCd: ""},{}]
isFinalBOState	Bo Name, Status	Returns true if there are no next valid states for a given business object and state, otherwise returns false.
getBOInfo	Bo Name	Returns all information(metadata) about a BO. A JSON object with description, owner code, each valid states and related info.
getAncestors	Bo Name	Returns an array of items with business object information for all business objects in the hierarchy. At 0th index is the top most parent and given business object at the end of the array.
getAncestorNames	Bo Name	Returns an array of business object names for all business objects in the hierarchy. At 0th index is the top most parent and given business object at the end of the array.

To read the value of a Business Object in deployment, you can use BOEntity APIs.

Properties

The following properties are downloaded at login.

- Properties
- KeyValue
- sessionId
- decimalSeparator
- API - ouml.Properties

Business Objects

Please reference the **Oracle Utilities Application Framework** documentation for details of server side implementation and metadata associated with business objects. Mobile client implementation of business objects includes the following artifacts:

- **Business Object JavaScript (BO JS)** - A JavaScript file located in the scripts/bo/ folder with the same name as the business object name.

- **Business Object User Inteface (BO UI)** - An HTML file for the BO UI located in the ui/ bo/ folder with the same name as the business object name.

- **Business Object User Inteface JavaScript (BO UI JS)** – A JavaScript file located in the ui/ bo folder with the same name as the business object name.

- **boFiles** – config.js mapping for the BO JS files.
  This is only specified if the business object name and the JavaScript name are not the same.

- **pageFiles** – cconfig.js mapping for the BO UI JS files.
  This is only specified if the file name is not same as the Page ID.

Business object data is received by Inbound Service and processed by inbound scripts. Please reference the Device Inbound Messages section for more information.

Javascript in the Oracle Utilities Mobile Library uses revealing module pattern or prototype pattern or a combination of both where prototype is wrapped in revealing module. Every Javascript class/ module is attached to a namespace that starts with "ouml".

Please make yourself comfortable with Object Oriented Programming in JavaScript which is a pre-requisite for writing new BO classes. (https://developer.mozilla.org/en-US/docs/Web/JavaScript/ Introduction_to_Object-Oriented_JavaScript

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model)

Business Object JavaScript (BO JS)

Every BO JS must extend GenericBusinessObject class, if there is no parent to this business object otherwise it must extend the parent class.

Example business object class:

```
ouml.BusinessObject["M1-Assignment"] = (function (ouml){
    //define private variables and functions here (vars and
functions that should not be accessible from anywhere else except
this module)
var util = ouml.Utilities;
    /**
     * M1-Assignment BO constructor (variable name m1Assignment can
be named anything).
     * @constructor
     * @memberof ouml.BusinessObject
     *
     */
    var m1Assignment = function(data){
        this.bo = "M1-Assignment";
        //Invoke the parent BO, if any or generic BO, and pass the
"this" reference. So that parent can use same "this" reference.
        ouml.GenericBusinessObject.call(this, data);
    };

    //set the prototype to parent BO, so we extend the parent's
functions.
    m1Assignment.prototype =
Object.create(ouml.GenericBusinessObject.prototype);
```

```
        //point the constructor property to this key (in case we need to
make use of it later)
        m1Assignment.prototype.constructor =  m1Assignment;

        return m1Assignment;
        })(ouml);
```

GenericBusinessObject APIs

API	Parameters	Description
<constructor>	Data – BO JSON data (optional)	If BO JSON data is provided it will be set to this.data property of this instance. To create a new BO instance you should use ouml.BusinessObjectFactory.getBusinessObject
addBO	{onSuccess: <success callback>, onFailure: <failure callback>, transaction: <transaction to be used>, input.outbound: <outbound message details>}	Inserts the BO data to F1_BIZOBJ tables.Use this API only when a record doesn't already exist for this BO in DB (otherwise use updateBO). If a save operation should also send an outbound message via guaranteed delivery to server, set the service name and payload in input.outbound argument. Input.outbound = {service: <service name>, payload: <service payload JSON> } This API invokes getDTO API of BO and uses the returned value to save to DB. (getDTO returns the mapping of BO JSON to F1_BIZOBJ column)
getData	JSON path of a filed	Returns the value of a JSON path on BO data. Directly accessing a value in a nested JSON structure might result into "undefined is not a function" error so to avoid that use this API.
getDTO		Every BO must implement this method. This should map the BO JSON data to columns in F1_BIZOBJ
hasUndefinedOr EmptyField	JSON path of a field	Returns true if the field is undefined or is empty.
setData	Field, Data	When supplied with only one argument, it should be the BO JSON data. If you need to set the value of a specific field pass both field and data.
setFieldData	Field,Data	Set the value of a specific field to given data.
updateBO	{onSuccess: <success callback>, onFailure: <failure callback>, transaction: <transaction to be used>, input.outbound: <outbound message details>}	It is like addBO but instead of inserting a new record it will update the data

Business Object Factory API

This API is used to create a new instance of a business object. The API ensures that before returning a business object instance, all the parent business object related files are loaded and then only a new instance is created and returned.

BOHelper API

API		Description
getBusinessObject		Accepts the business object name and data as arguments and returns a new instance of a BO initialized with data, if provided. Data can be set on an initialized business object instance by accessing data property of instance.
loadRawData	args = {transaction: , input: {boKey: , bocode: , mocode:, genCol1:}, onSuccess: , onFailure}	Specify column name value pairs in input arguments. Success callback will be called with an argument in format {output: <BOJSONdata>} if more than one record were fetched output will be an array of BO JSON data. Use this API when you need just the data but no BO instance. Creating a BO instance loads all BO related JS files, however if you are interested only in BO JSON data this API can be very light weight when compared with getBusinessObject API.
getRowCount	args = {transaction:<tx>, onSuccess:<success callback>, onFailure:<failure callback>, input: { <db column key value pairs>}}	Success callback is called with row count (based on filter criteria passed in input) as {count:<value>, transaction:<tx>}
getBOStatusDescr	BO Name, Status	Returns the description for a BO and its given status code.
checkProcedureStatus	args = {transaction:<tx>, onSuccess:<success callback>, onFailure:<failure callback>, input: {boKey: <value>}}	Success callback is called with a true or false as an argument, true when all procedures are complete or there are no procedures
getNewBOPrimeKey		Returns a new primary key. This API is useful for those business objects that are not received from the server but are created on the client first and then posted to the server.
isProcedurePending	args = {transaction:<tx>, onSuccess:<success callback>, onFailure:<failure callback>, input: {boKey: <value>}}	Success callback is called with a true or false as an argument, true when any procedure is pending or false when none is pending or none exists.

API		Description
getStoredProcedureData	args = {onSuccess: <success Callback>, onFailure:<failure callback>}	Success callback returns all the procedures having moCode=M1-PRCTYP and all procedures types with moCode=M1-PRCTYP
isProcedureFailed	args = {transaction:<tx>, onSuccess:<success callback>, onFailure:<failure callback>, input: {boKey: <value>}}	Success callback is called with a true or false as an argument, true when any procedure has failed or true if when none is in failed state or none exists.If no boKey is passed all procedures are checked

Business Object Entity API

This module is the lower level API to insert/update records in F1_BIZOBJ table. In most of the cases BusinessObject addBO/updateBO and other APIs from BOHelper should be used. However for exceptional scenarios following APIs can be used. Both BOHelper and BusinessObject uses these APIs internally.

Every column of F1_BIZOBJ table has a corresponding property in this class. Underscore in column name is removed and a camelCase approach is used to name the JS variables corresponding to a DB column. E.g. bo_key DB column maps to boKey inJS.

add		Inserts this record into f1_bizobj table
update		Updates this record in f1_bizobj table, uses boCode and boKey to uniquely identify a record
read		Read a record from table
purge		Deletes the record from DB
setOutboundMsg	args = {service: <service name>, payload: <JSON data>}	Invoking this method before calling update/add method would ensure that an outbound update is also sent to server using same transaction.

Business Object UI (HTML and Javascript)

Every BO has a top level UI page (UI Map/HTML file), which is the main landing page when you use navigateToBOPage API. This page usually has list of sections and each section point to a <DIV> in either same HTML file or other HTML fragments(files) included via an overridden API(loadPageFragments). HTML file name must match the BO name and the top level DIV which has data-role=page should have BO name as the value for div's id attribute. E.g. for M1-Assignment BO, file name would be M1-Assignment.html and the top level div in that page is:

```
<div data-role="page" id="M1-Assignment">
    <script src="M1-Assignment.js"></script>
```

For a non BO Page too the div Id should uniquely identify a page and if the JS file name, html file and the div id all are same then the Oracle Utilities Mobile Library can load the JS file automatically without any script include.

As you can see above we are including a script inside the div, this is important as with jQuery mobile, you cannot expect any script outside the page div to be loaded. So if you ever have to load a JS file for any page (BO or non BO) it should be included inside the page div.

Having said that if the JS file name is same as the bo name or the id of page div (for non-business objects), then we need not include this file in HTML code and the Oracle Utilities Mobile Library automatically includes a file with this name in same directory as the page.

Page View Model

Every UI JS must extend the BaseViewModel class, if there is no parent to this BO otherwise it must extend the parent class. Code for every UI page (bo/non bo) should be attached to a specific namespace (ouml.ViewModel) as shown in this example BO class:

```
ouml.ViewModel["M1-Assignment"] = (function(ouml) {
function m1Assignment() {
    ouml.ViewModel["M1-Common"].call(this);
  model = this;
};
//set the prototype to parent BO, so we extend the parent's
functions.
    m1Assignment.prototype = Object.create(ouml.ViewModel["M1-
Common"].prototype);

    //point the constructor property to this key (in case we need to
make use of it later)
    m1Assignment.prototype.constructor =  m1Assignment;
    return m1Assignment;

})(ouml);
```

Note: In this example we have a common parent for all business objects, which extends BaseViewModel hence we are extending M1-Common here.

This is the basic minimum code that every business object (nonBO) UI must have.

BaseViewModel API Properties

LABELS	A reference to all labels, can be used in HTML as: <span <u>data-bind</u>=*"text: LABELS.M1_SITE_ADDRESS"* id=*"siteAddress-label"*>
LOOKUPS	A reference to all Lookups. usage: <select class="ui-select" id="keepWithCrew" data-role="none" data-bind="value:stateSpecificFields.keepWithCrew, options: LOOKUPS.M1_SAME_CREW_FLG, optionsText: 'description', optionsValue: 'lookupValue', optionsCaption: 'Select One ...'"></select>
EXTLOOKUPS	A reference to all Extended lookups. Usage: <select id="customerContactType" data-bind="options: EXTLOOKUPS['M1-CustomerContactType'], optionsText: 'description', optionsValue: 'lookupValue', value: completionInfo.customerContactDetails.customerContactType, optionsCaption: ""data-role="none"></select>
pageTitle	This is a knockout observable array, so anytime value is changed it will automatically reflect on UI title. model.pageTitle(model.LABELS.M1_SHIFT_LBL)
pageButtons	This is a knockout observable array, all page buttons are stored here. Please reference the setPageButtons API under Page View Model for more information.

pageMenuItems	This is a knockout observable array containing all menu items.
pageIndicators	This is a knockout observable array containing all page indicators.

API	Parameters	Description
showBackButton	Boolean(true|false)	Sets the visibility of back button on a page. Visible when set to true.
showPanicButton	Boolean(true|false)	Sets the visibility of Panic Alert button on a page. Visible when set to true.
showMapButton	Boolean(true|false)	Sets the visibility of Map button on a page. Visible when set to true.
loadPageFragments		This API when implemented by a page will be invoked before displaying the page content. This API must return an array of HTML file names. The content of each file will be appended to the currently active mobile page's content..Use this API to create a UI from multiple HTML files, and reuse same HTMl file in multiple pages. Each HTML file should contain divs which can be navigated to by showSection API. All the Divs should have visibility of none, otherwise they will appear on UI as soon as a fragment (an html file) is loaded. Please reference the UI Layout and Navigation section for more details.
load	args = {onSuccess:<callback>}	This API when implemented by a page will be invoked just after page HTML/JS files are loaded. Implementer must invoke args.onSuccess() on completion of the work of this method. It is assumed that something asynchronous can happen in this overridden method hence the onSuccess callback is provide to indicate the completion of that work. E.g. loading appropriate data from DB and binding the UI via KO.
setPageButtons		Applicable only for business object pages. This API when implemented by a business object page will be invoked during page load process to allow the page to customize the page specific buttons. Overridden method must set pageButtonList property to an array of buttons, each button object should match this structure: {buttonLabel : <string value>, buttonAction: <click handler function on viewModel >}. Default Oracle Utilities Mobile Library implementation of this method adds next valid states of current BO's state to pageButtonList, and set the state name as handler function name, which means for every state the BO UI JS file (ViewModel) should have a corresponding method.

API	Parameters	Description
onInboundMessage	{inboundMsg } – inbound message received from server	This method will be invoked every time a new message is received via InboundWorker. Message received from server will be passed as an argument (inboundMsg.msgData). (each inbound message processing script decides whether or not to notify the current page)
navigateToBOPage	bo – Name of the BO args = {inputArgs: {key:value}, inputData: {key: value}}	Loads the BO JS files, BO UI page/html and BO UI JS files in that order. Even if your HTML file for a business object has no JS included via script tag it will load the matching file (same name as bo) in same folder as the main html file. It also loads the pageFiles specified in config.js for a given pageID (div's id)
navigateToPage	bo – Name of the BO args = {inputArgs: {key:value}, inputData: {key: value}}	Same as navigateToBOPage except that this can be used to load any non business object page UI html files. So no BO JS files are loaded.
showSection	args = {id:<id of the div>, title: <string for title>, processAction: {icon:< data-icon attribute to be set>, handler:<method in current viewModel to be invoked>}}	This API doesn't switch the currently loaded page however hides currently active section and displays the requested one. Left side icon will be a back button and right side icon will be set accordingly only if processAction is set. On click of right side icon handler will be invoked
goBack		Use this API to back to either previous section or to a previous page. Whatever was displayed before this UI.
showError	error – an instance of ouml.ClientError sectionId – optional, div ID of a section that should be displayed to show this error on	Displays an inline error message in RED color at the top of either currently displayed section or displays the section with given ID first to show the error
setDefaultSection	sectionId- div id of a section that is displayed by default when page loads	This API must be implemented in order for showSection to work. Default section is the Div that is displayed by default when page is loaded.
showDefaultSection		Show the default section and cleans the page's section history stack. So that using goBack on main page should not go back to previously displayed section of that page but previous page in history.
showProcedures	Key – BO primary key onComplete – success callback	
showCommonAttachments		

API	Parameters	Description
showAttachments		
dialNumber		
getFormattedDate	date – date to be formatted	Converts a base date to device date and formats it to a user specific format. Useful for displaying business object data in the UI.
getFormattedTime	dateTime – datetime to be formatted	Converts a base datetime to device datetime and formats it to a user specific format. Useful for displaying business object data in the UI.
getFormattedDateTime	dateTime – datetime to be formatted	Converts a base datetime to device date time and formats it to a user specific format. Useful for displaying business object data in the UI.
setPageMenuItems		Implement and override this method to add page specific menu items. Oracle Utilities Mobile Library calls it at appropriate time to render the page menus. Please reference addMenuItem under Page View Model for more details.
addMenuItem	menuItem – a menu item object , ouml.MenuItem({index:<index of item>,title: <label>, action:<callback function on viewModel>, active:<true|false> });	setPageMenuItems method if overridden must add individual menu items using this API. A Menu item object should be passed to this method. Please reference the Menu section for more details.
setAppMenuItems	an integer value	Returns an ouml.MenuItem object. This API helps your implementation to write a custom API for menus.

Buttons

The Page Buttons API automatically generates life cycle buttons for a BO User Interface. User can override this default behavior by overriding setPageButtons API of the Base View Model present in Oracle Utilities Mobile Library.

For automatically generation of Life Cycle buttons for a Bo UI, page specific model needs to extends ouml.BaseViewModel.

For generation of buttons for a Non BO UI developer needs to override setPageButtons API and either call addButton function of base view model or directly push button JSON into pagebUttons observable array.

JSON for Buttons

```
{
        buttonLabel: "Button Label",
        buttonAction: function
}
```
Menu API has been added as part of the Base View Model and will be available in child view model at different application layers if child view model extends base view model.

1. **setPageButton**

   Description
   Need to override in page specific view model to add page buttons.

   Specified by
   setPageButton in ouml. BaseViewMode()

   Parameters
   none

   Returns
   none

   Sample Uses
   cmModel.prototype.setPageButtons = function(){
        var sample = {

   buttonLabel: 'Sample',
   buttonAction: this.sampleAction
   };

                   this.addButton(sample);

          }

2. **addButton (json)**

   Description
   Returns a ouml.MenuItem object. This API will help developer/Cm to write there custom API for menu

   Specified by
   addButton in ouml. BaseViewMode()

   Parameters
   json: plain json object

   Returns
   none

   Sample Uses
   Var button = {
   buttonLabel: 'Sample',
   buttonAction: this.sampleAction
   }
          this.addButton(button);

Page Level Buttons

To add page level Buttons for a non BO UI, override setPageButtons() API of ouml.BaseViewModel.

```
cmModel.prototype.setPageButtons = function(){

        var sample = {
buttonLabel: 'Sample',
buttonAction: this.sampleAction
};

            this.addButton(sample);
```

}

Properties

Properties are used to represent configurable values such as Date formats, Sync Interval, Log File Size. These properties are fetched via a REST call to the server every time in online mode. Some of these properties can be set and retrieved at execution time too.

API - ouml.PropertyEntity

Public APIs

- **getMDTProperty** – Fetch the value against the key/name property.
  This is a synchronous call for all the properties except for "PurgeOnNextLogon". This takes in an input of Property name and an optional callback function which is only used in the case of Property Name = 'PurgeOnNextLogon'.

  Method Signature - function getMDTProperty(key, callbackFunc) {}

- **setMDTProperty –** Set the value against the key/name property.
  This is a synchronous call for all the properties except for "PurgeOnNextLogon". The parameters callbackFunc, errorFunc, transaction are optional for properties that aren't stored in the DB.

  Method Signature – function setMDTProperty(key, value , callbackFunc, errorFunc , transaction) {}

- **removeMDTProperty –** Delete a given property from the Property cache. This is a synchronous call for all the properties except for "PurgeOnNextLogon".

  Method Signature - function removeMDTProperty(key , onSuccess ,onFailure ,transaction) {}

Property Names

- **ASYNC_INTERVAL –** Defines the time interval (in seconds) between Device to Server data sync.

- **ATTACHMENT_STORAGE_SIZE –** Maximum (Sum of all the attachments) attachment storage size possible for the current MDT>

- **BASE_TIMEZONE_OFFSET –** Fetches the base time offset against GMT.

- **CURRENCY_CODE –** Preferred currency code as fetched from the user's 'Display Profile'

- **DATE_DISPLAY_FORMAT –** Preferred date display format as fetched from the user's 'Display Profile'

- **DECIMAL_SEPARATOR -** Decimal separator as fetched from the user's 'Display Profile' **(Un-used right now)**

- **DISPLAY_OPTION –** Display option set for the current MDT's MDT Type. This is not used in this framework as the screens are built responsive to deal with both Mobile and Laptop.

- **GPS_LOG_INTERVAL –** Time interval for capturing the device's current location.

- **GPS_SUPPORTED –** GPS Enabled or Disabled on the MDT Type.

- **GPS_SYNC_INTERVAL –** Logged GPS records will be synced across to the server at this time interval (in mins).

- **INITIAL_SERVICE_SCRIPT –** Not used right now but will have the initial script name to be executed. At the moment, the ouml.Config.getConfig("initScript") property.

- **IP_UPDATE_INTERVAL –** IP Address update interval. This is not used in the implementation yet as there is no server to device push communication (Only Device to server pull calls are supported).

- LOG_ARCHIVE_DAYS – Un-used – remove.

- **LOG_FILE_COUNT** - Number of active log files to keep before archival.

- **LOG_FILE_SIZE** - The maximal size in kilobytes of a log file. After the log file reaches this size, it's rolled over into a new file.

- **M1_CAPABILITY –** Stores the JSON format of all the capabilities defined on the MDT Type. Value should first be JSON Parsed before use. For using Capabilities use - ouml.Capabilities

- **MDT_ENCRYPTION_KEY –** Data encryption key is used to encrypt any transactional data on the device. The Key itself is encrypted with the user entered - user name and password.

- **MDT_LOG_LEVEL –** MDT's logging framework uses this Log level to conditionally log only selective log statements.

- **MDT_SESSION_ID –** Counter incremented each time a device is registered. This will be used for BO primary generation to ensure unique keys.

- **MONEY_DECIMAL_DIGITS –** Number of allowed decimal digits for Money fields. (Un-used right now)

- **MONEY_FORMAT –** Money format as fetched from the user's display profile. (Un-used right now)

- **NUMBER_FORMAT –** Number format as fetched from the user's display profile.

- **NUMBER_GROUP_SEPARATOR –** Number group separator symbol.

- **TIME_FORMAT –** Time format as defined in the user's display profile.

UI Layout and Navigation

The Oracle Utilities Mobile Library uses jQuery and Knockout APIs for UI Pages. Each UI page is either a single HTML file or a set of files (page fragments) combined together and displayed as one. Oracle Utilities Mobile Library uses jQuery ajax APIs to load HTML and JS content. Knockout is used to bind the JSON data to UI elements. All layout and navigation specific APIs are part of BaseViewModel class and are made available to a page specific ViewModel when it inherits the BaseViewModel.

HTML Content

Each HTML file that can be navigated by a direct link on the menu, an href in html, or via the navigateToPage API should follow standard jQuery page structure:

```
<div data-role="page" id="M1-BreakTask" >
<div data-role="header">...</div>
<div role="main" class="ui-content">...</div>
<div data-role="footer">...</div>
</div>
```

- The ID of the page div should match to the business object name if it is a BO UI otherwise it should be same as the html filename excluding the file extension.

- Each HTML file cannot have more than one div with data-role=page.

- Oracle Utilities Mobile Library uses a single page template structure of jQuery.

Headers

Header elements are automatically injected from the generic header.html on each page load. This forms the content of the jQuery Mobile page header (data-role="header");

If the SDK detects empty header DIVs with data-role="header" and only injects the header.html content, otherwise individual pages can define their own header html that remains untouched by the SDK.

It's advised to use the system headers on most screens with the following APIs to selectively show/ hide them on specific screens.

Contents

The following buttons that appear on the header (from left to right):

- **Back Button -** Displays a back button
- **Maps Button -** Toggles between the Timeline view and the MapView
- **Panic Alert -** Triggers a Panic Alert from a new UI screen
- **Indicator Bar -** Please reference the Indicators section for more information
- **Menu Bar -** Please reference to the Menu section for more information

Public APIs (via ouml.BaseViewModel)

- **showBackButton** – This is a knockout observable object. The default value is set to True. The value changes when in different UI screens.

    Home Page (Shift Start/Task List) → Invisible

    Task List Page → Shift Page → Visible

    Task List Page (Invisible) → Assignment Main Page (Visible) -> Assignment Details Section -→(Visible)

    Back button is visible when the Stack Size is > 2 (Current page occupies a place too)

    This method should be called by over-riding determinePageHeaderButtons() in your UI's ViewModel class.

    Example – model.showBackButton(false); // Would set it to false.

- **showPanicButton** – Controls the visibility of the Panic button. Default value is true. This is a KO Observable object. This button would be displayed on all the UI screens except for the Login screen.

    Example – model.showPanicButton(false); // hides it.

- **howMapButton** – Controls the visibility of the Map button. Default value is false. This button is only displayed on the Task List page in the application. Any UI requiring this method will have to toggle it ON in the determinePageHeaderButtons API.

- **determinePageHeaderButtons** – API that can be used to control the visibility of the header buttons. This API gets called even when coming out of the section pages using the back button. This is different than the ouml.ViewModel.load() method that is used for the first time initialization. This API can be optionally over-ridden by the child ViewModel UI screens.

Page Fragments

Page fragments are HTML files that cannot be directly navigated to but are combined with main HTML files and contain just individual divs but no complete page structures. Such divs are mostly hidden divs and referenced in showSection API or on default section of the page which has link to those sections embedded with showSection API. A page can override loadPageFragments method in it's viewModel to include such fragments on page load. Each fragment if present in same folder as main file can be listed as is without any folder path. However if the fragment file is in a different folder then a complete path starting from app folder should be given (e.g. m1/ui/bo/<filename>).

API

APIs are covered in the BaseViewModel API section.

Menu

One popup menu appears on each page. Each contains the following two types of menu items:

- Application Level Menu items
- Page Level Menu item.

For menu item we have one observableArray pageMenuItems in BaseViewModel. By default it will be populated with Application Level menu items once View Model is loaded, and it can be extended or appended from child ViewModel with page specific menu items.

Application Level menu items are generated from configuration file of the application and page specific menu will be implemented by the developer in the page specific models.

Menu Items (ouml.MenuItem)
This is menu item object. Create one object for each menu item.

Constructor
```
ouml.MenuItem({
    // Integer a unique id. It's also determines position of menu
item in menu.
    It's a required filed
    index;

    // Label of menu that will appear for menu item on UI. It's a
required property
    title;

    // Icon if we want an icon for menu item optional
    icon;

    // java script method or a URL optional
    Action;

    // Set active true or false if user you want to show hide the
menu item ->   It's
    required
    Active;
})
```

Menu API

Menu API is part of the Base View Model and will be available in the child view model at different application layers if the child view model extends the base view model.

API	Parameters	Description
ouml. BaseViewModel.getMenuItem	index: an Integer value	Returns a ouml.MenuItem object. This API will help write there custom API for menus. Return an object of ouml.MenuItem or **undefined var** menuItem = **this**.getMenuItem(201);
ouml. BaseViewModel.addMenuItem	menuItem : an object of ouml.MenuItms	Add menu item in the menu. This API checks for the menu item with the same index value. If menu Item exits its replace the menu items fields' value with new one otherwise add it to the list.
ouml. BaseViewModel. updateMenuItem	menuItem : an object of ouml.MenuItm	Update menu item values example: Label, action handler and visibility. This API checks for the menu item with the same index value. If menu Item exits its update the menu items fields' value with new one. Print an error log input parameter is not a valid ouml.MenuItem object.
ouml. BaseViewModel. showMenuItem	item : an object of **ouml**.MenuItem or index – index of menu item	Display a hidden menu item dynamically e.g. this.showMenuItem(201);
ouml. BaseViewModel. hideMenuItem	item : an object of **ouml**.MenuItem or index – index of menu item	Hide a visible menu item from menu dynamically. e.g. this.hideMenuItem(201);

Application/SDK level menu Items

To add application level menu item Developer/CM need to add ouml.MenuItem object in mainMenu Array List of config.js of the application.

```
var mainMenu = [
                    new ouml.MenuItem({
                        index:102, title: "Settings",
                        action: "ouml/ui/settings.html",
active:true}),
                ];
```

Page Level Menu Item

To add page level menu item Developer/CM need to override setPageMenuItems() API of ouml.BaseViewModel().

```
cmModel.prototype.setPageMenuItems = function(){
```

```
            var sample = new ouml.MenuItem({
                        index:201,
title: 'Sample Menu Item',
action: this.sample,
active:true
});

            this.addMenuItem(sample);

    }
```

Indicators

An indicator bar is displayed on the screen header as a popup. This bar displays various indicators to present device, server, network, user and activities states.

API (module - ouml.BaseViewModel)

API	Parameters	Description

ouml.Indicator.addIndicator	lookusValue | count: integer valueto represent counter if any	Add an indicator in the indicator list. Also sets a counter for this indicator.
		
		 ouml.Indicator.addIndicator('M1NCCON');
		 ouml.Indicator.addIndicator('M1NR' , 5);

ouml.Indicator.removeIndicator	lookusValue.	Removes an indicator from the indicator list. ouml.Indicator.removeIndicator('M1NCCON');

ouml.Indicator.addUpdateIndicator	lookusValue count: integer valueto represent counter if any	Description
		 Updates an indicator if it exists, otherwise adds an indicator in the same position.
		
		 ouml.Indicator.addUpdateIndicator('M1NCCON');
		 ouml.Indicator.addUpdateIndicator('M1NR' , 5);

ouml.Indicator.setCounter	lookusValue count: integer value to represent counter	Sets count value for an Indicator in Indicator list. Example: ouml.Indicator.setCounter('M1NR' , 5);
ouml.Indicator.getCounter	lookusValue	Returns count value for an Indicator in Indicator list. Example: var mailCount = ouml.Indicator.getCounter('M1NR');

Executing a single Javascript Asynchronous function is easy, however executing multiple asynchronous functions (one after another on success of previous one) requires a bit of extra code to manage the callbacks. The extra code is required to prevent recursive callbacks.

The AsyncWorker module can accept a list of functions to be executed. It returns (invokes your callback function) when the last function in the list is executed successfully or any one function fails. To be able to execute any functions using this AsyncWorker module you must follow the pattern below when executing your asynchronous functions:

Asynchronous Functions Pattern

Any method that can do ASYNC work has to accept arguments as a single obj/args ({key: value,…}) instead of fixed arguments. Please reference addBO under Business Object JavaScript (BO JS) or loadRawData under BOHelper API.

- Rransaction, input, onSuccess and onFailure are required keys on this object and will be same for all ASYNC methods.

  Real input required by business logic of the function will be part of "input" , each function can decide what should be in it.

  - onSuccess will always be called with a single argument obj (again {key: value,…})

    - transaction, output are required keys on this obj.

    - output can contain the real response that caller is expecting in as JSON.

  - onFailure will alwas be called with a single argument ouml.ClientError

  - Each such ASYNC API should have its own set of onSuccess and onFailure implementation to intercept the async response of API called by it.

    So that lower level API's async response should be first intercepted by your API and formatted in a format that caller of your API can understand

  - AsyncWorker can be used to execute N number of such methods in sequence.

    - Each method will use transaction returned by (via onSucess callback) previous method

Methods that are guaranteed to be executed SYNChronously need not follow the above approach.

AsyncWorker API

BO Plugins

new Ouml.AsyncWorker		Constructor to create an instance of AsyncWorker
addWork	args = {obj: <a reference of object on which a method will be executed>, method: <method name>, args: <arguments to be passed when executing a method on obj instance>}	Use this API to queue an async function to be executed. Function will not be executed right away, this API just collects the data required to execute a function later.
execute	args = {onSuccess:<Success callback >, onFailure:< failure callback >, transaction: <transaction to be used to execute all functions>}	This API must be called to start the execution of queued async functions. On successful execution of 1st function AsyncWorker will execute 2nd function and so on. On Failure of any function in queue the onFailure callback of this function (execute) will be invoked and error object returned by failing function will be passed as argument. On successful completion of all functions onSuccess callback of this function will be invoked.

Plugins approach of the Oracle Utilities Mobile Library is just a convention that is recommended approach to write client side equivalent of server side Enter and PostProcessing plugin scripts.

Enter and PostProcessing algorithms on a BO are written in JavaScript for execution on client. Each plugin is written as a Javascript Class attached to ouml.plugins namespace.<script-code>. This class should implement a process method. Structure of a plugin script:

```
ouml.plugins["M1-MCPTSUpd"] = (function (ouml){

var m1SendTaskUpdate = function () {

};
m1SendTaskUpdate.prototype.constructor =  m1SendTaskUpdate;
m1SendTaskUpdate.prototype.process = function(args) {
 }
 return m1SendTaskUpdate;
})(ouml);
Process method should accept an object as argument with following
atributes:
transaction-  transaction to be used for any DB operations
onSuccess – callback for successful execution of plugin
onFailure – callback for failure in execution, any exception or
error scenario

input – {bo: <BO instance reference>, action: <ADD/REPLACE>}
onFailure callback should be called with ouml.ClientError instance
```

Each BO Js should have a state transition method per state, this method should create instances of appropriate plugin scripts and invoke the process method of each. All enter plugins should be invoked

first and then BO's save method should be invoked which should decide whether to invoke bo.update or bo.add based on action on the business object (add or replace). The business object save method should execute all post processing plugins. This approach will ensure that whenever a state transition is done first Enter plugin scripts are executed and then Post processing scripts. As mentioned earlier, the Oracle Utilities Mobile Library does not enforce how a business object class should be implemented however if above approach is followed there are APIs that the Oracle Utilities Mobile Library provides which you can take advantage of and also be consistent with base product code. Please reference the AsyncWorker API section for more information.

Mobile Device APIs

The following sections describe APIs used to access various objects and modules of the mobile application.

Attachments

An attachment can be any file, such as a photo, a document with instructions or specifications, a spreadsheet, or any other supporting documentation. Mobile devices have the ability to upload attachments to the server application as well as receive attachments from the server. This processing uses restful web services.

API (module - ouml.BaseViewModel)

API	Parameters	Description
showAttachments	Pk value associated with the attachments, item id associated (optional)	Fetches and renders a list of attachments in the attachments section. The primary key and item id (optional) are used to query and load the attachments from the Database. The file API is used to query the attachment folder on the device and see if the attachment exists. If the attachment exists then open option is displayed for that attachment in the list and if it does not exist then download option is displayed in the list. It also fetches the common attachments associated with the activity.
getServerAttachment	PK value associated with the attachment, file path if file is already downloaded, attachment id , attachment filename	If the file path exists then it opens the local file on device. If filepath does not exist then a server call is made and the file is downloaded from server and stored on the device
deleteFileOnDevice	Complete file path with the file name	The file specified in the file path is deleted from the device
onPhotoDataSuccess	Image data as base64 binary	From the UI if the device camera is used to take a picture or picture is selected from gallery then this function is invoked and it gets the image data in base64 binary format as input. It in turn saves the file locally

API	Parameters	Description
createAttachmentMessage	Directory path, attachment file name, flag to indicate if its text or binary, persist flag (optional), file size	This function creates an attachment message and sends the data to the server for storage as part of the activities attachment list. If persist flag is set then it is an attachment uploaded from device to server so this function creates a record in the F1_BIZOBJ table for the attachment as well.
createTaskDirectory	Task ID	Creates a directory with the task id as the name of the directory

File

The file object is a wrapper for the Apache Cordova File. This object is used for read/write access to files residing on the device. It also has some other helper functions for file access/read/write.

API (module - ouml.File)

API	Parameters	Description
openLocalFile	url	Opens the local file from the device file system in native device viewer. The url passed in as the parameter is the complete path of the file to be opened.
base64ToArrayBuffer	base64String	Creates bytes buffer for a given base64 String
getFilesFromDirectory	Directory name, success callback and failure callback	Fetches list of files for a given directory and its sub directories on the device
createDirectoryStructure	Directory path, success callback and failure callback	Creates the structure for given directory on the device. It creates all the directories passed as part of the directory path.
writeFileData	filename, directory path for the file, success callback, failure callback, original callback, file data, appendEOF flag	Writes data to a given file on the Device. A file is created on the device file system and the data passed is written to the file. If original callback is passed it will supersede the success callback. The file url is passed as a parameter to the callback function. If appendEOF flag is passed then the data is appended to the file if the file exists.
deleteFile	Filepath including file name, success callback and failure callback	Deletes given file on the device. The file path is used to locate the file and delete it
getFileSize	Filepath including file name, success callback and failure callback	Returns the size of a given file on the device in the callback function. The file size is returned in bytes.

API	Parameters	Description
readLogsFile	filename, directory path for the file, success callback, failure callback	Reads log data from given file on the device
deleteFolder	directory path with directory name, success callback, failure callback	Deletes given folder on the device
fileFailure	error	Default file failure callback. Used if no failure callback is passed as input to the other API functions

Camera

The camera object is a wrapper for the Apache Cordova Camera plugin. This object is used for capturing a picture by opening the device picture gallery or by opening the device camera.

API (module - ouml.Camera)

API	Parameters	Description
openPictureLibrary	Callback function	Opens the device picture gallery from the User interface. User can select a picture from the picture gallery and it will be passed as a String containing the base64-encoded photo image in the callback function
openCamera	Callback function	Opens the device camera from the User interface. User can take a picture using the camera and it will be passed as a String containing the base64-encoded photo image in the callback function
onFail	Error message	Function to show failure message on Camera API failure

Maps

The Maps object is a wrapper for the Apache Cordova geolocation and also for the Javascript function to get the current device location. This object can be used to get the users current location and has helper method to get info window content.

API (module - ouml.Maps)

API	Parameters	Description

getCurrentLocation	Callback function, refresh flag	This returns the current location latitude and longitude in the callback function. The refresh flag is always false for now. If the users location cannot be determined then 0,0 is returned back.
getInfoWindowContent	Data area with activity data, callback function, latitude and longitude	This returns the info window content which needs to be displayed on click of the Activity marker on the map

Barcoding

The BaseBarCode object is a wrapper for the Apache Cordova Barcode scanner. This object can be used to get the barcode and barcode type for an item.

API (module - ouml.BaseBarcode)

API	Parameters	Description
scan	Success callback, failure callback	This returns the barcode result which has the barcode type and barcode in the success callback

Barcode Support

The application supports the following barcode types on supported Android, iOS, and Windows* devices:

- QR-Code
- Code 128
- Data Matrix ECC200 (* Android and iOS only)
- ITF-14
- UPC-A
- UPC-E

Note: All 12 numbers were returned in UPC-A testing on Android devices.

Signature

The signature object is a wrapper for the Signature plugin. This object is used for capturing a signature on the canvas and saving it as an image on the server.

API (module - ouml.Signature)

API	Description
init	Initializes the signature widget and makes it ready for use on the UI
reset	Clears the signature from the widget on the UI
getData	Gets the data for the signature image in base64 encoded format

Procedures

Procedures can be associated with shifts or activities. Procedures are checks which need to be performed during the shift start and during activity completion.

API (module - ouml.BaseViewModel)

API	Parameters	Description
showProcedures	Pk value associated with the procedures, callback function	Fetches and renders a list of procedures in the procedure section. The pk value key is used to query and load the procedures from the Database.
showProcedureDetails	Procedure ID, callback function	On click of an item in the procedure list this function is invoked. It queries the procedure based on the procedure id passed and displays the details.
onProcedureSave	Procedure ID	When the procedure is saved this function is invoked. This function validates the procedure based on the criteria set on the server in the procedure type and moves it to the passed, failed or overridden procedure state.
onProcedureCorrect	Procedure ID	Invoked when the user corrects and saves the procedure data.
onProcedureAccept	Procedure ID	This updates the database with the updated procedure details. The procedure details are also sent to the server for update.

UI Theme

The UI Theme defines the color scheme used in the mobile application. Please reference Chapter 5: Customization and Extension Methodology for information on working with UI Themes and the JQuery Mobile Theme Editor.

Logging

The Oracle Utilities Mobile Library Logging module exposes the APIs required by your implementation to facilitate system logging. Any application module that requires logging uses this module with the single log instance maintained for the complete application. Logs get the appropriate instance from **ouml.JSLogger** and use the exposed API.

For example to log an info message your implementation would use:

ouml.JSLogger.info('Your message ');

Extra public APIs exposed by this object (not part of the Oracle Utilities Mobile Library or Parent business object).

- **mdtdebug(message):** The module that needs to log a **framework level debug** message calls this method.
  Passes the log message arguments to the methods.

- **debug(message)**: The module that needs to log a **debug** message calls this method.
  Passes the log message arguments to the methods.

- **info(message)**: The module that needs to log an **info** message calls this method.
  Passes the log message arguments to the methods.

- **warn(message):** The module that needs to log a **warn** message calls this method.
  Passes the log message arguments to the methods.

- **error(message)**: The module that needs to log an **error** message calls this method.
  Passes the log message arguments to the methods.

- **perf(message)**: The module that needs to log a **perf** message calls this method.
  Passes the log message arguments to the methods

- **fatal(message)**: The module that needs to log a **fatal** message calls this method.
  Passes the log message arguments to the methods

- **setLevel(level)**: These methods set the logging level of the logger instance that the application has acquired initially. The level that is to be set should be within the set of levels supported by Logger. Else default logging level will be used

- **syncLogFile()** : This method synchronizes the log files to the server.

Error Handling

All error messages that gets displayed to user on UI must be created as Message object on server side and downloaded to client as deployment. Any error situation that occurs on client has to create an instance of ouml.ClientError. Please reference the API description below for more information.

All onFailure/error callbacks should return an instance of ouml.ClientError. The Oracle Utilities Mobile Library includes API to display an error on UI in two forms as described in API description below.

API	Parameters	Description
new ouml.ClientError	args = {msgCat:<message category>, msgId:<message id>,params:<parameters to be set on message>}	Create a new instance by passing the message category, id and parameters
ouml.ViewModel.showError	error – an instance of ouml.ClientError sectionId – optional, div ID of a section that should be displayed to show this error on	Displays an inline error message in RED color at the top of either currently displayed section or displays the section with given ID first to show the error
ouml.Notification.showAlert	ouml.ClientError – an instance of this class	Displays the closeable error on a popup at the top of current UI page.

Chapter 4

Mobile Application APIs

The following sections describes APIs used to access various objects and modules of Oracle Real-Time Scheduler Mobile Application (M1).

Inbound Scripts

Once a message is downloaded and saved to intermediate table it is handed over to inbound processing script. Below is a list of processing scripts to handle various inbound requests.

Script Name	Short Description
M1-MCPDpAsgn	Handles the inbound requests of assignment data coming in.
M1-MCPDpTask	Handles inbound requests for POU, Break, Depot Task, Depot Task Items, Non Productive Task
M1-MCPDspShf	Handles inbound requests for updates made to the shift once the shift has started.
M1-MCPMalUpd	Handles inbound requests for mail updates from the server.
M1-MCPREntUp	Handles inbound requests for updates to Related Entities including attachments.
M1-MDTSetCLL	Handles requests to dynamically increase log level on the client side.
M1-MDTSndLog	Handles requests from server to send log files of mobile device to a known location on server.
M1-MCPRclAsg	Handles M1-Assignment recall.

Plugins

Appropriate "enter" or "post processing" scripts are called on save and enter state methods of business objects.

Plugin scripts are located in the m1/scripts/plugins/plugins.js file.

 Please reference Oracle Utilities Application Framework documentation and the *Oracle Real-Time Scheduler Administration Guide* for more information on business object lifecycles and lifecycle plugins.

Images

Images specific to M1 are located in the m1/images and m1/themes/images folder.

Task List

The task list provides a listing of the crew's assigned work for the shift.

UI JavaScript

API	Short Description
setPageMenuItems	Sets the Page Menu Items for a given BO page
load	Invoked by SDK on page load. Loads the task list for the given shift id. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the data passed in case of inputData)
getTaskList	Fetch tasks from the database and update the model with the list of tasks. Depending on the tab selected, it displays either the Open tasks or the Completed tasks.
getMoreTaskList	Handles the "Show More" button being clicked and fetches the next tasks set from the database and update the model. Again depending on the tab selected, it displays either the Open tasks or the Completed tasks.
endShift	Handles the End of Shift being invoked and navigates to the End of Shift Page.
onInboundMessage	This API is used to refresh the model once any task is received from the server
showMap	This is the API triggered to showMap and is used to navigate to the appropriate map page passing in the boInstance data as input.
getDTO	Returns the business object data mapped to offline columns. This is overwritten.
setStatus	Change the status of the business object. This is overwritten.

HTML Pages

Html Page	Short Description
taskList.html	This page handles the display of all the tasks available for a given Crew Shift. It displays the tasks in either the "Open" tab or the "Completed" tab depending on the State of the task. For each task in the task list, it shows the task type Description, location associated with the task (if exists), state of the task and the Arrival date time for an Open Task and BO Status Date Time for a Completed task. It also displays an icon, if there are attachments associated with a task. The page is setup to show a pre-configured number of tasks by default. Clicking on "Show More" triggers the next set of tasks to be retrieved.
endOfShift.html	When User requests end of shift, this page has UI sections to display Vehicle information and enter the start and end odometer. It displays information showing that the activities that have not been marked to keep for a future shift will be rescheduled to any crew. It also shows the Complete button to get confirmation from user to complete the shift.

Html Page	Short Description
mailInbox.html	This page creates UI sections for mail inbox showing list of all received and sent mails. This UI has two tabs – a) Received: this shows list of all messages received by user. b) Sent: this shows list of all messages sent by user. User may navigate to detail view of received of sent messages from this screen. The list in this screen also supports pagination.
M1-OracleMap.html	This html contains section to show map canvas and map directions
commonAttachments. html	This page handles attachments and shows all the information associated with attachments.

This html contains UI sections to show attachments list, common attachments list, gallery to open picture library and camera to be able to take pictures. |

Panic Alert

Panic Alerts are generated when a crew in distress presses the Panic button on their mobile device.

UI JavaScript

API	Short Description
setPageButtons	Overrides the base API that controls the Page Buttons. It shows the Send and Back buttons.
load	Invoked by SDK on page load. Starts and stop the counter and indicates to the user if the device is offline and cannot send the alert.
sendPanicAlert	Handles the sending of the Panic Alert to the server.
showMap	This is the API triggered to showMap and is used to navigate to the appropriate map page passing in the boInstance data as input.

HTML Pages

Html Page	Short Description
panicAlert.html	This page handles the display of Panic Alert indicator.
It shows the countdown in seconds indicating when the Request for Help Will be Sent.
If the countdown reaches 0, the panic alert is automatically sent to server.
This page also shows the Send button and Back button.
The Send button is used to force Send the panic alert to Server at any time.

Assignments

An assignment is a copy of an activity created by the system to track the assigned crew's progress in accomplishing the work.

Business Object: M1-Assignment

Business Object JavaScript

API	Short Description
getDTO	Returns the business object data mapped to offline columns.
Save	Handles all the post processing plugin scripts called after each state
enterState	Changes the state of BO. Depending on the state of the BO, one of the enterState methods below are invoked.For example enterSateENROUTE is invoked when bo state of Enroute is entered
enterStateENROUTE	Handles the enter state Enroute - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateARRIVED	Handles the enter state Arrived -invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateBREAK	Handles the enter state Break - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateONSITE	Handles the enter state Start/On Site - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateRETURNED	Handles the enter state Returned- invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStatePOSTPONED	Handles the enter state Postponed - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateSUSPENDED	Handles the enter state Suspended - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateCOMPLETED	Handles the enter state Completed - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStatePendingDispatch	Handles the enter state Pending Dispatch- invokes the appropriate enter plugin scripts and at the end calls the save to invoke post-processing scripts method.

Business Object UI JavaScript

API	Short Description
setPageButtons	Overrides the base API that controls the Page Buttons of various states of the BO Lifecycle.
setPageMenuItems	Sets the Page Menu Items for a given BO page

API	Short Description
loadPageFragments	Loads a list of fragments that should be added on the UI of a given page in a specific sequence: commonAssignment.html, commonProcedures.html,commonAttachments.html
load	Invoked by SDK on page load. Loads the task data depending on the input task id passed in. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the data passed in case of inputData)
enterState	Triggered by change of BO State on UI. Depending on the BO State clicked, one of the methods below are invoked.For example ENROUTE is invoked when bo state of Enroute is clicked on.
ENROUTE	Handles the Enroute button being clicked. It is invoked by the enterState of BO UI JS method above. Appropriate validations of various UI fields if required are performed and at the end of it invoke the enterStateAPI of the BO JS passing in the input of state and action.
ARRIVED	Handles the Arrived button being clicked. It is invoked by the enterState of BO UI JS method above. Appropriate validations of various UI fields if required are performed and at the end of it invoke the enterStateAPI of the BO JS passing in the input of state and action.
ONSITE	Handles the Start button being clicked. It is invoked by the enterState of BO UI JS method above. Appropriate validations of various UI fields if required are performed and at the end of it invoke the enterStateAPI of the BO JS passing in the input of state and action.
RETURNED	Handles the Decline button being clicked. It is invoked by the enterState of BO UI JS method above. Appropriate validations of various UI fields if required are performed and at the end of it invoke the enterStateAPI of the BO JS passing in the input of state and action.
POSTPONED	Handles the Decline button being clicked. It is invoked by the enterState of BO UI JS method above. Appropriate validations of various UI fields if required are performed and at the end of it invoke the enterStateAPI of the BO JS passing in the input of state and action.
SUSPENDED	Handles the Suspend button being clicked. It is invoked by the enterState of BO UI JS method above. Appropriate validations of various UI fields if required are performed and at the end of it invoke the enterStateAPI of the BO JS passing in the input of state and action.
COMPLETED	Handles the Complete button being clicked. It is invoked by the enterState of BO UI JS method above. Appropriate validations of various UI fields if required are performed and at the end of it invoke the enterStateAPI of the BO JS passing in the input of state and action.

API	Short Description
SENDNOW	Handles the Send Now button being clicked. It is invoked by the enterState of BO UI JS method above. Appropriate validations of various UI fields if required are performed and at the end of it invoke the enterStateAPI of the BO JS passing in the input of state and action.
getStatusReasons	Get the status reasons available for the BO
showAddRemarkTypeSection	API triggered when user shows selected remark types
setSelectedRemarkType	API triggered when user needs to add selected remark types
deleteSelectedRemarkType	API triggered when user needs to delete selected remark types
saveStateFields	This API is triggered when user tries to save the UI fields populated for the RETURNED, ARRIVED, SUSPENDED, POSTPONED states
saveCompCompl	This API is triggered when user tries to save the UI fields populated for Common Completion Information section
showMap	This is the API triggered to showMap and is used to navigate to the appropriate map page passing in the boInstance data as input.

HTML Pages

Html Page	Short Description
M1-Assignment.html	This page handles the display of all the Assignment information. This includes showing the buttons associated with the lifecycle of the BO. Upon completion of Assignment, user is navigated to the taskList.html page.
commonAssignment.html	This html contains UI sections to display the Account Information, Customer Information, Scheduling Information and Common Completion Information, section to handle specific states such as declined, Postponed, Suspended and Arrived.
commonProcedures.html	This html contains 2 UI sections to display: procedure list and procedure details once a particular procedure is selected from the list.
commonAttachments.html	This html contains UI sections to show attachments list, common attachments list, Gallery to open picture library and camera to be able t take pictures.
M1-OracleMap.html	This html contains section to show map canvas and map directions

Page Menu Items
- Attachments – Only on Device
- Procedures
- Maps

Depot Related Assignment

Depot related assignments provide a copy of an activity created by the system to track the assigned crew's progress in accomplishing work specifically related to picking up and dropping off goods.

- Business Object: M1-DepotRelatedAssignment
- Parent Business Object: M1-Assignment

Business Object JavaScript

API	Short Description
getDTO	Returns the business object data mapped to offline columns.

Business Object UI JavaScript

API	Short Description
setPageMenuItems	Sets the Page Menu Items for a given BO page
load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the passed data in case of inputData)
showItemAssignment	Handles the click of a specific item from a list of items for a given assignment. If the item is a package item then it shows the lists of items under it. If the item is a single item, then it shows the item details.
showPackageItemAssignment	Handles the click of a specific item from a list of items within a given package. When the item is clicked it shows the item details.
showItems	Handles the population of all items that are available for a given assignment
getItemsList	This API takes the input of "DELIVERED" or "NOTDELIVERED" and populates the tab of Delivered /Not Delivered items for a given assignment
showPackageItems	Handles the population of all items that are available for a given package item within the list of items in an assignment
getPackageItemsList	This API takes the input of "DELIVERED" or "NOTDELIVERED" and populates the tab of Delivered /Not Delivered items for a given package item within the list of items in an assignment
scanAnyItem	This API is used to handle the Scan Any feature available on the Item list page as well as from the page menu. The Scan Any button/menu item only shows up if there are any items within the item list with a barcode and it is a device and the device has the scanning capability.
scanNext	Handles the Scan Next button once the first item has been scanned. This feature is only available when there are more items in the item list with a barcode and it is a device and the device has the scanning capability.

API	Short Description
scanItemDetail	This API is used to handle the scanning of item once a specific item has been selected. The API is invoked when the Scan Button is clicked from the Item Details page. The Scan button shows up only if the item has a barcode and it is a device and the device has the scanning capability.
scanPackageItemDetail	This API is used to handle the scanning of item once a specific item has been selected within a package. The API is invoked when the Scan Button is clicked from the Item Details page within a package item list. The Scan button shows up only if the item has a barcode and it is a device and the device has the scanning capability.
scanItemInAnchor	This API is used to handle the scanning of item once a specific item has been selected. The API is invoked when the scan icon is clicked on a particular item on the item list page. The icon shows up only if the item has a barcode and it is a device and the device has the scanning capability.
scanPackageItemInAnchor	This API is used to handle the scanning of item once a specific item within a package has been selected. The API is invoked when the scan icon is clicked on a particular item on the item list page within the package. The icon shows up only if the item has a barcode and it is a device and the device has the scanning capability.
saveCustAcceptance	This API is used to save the Customer Acceptance when the check icon is clicked. This includes the receipt option and signature acceptance.
saveItemCompl	This API is used to save the Item Completion Details once the save/check icon is clicked.
savePackageItemCompl	This API is used to save item Completion Details within a Package once the save/check icon is clicked.
markAllItems	This API takes "DELIVERED" or "DECLINED" as possible inputs and marks all the items that have not yet been delivered/declined with the input.
updateDeliveryStatus	Handles the checkbox of each item in the item list page, if checkbox is checked then the item is marked as delivered.
updatePackageDeliveryStatus	Handles the checkbox of each package item in the package item list page, if checkbox is checked then the item in the package item list is marked as delivered.
showItemLevelAttachments	Handles the attachments at item level. It passes in the task id and item id as inputs to the showAttachments API
ENROUTE	Overrides the parent M1-Assignment BO UI JS. It invokes the parent BO's ENROUTE BO UI JS to change state and then show/hide item/signature specific menu items.
ARRIVED	Overrides the parent M1-Assignment BO UI JS. It invokes the parent BO's ENROUTE BO UI JS to change state and then show/hide item/signature specific menu items.
ONSITE	Overrides the parent M1-Assignment BO UI JS. It invokes the parent BO's ENROUTE BO UI JS to change state and then show/hide item/signature specific menu items.

API	Short Description
POSTPONED	Overrides the parent M1-Assignment BO UI JS. It invokes the parent BO's ENROUTE BO UI JS to change state and then show/hide item/signature specific menu items.
SUSPENDED	Overrides the parent M1-Assignment BO UI JS. It invokes the parent BO's ENROUTE BO UI JS to change state and then show/hide item/signature specific menu items.
COMPLETED	Overrides the parent M1-Assignment BO UI JS. It does item specific/customer acceptance validations and then invokes the parent BO's COMPLETED BO UI JS to change.
showMoreItems	By default the number of items displayed in the Item Information section is restricted to the page size configuration parameter. Handles showing of additional items if present when Show More is clicked.
showMap	This is the API triggered to showMap and is used to navigate to the appropriate map page passing in the boInstance data as input.

HTML Pages

Html Page	Short Description
M1-DepotRelatedAssignment.html	This page handles the display of all the Depot Related Assignment information. Since the parent BO of M1-DepotRelatedAssignment is M1-Assignment, it inherits the html pages in M1-Assignment including commonAssignment.html, commonProcedures.html, commonAttachments.html.

This html shows Activity Information, Customer Information, Scheduling Information, Item Information, Common Completion Information, and Customer Acceptance Information.

In addition it has UI sections to display Item List, Item Details, Package Items list and Package Item details, Customer Acceptance and Scan Section.

When displaying the Item List or Package Item List, the user can choose between showing "All", "Delivered" and "Not Delivered" tabs.

Upon completion of Assignment, user is navigated to the taskList.html page. |
| M1-OracleMap.html | This html contains section to show map canvas and map directions. |

Page Menu Items
- Attachments – Only on Device
- Procedures
- Maps

- Customer Acceptance – Only on Device

- Deliver All Items – Appears only after assignment status is OnSite

- Deliver All Items – Appears only after assignment status is OnSite

- Scan Any – Appears only if it is a device and bar coding capability is enabled and there are items in the assignment with a barcode.

Depot Task

Depot tasks represent a scheduled visit to a depot, with a task duration for loading or off-loading the goods.

Note: A UI HTML Pages is not supported for M1-DepotTask. The below mentioned Business Object JavaScript and Business Object UI JavaScript are specified as they may be inherited/used by child BO M1-DepotTaskItems.

Business Object: : M1-DepotTask

Business Object JavaScript

API	Short Description
getDTO	Returns the business object data mapped to offline columns.
save	Handles all the post processing plugin scripts called after each state
enterState	Change the state of BO. Depending on the state of the BO, one of the enterState methods below are invoked.For example enterSateENROUTE is invoked when bo state of Enroute is entered
enterStateENROUTE	Handles the enter state Enroute - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateARRIVED	Handles the enter state Arrived -invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateONSITE	Handles the enter state Start/On Site - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateCANCELED	Handles the enter state Canceled- invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateCOMPLETED	Handles the enter state Completed - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStatePendingDispatch	Handles the enter state Pending Dispatch- invokes the appropriate enter plugin scripts and at the end calls the save to invoke post-processing scripts method.

Business Object UI JavaScript

API	Short Description

load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the passed data in case of inputData)
enterState	Triggered by change of BO State on UI. It invokes the enterStateAPI of the BO JS passing in the input of state and action.

Depot Task Items

- Business Object: M1-DepotTaskItems

- Parent Business Object: M1-DepotTask

Parent Business Object JavaScript

API	Short Description
enterStateCOMPLETED	Handles the enter state Completed - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts. Override this state transition from parent BO as there is a need to execute additional plugin.

Business Object UI JavaScript

API	Short Description
load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the passed data in case of inputData)
getTaskList	This API is used to populate the task list (Depot Related Assignments) within the depot task. Depending on the input tab selected, it shows the "All" (default),"Loaded" and "Not Loaded" tasks.
getMoreTaskList	By default, only specific number of tasks(configurable) in the task list are shown. If there are more than the configured number of tasks then the "Show More" button is shown and Handles it.
setPageMenuItems	Sets the Page Menu Items for a given BO page
ENROUTE	Handles the Enroute button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action
ARRIVED	Handles the Arrive button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action
ONSITE	Handles the Start button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action
CANCELED	Handles the Cancel button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action
COMPLETED	Handles the Complete button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action
showMap	This is the API triggered to showMap and is used to navigate to the appropriate map page passing in the boInstance data as input.

HTML Pages

Html Page	Short Description
M1-DepotTaskItems.html	This page handles the display of information associated with the depot task items such as Depot Information, Scheduling Information, Activities for the given Depot including Tabs to display "All", "Loaded", "Not Loaded" activities. The "Show More" button is shown to display more than the pre-configured number of activities if they exist for a given depot task. The Status Reason section is shown when status of Arrive is chosen. This page also includes showing the buttons associated with the lifecycle of the BO.

Upon completion/return of Depot Task, user is navigated to the taskList.html page. |
| M1-OracleMap.html | This html contains section to show map canvas and map directions |

Page Menu Items
- Attachments – Only on Device

- Procedures

Depot Task Assignments

Business Object: M1-DepotTaskAssignment

Business Object Java Script

API	Short Description
getDTO	Returns the business object data mapped to offline columns.
Save	Handles all the post processing plugin scripts called after each state
enterState	Change the state of BO. Depending on the state of the BO, one of the enterState methods below are invoked.For example enterSateCOMPLETE is invoked when bo state of Complete is entered
enterStatePENDING	Handles the enter state Pending - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateRETURNED	Handles the enter state Returned- invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateCOMPLETE	Handles the enter state Complete - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStatePendingDispatch	Handles the enter state Pending Dispatch - invokes the appropriate enter plugin scripts and at the end calls the save to invoke post-processing scripts method.

Business Object UI JavaScript

API	Short Description
setPageMenuItems	Sets the Page Menu Items for a given BO page
load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the passed data in case of inputData)
getStatusReasons	Get the status reasons available for the BO
getTaskTypeDesc	Gets the task type description of Depot Related Assignment and appends label "AT DEPOT" to it
showLoadingAssignment	Handles the click of a specific item from a list of items for a given loading assignment. This method loads "itemDetailsSection" section to show the item details.
saveCompl	This function is invoked on save of the Item details section. This function updates the load status,decline status, decline reason and comments to the database through the updateLoadStatus function
updateLoadStatus	This function updates the load status,decline status, decline reason and comments to the database
selectAllItems	This function is called on "Mark All as Loaded" menu item. This updates the load status of all unloaded items. Declined items are not modified.
getTaskList	This function is called on "LOADED" and "NOT LOADED" to load items belonging to that particular tab.
showActivities	This function is called on "ALL" tab
saveStatusReason	This function is called when saving Status Reason
PENDING	Resets the status Reason. Invokes the BO's enterState method with "PENDING" status
RETURNED	Checks if Status Reason is entered. If status reason is not entered, it redirects to Status Reason page. If Status Reason is populated, then invokes the BO's enterState method with "RETURNED" status. On success Control returns back to Depot Task Items screen.
COMPLETE	Checks if all the items are loaded. Partial loading is not supported. If not loaded, then a message is shown. If all the items are loaded then it invokes the BO's enterState method with "COMPLETE" status. On success control returns back to Depot Task Items screen.
declineActivity	This function is called when "RETURNED" button is clicked from the screen or "Decline Activity" menu item is selected. This function calls the RETURNED method.
completeActivity	This function is called when "COMPLETE" button is clicked from the screen or "Complete Activity" menu item is selected. This function calls the COMPLETE method
scanAnyItem	Handles the Scan Any button. This function in turn invokes ouml.Capabilities.executeCapability('M1CAPBARCODE') passing in the success method to process the barcode returned. The success method iterates over the list of items , loads the detail screen of the item whose barcode is matched and marks the load status of item found as loaded.

API	Short Description
scanNext	On successful match of the item found when scanAnyItem method was invoked, if there are any unloaded items, then "Scan Next" button is displayed on the screen to scan any unloaded items. On clicking the button, scanNext function is invoked, which in turn invokes scanAnyItem.
scanItemInDetail	This API is used to handle the scanning of the selected item in the detail screen. The API is invoked when the Scan Button is clicked from the Item Details page. If the barcode returned matches the barcode of the selected item, the item's load status is set to loaded.
scanItemInAnchor	This API is used to handle the scanning of a particular item. The API is invoked when the scan icon next to a item is clicked on the item list screen. If the barcode returned matches the barcode of the selected item, the item's load status is set to loaded.
showMoreItems	By default the number of items displayed in the Item Information section is restricted to the page size configuration parameter. Handles showing of additional items if present when Show More button is clicked

HTML Pages

Html Page	Short Description
M1-DepotTaslAssignment.html	This page handles the display of all the Depot Task Assignment information once an activity is clicked on from the Depot Task. This page has UI sections to display Item List and Item Details, Decline Section and Scan Section. When displaying the Item List, the user can choose between showing "All", "Loaded" and "Not Loaded" tabs. This also includes showing the buttons associated with the lifecycle of the BO.

Page Menu Items

Mark All as Loaded	Appears only after assignment status is Pending.
Scan Any	Appears only if it is a device and bar coding capability is enabled and there are items in the assignment with a barcode.
Complete Activity	Appears only after assignment status is Pending
Decline Activity	Appears only after assignment status is Pending

Break Task

Break tasks provide information about tasks designated for the crew taking a break.

Business Object: M1-BreakTask

Business Object JavaScript

API	Short Description
getDTO	Returns the business object data mapped to offline columns.
Save	Handles all the post processing plugin scripts called after each state
enterState	Change the state of BO. Depending on the state of the BO, one of the enterState methods below are invoked.For example enterStateCANCELED is invoked when bo state of Canceled is entered
enterStateSTARTED	Handles the enter state Started - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateCANCELED	Handles the enter state Canceled- invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateCOMPLETED	Handles the enter state Completed - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStatePendingDispatch	Handles the enter state QD-DISPATCH- invokes the appropriate enter plugin scripts and at the end calls the save to invoke post-processing scripts method.

Business Object UI JavaScript

API	Short Description
setPageMenuItems	Sets the Page Menu Items for a given BO page.
load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the passed data in case of inputData)
STARTED	Handles the **Start** button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.
COMPLETED	Handles the **Complete** button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.
CANCELED	Handles the **Cancel** button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.

HTML Pages

Html Page	Short Description
M1-BreakTask.html	This page handles the display of Break Task information. This html shows Arrival Date Time, Break Duration and BO Status. This includes showing the buttons associated with the lifecycle of the BO.

Non Productive Tasks

Non productive tasks (NPTs) indicate a window of time as well as location where the crew must attend to some task unrelated to shift work.

Business Object: M1-NonPrdTask

Business Object JavaScript

API	Short Description
getDTO	Returns the business object data mapped to offline columns
save	Handles all the post processing plugin scripts called after each state
enterState	Change the state of BO. Depending on the state of the BO, one of the enterState methods below are invoked.For example enterStateCANCELED is invoked when bo state of Canceled is entered
enterStateENROUTE	Handles the enter state Enroute - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateONSITE	Handles the enter state Start - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateCANCELED	Handles the enter state Canceled- invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateCOMPLETED	Handles the enter state Completed - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStatePendingDispatch	Handles the enter state QD-DISPATCH- invokes the appropriate enter plugin scripts and at the end calls the save to invoke post-processing scripts method.

Business Object UI JavaScript

API	Short Description
setPageMenuItems	Sets the Page Menu Items for a given BO page
load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the passed data in case of inputData)
ENROUTE	Handles the Enroute button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.

API	Short Description
ONSITE	Handles the Start button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.
COMPLETED	Handles the Complete button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.
CANCELED	Handles the Cancel button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.
showMap	This is the API triggered to showMap and is used to navigate to the appropriate map page passing in the boInstance data as input.

HTML Pages

Html Page	Short Description
M1-NonPrdTask.html	This page handles the display of Non Productive Task information. This html shows Location, Arrival Date Time, Break Duration and BO Status. This includes showing the buttons associated with the lifecycle of the BO.
M1-OracleMap.html	This html contains section to show map canvas and map directions

Page Menu Items
- Maps

Period of Unavailability Task

Period of unavailability tasks define a specific period where the crew is unavailable for shift work.

Business Object: M1-POUTask

Business Object JavaScript

API	Short Description
getDTO	Returns the business object data mapped to offline columns
save	Handles all the post processing plugin scripts called after each state
enterState	Change the state of BO. Depending on the state of the BO, one of the enterState methods below are invoked.For example enterStateCANCELED is invoked when bo state of Canceled is entered
enterStateENROUTE	Handles the enter state Enroute - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateONSITE	Handles the enter state Start - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts

API	Short Description
enterStateCANCELED	Handles the enter state Canceled- invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateCOMPLETED	Handles the enter state Completed - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStatePendingDispatch	Handles the enter state QD-DISPATCH- invokes the appropriate enter plugin scripts and at the end calls the save to invoke post-processing scripts method.

Business Object UI JavaScript

API	Short Description
setPageMenuItems	Sets the Page Menu Items for a given BO page
load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the passed data in case of inputData)
ENROUTE	Handles the Enroute button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.
ONSITE	Handles the Start button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.
COMPLETED	Handles the Complete button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.
CANCELED	Handles the Cancel button being clicked. Saves the old BO data before the state change and invokes the enterStateAPI of the BO JS passing in the input of state and action.
showMap	This is the API triggered to showMap and is used to navigate to the appropriate map page passing in the boInstance data as input.

HTML Pages

Html Page	Short Description
M1-POUTask.html	This page handles the display of all the Period Of Unavailability information. This html shows Location, Arrival Date Time, Start Date Time, Completion Date Time, Calculated Travel Time, Calculated Travel Distance and BO Status. This includes showing the buttons associated with the lifecycle of the BO.
M1-OracleMap.html	This html contains section to show map canvas and map directions

Page Menu Items
- Maps

Mail

Mail is defined as messages that can be sent to or sent from the crew from within the mobile application.

Business Object: M1-MainMail

Business Object JavaScript

API	Short Description
getDTO	Returns the BO data mapped to offline columns.
save	Handles all the post processing plugin scripts called after each state
enterState	Change the state of BO. Depending on the state of the BO, one of the enterState methods below are invoked.For example enterSateSENT is invoked when bo state of Sent is entered
enterStateSENT	Handles the enter state Sent - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateERROR	Handles the enter state Error- invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts

Business Object UI JavaScript

API	Short Description
setPageMenuItems	Sets the Page Menu Items for a given BO page
setPageButtons	Overrides the base API that controls the Page Buttons of various states of the BO Lifecycle.
load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the passed data in case of inputData)

HTML Pages

Html Page	Short Description
M1-MainMail.html	This page handles the display of all the Sent Mail information. The page contains action button to 'Resend' the same mail. User may choose to alter subject, message and priority queue while resending the same message. User may resend to the same user or group as in the original mail.

Html Page	Short Description
mailInbox.html	This page creates UI sections for mail inbox showing list of all received and sent mails. This UI has two tabs – a) Received: this shows list of all messages received by user. b) Sent: this shows list of all messages sent by user. User may navigate to detail view of received of sent messages from this screen. The list in this screen also supports pagination. This page also has a link to taskList.html in case user wants to switch to task list page.
mailComposePage.html	This page creates UI sections for drafting and sending a new mail. User may enter recipient user and group, subject, message and mention priority queue. This screen contains 'Send Mail' button which will send the mail to intended recipients.

Page Menu Items
- Compose Mail

- End Shift

Recipient Mail

Mail messages that are received by mobile application users.

Business Object: M1-RecipientMail

Business Object JavaScript

API	Short Description
getDTO	Returns the business object data mapped to offline columns.
save	Handles all the post processing plugin scripts called after each state
enterState	Change the state of BO. Depending on the state of the BO, one of the enterState methods below are invoked.For example enterStateSENT is invoked when bo state of Sent is entered
enterStateSENT	Handles the enter state Sent - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts
enterStateACKNOWLEDGED	Handles the enter state Acknowledged - invokes the appropriate enter plugin scripts and at the end calls the save method to invoke post-processing scripts

Business Object UI JavaScript

API	Short Description
setPageButtons	Overrides the base API that controls the Page Buttons of various states of the BO Lifecycle.

setPageMenuItems	Sets the Page Menu Items for a given BO page
load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the passed data in case of inputData)

HTML Pages

Html Page	Short Description
M1-RecipientMail.html	This page handles the display of Received mail. This UI contains action buttons to 'Reply', 'Delete' and 'Acknowledge' the mail. 'Acknowledge' button will appear only if the mail is marked for acknowledgement required.
mailInbox.html	This page creates UI sections for mail inbox showing list of all received and sent mails. This UI has two tabs – a) Received: this shows list of all messages received by user. b) Sent: this shows list of all messages sent by user. User may navigate to detail view of received of sent messages from this screen. The list in this screen also supports pagination.
mailComposePage.html	This page creates UI sections for drafting and sending a new mail. User may enter recipient user and group, subject, message and mention priority queue. This screen contains 'Send Mail' button which will send the mail to intended recipients.
mailReplyPage.html	This page creates UI sections for replying to a received mail. This page contains action button for 'Send Mail' which will send the mail in reply to the received mail. The field for 'Send To' is pre-populated with the senders ID, other fields are editable that user can alter before sending the reply.

Page Menu Items

- Compose Mail
- End Shift

Crew Shift

A crew shift is a planned period of time in which a crew (one or more mobile workers and vehicles) is scheduled to perform work.

Business Object: M1-CrewShift

Business Object JavaScript

API	Short Description
getDTO	Returns the business object data mapped to offline columns
startShift	Before initiating the Starting the Shift checkin process, this function will check and delete if any preview tasks downloaded to MDT.
previewShift	Get the preview shift details from server.

Business Object UI JavaScript

API	Short Description
setPageButtons	Overrides the base API that controls the Page Buttons of various states of the BO Lifecycle.
setPageMenuItems	Sets the Page Menu Items for a given BO page
load	Invoked by SDK on page load. Page Developer to read the inputArgs (or inputData) to load the offline data (or render the data passed in case of inputData)
loadPrimaryFunctions	This API forms the list of Service Classes to be displayed under the Primary Function section
showWorkerSection	This API sets selectedWorker value with current row data and make shiftWorkerSection visible
showAddEditWorkerSection	This API resets the selectedWorker and make addEditWorkerSection visible.
addWorker	This API first deletes the selectedWorker from resourceAllocationList if it already exists and then adds the updated selectedWorker
editWorkerDetails	This API is used to edit worker details once a specific worker has been selected from the resourceAllocationList.
editWorker	This API is the handler for showWorkerSection. It displays addEditWorkerSection with current selected data populated on UI.
removeWorker	Handles the deletion of currently selected worker from the resourceAllocationList
showVehicleSection	This API sets selectedVehicle value with current row data and make shiftVehicleSection visible
showAddEditVehicleSection	This API resets the selectedVehicle and make addEditVehicleSection visible.
addVehicle	This API first deletes the selectedVehicle from resourceAllocationList if it exists and then adds the updated selectedVehicle
editVehicleDetails	This method first deletes the selectedVehicle from resourceAllocationList if exist and then adds the updated information of selectedVehicle
removeVehicle	This API deletes the currently selected Vehicle from the resourceAllocationList
editVehicle	This API is the handler for showVehicleSection. It displays addEditVehicleSection with currently selected data populated on UI.
start	This API starts the Shift
logoff	This API does logoff of the Shift
setSelectedShiftFunction	This API sets the service class/Primary function of the shift.
preview	Handles the preview mode of Shift.
endShift	Handles the end of shift.
loadPageFragments	
showShiftProcedures	Handles the showing of Procedures that exist for a given shift.

API	Short Description
showServiceState	This API shows the service status possible if the current shift status is "In Service" or "Out of Service"
setServiceStatus	This API sets the service status to "In Service" if it is currently "Out of Service" and vice versa.
setShiftStatus	This API sets the shift status to whatever is passed in as input.
setStatusReasonFunction	This API is used to set the input status reason code and description.

HTML Pages

Html Page	Short Description
M1-CrewShift.html	This page handles the crew shift page and shows all the information associated with the Crew Shift. This html shows Crew Name, Shift Planned Date Time, Shift Planned End Date Time, Primary Function Section, Worker Section, Vehicle Section. User has the ability to change primary function. User has the ability to add, edit and remove worker or vehicle information from the crew shift. User has the ability to go out of service on the shift by entering the Status Reason and Estimated duration. In addition it has UI sections to display and complete Procedures associated with the Crew Shift. This also includes showing the buttons associated with the lifecycle of the BO. Upon start of Shift, user is automatically navigated to the taskList.html page.
taskList.html	Once the shift is successfully Started, the task list page is invoked. This page handles the display of all the tasks available for a given Crew Shift. It displays the tasks in either the "Open" tab or the "Completed" tab depending on the State of the task. For each task in the task list, it shows the task type Description, location associated with the task (if exists), state of the task and the Arrival date time for an Open Task and BO Status Date Time for a Completed task. It also displays an icon, if there are attachments associated with a task. The page is setup to show a pre-configured number of tasks by default. Clicking on "Show More" triggers the next set of tasks to be retrieved.
endOfShift.html	When User requests end of shift, this page has UI sections to display Vehicle information and enter the start and end odometer. It displays information showing that the activities that have not been marked to keep for a future shift will be rescheduled to any crew. It also shows the Complete button to get confirmation from user to complete the shift.

Page Menu Items

Start Shift	Shown only when Shift is in Planned State
Preview Shift	Shown only when Shift is in Planned status and "advancedDispatchTasks" is set on model.
End Shift	Shown after Shift is started
Procedures	
Change Service State	Shown after Shift is started

Depot Related Shift

A depot related shift is a planned period of time in which a crew (one or more mobile workers and vehicles) is scheduled to perform work specifically related to the pick up or drop off of goods.

- Business Object: M1-DepotRelatedShift

- Parent Business Object JavaScript – M1-CrewShift

Business Object UI JavaScript

API	Short Description
loadPageFragments	This API return a list of fragments that should be added in same sequence – commonShift.html

HTML Pages

Html Page	Short Description
commonShift.html	This page handles the crew shift page and shows all the information associated with the Depot Related Crew Shift. This html shows Crew Name, Shift Planned Date Time, Shift Planned End Date Time, Primary Function Section, Worker Section, Vehicle Section. User has the ability to change primary function. User has the ability to add, edit and remove worker or vehicle information from the crew shift. User has the ability to go out of service on the shift by entering the Status Reason and Estimated duration. In addition it has UI sections to display and complete Procedures associated with the Crew Shift. This also includes showing the buttons associated with the lifecycle of the BO.

Simple Procedure

Procedures represent steps or tasks that crews may need to complete prior to starting a business activity such as starting their shift, using a vehicle or starting work on an activity.

Business Object: M1-SimpleProcedure

Business Object JavaScript

API	Short Description
getDTO	Returns the BO data mapped to offline columns
setStatus	Change the status of the business object.

HTML Pages

Html Page	Short Description
commonProcedures.html	This page handles the Procedures and shows all the information associated with Procedures both at the activity left as well as the shift level. This has UI sections to display the Procedure List and Procedure Details. The procedure details UI section gets the details of the procedures by invoking the procedure template specified in index.html

Procedure Type

Procedure types define procedures of a certain type.

Business Object: M1-ProcedureType

Business Object

API	Overridden	Short Description
getDTO	Yes	Returns the business object data mapped to offline columns
setStatus	No	Change the status of the business object.

Oracle Map

The Oracle Fusion Middleware MapViewer is used to render maps. The out of box solution uses Oracle MapViewer Javascript V2 HTML5 API.

UI JavaScript

API	Short Description
load	The first function which gets called on the Map page. This checks if it is an Activity specific map or for all the activities in the task list.
loadLocationData	Loads the Locations which are configured in the system. These locations are used to get location of activities without any site address like NPT and POU

API	Short Description
loadMapConfiguration	This function loads the Map configuration which is defined as a Feature Configuration on the server. The properties in the Map configuration provide things like the Oracle Mapviewer URL, default latitude and longitude etc
populateTaskTypeMap	This method loads the Task Type descriptions
renderMap	Set the map height and calls the initOracleMap function
initOracleMap	This method initializes the Oracle Mapviewer map and sets the properties like URL, Datasource and tile layers. This also plot the Crew location and the activities as markers on the maps and add information windows for these markers
getDirectionsFromServer	This function gets the directions from the server from the crew location to the activity(ies).
renderOracleMapv2	This function plots the route based on those directions returned by the server on the map.
featureClick	This function shows an information window with the Activity details when activity marker is clicked on the Map.
showDirections	This functions renders the directions coming from the server in the Directions panel.
setMapHeight	Sets the height of the map
openNativeMaps	This function makes a javascript call when the Native maps button on an activity info windows is clicked. This calls the native map application on the device and passes the activity co-ordinates as parameters.
getLegend	This function creates the legend table displayed on the map.

HTML Pages

Html Page	Short Description
M1-OracleMap.html	This html contains section to show map canvas and map directions

Attachments

An attachment can be any file, such as a photo, a document with instructions or specifications, a spreadsheet, or any other supporting documentation. Mobile devices have the ability to upload attachments to the server application as well as receive attachments from the server.

Business Object: M1-Attachment

Business Object JavaScript

API	Overridden	Short Description
getDTO	Yes	Returns the business object data mapped to offline columns.
setStatus	No	Change the status of the business object.

HTML Pages

Html Page	Short Description
commonAttachments.html	This page handles the Attachments and shows all the information associated with Attachments. This html contains UI sections to show attachments list, common Attachments list, Gallery to open picture library and camera to be able to take pictures.

The following table provides a list of the various supported attachments and the associated business object. All of these business objects have the parent business object: M1-Attachment.

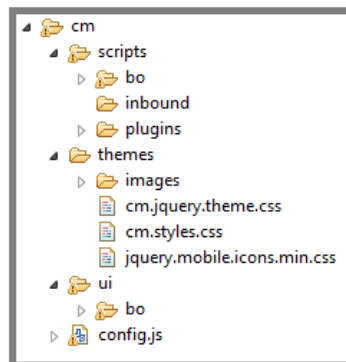| Type of Attachment	Business Object
Captured Picture Attachment	M1-CapturedPicture
Excel Attachment	M1-Excel
PDF Attachment	M1-PDF
Text File Attachment	M1-TXT
Word File Attachment	M1-Word
Captured Sound Attachment	M1-CapturedSound
MP3 Audio Attachment	M1-Mp3Audio
MPEG Video Attachment	M1-MPEGVideo
Microsoft Video Attachment	M1-MicrosoftVideo

Chapter 5

Customization and Extension Methodology

This chapter provides information on extending the Oracle Real-Time Scheduler Mobile Application.

Ensure that the cm folder is created under www before you begin customization. Please reference the *Oracle Real-Time Scheduler Mobile Application Installation and Deployment Guide* in the chapter regarding building the mobile application for information on the required files that are needed under cm.

Reference the configurable properties defined in the OUML.config.js API for the properties that can be overridden in config.js. This is described in Chapter 3: Oracle Utilities Mobile Library.

We suggest that the cm directory mirrors the directory structure of the M1 directory with specific js in each folder.

Directory	Description
www/cm/scripts	Device plugins
www/cm/config.js	Custom config.js to define configurable properties
www/cm/themes	Custom css, theme css files The following are files required under cm/themes: • cm.jquery.theme.css: custom themes • cm.styles.css: custom styles • jquery.mobile.icons.min.css: Icon definitions defined for custom theme. This also gets downloaded as part of downloading custom theme from JQuery Mobile Theme Roller

Directory	Description
www/cm/themes/images	Custom images
www/cm/ui/bo	UI related javascript, html files for a specific BO, and html files loaded as page fragments as part of the main BO html
www/cm/ui	Html and Javascript files not related to a BO. Custom common.js files to specify custom scripts and custom logic
www/cm/scripts/bo	Custom business object related java script files to handle lifecycle logic
www/cm/scripts/inbound	Custom inbound processing scripts
www/cm/scripts/plugins	Javascript files to specify custom plugin scripts

Themes and Images

The following sections describe how to change custom themes and images.

Setting Custom Themes

You can create your own custom theme using JQuery Mobile Theme Roller (http://themeroller.jquerymobile.com/).

- Oracle theme is set in www/ ouml/themes/theme-sample.css file.

- You can either import theme-sample.css file in JQuery Mobile Theme Roller and modify it or create your own.

- Make sure the custom theme downloaded from JQuery Mobile Theme Roller is named as cm.jquery.theme.css and is placed under www/cm/themes along with jquery.mobile.icons.min.css which is part of the download.

- The corresponding images are placed under www/cm/themes/images.

- The new theme should be mapped to the uiTheme property in www/cm/config.js Example : uiTheme: "f"

Changing Images on Index.html

The images on the index.html are defined as a css property as shown below.

```
/* DeskTop version */
#bannerImage {
   content:url('images/OracleBanner.png');
}

/* Mobile version */
@media all and (max-width:500px) {

    #bannerImage {
        content:url('images/OracleBanner_mobile.png');
    }
}
```

The **bannerImage** property can be overridden in www/cm/themes/cm.styles.css to reference custom images.

Changing Images of Icons on Maps

Map icons are defined as part of a configurable property called **oracleMapProperties** in the config.js file. This property can be overridden in custom config.js to define custom properties for Oracle Map including images.

Overriding Icons

You can override icon properties in www/cm/themes/cm.styles.css.

For example:
To specify custom image to paperclip icon used in Item level Attachment in Depot Related Assignment screen.

```
 .ui-icon-paperclip{
background-image: url("images/icons-png/delete-black.png");
}
```

Extending Navigation

Application Level Menu Items

Application level menu items which is shown in all the screens are defined through the **mainMenu** property in config.js.

Default value:

```
var mainMenu = [
                  new ouml.MenuItem({ index:102, title: "Settings",
action: "ouml/ui/settings.html", active:true}),
                  new ouml.MenuItem({ index:101, title: 'Logoff',
action: function(){ouml.Utilities.logout();}, active:true})
            ];
```
This property can be extended in www/cm/config.js to add new application level menu items.

```
var mainMenu = [
                  new ouml.MenuItem({ index:104, title: 'Contact
Us', action: "CM/ui/ContactUs.html", active:true})
            ];
```
Application level menu items can be removed from individual pages by overriding setPageMenuItems in each page JS. Please reference the example in the next section for more information.

Page Level Menu Items

Menu items for a particular screen are set in the UI javascript by overriding setPageMenuItems function.

In this example we will hide the "Decline Activity" menu option and introduce a new menu item called "New Function" in the M1-DepotTaskAssignment screen.

1. Create a js file under www/cm/ui/bo/M1-DepotTaskAssignment.js.

```
ouml.ViewModel["M1-DepotTaskAssignment"].CM = (function(ouml) {
var model = undefined;

 function cmDepotTaskAssignment() {
 model = this;
 ouml.ViewModel["M1-DepotTaskAssignment"].call(this);
   };

    //set the prototype to parent BO, so we extend the parent's
functions.
```

```
        cmDepotTaskAssignment.prototype =
Object.create(ouml.ViewModel["M1-
DepotTaskAssignment"].prototype);;


        cmDepotTaskAssignment.prototype.constructor =
cmDepotTaskAssignment;
        cmDepotTaskAssignment.prototype.newFunc = function()
        {
        //Custom code for the new function
        ouml.Utilities.log("New Function");

        }

        cmDepotTaskAssignment.prototype.setPageMenuItems = function(){

        ouml.ViewModel["M1-
DepotTaskAssignment"].prototype.setPageMenuItems.call(this);
            // To hide menu item
        model.hideMenuItem(203);
            // To add new menu item
        var newDTMenuItem = new ouml.MenuItem({index:201,title: 'New
Function', action:model.newFunc,active:true});
          model.addMenuItem(newDTMenuItem);

        }

    return cmDepotTaskAssignment;

})(ouml);
```

2. Create a mapping for this js in www/cm/config.js mapping pageId to the js file through the **pageFiles** property.

```
var pageFiles = {
 "M1-DepotTaskAssignment": ["cm/ui/bo/M1-DepotTaskAssignment.js"]
    }
```

Extending Existing Screens and Functions

Hiding Menu Items And Overriding Functionality

Depot Related Assignments have attachments which are shown at page level on the menu and at item level within the item section. Attachments are shown based on a positive check for Cordova.

In this example we will add a capability check. The showAttachments functionality is also overriden.

1. Extend M1 DepotRelatedAssignment.js by creating it under cm/ui/bo/M1-DepotRelatedAssignment.js.

```
ouml.ViewModel["M1-DepotRelatedAssignment"].CM = (function(ouml) {
var model = undefined;

 function cmDepotRelatedAssignment() {
 model = this;

    ouml.ViewModel["M1-DepotRelatedAssignment"].call(this);
    if(enableAttachmentSupport())
      model.showAttachmentIcon(true);
```

```
else
  model.showAttachmentIcon(false);

};

function enableAttachmentSupport()
{
if(ouml.Capabilities.isCapabilitySupported('M1ATTJPEG') &&
ouml.Device.isCordova())
return true;
else
return false;
}
cmDepotRelatedAssignment.prototype =
Object.create(ouml.ViewModel["M1-
DepotRelatedAssignment"].prototype);;

cmDepotRelatedAssignment.prototype.constructor =
cmDepotRelatedAssignment;

cmDepotRelatedAssignment.prototype.setPageMenuItems =
function(){
ouml.ViewModel["M1-
DepotRelatedAssignment"].prototype.setPageMenuItems.call(this);
model.hideMenuItem(301);
var input =  {pkValue :
ouml.App.getPageContext().inputArgs['taskId']};
var attachmentsMenuItem = new ouml.MenuItem({index:301,title:
model.LABELS.M1_ATTACHMENT,
action:model.showAttachments.bind(this,input), active:true});

    if(enableAttachmentSupport())
      model.addMenuItem(attachmentsMenuItem);

 };

 cmDepotRelatedAssignment.prototype.showAttachments = function
(keys) {

// Custom Attachment logic goes here.
};

 return cmDepotRelatedAssignment;

})(ouml);
```

2. Create a mapping for this js in www/cm/config.js mapping pageId to the js file through the **pageFiles** property.

```
var pageFiles = {
"M1-DepotRelatedAssignment": ["cm/ui/bo/M1-
DepotRelatedAssignment.js"]
}
```

Extending BO Files

In this example, we will extend M1-Assignment BO js file.

1. Create M1-Assignment.js file under www/cm/scripts/bo/M1-Assignment.js.
    www/cm/scripts/bo/M1-Assignment.js.

    ```
    ouml.BusinessObject["M1-Assignment"].CM = (function (ouml){

    //define private variables and functions here (vars and
    functions that should not be accessible from anywhere else except
    this module)

    var m1CMAssign = function(data){
        this.bo = "M1-Assignment";
          //Invoke the parent BO, if any or generic BO, and pass the
    "this" reference. So that parent can use same "this" reference.
          ouml.BusinessObject["M1-Assignment"].call(this, data);
      };

      //set the prototype to parent BO, so we extend the parent's
    functions.
      m1CMAssign.prototype = Object.create(ouml.BusinessObject["M1-
    Assignment"].prototype);

      //point the constructor property to this key (in case we need to
    make use of it later)
      m1CMAssign.prototype.constructor =  m1CMAssign;


      m1CMAssign.prototype.enterStateENROUTE = function(args) {
              ouml.BusinessObject["M1-
    Assignment"].prototype.enterStateENROUTE.call(this, args);
          // Custom logic goes here

      };

      //return the reference to a BO class(function here). When you do
    "new ouml.BusinessObject["M1-Assignment"]()" it would mean new
    m1Assignment(), which is what we want.

      return m1CMAssign;
    })(ouml);
    ```

2. Map cm M1-Assignment.js to the **boFiles** property in www/cm/config.js.

    ```
    var boFiles = {
            "M1-Assignment": ["M1-Assignment.js"]
    };
    ```

Extending HTML Pages

You can hide or show html elements using jquery API's

In this example we will customize the scheduling information section in M1-Assignment and hide the completion information section.

1. Create a custom M1-Assignment.js under cm/ui/bo/M1-Assignment.

2. Create a mapping in www/cm/config.js to map the custom **M1-Assignment.js** to **M1-Assignment pageid**.

```
var pageFiles = {
 "M1-Assignment": ["cm/ui/bo/M1-Assignment.js"]
                    }
```

3. Create a html fragment www/cm/ui/bo/cmshiftimpl.html with the div element which has the same div ID as M1 Scheduling information div (section) ID=*schinfo*. Add custom html elements required with in this div.

4. Override the loadFragment method to include cmshiftimpl.html.

5. Rename the Scheduling information div section ID, "schinfo", to a different ID so that the custom div with ID schinfo takes effect.

```
www/cm/ui/bo/M1-Assignment.js
ouml.ViewModel["M1-Assignment"].CM = (function(ouml) {

var m1CMAssignmentViewModel = function(){
        ouml.ViewModel["M1-Assignment"].call(this);
    };

    m1CMAssignmentViewModel.prototype =
Object.create(ouml.ViewModel["M1-Assignment"].prototype);
    m1CMAssignmentViewModel.prototype.constructor =
m1CMAssignmentViewModel;

    m1CMAssignmentViewModel.prototype.load = function(args) {
    args.onSuccess = onLoadSuccess.bind(this, args.onSuccess);
    ouml.ViewModel["M1-Assignment"].prototype.load.call(this,
args);
    };

    m1CMAssignmentViewModel.prototype.loadPageFragments =
function() {
        //return a list of fragments that should be added in same
sequence
      //default path is the same dir which hosts the main page HTML
file
          return ["commonAssignment.html", "cm/ui/bo/
cmshifttmpl.html"];
    };

    function onLoadSuccess(onSuccess) {
    //Hide Common Completion section.
    $.mobile.activePage.find("#saveCompCompl-
href").parent("LI").hide();
    //Rename the Scheduling infromation section Id of schinfo to a
different ID, so that we can use custom section with same ID
    $.mobile.activePage.find("#schinfo").attr("id","schinfo-m1");
    onSuccess();
    }
      return m1CMAssignmentViewModel;

})(ouml);
```

Overriding M1 Plugins and Creating Custom Plugins

You can override plugins defined in www/m1/plugins/plugins.js or create custom plugins by defining custom plugin scripts in www/cm/plugins/plugins.js

In this example we will override logic in the M1-MCPSTCom script in www/cm/plugins/plugins.js.

1. In plugin.js define:

```
ouml.plugins["M1-MCPSTCom"] = (function(ouml) {
var m1SendTaskComplCall = function() {
};
m1SendTaskComplCall.prototype.constructor = m1SendTaskComplCall;

m1SendTaskComplCall.prototype.process = function(args) {
            //Custom logic goes here
};
return m1SendTaskComplCall;
})(ouml);
```

2. Define the mapping in custom config.js in the **commonJSFiles** property.

```
var commonJSFiles = ["cm/scripts/plugins/plugins.js"];
```

Custom Screens and Functions

Creating a Custom Page Not Related To a Business Object

Create the custom page not related to a BO and using Oracle Utilities Mobile Library APIs

1. Create your html and the corresponding js file under www/cm/ui/.

2. Define the mapping of the page ID and the corresponding js file in www/cm/config.js.

   Example: About.html

```
<html>
<head>
<title></title>
<meta name="viewport" content="user-scalable=no, width=device-
width">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
</head>
<body>
<div data-role="page" ID="About">
<script src="About.js"></script>
<div data-role="header" data-position="fixed"> </div>
<div>
About Home Delivery Product
</div>
</body>
</html>
```

   Example: About.js

```
ouml.ViewModel["About"] = (function(ouml) {
    var model = undefined;

function aboutUs() {
 model = this;
```

```
ouml.ViewModel["M1-Common"].call(this);

    };

    aboutUs.prototype = Object.create(ouml.ViewModel["M1-
Common"].prototype);

    aboutUs.prototype.constructor =  aboutUs;

    aboutUs.prototype.load = function (args) {
    var callbackFunc = args["onSuccess"];
        callbackFunc();
        model.pageTitle("About Us");

    };
 return aboutUs;

})(ouml);
```

Configuration Property

 set in www/cm/config.js

```
var pageFiles = {
"About":["cm/ui/About.js"]
}
```

Creating Custom Screens for a Child BO

In this scenario we will create a child BO, cm-break (which has a new field), for the parent Break BO.

1. Create a child BO on the server side with the required changes and the corresponding changes in the deployment necessary to receive the BO on the client side.

2. Create the html page and the corresponding UI js file under www/cm/ui/bo with the same name as the child BO.

3. Create a mapping in www/cm/config.js for variable boFiles:

    ```
    var boFiles = {
    "cm-break":["cm-break.js"]
            };
    ```
4. Create www/cm/ui/bo/cm-break.html and cm/ui/bo/cm-break.js.

cm-break.html to show the new field

```
<html>
<head>
<title></title>
<meta name="viewport" content="user-scalable=no, width=device-
width">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
</head>
<body>
<div data-role="page" id="cm-break">
<div data-role="header" data-position="fixed"></div>


<div data-role="content">
```

```html
<ul data-role="listview" data-inset="false" data-filter="false">
<li data-icon="false"><a>
<h2>
<span data-bind="text:LABELS.M1_ARR_TM" id="m1Arr-label">
</h2>
<aside class="ui-li-aside">
<spandata-
bind="text:getFormattedTime(scheduleDetails.arrivalDateTime)"
id="arrivalDateTime"></span>
</aside>
</a></li>
<li data-icon="false"><a>
<h2>
<span data-bind="text:LABELS.M1_DURATION_LBL"
id="m1Duration-label">
  </h2>
<aside class="ui-li-aside">
<span id="breakDuration" data-
bind="text:getFormattedTimeFromSecs(breakDuration)">
   </span>
</aside>
</a></li>
<li data-icon="false"><a>
<h2>
<span data-bind="text: LABELS.M1_TASK_STATUS_FLG"
id="m1TaskStatus-label">
</h2>
<aside class="ui-li-aside">
<span id="boStatus" data-bind="text: getBOStatusDescr(boStatus)">
</span>
</aside>
</a></li>
<li data-icon="false"><a>
 <h2>CM Field</h2>
<aside class="ui-li-aside">
<input id="boStatus2" data-bind="value: sample"></input>
</aside>
</a></li>

 </ul>
 </br>

 <div data-bind="template: {name: 'buttonTemplate'}, refresh: true"
class="ui-grid-a" id="breakTaskButtonTemplate"></div>
</div>

</div>
</body>
</html>
```

cm-break.js

```javascript
ouml.ViewModel["cm-break"] = (function(ouml) {

var model = undefined;
var boInstance = undefined;
 var app = ouml.App;

 function cmBreak() {
 model = this;
```

```
        ouml.ViewModel["M1-BreakTask"].call(this);


    };

    //set the prototype to parent BO, so we extend the parent's
functions.

    cmBreak.prototype = Object.create(ouml.ViewModel["M1-
BreakTask"].prototype);;

    cmBreak.prototype.constructor =  cmBreak;

return cmBreak;

})(ouml);
```

The js can be extended to handle any writable fields by overriding STARTED,COMPLETED and CANCELED methods and making required changes in cm/bo/cm-break.js if necessary based on the algorithms defined.

Creating Custom Screens for a New Business Object

Create BO javascript and html files for a new BO, CM-NewBO.

1. Create the html page and the corresponding UI js file under cm/ui/bo with the same name as the new BO.

2. Create a mapping in cm/config.js to map the BO name and the corresponding name of the BO js file. This mapping is defined in the **boFiles** variable.

The file content should have the structure as below for the BO:

www/cm/scripts/bo/CM-NewBO.js

```
ouml.BusinessObject["CM-NewBO"] = (function (ouml){
    var cmNewBO = function(data){
        this.bo = "CM-NewBO";
        ouml.GenericBusinessObject.call(this, data);
    };

     cmNewBO.prototype =
Object.create(ouml.GenericBusinessObject.prototype);

     cmNewBO.prototype.constructor =  cmNewBO;     return
cmBreakTask;
     cmNewBO.prototype.getDTO = function() {
// Refer to the getDTO API method for the description

    };
// Logic to process the lifecycle methods go here

})(ouml);
```

www/cm/bo/ui/CM-NewBO.js

```
ouml.ViewModel["CM-NewBO"] = (function() {

    function cmNewBO() {
        ouml.BaseViewModel.call(this);
```

```
                    model = this;


                };


            //set the prototype to parent BO, so we extend the parent's
        functions.
            cmNewBO.prototype = Object.create(ouml.ViewModel["M1-
        Common"].prototype);


            cmNewBO.prototype.constructor =  cmNewBO;


        }
```

www/cm/bo/ui/CM-NewBO.html

```html
<html>
<head>
<title></title>
<meta name="viewport" content="user-scalable=no, width=device-
width">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
/>
</head>
<body>
    <div data-role="page" id="CM-NewBO">
<!—UI elements →
</body>
</html>
```

Device Plugins

Device plugins can be overridden by specifying custom plugins under www/cm/scripts and map the
new plugin file names in www/cm/config.js.

Barcode Plugin
In this example we will override the barcode plugin with the custom Barcode plugin, CMBarcoding.js.

www/cm/scripts/CMBarcoding.js

```javascript
ouml.BaseBarcode = (function(ouml)

    // module dependency
    var util = ouml.Utilities;

    var model = function() {
};

  // Define custom plugin specific methods
    return model;

})(ouml);

ouml.BarcodeObjectFactory  = (function(ouml) {

    var apps = ouml.Config.apps;

    function getBarcodeObject(){
        var barcodeObject = new ouml.BaseBarcode();
        return barcodeObject;
    }
```

```
                                    return {
                                    getBarcodeObject : getBarcodeObject
                                    };
                        })(ouml);
```

File Plugin

In this example we will override the File plugin with a custom File plugin, CMFile.js

www/cm/scripts/CMFile.js

```
ouml.File = (function(ouml) {

var util = ouml.Utilities;
var device = ouml.Device;

var model = function() {};

// Method relate to custom file plugin goes here

ouml.FileObjectFactory  = (function(ouml) {

    var apps = ouml.Config.apps;

    function getFileObject(){

        var fileObject = new ouml.File();
        return fileObject;
    }

    // Return public method(s)
    return {
    getFileObject : getFileObject
    };
})(ouml);
```

www/cm/config.js

```
var commonJSFiles =["cm/scripts/cmBarcoding.js","cm/scripts/
cmFile.js"];
```

Custom Script for Barcode

In this example we will override the custom script for Barcode.

The barcode plugin is invoked through a script defined in **www/cm/ui/common.js**. This script is mapped in cm config.js through a **capabilitiesMapping** property:

```
var capabilitiesMapping = {
    "M1CAPBARCODE" : "CM-MCPBarCodeScan"
};
var commonJSFiles = ["cm/ui/common.js"]
```

www/cm/ui/common.js

```
ouml.ServiceScripts["CM-MCPBarCodeScan"] = (function (ouml){
    var m1BarCode = function () {

    };

    m1BarCode.prototype.constructor =  m1BarCode;

    m1BarCode.prototype.process = function(args) {
```

```
      // Custom logic to invoke methods of Barcode plugin goes here

      }

      return m1BarCode;
})(ouml);
```

Testing to Determine If the /cm Layer is Delivered

> **Important**: The URL field in this procedure is intended for
> testing only in Chrome (in No Security Mode). It should be used
> for development and testing purposes only and should *not* used
> for production purposes or for Windows 10 runtimes (these are
> built on-premise by the customers). If you don't wish to use
> either of the two runtimes/modes – then these fields are not
> required in cm/config.js. You can instead define an empty cm/
> config.js without any of these properties and continue to extend
> the product/screens etc.

In cases in which it has been determined that the product does not deliver the cm layer (nothing in cm/ * is delivered), the default MDT URL and capabilities can be tested by changing the cm/config.js file as follows:

1. Enable Refresh Deployment to get the latest deployment data to the device (even though there is transaction data in the application) or Refresh Application (to get the latest application bundle to the device).

2. Edit cm/config.js as follows (using the appropriate server and port values in place of serverURL and newPortNumber):

```
ouml.Config.CM = (function(ouml) {
  var DEFAULT_MDT_URL = "http://<serverURL>:<newPortNumber>/ouaf/
mobility";
  var capabilitiesMapping = {
      "M1REFDEP" :{scriptName:"M1-MCPRefDep",boName:""},
      "M1REFAPP":{scriptName:"M1-MCPRefApp",boName:""}
    }
  return {
    DEFAULT_MDT_URL: DEFAULT_MDT_URL,
    capabilitiesMapping: capabilitiesMapping
  };
})(ouml);
```

Customizable Indicators

We can add a new indicator and show that inside indicator bar visible at header section of the page. List of indicators shown inside indicator bar is maintained as Extendable Lookup in Oracle Utilities Application Framework. Please follow the steps below to add/hide an indicator in indicator bar.

The following examples are for demonstration purpose only and applicable for manipulation of indicators in indicator bar during state transition. These changes will not be persisted. Customization approach may differ based on actual requirement, even though the API for handling indicators in indicator bar will remain the same.

Adding a Custom Indicator

1. Login to Oracle Utilities Application Framework with a CM system user ID and navigate to list of Extendable Lookups following the path:

   Menu -> Admin Menu -> E -> Extendable Lookup

2. Search for the business object M1-MCPIndicator.

3. Select the extendable lookup from search result.
   This lookup contains indicator information for those which need to be added to indicator bar.

4. Add new indicators that you want to show in indicator bar.
   For example:

   a. Click the Add link on the Extendable Lookup Value List section.

   b. Add an indicator for showing Crew Onsite.

      The indicator code must start with 'CM', which designates these indicators as custom.

   c. Add a second (similar) indicator for showing Crew Enroute.

5. Add the new indicator icons in the path <base _dir>\m1Mobile\www\cm\images.
   The position of the indicator (with respect to the other indicators) depends on the value of: M1-MCPIndicator -> businessObjectDataArea -> position.

6. Call the API to add the indicator in the indicator list:

   ```
   ouml.Indicator.addUpdateIndicator(extendable_lookup_value);
   //e.g. ouml.Indicator.addUpdateIndicator("CMCRENR");
   ```

   The code for indicator bar customization should be added within the CM layer.

   For example, you may want to show the 'Crew Enroute' indicator when crew is working in an

Assignment and is in Enroute state. To implement that, you must first create custom BO M1-Assignment within CM layer, followed by overriding the method 'ENROUTE'.

Within the overridden ENROUTE function, invoke the API to add this indicator in indicator bar.

```
 m1CMAssignmentViewModel.prototype.ENROUTE = function () {
ouml.Indicator.addUpdateIndicator("CMCRENR");
ouml.ViewModel["M1-Assignment"].prototype.ENROUTE.call(this);
}
```

Now the indicator is added to the indicator bar.
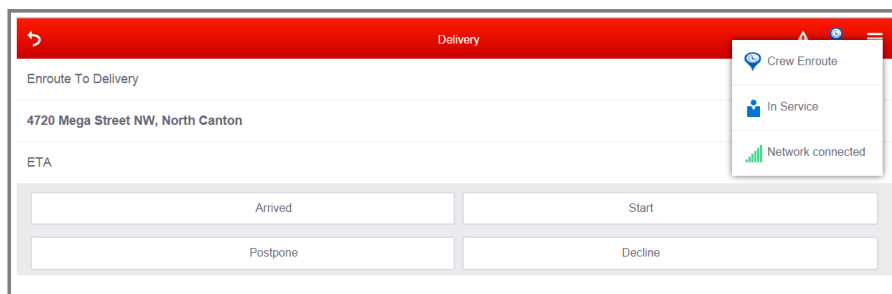
Switching Between Indicators

We can switch between indicators based on specific condition. We need to add the indicators first as specified in the section above before we can perform switch. Please note that all the indicator should have same 'position' value to enable them to switch between themselves.

In this example, we will extend the previous example to switch between the indicators for 'Crew Onsite' and 'Crew Enroute'.

To show the indicator for 'Crew Onsite', we need to override ONSITE state transition in the custom BO for M1-Assignment in the same way as we had overriden ENROUTE method in the previous section.

1. Within the overridden ONSITE function, add the API to add indicator for 'Crew Onsite'.

```
m1CMAssignmentViewModel.prototype.ONSITE = function () {
        ouml.Indicator.addUpdateIndicator("CMCRONS");
    ouml.ViewModel["M1-Assignment"].prototype.ONSITE.call(this);
        }
```

2. Since both 'Crew Enroute' and 'Crew Onsite' share same value in 'position', they will replace each other based on the state of the assignment.

Removing an Indicator

You also have the option to remove an indicator. For example, you can hide the 'Network connected / disconnected' indicator.
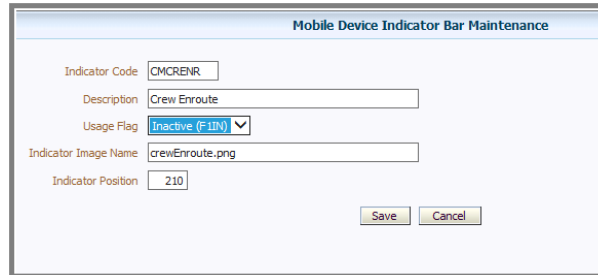
- To remove an indicator, call the API to remove an indicator from indicator bar:

```
ouml.Indicator.removeIndicator("extendable_lookup_value ")
```

- To remove the network indicator while the crew is enroute, override the ENROUTE state transition method in custom BO of M1-Assignment in CM layer.

```
m1CMAssignmentViewModel.prototype.ENROUTE = function () {
        ouml.Indicator.addUpdateIndicator("CMCRENR");
```

```
ouml.Indicator.removeIndicator("M1NCCON");
ouml.Indicator.removeIndicator("M1NCDSCN");
ouml.ViewModel["M1-
Assignment"].prototype.ENROUTE.call(this);
        }
```

- Indicators can also be removed from indicator bar by changing the 'Usage Flag' of this extendable lookup to 'Inactive'.

Mobile Device Indicator Bar Maintenance

Indicator Code	CMCRENR
Description	Crew Enroute
Usage Flag	Inactive (F1IN)
Indicator Image Name	crewEnroute.png
Indicator Position	210

Save   Cancel