Oracle Mobile Application Framework

MAF MCS Utility Developer Guide

ORACLE WHITE PAPER | AUGUST 2015

Table of Contents

Introduction

The MAF MCS Utility is an Oracle Mobile Application Framework (MAF) public sample and library that simplifies access to Oracle Mobile Cloud Services (MCS) from mobile applications built with MAF.

Using MAF MCS Utility is Simple.

Like the Oracle MCS mobile client SDKs such as those for Android and iOS, which are available as part of Oracle MCS, the MAF MCS Utility both wraps REST service requests to Oracle MCS and provides an infrastructure that transforms REST responses and request payloads into Java entities.

MAF MCS Utility is part of the public samples provided with Oracle MAF 2.1.3 and onwards in Oracle JDeveloper and Oracle Enterprise Packs for Eclipse (OEPE). The samples contain the utility sources, as well as a compiled version in a Java archive file (mafmcsutility.jar) for immediate use. In addition, and to demonstrate to developers how to use MAF MCS Utility, a MAF public sample application is provided that is a generic Oracle MCS tester that can connect to any Oracle MCS public cloud service.

This whitepaper provides and overview of MAF MCS Utility and explains MAF application developers how to use the utility in custom MAF application developments against Oracle MCS public cloud. The paper also points out where in the MAF MCS Utility public sample application you find additional code examples.

Oracle MCS Platform API

Oracle Mobile Cloud Service exposes REST interfaces for all its features, including access to user management services, data offline synchronization, storage, analytics, push notification and custom APIs.

Figure 1 shows the Oracle MCS platform APIs, highlighting those APIs that are directly accessible from mobile client applications. APIs that are not directly accessible from the mobile client application can be exposed using the Oracle MCS custom API feature.

Oracle Mobile Cloud Service exposes standard REST interfaces. Any REST aware mobile technology or framework capable of handling JSON messages thus can access Oracle MCS cloud service functionality as a client.

Since all mobile-client to Oracle MCS interaction is performed through standard REST calls, any mobile application development technology and framework capable of handling REST calls with a JSON (JavaScript Object Notation) payload can be used to build mobile applications that leverage the powers of Oracle MCS.

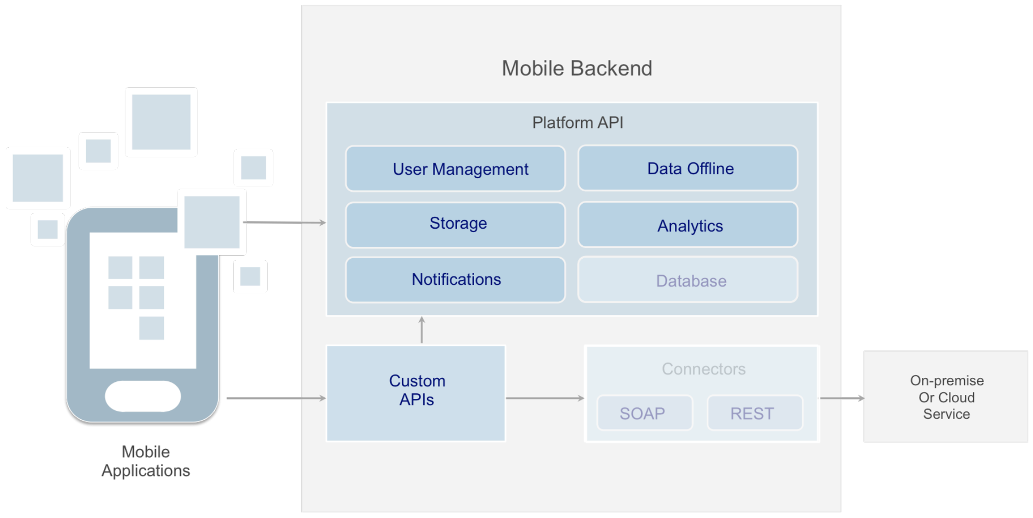


Figure 1: Oracle MCS Platform API Overview

To simplify the development of mobile client applications accessing Oracle MCS, Oracle MCS provides platform specific software development kits (SDK) that wrap REST calls to Oracle MCS REST API in native API. In its initial public release, Oracle MCS provides SDKs for Android and Objective C.

Oracle MCS does not provide an SDK for Oracle MAF. This is where MAF MCS Utility comes into play, providing functionality similar to the Android and iOS SDK to Oracle MAF application developers.

MAF MCS Utility Overview

Oracle MAF MCS Utility is a public sample library for Oracle Mobile Application Framework to easily access Oracle MCS public cloud services. REST is the communication transport mechanism used in Oracle MCS to interact with mobile clients through its exposed platform APIs. However, REST as a technology is verbose in its use and ideally is shielded from the application's developer view for them to only work within the programing language that is used by the mobile platform. Java is the programming language of the Oracle MAF mobile platform and as such MAF MCS Utility is a tool that allows MAF application developers to access Oracle MCS from Java directly.

MAF MCS Utility allows MAF mobile application developers to think in Java, not REST or raw HTTP.

To use MAF MCS Utility in custom Java code added to AMX data control or managed bean classes, MAF application developer only need to reference the MAF MCS Utility Java Archive (JAR) file within the MAF application- or view controller project.

The MAF MCS Utility is provided as a public sample in Oracle MAF and ships with source code, for MAF application developer to be able to customize and extend the utility out-of-the box behavior and functionality.

Note: By design, the initial version of MAF MCS Utility works with MAF 2.1.2 and later.

MAF MCS Utility Architecture

The MAF MCS Utility not only provides functionality similar to the Oracle MCS SDK for Android and iOS, but also implements a similar architecture. Figure 2 shows the MAF MCS Utility architecture put into context to the MAF AMX data control and managed bean classes, as well as the Oracle MCS platform REST APIs.

Oracle MAF MCS Utility matches the functionality and architecture of the Oracle MCS SDK for Android and iOS

MBE Manager – The Mobile Backend (MBE) Manager is a singleton object that manages instances of MBE objects that represent mobile backend (MCS MBE) instances configured and exposed on the Oracle MCS cloud instance.

Each MBE object is configured with the Oracle MCS MBE root URL, the MCS MBE id, the MCS MBE anonymous key, MCS MBE client application key(s) and other optional settings. To access functionality exposed on the Oracle MCS MBE, MAF application developers create a MBE Configuration object that then is passed to the MBE manager singleton to create or retrieve a MBE object instance.

Service Proxies – Each MBE object exposes methods for MAF application developers to obtain a handle to proxy objects that wrap the MAF REST call interaction to Oracle MCS for a specific platform API

- **User Management** – Allows application developers to verify user provided username and password pairs with the account information available in Oracle MCS. Further more, mobile applications have access to the authenticated user information that is stored in the Oracle public cloud realms. The information can be fully read and, based on user permission, partially updated.
- **Storage** – Allows mobile applications to upload and download opaque objects like documents or images to share with other mobile users or to save them for a specific user only. Storage saves content specific for a mobile use case, for example to save photos related to a customer service request.
- **Notifications** – Allows mobile clients to register with the Oracle MCS MBE to receive push messages from the MCS MBE instance. Messages are not sent directly from Oracle MCS but routed to the mobile device specific push message provider, like Google Messaging Cloud (GMC) or Apple Push Notification Service (APNS). For MAF applications to use the Notification Proxy Service, additional configuration steps may be required as documented in the Oracle Mobile Application Framework product documentation.

A second functionality of the Oracle MCS MBE Notifications platform services is not directly accessible to mobile clients: to queue messages in Oracle MCS for sending to GMC or APNS. To access this Oracle MCS API from mobile clients, mobile application developers need to write an Oracle MCS custom API that then is exposed on the MCS MBE for MAF applications to access (see Custom API service proxy)

- **Analytics** – Oracle MCS supports analytic events, which are used by mobile clients to send information about the mobile application health and usage, as well as business related information, like information a customer service consultant would enter to indicate whether he or she accepts a customer call and, if not, why the call was rejected. Analytic events are collected on the mobile client and sent in batches, called a session, to the MCS MBE instance where the Oracle MCS Analytics engine saves and aggregates events for graphical display in the Oracle MCS administration portal.
  The Analytics service proxy in MAF MCS Utility allows MAF application developers to create and send events to Oracle MCS. Events that fail sending due to network disruption are saved locally for later when the network becomes available.

- **Custom API** – Allows MAF application developers to invoke any REST API exposed on a MCS MBE. The feature is primarily designed for accessing custom APIs exposed in Oracle MCS but can be used with any of the platform APIs too. The difference between the custom API service proxy and other service proxies is that the JSON REST messages are not automatically serialized into Java objects but provided in their raw Java String or byte[] formats. It's the responsibility of the MAF application developer to parse the message accordingly. All that MAF MCS Utility provides in this case is the access to Oracle MCS, including sending the required Authorization headers, as well as the error handling routine.

Rest Service Adapter – MAF MCS Utility wraps the Oracle MAF Rest Service Adapter to simplify REST interaction with Oracle MCS. As part of this wrapping, Oracle MAF MCS Utility ensures authorization headers to be added to the call, as well as MBE instance specific logging that can be switched on and off dynamically at runtime. The REST Service Adapter wrapper is considered to be MAF MCS Utility implementation specific and thus not exposed to the mobile application developers. Its providing the wiggle room needed for the MAF MCS Utility to be able to adapt to future changes in Oracle MAF or Oracle MCS without impacting mobile applications that are built using the utility.

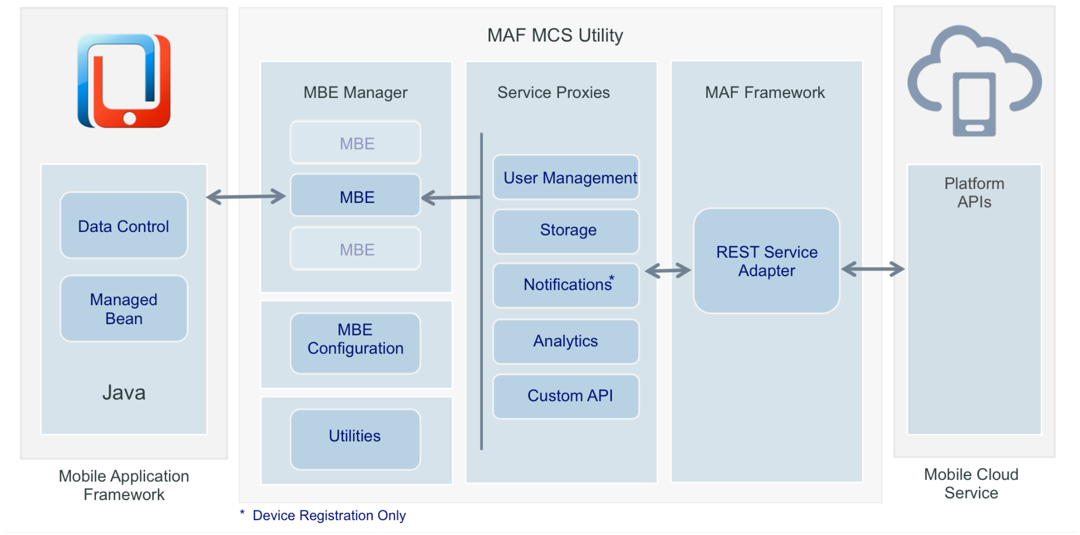


Figure 2: MAF MCS Utility Architecture

MAF MS Utility provides a set of utility classes and methods that mobile applications can use to e.g. dump the content of a hash map for logging, format dates and other related common tasks.

Note: Figure 2 shows the MAF application and the MAF MCS Utility in two separate boxes. The boxes are to show what belongs to MAF MCS Utility and what belongs to MAF. Physically MAF MCS Utility is deployed with the MAF application to the mobile devices.

Functionality provided by the Oracle MCS SDK for Android and iOS and that is not in Oracle MAF MCS Utility is offline storage and synchronization. At current MAF mobile application developers who require offline capabilities for their mobile application need to implement on-device content persistence and -synchronization manually. MAF MCS Utility still can be used in this context.

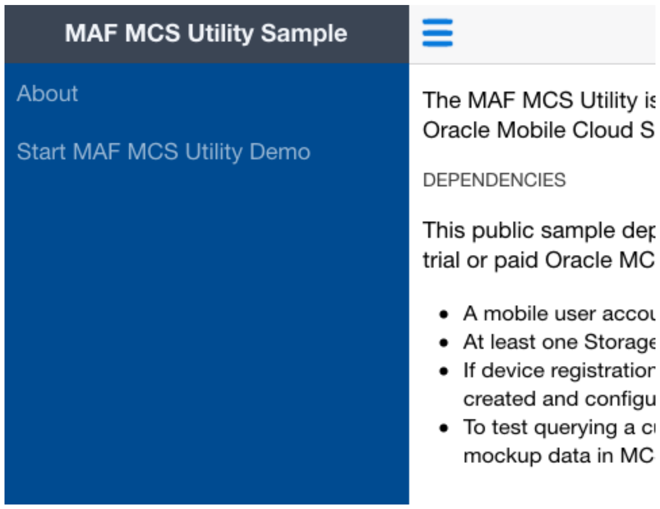
MAF MCS Utility Sample Application

MAF MCS Utility also ships with an Oracle MAF public sample that provides example code for how to use the utility to access Oracle Mobile Cloud Services. The public sample application is a generic MCS tester that has the MAF MCS Utility configured in its Application Controller project. The MAF MCS Utility functionality is exposed on a data control for developers to find all the example code in a single place.

The MAF MCS Utility public sample in Oracle MAF is a generic Oracle MCS tester that runs against any trial or paid subscription instance of Oracle MCS public cloud.

The MAF MCS Utility public sample can be executed in the Apple Simulator, the Android Emulator or deployed onto an iOS or Android device. For the best user experience it is recommended to run the MAF MCS Utility public sample on a mobile tablet or an equivalent form factor used by the Simulator or Emulator to make use of the larger screen realestate.

Figure 3 shows the initial sample application screen guides application users to pre-requisite information that needs to be provided immediately when starting the application. After reading the information, application users press the "hamburger" icon, to open a sliding menu that contains the demo launch entry.



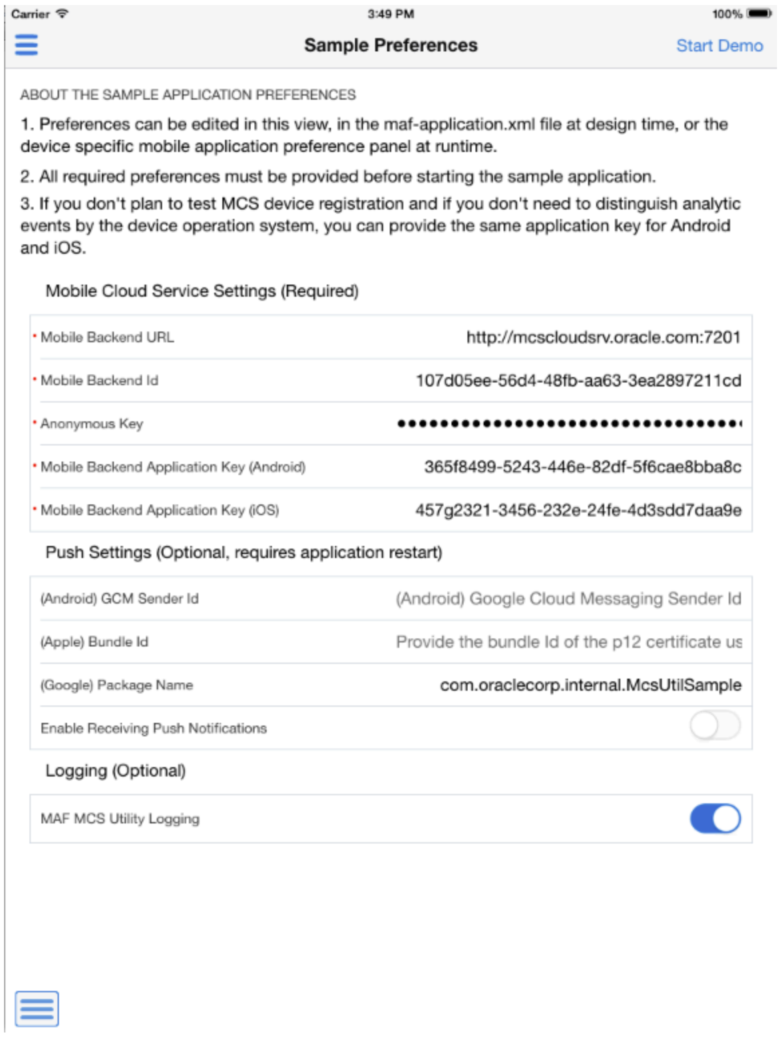
The MAF MCS Utility is
Oracle Mobile Cloud S

This public sample dep
trial or paid Oracle MC

- A mobile user accou
- At least one Storage
- If device registration created and configu
- To test querying a c mockup data in MC

Figure 3: MAF MCS Utility Sample Welcome Screen

Upon the first time the user runs the application, a preference screen shown in Figure 4 is shown for the user to provide information about the Oracle MCS mobile backend (MBE) instance to access and test via the sample application.


Carrier 3:49 PM 100%

Sample Preferences

Start Demo

ABOUT THE SAMPLE APPLICATION PREFERENCES

1. Preferences can be edited in this view, in the maf-application.xml file at design time, or the device specific mobile application preference panel at runtime.

2. All required preferences must be provided before starting the sample application.

3. If you don't plan to test MCS device registration and if you don't need to distinguish analytic events by the device operation system, you can provide the same application key for Android and iOS.

Mobile Cloud Service Settings (Required)

* Mobile Backend URL	http://mcscloudsrv.oracle.com:7201
* Mobile Backend Id	107d05ee-56d4-48fb-aa63-3ea2897211cd
* Anonymous Key	••••••••••••••••••••••••••••••••
* Mobile Backend Application Key (Android)	365f8499-5243-446e-82df-5f6cae8bba8c
* Mobile Backend Application Key (iOS)	457g2321-3456-232e-24fe-4d3sdd7daa9e

Push Settings (Optional, requires application restart)

(Android) GCM Sender Id	(Android) Google Cloud Messaging Sender Id
(Apple) Bundle Id	Provide the bundle Id of the p12 certificate us
(Google) Package Name	com.oraclecorp.internal.McsUtilSample
Enable Receiving Push Notifications	Off

Logging (Optional)

MAF MCS Utility Logging	On

Figure 4: MAF MCS Utility sample preference screen

Note: The preference settings can always be changed later by clicking the "Edit Preferences" link displayed on top of the login screen (Figure 5), or by navigating to the mobile device specific preference settings.

The login screen shown in Figure 5 is the first screen application users see when starting the sample application after setting the application preferences. Any subsequent application start will automatically omit the preference screen.

The login screen in Figure 5 allows users to authenticate against MCS with a mobile username and password, or to authenticate using the anonymous user credentials.

Note that anonymous users don't see the same menu as shown in Figure 6 but a reduced choice. Not all Oracle MCS platform APIs are available to the anonymous user and the sample application menu reflects this.

Note: Anonymous users have access to the custom API tester. However, be aware that invoking custom APIs that require an authenticated user account will fail with an error.

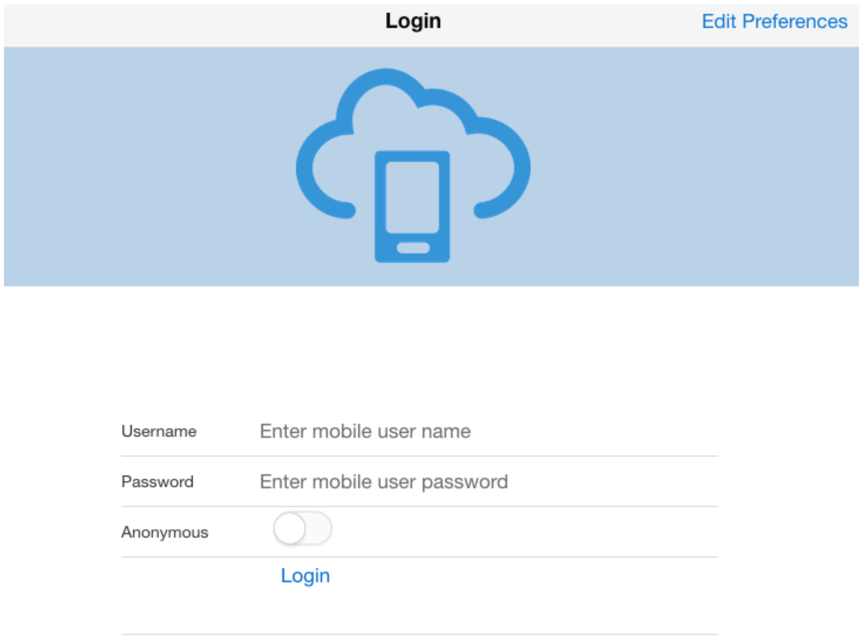


Figure 5: MAF MCS Utility sample Login

Figure 6 shows the public sample tester menu that allows users to select a specific API to test against the remote Oracle MCS MBE instance. **The Analytics API menu** item leads to a view that allows queuing analytic events that report shopping card selections to the remote MCS analytic engine.

Note**:** For sending analytic events, Oracle MAF attempts to obtain geo-location information, which however is not available on the simulator and the emulator. In this case, and only the first time the analytic event API is invoked, Oracle MAF waits for a time-out to happen after 1 minute. This then causes a delay of 1 minute between the user sending analytic events to MCS and the analytic dashboard in the MCS portal displaying the event. This is not the case for when the application is deployed to a mobile device.

The **MCS Storage API menu entry** navigates to a set of page that display a list of available collections exposed on a MCS MBE instance and that allow to upload, download and delete stored content. To demonstrate upload of content from the mobile device to Oracle MCS, the MAF sample application is deployed with example documents including PNG, PDF and MP4 files.

The **Custom API menu entry** navigates to a view in which the user provides information about the custom API URI as well as the http method to use and the (optional) JSON payload to send. The message returned from Oracle MCS is displayed in its raw format.

The **MCS Device Registration API menu entry** navigates to a view with additional instructions that inform users about pre-requisite configurations required by Oracle MAF to receive messages from Google and Apple.

Once configured, the sample application shows the client token obtained from Google or Apple, the MCS client registration confirmation and the content of incoming push message queued by Oracle MCS.

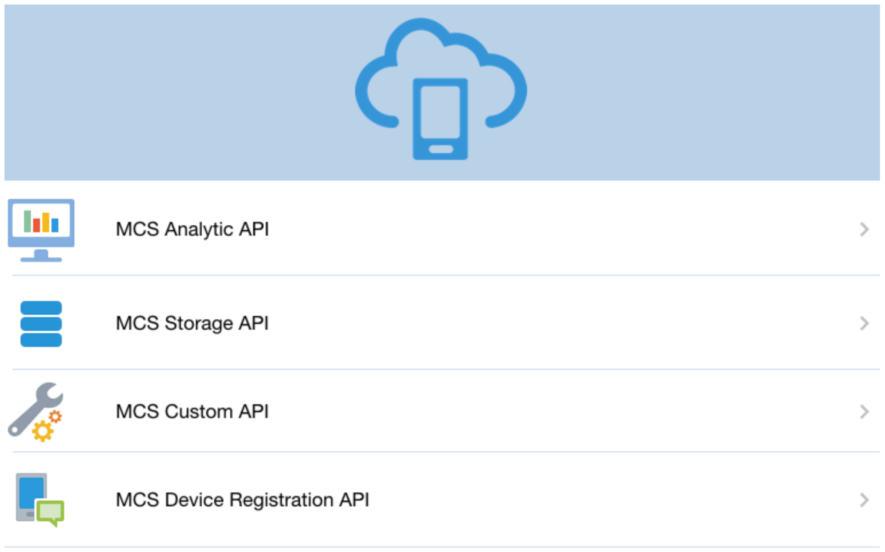
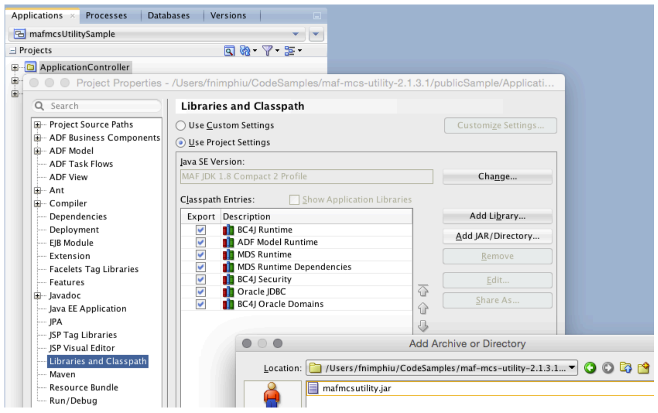


Figure 6: MAF MCS Utility Tester Menu

The MAF MCS Utility public sample provides MAF application developers with sample code, including synchronous and asynchronous invocation examples of the MAF MCS Utility APIs.

MAF MCS Utility Configuration

To use MAF MCS Utility in a custom MAF application development you need to add the mafmcsutility.jar archive file to the application's application controller and the view controller project. For MAF applications that use the default MAF project names, the application controller project name is "ApplicationController" and the view-controller project name is "ViewController".



Project Properties - /Users/fnimphiu/CodeSamples/maf-mcs-utility-2.1.3.1/publicSample/Applicati...

Libraries and Classpath

Use Custom Settings [✓ Use Project Settings]

Java SE Version: MAF JDK 1.8 Compact 2 Profile

Classpath Entries: Show Application Libraries

Export	Description
✓	BC4J Runtime
✓	ADF Model Runtime
✓	MDS Runtime
✓	MDS Runtime Dependencies
✓	BC4J Security
✓	Oracle JDBC
✓	BC4J Oracle Domains

Add Archive or Directory

Location: /Users/fnimphiu/CodeSamples/maf-mcs-utility-2.1.3.1...

mafmcsutility.jar

Figure 7: mafmcsutility.jar configuring

To add the mafmcsutility.jar file to the "ApplicationController" project, in Oracle JDeveloper, select the project node and choose "Project Properties" from the right mouse menu. In the project properties dialog, select the "Libraries and Classpath" entry and press the "Add JAR / Directory" button to find and select the mafmcsutility.jar file on your computer's file system. With the archive file selected, press "Open" followed by "OK". Figure 7 shows a screenshot of the before mentioned dialogs.

Note: Though explained for Oracle JDeveloper, MAF MCS Utility can also be configured for MAF applications developed with Oracle Enterprise for Eclipse (OEPE). In OEPE too the mafcsutilit.jar can be configured for the application- or the view controller project.

The configuration of the mafmcsutility.jar file in the "ViewController" project is similar to the configuration of the ApplicationController project shown in Figure 7. The only difference is that you choose the "Project Properties" menu entry on the view controller project node.

About MAF MCS Utility Scopes

Data controls that are configured and instantiated in the application controller project are available throughout a MAF application, like application scoped managed beans do. Data queried from Oracle MCS thus is shared across MAF features.

On the other side, data controls defined in the view controller project don't share their instances across features in MAF so that data too cannot be shared.

As usual in application development, use case matters and there is no easy right or wrong as of where to define a data control. In the case of MAF MCS Utility, if you prefer a one-to-one mapping between a feature in MAF and an Oracle MCS MBE, you create the data control on the view controller project, which then also means you configure the mafmcsutility.jar file in the ViewController project. In this case, each data control usage in a MAF feature creates its own instance of the MAF MCS Utility MBE Manager and the contained MBE instances.

If however the Oracle MCS MBE functionality, and therefore the data queried from MCS, needs to be accessed from different MAF features, you create the data control in the ApplicationController project and configure the MAF MCS Utility archive in the application controller project and optionally the ViewController project. The ViewController only needs the mafmcsutility.jar configured if MAF MCS Utility classes are exposed on the data control for use in MAF features. Note that when using this setup you need to ensure that the MBEManager instance is exposed through the data control only and not directly accessed from e.g. a managed bean in a feature.

MAF MCS Utility Configuration in the Public Sample

The MAF MCS Utility public sample application that ships with Oracle MAF 2.1.3 and later versions has the utility library configured on the ApplicationController project and uses it within the `MobileBackendDC.java` data control class. Further more, the ViewController project does not use the mafmcsutility.jar and instead the data control does expose generic objects for the results queried from Oracle MCS. If you want to avoid data type transformation, you would configure mafcsutility.jar on the ViewController project instead and parse the MAF data control results to the MAF MCS Utility types (assuming access to the data controls from managed beans). Again, there is no right or wrong implementation here and the use case defines were to implement data controls and where to configure the MAF MCS Utility library.

The MAF MCS Utility public sample is a first class resource for developers to find code examples and to read up on what the utility does

The MAF MCS Utility public sample is there to show a working example and is filled with code comments to explain how to work with Oracle MAF MCS Utility. It’s a first class resource for developers to find code examples and to read up on what the utility does. Also note that the MAF MCS Utility source code that is also made available as part of the MAF public samples contains detailed code comments.

Getting Started: The MAF MCS Utility Base Classes

The three MAF MCS Utility classes that are the starting point of any MAF application development against Oracle MCS public cloud are

- com.oracle.maf.sample.mcs.shared.mbe.MBEConfiguration
- com.oracle.maf.sample.mcs.shared.mbe.MBEManager
- com.oracle.maf.sample.mcs.shared.mbe.MBE

The following sections explain what the classes are used for and what the methods are they expose

MBEConfiguration class

The `MBEConfiguration` object needs to be created to pass initial configuration settings for the requested `MBE` instance to the `MBEManager` class. The MBEConfiguration object is copied into the MBE instance from where it can be accessed at runtime to, for example, change logging and analytic event behaviors.

Constructor

The constructor requires information about the MAF REST connection that points to the Oracle MCS MBE base URL, the unique MCS MBE ID, the MCS MBE anonymous key, the client application key for Android and iOS,and the authentication type. The first release of Oracle MCS public cloud only supports basic authentication so that the authentication type needs to be set to MBEConfiguration.AuthenticationType.BASIC_AUTH.

```
MBEConfiguration(String mafRestConnectionName, String mobileBackendId,
                 String mbeAnonymousKey, String mbeClientApplicationKey,
                 AuthenticationType authenticationType)
```

Note: Before creating the configuration object, you need to create a MAF REST connection to hold the Oracle MCS MBE base URL (similar to http://mcscloudsrv.oracle.com:7201).

The constructor throws `oracle.adfmf.framework.exception.IllegalArgumentException` exception if one of the arguments has a null value or if the value has a zero length.

Methods

The following table documents common methods MAF developers may make use of and is not the definitive list

Method Name	Description
getAppleBundleId	Returns the application bundle Id of MAF applications deployed to Apple iOS devices. The bundles Id is used to register MAF applications to MCS to receive push messages
getAuthenticatedUsername	Returns the username of the authenticated user or null if the MBE instance is not authenticated
getAuthorizationToken	The base 64 encoded authorization token that is added to each REST request to Oracle MCS for authorization
getAuthtype	Whether the MBE instance is authenticated using basic authorization or OAUTH. In the initial release of Oracle MCS, authentication is always

	basic.
getDeviceToken	If a notification is enabled, returns the device token returned by Google GMC or Apple APNS push providers. The information is used by MAF MCS Utility to register the MAF application with MCS to register push messages
getGooglePackageName	Returns the application package name of MAF applications deployed to Android devices. The package name is used to register MAF applications to MCS to receive push messages
getLogger	Provides access to the MBE specific logger to write log messages that then print in the context of the MBE instance the configuration object is used with. MBE logger output always has the MBE ID added to the log message
getMobileBackendBaseURL	Allows MAF application developers to obtain the Oracle MCS MBE base URL string that is configured in the MAF REST Connection used with a MBE. This information is useful e.g. to debug MCS connection problems
getMobileBackendClientApplicationKey	Returns the client application key defined in MCS MBE for the MBE instance the MBEConfiguration object is used with. The key usually is different for iOS and Android applications and is used with analytics and notifications
getMobileBackendId	Returns the MCS MBE id of the Mobile Backend in Oracle MCS that the MBE instance of MAF MCS Utility connects to
isEnableAnalytics	Analytics can be enabled and disabled at runtime to reduce the amount of information sent to Oracle MCS analytic engine. MAF applications can check this setting to queue analytic events or ignore analytics
isLoggingEnabled	Logging can be enabled and disabled at runtime. This method allows MAF developers to check whether log messages are written or not
isManualAuthentication	MAF MCS Utility supports manual authentication in which authentication to MCS is handled through MAF MCS Utility, as well as authentication on the MAF Feature level, in which case authentication is handled by MAF. This method allows MAF developers to tell what type of authentication is used with a specific MBE

MBEManager class

The MAF MCS Utility `MBEManager` class is a factory class and façade to start with. The `MBEManager` class creates and manages `MBE` instances based on a provided `MBEConfiguation` object. The `MBE` instance then provides Java access to all Oracle MCS platform API functionality through the service proxy classes it exposes.

Constructor

The `MBEManager` is a singleton and is called through a factory method. `MBEManager` instances in MAF are created per class loader, which means that an `MBEManager` instance defined on the application controller project is different from `MBEManager` instances defined in the view controller project where they are created for each MAF feature.

Recommended practice is to define the MAF MCS Utility reference in a MAF data control in the application controller project and always work through the data control. This way you keep the number of `MBE` instance to a minimum and also share data queried from Oracle MCS across features.

However, if the mobile application use case foresees a 1-1 mapping of a MAF features with a server side MCS MBE, you define the `MBEManager` in the view controller project either in a data control or managed bean.

```
MBEManager.getManager()
```

Methods

The following table documents common methods MAF developers may make use of and is not the definitive list

Method Name	**Description**
createOrRenewMobileBackend	Creates a new `MBE` instance based on `MBEConfiguration` object settings. If the `MBE` instance with the provided name exists, it will be dismissed and re-created with the new configuration settings. Note: To ensure unique names for the registered MBE instances you could use the MCS MBE id as the name.
existMobilBackendWithName	Allows MAF application developers to test if a MBE with the provided name already exists in the MBEManager instance.
getMobileBackend	Retrieves the `MBE` instance associated with the provided name or null if not `MBE` instance is found for the name
releaseAllMobileBackend	Releases and removes all `MBE` instances managed by this `MBEManager`
releaseNamedMobileBackend	Releases and removes a named `MBE` instance managed by this `MBEManager`

MBE class

The Mobile Backend (`MBE`) class is an access façade to a specific Oracle MCS MBE in the public cloud. The `MBE` class exposes service proxy classes (discussed later) that wrap the functionality of the Oracle MCS platform APIs.

Constructor

The `MBE` constructor is called by the `MBEManager` and should not be called by MAF application developers.

```
MBE(String mobileBackendName, MBEConfiguration mbeConfig)
```

Methods

The following table documents common methods MAF developers may make use of and is not the definitive list

Method Name	Description
getApplicationFeatureName	Gets the name of the current feature this MBE is used in. This method is of interest in use cases where there is a 1 – 1 mapping in Oracle MAF between a MAF Feature and a MCS MBE
getAuthorizationProvider	Returns an instance of the authorization provider that MAF application developer use to manually authenticate an `MBE` to the MCS MBE public cloud. In the initial release, only basic authentication is supported. A future version will also support OAUTH. `com.oracle.maf.sample.mcs.shared.authorization.auth.Authorization`
getMbeConfiguration	Returns the `MBEConfiguration` object of the `MBE` instance.
getServiceProxy	Generic method to access one of the service proxies exposed by the `MBE`. It is recommended that MAF developer use the type safe methods to access the service proxy instances.
getServiceProxyAnalytics	Returns an instance of the service proxy class for accessing the Analytics platform API in Oracle MCS
getServiceProxyStorage	Returns an instance of the service proxy class for accessing the Storage platform API in Oracle MCS
getServiceProxyUserInfo	Returns an instance of the service proxy class for accessing the User Management API platform API in Oracle MCS
getServiceProxyCustomAPI	Returns an instance of the service proxy class for accessing custom APIs defined in Oracle MCS MBE
getServiceProxyNotifications	Returns an instance of the service proxy class for accessing the Notifications platform API in Oracle MCS

Example: Creating a MAF MCS Utility MBE Instance

The code example in this section starts with creating and configuring and instance of `MBEConfiguration`. As you will, defining the configuration object is the bigger part involved in creating MAF MCS Utility access to Oracle MCS public cloud.

```
/* Create a handle to an MBE class instance */

mobileBackend = MBEManager.getManager().createOrRenewMobileBackend(
                                   mobileBackendId,mbeConfiguration);

//access the MBE specific logger instance
logger = mobileBackend.getMbeConfiguration().getLogger();
```

Though the `MBE` instance is created and managed by the `MBEManager`, it's configured through a configuration object, `MBEConfiguration`. Before creating the configuration object, you need to create a MAF REST connection to hold the Oracle MCS MBE base URL (similar to http://mcscloudsrv.oracle.com:7201). In this sample the REST connection name is assumed as " mcsmbeurl"

```
/* global variable declaration e.g. in data control

private MBE mobileBackend = null;
// If you need to write log messages for an MBE instance
private MBELogger logger = null;

…
/* Create MBEConfig configuration object */
String mobileBackendId = "107d05ee-56d4-48fb-aa63-3ea2897211cd"
String mbeAnonymousKey = "UFJJTUVfREVDRVBUSUNPTl9NT0JJTE"


// Provide the client application keys created in MCS MBE. The client application
// is a configuration in MCS MBE that is mainly used for configuring push messages
// MAF MCS Utility however needs a the key for Analytics also, even if push is not
// configured
String appKeyAndroid = "365f8499-5243-446e-82df-5f6cae8bba8c";
String appKeyiOS = "457g2321-3456-232e-24fe-4d3sdd7daa9e";


// MBE specific logging in MAF MCS Utility can be enabled / disabled at runtime
boolean loggingEnabled = false;


// Detect platform MAF runs on
DeviceManager deviceManager = DeviceManagerFactory.getDeviceManager();
String _toUppercaseManufacturerOS = deviceManager.getOs().toUpperCase();
boolean isIOS = _toUppercaseManufacturerOS.equals("IOS")?true:false;
// Assign application client key
String appKey = isIOS? appKeyiOS : appKeyAndroid;

// MBE instance need to be configured to support a specific authentication mechanism.
// At the moment only Basic Authentication is supported.

MBEConfiguration mbeConfiguration = new MBEConfiguration("mcsmbeurl", mobileBackendId,
                 mbeAnonymousKey,appKey, MBEConfiguration.AuthenticationType.BASIC_AUTH);
```

```
// Enable analytic event feature for MBE
mbeConfiguration.setEnableAnalytics(true);
// logging can be enabled / disabled at runtime for a MBE instance. Note that logging for
// An MBE requires logging to be enabled for the MAF application too.
mbeConfiguration.setLoggingEnabled(loggingEnabled);

// Try to identify the device so that analytics can distinguish between the devices owned
// by a person
mbeConfiguration.setMobileDeviceId(DeviceManagerFactory.getDeviceManager().getName());

/* Get handle to MBE instance */

mobileBackend = MBEManager.getManager().createOrRenewMobileBackend(
                                   mobileBackendId, mbeConfiguration);

//access the MBE specific logger instance
logger = mobileBackend.getMbeConfiguration().getLogger();
```

With the code above, your MAF application is ready to access Oracle MCS through the proxy services exposed on the `MBE` instance. Most of the code above is for defining the configuration for an MBE instance.

The configuration object for an MBE object is accessible at runtime in a call to `getMbeConfiguration()` on the MBE instance, allowing developers to switch functionality, like logging, on and off, or to access to the configuration settings.

Note: Changing the originally created configuration object has no impact to the MBE instance that was created from it. To change MBE configurations, you always need to access the `MBEConfiguration` through the `MBE` instance or one of its exposed service proxies.

Where to find it in the MAF MCS Utility Public Sample Application

The MAF MCS Utility public sample instantiates the MAF MCS Utility `MBE` instance in the `MobileBackendDC.java` class located in the application controller project. The `prepareMCSAccess()` method is invoked from a task flow method activity after application users provided detail information about the MCS MBE to access in the application preferences page. The sample code also shows how to configure an `MBEConfiguration` object from application preferences and includes how to override the MAF Rest connection with information from the preferences.

Working with MAF MCS Utility MBE Service Proxies

Oracle MCS MBE platform APIs are represented by service proxy classes in MAF MCS Utility. The services proxy classes not only wrap the underlying REST call to Oracle MCS but also provide infrastructure classes to handle the server response in Java. Services proxy classes included in MAF MCS Utility are

- com.oracle.maf.sample.mcs.apis.userinfo.UserInfo
- com.oracle.maf.sample.mcs.apis.analytics.Analytics
- com.oracle.maf.sample.mcs.apis.storage.Storage
- com.oracle.maf.sample.mcs.apis.custom.CustomAPI
- com.oracle.maf.sample.mcs.apis.notifications.Notifications

UserInfo Service Proxy

The `com.oracle.maf.sample.mcs.apis.userinfo.UserInfo` proxy class allows MAF applications to access the authenticated user profile information saved in the Oracle MCS security realm, including the user's custom properties. For realm attributes that are updateable, the UserInfo proxy can be used to change user information.

Constructor

MAF application developers should not call the constructor directly but access the proxy class through a `MBE` instance

```
mbe.getServiceProxyUserInfo()
```

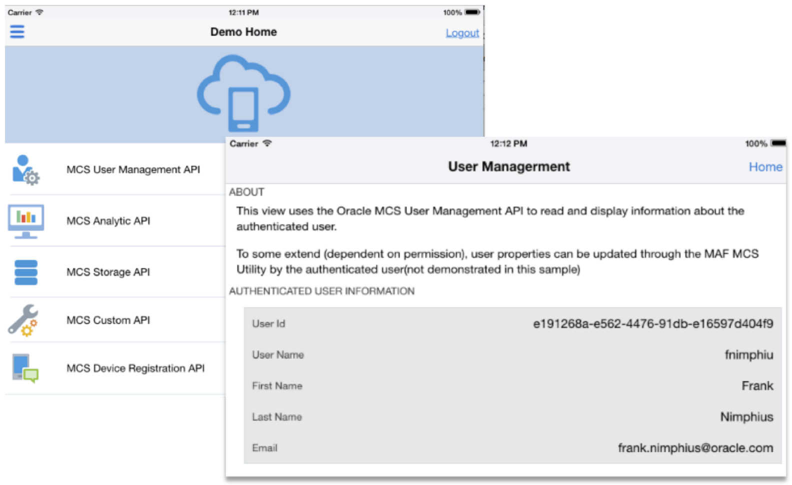
Methods

The following table documents common methods MAF developers may make use of and is not the definitive list

Method Name	Description
getCurrentUserInformation	Returns an object instance of `com.oracle.maf.sample.mcs.apis.userinfo .User` containing named user properties and a HashMap with additional properties that are custom to a specific security realm
updateCurrentUserInformation	Allows MAF application developers to update user information in the MCS security realm through a HashMap object that contains property names and values. Note that not all realm properties, for example the username, are updateable. To not run into errors its recommended to test property updates in a test tool like the MCS MBE API tester, cURL or Google Postman

Where to find it in the MAF MCS Utility Public Sample Application

The MAF MCS Utility public sample accesses the `UserInfo` proxy class from the `MobileBackendDC` data control class located in the application controller project. The getUserInformation method returns a JSON string that is accessed and parsed in the `UserManagementBacking` bean located in the view controller project. Figure 8 shows the UserManagement.amx view at runtime.



ABOUT

This view uses the Oracle MCS User Management API to read and display information about the authenticated user.

To some extend (dependent on permission), user properties can be updated through the MAF MCS Utility by the authenticated user(not demonstrated in this sample)

AUTHENTICATED USER INFORMATION

User Id	e191268a-e562-4476-91db-e16597d404f9
User Name	fnimphiu
First Name	Frank
Last Name	Nimphius
Email	frank.nimphius@oracle.com

Figure 8: UserInfo proxy in the MAF MCS Utility public sample

Example: Read user profile information

```
MBE mbe = … get MBE instance from MBEManager
UserInfo userInfo = mbe.getServiceProxyUserInfo();

User user = userInfo.getCurrentUserInformation();

//get information from User object properties or dump user information into
//JSON string. The latter is shown below
userInfoJson = user.toJSONString();
```

Analytics Service Proxy

The `com.oracle.maf.sample.mcs.apis.analytics.Analytics` proxy class allows MAF applications to queue and send custom events to the Oracle MCS analytic engine. Events are sent in the context of an analytic session, which is not the same as an application session. Analytic sessions match to a mobile task or use case that mobile project leads or administrators want to gather information about. For example, a customer service technician may use her mobile phone to accept or reject customer service requests, or close requests with information about the fix. This information may be gathered in analytic events and sent to MCS for the mobile project lead to learn about how the mobile application or the support service can be improved to operate more efficient. Another use case is to use events to report in-mobile application purchases for mobile project leads to understand how application users search for a product and which product categories are selected the most.

Constructor

MAF application developers should not call the constructor directly but access the proxy class through a `MBE` instance

```
mbe.getServiceProxyAnalytics()
```

Methods

The table below only lists those public methods that are likely to be used by MAF application developers. Those methods that are relevant for MAF internal operations are not discussed.

Method Name	Description

addCustomEvent	Adds a custom event to the event queue. If no analytic session exists it is created. The method argument is an instance of
	`com.oracle.maf.sample.mcs.apis.analytics.Event`
	Example:
	`Event customEvent =`
	`      new Event(customEventName, null, properties);`
	`analyticsProxy.addCustomEvent(customEvent);`
	The "properties" argument is a HashMap<String, String> of custom name value pairs that describe event content.

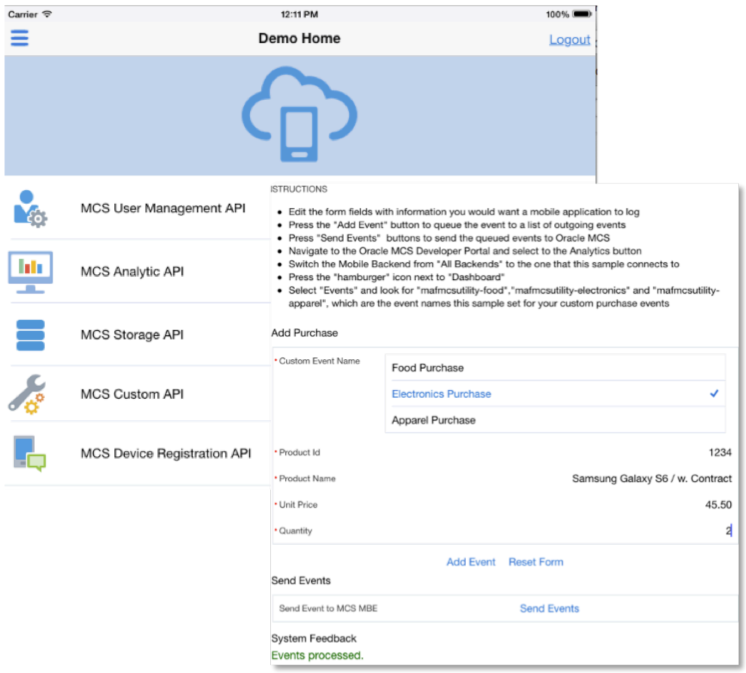
endSession	Ends the recording of the analytic session and sends the session events to the MCS analytic engine. A session not only contains custom

	events but also system and context events generated by MAF MCS Utility.
purgeAnalyticMessagesForMobileBackend	Analytic events are saved in a SQLite database if a session cannot be sent to MCS because of a network failure. This housekeeping method allows MAF application developers to clear all saved session for a MBE instance
refreshGeoLocationInformation	As part of the analytic session context event, the current GEO location is passed to MCS. MAF applications call this method to make MAF MCS Utility to re-read the GEO location
startSession	Starts an analytic session. If this method is not called before a custom event is added, the session will be created automatically. A time stamp is taken when calling this method

Where to find it in the MAF MCS Utility Public Sample Application

The MAF MCS Utility public sample accesses the `Analytics` proxy class from the `MobileBackendDC` data control class located in the application controller project. The data control exposes the following method you should have a look at: `addCustomAnalyticEvent`, `postEventsToServer`.

The public sample shows an example of a shopping cart in the Analytics.amx. Figure 9 shows the runtime view of this page.



ISTRUCTIONS

- Edit the form fields with information you would want a mobile application to log
- Press the "Add Event" button to queue the event to a list of outgoing events
- Press "Send Events" buttons to send the queued events to Oracle MCS
- Navigate to the Oracle MCS Developer Portal and select to the Analytics button
- Switch the Mobile Backend from "All Backends" to the one that this sample connects to
- Press the "hamburger" icon next to "Dashboard"
- Select "Events" and look for "mafmcsutility-food","mafmcsutility-electronics" and "mafmcsutility-apparel", which are the event names this sample set for your custom purchase events

Add Purchase

* Custom Event Name	Food Purchase Electronics Purchase ✓ Apparel Purchase
* Product Id	1234
* Product Name	Samsung Galaxy S6 / w. Contract
* Unit Price	45.50
* Quantity	2

Add Event   Reset Form

Send Events

Send Event to MCS MBE   Send Events

System Feedback
Events processed.

Figure 9: Analytics proxy in the MAF MCS Utility public sample

Example: Queue custom events and send analytic session to MCS

```
HashMap<String, String> properties = new HashMap<String, String>();

// Some sample data
String productId = "12345";
String productName = "iPhone";
String category = "electronics-purchase";
Double singlePrice = new Double(588);
Double quantity = new Double(2);
Double total = singlePrice * volume;
String totalString = total.toString();

// Create HashMap with values
properties.put("ProductId", productId);
properties.put("Name", producName);
properties.put("ProductType", category);
properties("Price",singlePrice.toString());
properties.put("Quantity", quantity.toString());
properties.put("Discount", "0");
properties.put("Total", totalString);

// Queue custom event locally
Analytics analyticsProxy = this.mobileBackend.getServiceProxyAnalytics();

if (isAnalyticEventOngoing == false) {

   analyticsProxy.startSession();
   isAnalyticEventOngoing = true;
}

Event customEvent = new Event(customEventName, null, properties);
analyticsProxy.addCustomEvent(customEvent);


// Repeat the steps above for as long as the analytic session (or mobile
// Use case) lasts

…

// Finally, send event to MCS - this would be in a separate method unless you
// immediately want to post events.

analyticsProxy.endSession();
```

Storage Service Proxy

The `com.oracle.maf.sample.mcs.apis.storage.Storage` proxy class exposes methods and helper objects to read Oracle MCS collection information and to update their content. The Storage proxy allows application users to access and write to shared or isolated collections they have access permission to.

Constructor

MAF application developers should not call the constructor directly but access the proxy class through a `MBE` instance

```
mbe.getServiceProxyStorage()
```

About HTTP ETag uses

HTTP entity tag (ETag) ensures read and write consistency for web clients querying or updating server side objects. To ensure read consistency and to increase query performance, clients can send a conditional request that contains information about the version of the requested object in the client cache. If the object on the server is newer than the object cached on the client, the server responds with a full reply that contains the new version of the object. If the version on the server is the same as in the cache, a shortened response is sent to indicate this.

Similar, to ensure write consistency, update requests issued from the client can indicate the version of the updated object for the server to compare it with the version of object it knows about. If the object versions match, the server allows the update, whereas if not, it returns an error message for the client to query the latest version of the object before sending an update.

The Storage Service Proxy supports ETag parameters in methods exposed on the `StorageCollection` object, allowing MAF application developers to implement conditional update and delete operations on collection content objects. Where supported, ETag parameters are optional arguments provided as key/value airs of a `HashMap`.

Methods

The following table documents common methods MAF developers may make use of and is not the definitive list

Method Name	Description
getThisUserId	Convenience method that returns the authenticated user Id. The MCS user is otherwise accessible through the UserManagement API. The user id is needed for querying a specific user's shared collection objects and may be used for this reason.
querySingleCollection	Returns an instance of the `StorageCollection` class that presents collection attributes and the collection object information for a named collection. The `StorageCollection` class also exposes methods to read (download) and update (create, modify and delete) the saved collection objects. `com.oracle.maf.sample.mcs.apis.storage.StorageCollection`
querySingleCollectionForUserById	Retrieves the collection with a given Id that is owned by a specified user identified by the userId passed as the second argument. The userId is not the same as the username used for login but the internal user Id of the mobile user account in MCS. For user-isolated collections, specifying the user id in the query allows the authenticated users with READ or

	READ_ALL permission to access this user's collection objects. Note that READ permission only entitles for reading the user's own isolated objects. To read other user's isolated objects, READ_ALL is required
queryStorageInformation	Queries a list of collections associated with a specific MCS MBE. The collection information query can be restricted in the fetch size and start index, allowing incremental fetches. The query returns an instance of `com.oracle.maf.sample.mcs.apis.storage.StorageInformation`. The `StorageInformation getItems()` method returns a list of `StorageCollection` with information about each collection, including the shared/isolated state or the total size of the collection contents

StorageCollection

The `com.oracle.maf.sample.mcs.apis.storage.StorageCollection` class is the object that MAF application developers use to download and upload collection object content.

The following table documents common methods MAF developers may make use of and is not the definitive list

Method Name	Description
contains	Checks if a specific object id is part of the collection
createObject	Overloaded method to upload content objects (like images) from the mobile device to MCS. MAF application developers pass the byte[] array of the object to upload along with either a `StorageObject` object or the displayname and mime type information. The Id the uploaded content as in the Oracle MCS collection is auto-generated by Oracle MCS.
createOrUpdateObject	Similar to the `createObject` method. This method also allows to update existin objects and only creates a new object in the MCS collection if no objet exists that matches the provided object id. This method also allows MAF application developers to define the unique id used when saving the content in Oracle MCS.
downloadByteContentForObjectId	Ovreloaded method that downloads the opaque object saved in Oracle MCS for a objectId or a object URI passed as the argument. An second method argument allows developers to specify the mime type for the http Accept header send to MCS. The response is a byte[] array.
dump	Returns a String representation of the collection object information (not its downloaded content)
isUserIsolated	Provides information whether or not the collection is user isolated

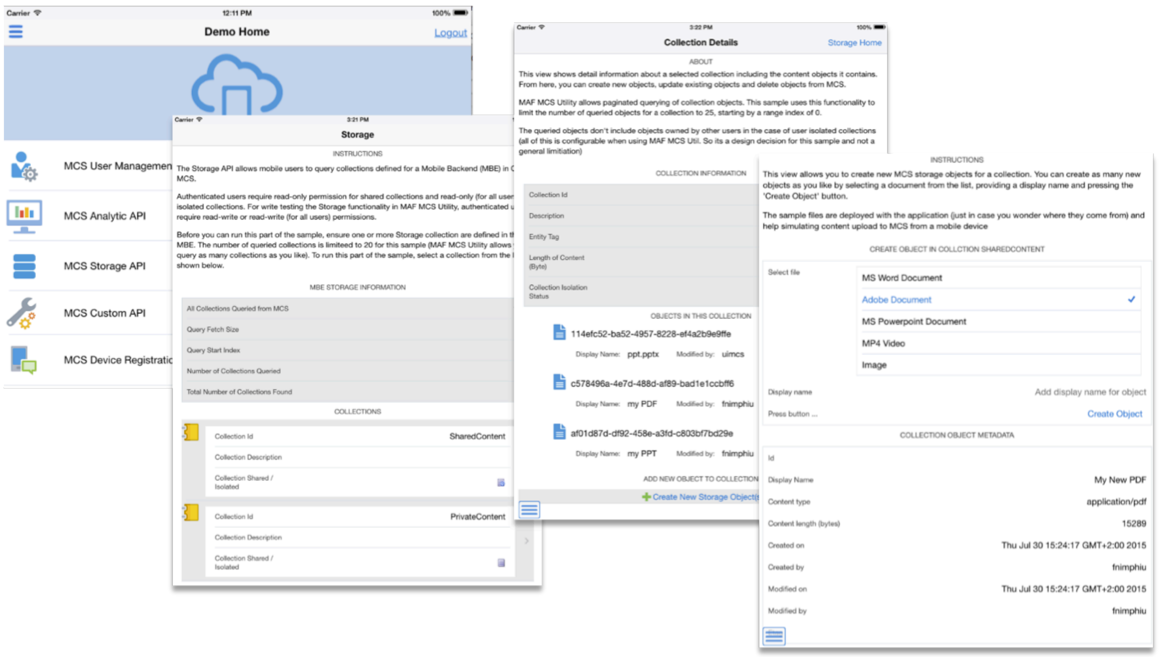
querySingleStorageObjectById	Returns information about the collection identified the collection id. The method returns an instance of `StorageInformation`
queryStorageObjectsByRange	Queries collection object information by fetch size and start index. The method returns a list of `StorageInformation`
removeCollectionObjectWithURI	The canonical link property of a StorageObject in a StorageCollection uniquely identifies the object in the remote MCS collection. For isolated collections this link contains e.g. the user id of the user owning the collection. The removeCollectionObjectWithURI method deletes the server side storage object. A second, optional, argument accepts a HashMap with HTTP ETags key-value pairs to implement conditional delete.
removeCollectionObject	Removes a storage object in a collection based on the following arguments: String objectId The id value of the object to delete HashMap<String,String> etagHashMap A HashMap with ETag parameter property and value to e.g. only remove object if it doesn't match the ETag sent with the request (If-Match).
setObjectOwnerUserID	StorageCollection represents a collection in the server side Oracle MCS cloud instance. If the collection is an isolated collection, then all queries must append a "user" query parameter (?user=<user id>). MAF MCS Utility automatically appends the user id of the authenticated user. For users with read_all, or read_write_all permission to access other user's isolated collection space this method allows changing the user query parameter value to the user Id of the other user (the one which collection objects should be updated or deleted). The `resetObjectOwnerUserIdToAuthenticatedUserId` method then can be used to switch the user query parameter value back to the user id of the authenticated user
updateCollectionObjectWithURI	The canonical link property of a StorageObject in a StorageCollection uniquely identifies the object in the remote MCS collection. The updateCollectionObjectWithURI method updates the server side storage object. The displayName argument allows developers to define the name of the object when its shown in the MCS portal. The contentType argument defines the MIME type of the content, for example image/png if the uploaded content is an image in PNG format

	A 4th, optional, argument accepts a HashMap with HTTP ETags key-value pairs to implement conditional delete.

Where to find it in the MAF MCS Utility Public Sample Application

The MAF MCS Utility public sample accesses the `Storage` proxy class from the `MobileBackendDC` data control class located in the application controller project. The sample application provides sample code for querying MCS MBE instance for contained collections, for listing collections and the contained object information, as well as for reading (download) creating (upload), deleting and updating (upload) objects.

The Storage.amx page in the view controller project is the landing pad for all Storage operations. Associated Java logic is stored in managed beans located in the `com.oracle.maf.sample.mobile.mbeans.storage` package of the view controller project. Figure 10 shows the runtime views of the MAF MCS Utility public sample



The Storage API allows mobile users to query collections defined for a Mobile Backend (MBE) in C MCS.

Authenticated users require read-only permission for shared collections and read-only (for all user isolated collections. For write testing the Storage functionality in MAF MCS Utility, authenticated u require read-write or read-write (for all users) permissions.

Before you can run this part of the sample, ensure one or more Storage collection are defined in th MBE. The number of queried collections is limiteed to 20 for this sample (MAF MCS Utility allows query as many collections as you like). To run this part of the sample, select a collection from the l shown below.

MBE STORAGE INFORMATION

All Collections Queried from MCS
Query Fetch Size
Query Start Index
Number of Collections Queried
Total Number of Collections Found

COLLECTIONS

Collection Id	SharedContent
Collection Description	
Collection Shared / Isolated	
Collection Id	PrivateContent
Collection Description	
Collection Shared / Isolated	

This view shows detail information about a selected collection including the content objects it contains. From here, you can create new objects, update existing objects and delete objects from MCS.

MAF MCS Utility allows paginated querying of collection objects. This sample uses this functionality to limit the number of queried objects for a collection to 25, starting by a range index of 0.

The queried objects don't include objects owned by other users in the case of user isolated collections (all of this is configurable when using MAF MCS Util. So its a design decision for this sample and not a general limitiation)

COLLECTION INFORMATION

Collection Id
Description
Entity Tag
Length of Content (Byte)
Collection Isolation Status

OBJECTS IN THIS COLLECTION

114efc52-ba52-4957-8228-ef4a2b9e9ffe
Display Name: ppt.pptx Modified by: uimcs

c578496a-4e7d-488d-af89-bad1e1ccbff6
Display Name: my PDF Modified by: fnimphiu

af01d87d-df92-458e-a3fd-c803bf7bd29e
Display Name: my PPT Modified by: fnimphiu

ADD NEW OBJECT TO COLLECTION
Create New Storage Object(s

INSTRUCTIONS

This view allows you to create new MCS storage objects for a collection. You can create as many new objects as you like by selecting a document from the list, providing a display name and pressing the 'Create Object' button.

The sample files are deployed with the application (just in case you wonder where they come from) and help simulating content upload to MCS from a mobile device

CREATE OBJECT IN COLLCTION SHAREDCONTENT

Select file: MS Word Document / Adobe Document ✓ / MS Powerpoint Document / MP4 Video / Image
Display name: Add display name for object
Press button ... Create Object

COLLECTION OBJECT METADATA

Id	
Display Name	My New PDF
Content type	application/pdf
Content length (bytes)	15289
Created on	Thu Jul 30 15:24:17 GMT+2:00 2015
Created by	fnimphiu
Modified on	Thu Jul 30 15:24:17 GMT+2:00 2015
Modified by	fnimphiu

Figure 10: Storage proxy views in the MAF MCS Utility public sample

Example: Create, Update, Delete Storage objects owned by other users

Isolated collections save user objects within a private space that is specific to a user. To address this private space in an isolated collection, Oracle MAF MCS Utility appends the authenticated user's user Id as a query parameter to all object manipulation and query operations. This happens automatically and developers don't need to code for this.

If the use case however requires the authenticated user to access collection objects in the private space of another user (assuming the authenticated user has the required read_all or read_write_all privileges), then the user id that is appended to the object manipulation an read operations, must be the Id of the object owner.

MAF MCS Utility knows of two options for accessing other users objects in an isolated collection

- Using the canonical URI of the object to access. The canonical URI of an object contains everything that is required to access the object in its private space. In MAF MCS Utility, each instance of `StorageObject` exposes a method to obtain the canonical URI.

  Note: See the example titled "Content Download using the canonical URI" later in this paper

- The `StorageCollection` objects exposes a property to set the object owner user Id. It's a context switch that allows developers to change the private space in which content objects are queried and manipulated. All operations executed on a `StorageCollection` happen in the context of the owner Id.

The following code fragment shows two options for setting the object owner Id to a user that is not the authenticated user:

1. Setting the user Id in the constructor

```
StorageCollection privateCollection = storageProxy.querySingleCollectionForUserId(
                                                   "<collection name>", "<user id>")
```

2. Setting the user Id for an existing collection

```
StorageCollection privateCollection = storageProxy.querySingleCollection(
                                                                 "<collection name>")
```

```
privateCollection.setObjectOwnerUserID("<user id>");
```

In both cases you can change the owner Id back to the authenticated user in a call to

```
privateCollection.resetObjectOwnerUserIdToAuthenticatedUserId();
```

Note: The above code does not impersonalize the user who owns the Storage content objects. The authenticated user performs operations in other user's private space of an isolated collection with her own permissions only.

Example: Content Download

```
String objectId = "… object Id of content in collection …";
String mimeTye  = "… e.g. image/png";
StorageCollection storageCollection = storageProxy.queryCollection("…a collection Id …");

byte[] downloadedContent =
     storageCollection.downloadByteContentForObjectId(objectId,mimeType);

//write to file. Define new file in download directory
String outFile =
             AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.DownloadDirectory)
             + "/"+targetFileName;
FileOutputStream fos = new FileOutputStream(outFile);
fos.write((byte[]) downloadedContent);
fos.close();
return outFile;
```

Example: Content Download using the canonical URI

Every object in a collection is unambiguously addressed by its canonical URI. For objects in isolated collections, for example, the canonical URI already contains the "?user=" parameter with the user id of the mobile user who created the object.

```
String mimeTye  = "… e.g. image/png";
StorageCollection storageCollection = storageProxy.queryCollection("…a collection Id …");
List<StorageObject> storageObjectList = new ArrayList<StorageObject>(); ;
//query the first 25 objects in the collection. If the collection is isolated (assuming
//the authenticated user has //read_write_all permission), then query objects of other
//users as well. Note that in such a use case it makes sense to use the canonical URI as
//you don't need to switch the user owner Id in the StorageCollection


storageObjectList = currentStorageCollection.queryStorageObjectsByRange(
                                          0, 25, this.showObjectsOwnedByOtherUsers, null);
//… lets assume the user selected an object from this list
StorageObject storageContentObject = storageObjectList.get(… index of object …)
String canonicalLink = storageContentObject.getCanonicalLink();
byte[] downloadedContent =
        currentStorageCollection.downloadByteContentForObjectUri(canonicalLink, mimeType);

//... see previous example for how to save files
```

Example: Content Upload

```
String displayName = "fnimphiu photo";
String contentType = "image/png";
byte[]byteContent = … photo from camera or file …
StorageCollection storageCollection = storageProxy.queryCollection("…a collection Id …");
StorageObject updateStorageObject = storageCollection.createObject(
                                                    displayName, contentType, byteContent);

// Access JSON object string with new item's meta-data
String json = updateStorageObject.toJSONString();
```

Example: Delete Content

```
String objectId = "… object Id of content in collection …";
StorageCollection storageCollection = storageProxy.queryCollection("…a collection Id …");
… browse collection …
// No entity tag, passing null instead
storageCollection.removeCollectionObject(objectId, null);
…
```

CustomAPI Service Proxy

The `com.oracle.maf.sample.mcs.apis.custom.CustomAPI` proxy class is a generic REST proxy that adds MCS required HTTP headers like Authorization, Oracle-Mobile-Backend-Id and Accept to any requests issued from MAF to Oracle MCS.

Technically, the `CustomAPI` proxy wraps the Oracle MAF `RestServiceAdapter` class, adding two convenience classes, `MCSRequest` and `MCSResponse`, to simplify the MAF REST request configuration and Response handling.

Though capable of accessing any Oracle MCS platform API, the `CustomAPI` should be primarily used to access custom APIs exposed on the Oracle MCS MBE. The benefit MAF application users get from using the `CustomAPI` proxy instead of calling the MAF MCS custom API directly from the MAF `RestServiceAdapter` is, beside of MCS specific headers that are set, a consistent error handling (see "ServiceProxyException class" later in this paper) as well as MBE specific logging.

Constructor

MAF application developers should not call the constructor directly but access the proxy class through a `MBE` instance

```
mbe.getServiceProxyCustomApi)
```

Methods

The `CustomAPI` proxy exposes two methods

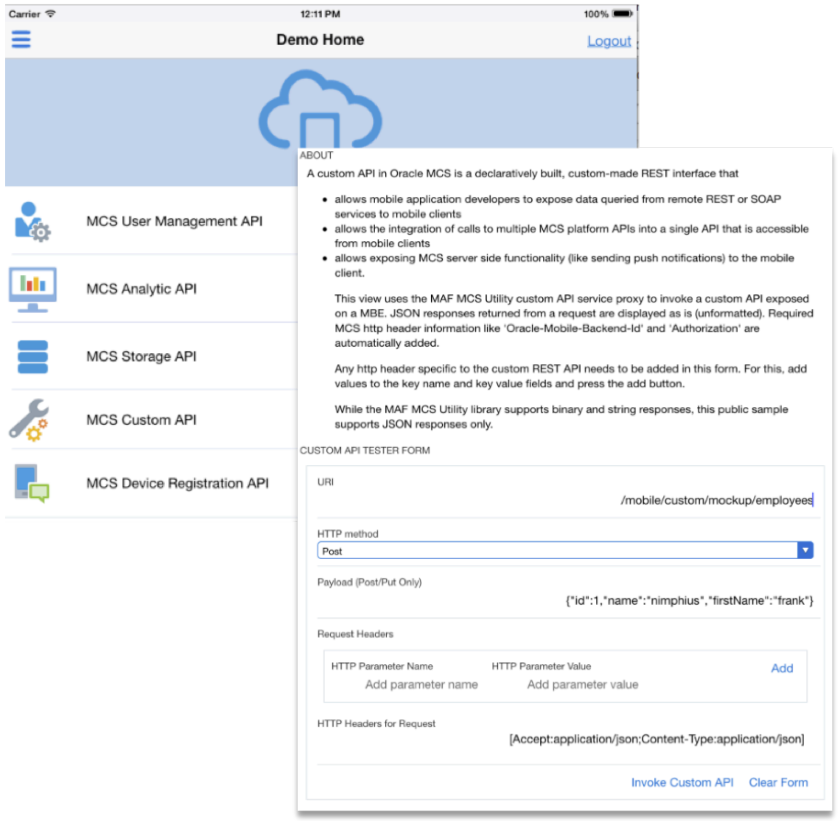
Method Name	Description
sendForStringResponse	Method to send a synchronous custom REST request to the MCS server instance for a String response. The method treats all response codes that are not within a HTTP 2XX range as application errors. `sendForStringResponse(MCSRequest request)` The `MCSRequest` object holds all the configurations require to issue the call, including the MAF REST connection name, the request URI, header parameters and more. The method returns `MCSResponse`, an object that holds the response payload and additional information. MAF applications call `getMessage()` on the `MCSResponse` object to obtain the response payload, which is an Object that can be casted to String.
sendReceiveBytes	Method to send a synchronous custom REST request to the MCS server instance for a byte[] response. The method treats all response codes that are not within a HTTP 2XX range as application errors. `sendReceiveBytes(MCSRequest request)` The `MCSRequest` object holds all the configurations require to issue the call, including the MAF REST connection name, the request URI, header parameters and more. The method returns `MCSResponse`, an object that holds the response payload and additional information. MAF applications call `getMessage()` on the `MCSResponse` object to obtain the response payload, which is an Object that can be casted to byte[].

Where to find it in the MAF MCS Utility Public Sample Application

The MAF MCS Utility public sample accesses the `CustomAPI` proxy class from the `MobileBackendDC` data control class located in the application controller project. The `invokeCustomMcsAPI` method is invoked from the view layer in the CustomAPIBacking backing bean that is located in the `com.oracle.maf.sample.mobile.mbeans.custom` package. Figure 11 shows the runtime view of the CustomAPI.amx page in the public sample.

In the sample, users select a HTTP method (Get, Post, Put, Delete) and provide a URI to the custom API to invoke in MCS. If the HTTP method is Put or Post, a payload may be required. The sample application only supports string payloads and responses to keep it simple. The underlying MAF MCS Utility proxy however also allows you to upload or download binary content if this is what the custom API is designed for.

The http header parameter fields (one of the name, one for the value) allow users to add any header parameter and value to be sent with the request. The response returned from the custom API invocation is then displayed at the bottom of the screen in its raw format, which is JSON or text.



ABOUT

A custom API in Oracle MCS is a declaratively built, custom-made REST interface that

- allows mobile application developers to expose data queried from remote REST or SOAP services to mobile clients
- allows the integration of calls to multiple MCS platform APIs into a single API that is accessible from mobile clients
- allows exposing MCS server side functionality (like sending push notifications) to the mobile client.

This view uses the MAF MCS Utility custom API service proxy to invoke a custom API exposed on a MBE. JSON responses returned from a request are displayed as is (unformatted). Required MCS http header information like 'Oracle-Mobile-Backend-Id' and 'Authorization' are automatically added.

Any http header specific to the custom REST API needs to be added in this form. For this, add values to the key name and key value fields and press the add button.

While the MAF MCS Utility library supports binary and string responses, this public sample supports JSON responses only.

CUSTOM API TESTER FORM

URI
/mobile/custom/mockup/employees

HTTP method
Post

Payload (Post/Put Only)
{"id":1,"name":"nimphius","firstName":"frank"}

Request Headers

HTTP Parameter Name: Add parameter name
HTTP Parameter Value: Add parameter value
Add

HTTP Headers for Request
[Accept:application/json;Content-Type:application/json]

Invoke Custom API　Clear Form

Figure 11: CustomAPI view in the MAF MCS Utility public sample

Example: Custom API Invocation

```
//gGet access to the service proxy for custom API calls
CustomAPI customApi = mobileBackend.getServiceProxyCustomApi();

// MAF MCS Utility provides to configuration objects, MCSRequest and
// MCSResponse, that guide you through setting up the MCS request and
//to read the MCS response. First, a MCSRequest object is created
```

```
MCSRequest request = new MCSRequest(mobileBackend.getMbeConfiguration());

// The MSF REST connection as defined in Application Resources --> Connections --> REST.
// In this example the REST connection name is "mcsrestconn"
request.setConnectionName("mcsrestconn");

// Save the custom MCS API URI
request.setRequestURI("/moile/custom/mockup/employees");
// Set the request method
request.setHttpMethod(MCSRequest.HttpMethod.POST);

// Set payload - a JSON string in this example
request.setPayload("{\"id\":\"1\"\"name\":\"nimphius\",\"firstName\":\"frank\"}");
request.setRetryLimit(0);

// Define the headers that need to be sent with the custom API request
HashMap<String, String> headers = new HashMap<String, String>();
headers.put("Content-Type","application/json");
request.setHttpHeaders(headers);


// This call returns a String response. For binary responses (binary[]) you use
// CustomApi.sendReceiveBytes(request) instead.


MCSResponse response = customApi.sendForStringResponse(request);
String jsonResponse = (String) response.getMessage();
```

Notifications Service Proxy

The Notifications platform API in Oracle MCS provides client and a server side services. The server side service is for custom API code written in Node.js to queue and send push messages to the Google Cloud Messaging (GCM) service or Apple Push Notification Service (APNS).

The client service allows mobile clients to register the mobile device with Oracle MCS so Oracle MCS can queue push messages with the Google GMC or Apple APNS push providers for delivery to the registered device when the messages are raised at a later point in time.

As the name indicates, only the client service is accessible from mobile clients directly, for which the MAF MCS Utility provides the `com.oracle.maf.sample.mcs.apis.notifications.Notifications` proxy class.

Unlike the other proxy classes exposed on the `MBE` in MAF MCS Utility, the `Notifications` proxy does not work out of the box and requires MAF application developers to obtain a Google sender Id and an Apple p12 certificate and bundle Id needed for MAF applications to request a device token from GMC or APNS upon application startup.

Note: The MAF MCS Utility public sample application has the code to obtain the device token readily setup in the application controller project: see the `PushEventListener` and `LifeCycleListenerImpl` classes. Please refer to the Oracle MAF product documentation for more information about how to obtain the sender Id and p12 certificate.

The `Notifications` proxy service allows mobile application clients to register and deregister mobile devices, ensuring the MCS message payloads to be well formatted and configured properly for Android and iOS.

Constructor

MAF application developers should not call the constructor directly but access the proxy class through a `MBE` instance

```
mbe.getServiceProxyNotifications()
```

Methods

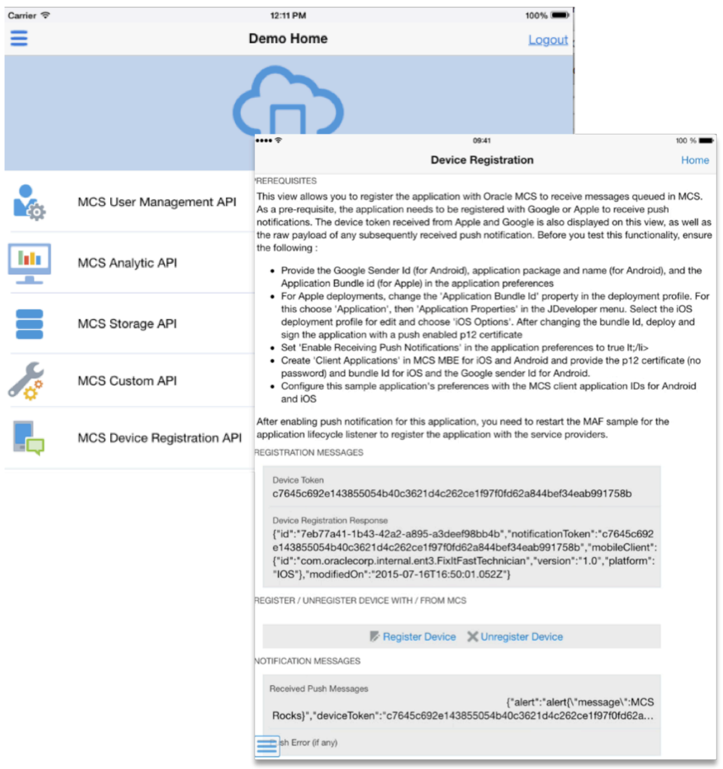
Method Name	Description
deregisterDeviceFromMCS	Attempts to de-register the device from MCS so no further messages are sent to Google or Apple push service for this device. The configuration information is read from the `MBE`'s `MBEConfiguation` object so MAF application developers don't need to pass any argument when invoking this method. This method should be called upon closing an application when no messages should be sent while the application is not active.
isDeviceRegisteredForPush	Convenience method for MAF application developers to check if an application has been registered with MCS.
registerDeviceToMCS	Attempts to register a device from MCS so messages are sent to Google or Apple push service through MCS for this device. The configuration information is read from the `MBE`'s `MBEConfiguation` object so MAF application developers don't need to pass any argument when invoking this method. This method should always be called upon application start. Note that MCS does not keep device registration forever and cleans-up unused tokens. The method returns a JSON string with the MCS device registration information. This information is not required for obtaining and push messages and can be ignored.

Where to find it in the MAF MCS Utility Public Sample Application

The MAF MCS Utility public sample accesses the `Notifications` proxy class from the `MobileBackendDC` data control class located in the application controller project. In addition, the sample application has a lifecycle and push event listener configured to receive a device token from Google or Apple.

Note: Because of Apple and Google license restrictions, the MAF MCS Utility sample application does not ship with a Google sender Id or an Apple bundle Id and p12 certificate for MAF developers to immediately test notifications. You need to obtain your own sender Id and p12 certificate and – for Apple deployment – sign the application upon deployment and change the bundle Id for the deployment.

Figure 12 shows the runtime view of the DeviceRegistration.amx page located in the view controller project. If push is setup and enabled for the MAF sample application then this page shows information like the device token that the application received from Google or Apple, the JSON object that indicates a successful device registration, as well as the raw payload of incoming push message. Figure 12 shows the page with a (meanwhile invalid) device token, a MCS device registration notification, as well as a push message sent from MCS.



PREREQUISITES

This view allows you to register the application with Oracle MCS to receive messages queued in MCS. As a pre-requisite, the application needs to be registered with Google or Apple to receive push notifications. The device token received from Apple and Google is also displayed on this view, as well as the raw payload of any subsequently received push notification. Before you test this functionality, ensure the following :

- Provide the Google Sender Id (for Android), application package and name (for Android), and the Application Bundle id (for Apple) in the application preferences
- For Apple deployments, change the 'Application Bundle Id' property in the deployment profile. For this choose 'Application', then 'Application Properties' in the JDeveloper menu. Select the iOS deployment profile for edit and choose 'iOS Options'. After changing the bundle Id, deploy and sign the application with a push enabled p12 certificate
- Set 'Enable Receiving Push Notifications' in the application preferences to true lt;/li>
- Create 'Client Applications' in MCS MBE for iOS and Android and provide the p12 certificate (no password) and bundle Id for iOS and the Google sender Id for Android.
- Configure this sample application's preferences with the MCS client application IDs for Android and iOS

After enabling push notification for this application, you need to restart the MAF sample for the application lifecycle listener to register the application with the service providers.

REGISTRATION MESSAGES

Device Token
c7645c692e143855054b40c3621d4c262ce1f97f0fd62a844bef34eab991758b

Device Registration Response
{"id":"7eb77a41-1b43-42a2-a895-a3deef98bb4b","notificationToken":"c7645c692e143855054b40c3621d4c262ce1f97f0fd62a844bef34eab991758b","mobileClient":{"id":"com.oraclecorp.internal.ent3.FixItFastTechnician","version":"1.0","platform":"IOS"},"modifiedOn":"2015-07-16T16:50:01.052Z"}

REGISTER / UNREGISTER DEVICE WITH / FROM MCS

Register Device Unregister Device

NOTIFICATION MESSAGES

Received Push Messages
{"alert":"alert{\"message\":MCS Rocks}","deviceToken":"c7645c692e143855054b40c3621d4c262ce1f97f0fd62a...

sh Error (if any)

Figure 12: CustomAPI view in the MAF MCS Utility public sample

Example: Device Registration

```
// Get the _token retrieved from Google or Apple, assuming the
// MAF lifecycle listener stored this information in memory: //
("#{appliationScope.devicetoken}");
String _token =
    (String) AdfmfJavaUtilities.getELValue("#{appliationScope.devicetoken}");

// If we have a _token, set it to the MBE configuration for the Notifications
// service proxy to access when registering and de-registering the application
// to MCS

if(_token != null && !_token.isEmpty()){
   mobileBackend.getMbeConfiguration().setDeviceToken(_token);
} else {

    // No _token -- cancel request
    return;

}

// Does MAF run on iOS or Android ?
DeviceManager deviceManager = DeviceManagerFactory.getDeviceManager();
```

```
String _toUppercaseManufacturerOS = deviceManager.getOs().toUpperCase();
boolean isIOS = _toUppercaseManufacturerOS.equals("IOS") ? true : false;

if (isIOS) {
  //assume Apple bundle Id is available in "#{applicationScope.bundleId}"
   String appleBundleId = (String)
   AdfmfJavaUtilities.getELValue("#{applicationScope.bundleId}");
   mobileBackend.getMbeConfiguration().setAppleBundleId(appleBundleId);
// else Android
} else {
  // Assume Google package name is available in "#{applicationScope.packageName}"
  String packageName =
     (String) AdfmfJavaUtilities.getELValue("#{applicationScope.packageName}");
   mobileBackend.getMbeConfiguration().setGooglePackageName(packageName);
}

Notifications notification = mobileBackend.getServiceProxyNotifications();
String registrationInfo = notification.registerDeviceToMCS();

…
```

ServiceProxyException class

MAF MCS Utility throws two types of exceptions for MAF application developers to handle errors

- `oracle.adfmf.framework.exception.IllegalArgumentException`

  Thrown when required method or constructor arguments are missing (e.g. set to null) or if arguments have a zero length.

- `com.oracle.maf.sample.mcs.shared.exceptions.ServiceProxyException`.
  Thrown for both application errors or REST transport layer exceptions. Application errors are failures in invoking Oracle MCS. For example a user attempts to access objects in a collection that she doesn't have READ or READ_ALL permission for.

  REST transport layer exceptions are runtime exceptions, like network or server access failures, that MAF MCS Utility doesn't know how to handle.

The `ServiceProxyException` provides convenient methods that allow MAF application developers to access the MCS error message details, like message title and description, and is thrown within all methods in MAF MCS Utility.

Methods

Method Name	Description
getErrorResponseHeaders	If an error response contains HTTP header information then this is returned in a HashMap
getExceptionClassName	Returns the exception class wrapped by the `ServiceProxyException` object
getHttpResponseCode	Returns the HTTP error response for application errors. Oracle MCS, for example, returns HTTP status codes in the 4xx range for failed platform API invocations. A list of status codes returned in case of an error for each MCS API can be obtained in the MAF MCS Utility source code or the Oracle MCS API tester
getMessage	Returns the error message, which either is a JSON payload string (if the error is an MCS API invocation failure) or a text message (if the error is a runtime or network error)
isApplicationError	Indicates the error to be an Oracle MCS application error. In this case you can call

```
String errorMessage = "";
if (e.isApplicationError()) {
   // If this is a well formatted Oracle Mobile error,
   // we can display a user friendly error message
    try {
    // OracleMobileError is a MAF NCS Utility class
    OracleMobileError mobileError =
        OracleMobileErrorHelper.getMobileErrorObject(
                                e.getMessage());
         errorMessage = mobileError.getTitle();

    } catch (JSONException f) {
      //not a JSN formatted MCS error message
      errorMessage = e.getMessage();
    }
 …
}
```

isException	Indicates the error not to be related to MCS but the runtime or network

Where to find it in the MAF MCS Utility Public Sample Application

The MAF MCS Utility public sample uses the `ServiceProxyException` class in the `MobileBackendDC` data control class located in the application controller project.

All MAF MCS Utility service proxy invocations in the data control class are surrounded by a try-catch block that, in case of a failed invocation, checks whether the error is an application error or runtime exception. In the case of an application error, the sample uses the MAF MCS Utility `OracleMobileErrorHelper` utility class to read the MCS error message details into a Java Object, the `OracleMobileError` class to then display the error title in the user interface.

Example: Error Handling

Sample code taken from `MobileBackendDC`:

```
if ((collectionId != null) && (!collectionId.isEmpty())) {
  //use of the typed interface to access a service proxy
  Storage storageProxy = this.mobileBackend.getServiceProxyStorage();
  try {
     currentStorageCollection = storageProxy.queryCollection(collectionId);
     // Indicates successful change of storagee selection
     currentStorageCollectionChangedFlag = true;
     return currentStorageCollection;
  } catch (ServiceProxyException e) {
      if (e.isApplicationError()) {
         try {
            OracleMobileError mobileError =
                     OracleMobileErrorHelper.getMobileErrorObject(e.getMessage());
                     // Print short description of error
                     logDcErrorForUserInterfaceDisplay(mobileError.getTitle());
         } catch (JSONException f) {
                logDcErrorForUserInterfaceDisplay(e.getMessage());
         }
      } else {
            logDcErrorForUserInterfaceDisplay(e.getMessage());
      }
  }

} else {
     logDcErrorForUserInterfaceDisplay("collectionId value cannot be null or empty");
}
```

…

Asynchronous API Invocation

By design, all MAF MCS Utility methods are invoked synchronously, except for sending Analytic sessions to Oracle MCS, which always executes asynchronously.

When designing Java libraries, developers have two choices to handle asynchronous method invocations

- Have methods exposed by the library to execute asynchronously and provide a callback mechanism that invokes a callback method in an object passed to the method. This option provides a clean design and passes control back to the calling application when the asynchronous task completes. The disadvantage is that creating the callback handler is a programming overhead, especially if a callback is needed for the method invocation success and error case. Using this approach also requires the application developer to ensure methods are executed in the required sequence, which could be tricky. Note that callbacks are very common in JavaScript development.
- Have methods in the library executing synchronously and explain to the application developer how the APIs can be wrapped in an asynchronous method invocation. The advantage of this option is that the application developer stays in control of the order in which methods are executed and also get a better handle to the UI controls, which usually are executed in the main thread, to refresh in response to a method invocation.

For MAF MCS Utility, the second choice, to optionally wrap synchronous methods in asynchronous calls, has been chosen. This section explains how MAF application developers can invoke MAF MCS Utility service proxy methods asynchronously.

MAF is a Java framework and as such can start additional execution threads in addition to the main application thread. For this, the MAF application developer wraps the MCS MCS Utility service proxy method invocation in an object that implements `Runnable`. The `Runnable` object is then executed from a `ExecutorService` object. Figure 13 shows a block diagram of this invocation.

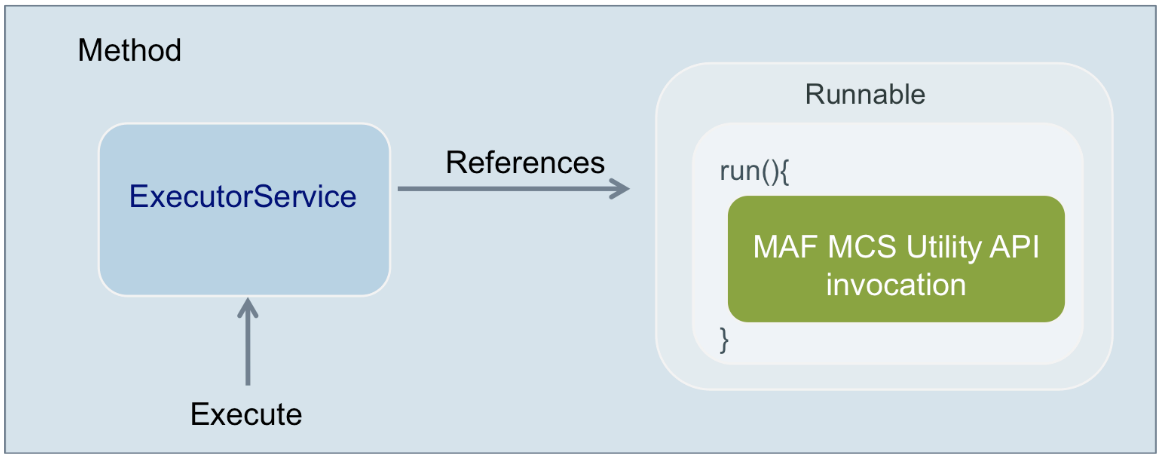


Figure 13: MAF MCS Utility Architecture

Lets have a look at an example. Note that all code below could be added to the same method e.g. exposed on a data control or managed bean:

```
Runnable mcsutilityJob = new Runnable(){
     public void run(){

       //this is where MAF developer put the MAF MCS Utility method
       //invocation …
        … do some more work …
       //refresh UI controls that are impacted by the data changes
```

```
        //triggered by the method invocation. For this in MAF you call
        //AdfmfJavaUtilities.flushDataChangeEvent()

    }

}

… optionally do some other work here …


//execute runnable
ExecutorService executor = Executors.newFixedThreadPool(2);
executor.execute(mcsutilityJob);
executor.shutdown();
```

Note that the application continues with the code lines coming after the `ExecutorService` execution. To ensure UI changes are displayed in the MAF UI, you need to call `AdfmfJavaUtilities.flushDataChangeEvent()` to make the main thread refresh its content. In addition, if a method updates a collection in the main thread, you need to raise a provider change event in MAF.

Note: Provider change events in MAF are well explained in this video published on the Oracle Mobile Application Framework YouTube channel: https://www.youtube.com/watch?v=ZJePFhfVqMU

Where to find it in the MAF MCS Utility Public Sample Application

The MAF MCS Utility public sample showcases asynchronous method invocation for device registration (see the `registerDeviceWithMCS` method) and the custom API invocation (see the `invokeCustomMcsAPI` method) in the `MobileBackendDC` data control class located in the application controller project.

Note: If you look at the MAF MCS Utility source code, you find the same asynchronous method invocation used in the `Analytics` proxy. Posting analytic sessions to MCS does not update values in the calling application, so that it was save to implement this call as asynchronous in the MAF MCS Utility directly without causing unwanted effects, like the UI not properly refreshing.

Example: Asynchronous MAF MCS Utility method invocation

```
Runnable mcsJob = new Runnable(){
  public void run(){
    try {
      Notifications notification = mobileBackend.getServiceProxyNotifications();
      notification.deregisterDeviceFromMCS();
      logDcInfoForUserInterfaceDisplay("Application has been deregistered from MCS");
      AdfmfJavaUtilities.setELValue(PushConstants.MCS_REGISTRATION_STRING, "");
   } catch (ServiceProxyException e) {
       if (e.isApplicationError()) {
           …
        }

          else {,

            logDcErrorForUserInterfaceDisplay(e.getMessage());
        }

    }
   //ensure main thread is synchronized with result

   AdfmfJavaUtilities.flushDataChangeEvent();
  }
 };


//Invoke asynchronous invocation

  ExecutorService executor = Executors.newFixedThreadPool(2);
  executor.execute(mcsJob);
  executor.shutdown();
}
```

User Authentication

Authentication in Oracle MCS public cloud version 1.0 is through basic authentication only and can be configured on the MAF Feature level, or performed manually using the MAF MCS Utility Authorization provider.

If user authentication is not explicitly performed through the MAF MCS Utility, then the utility expects the HTTP Authorization header to be set by Oracle MAF for each request to Oracle MCS.

Authentication through MAF

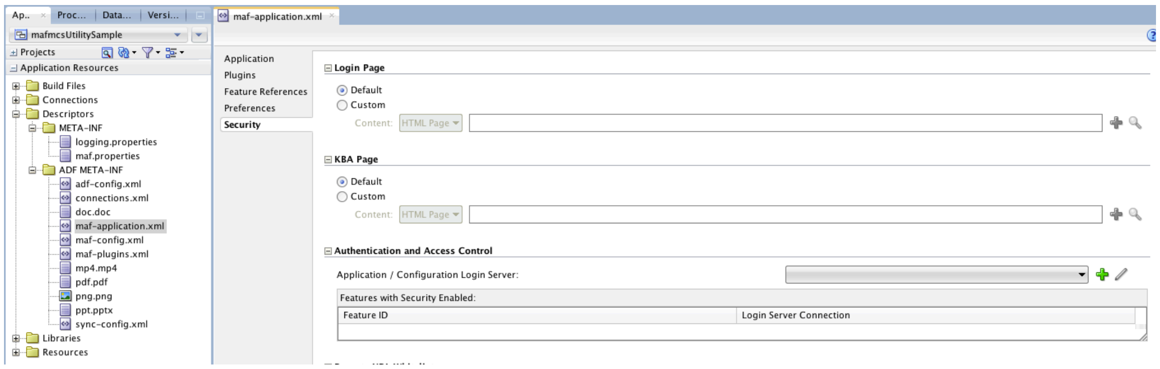
To use the MAF security framework for authenticating MAF MCS Utility calls, a 1-1 mapping between a MAF MCS Utility MBE instance and a feature in MAF needs to be implemented. To create a 1-1 mapping of a feature in MAF with an MBE instance in MAF MCS Utility, you configure the mafmcsutility.jar file in the view controller project and access the `MBEManager` from a data control or managed bean defined in the view controller project.

In doing so you ensure the `MBEManager` instance and the `MBE` instance(s) are created and managed in the child class loader associated with a MAF Feature.

Next you need to enable security on the MAF feature (maf-features.xml dialog) and use the maf-application.xml editor to create a Login Server pointing to Oracle MCS MBE for authentication.

Also in the maf-application.xml dialog, the MAF login server configuration needs to be associated with the MAF feature that should be authenticated against the Oracle MCS MBE.

Figure 14 shows themaf-application.xml dialog and the security panel you use to create a login server connection and associate it with the MAF feature to authenticate against the MAF MCS MBE



Login Page

[✓ Default] Custom

Content: HTML Page

KBA Page

[✓ Default] Custom

Content: HTML Page

Authentication and Access Control

Application / Configuration Login Server:

Features with Security Enabled:

Feature ID	Login Server Connection

Figure 14: maf-application.xml dialog with security panel

The benefit of using MAF declarative security for authentication, beside it being declaratively configured, is that it provides offline-authentication and "remember-me" functionality for basic authentication.

Note: To be able to authenticate MAF Features to Oracle MCS MBE you must use MAF 2.1.3 or later

Authentication through MAF MCS Utility

MAF MCS Utility allows application developers to authorize requests to Oracle MCS MBE through the MAF MCS Utility `Authorization` provider class. Manual MCS MBE authorization in MAF MCS Utility works with MAF 2.1.2 and later and ensures the HTTP Authorization header to be set by MAF MCS Utility for every request to Oracle MCS.

Note: MAF MCS Utility authorization works for MBE instances that are mapped to a MAF feature and MBE instance that are created in the application controller project for global application use through a MAF data control.

```
Authorization authorization = mobileBackend.getAuthorizationProvider();

Boolean authenticationSuccess = Boolean.FALSE;

try {

   // ServiceProxyException is thrown if authentication fails
   authorization.authenticate(username, password);
   authenticationSuccess = Boolean.TRUE;
} catch (Exception e) {
   authenticationSuccess = Boolean.FALSE;
     …
}
```

The benefit of using manual code-centric authentication through MAF MCS Utility is that application developers can switch between anonymous authentication and username-based authentication. This is easier to achieve using MAF MCS based authentication than declarative authentication in MAF.

In addition, programmatic authentication implicitly sets the authenticated user's user Id to the `MBEConfiguration` object. This setting is required for accessing isolate collections within the Storage Service Proxy. If you authenticate through MAF declarative security features, you need to immediately access the `UserInfo` Service Proxy and obtain the authenticated user's userId (not the username) and then set it manually to the `MBEConfiguration` object in a call to `setAuthenticatedUserMCSUserId()`.

Where to find it in the MAF MCS Utility Public Sample Application

Earlier in this paper, figure 5 shows the manual login screen in the MAF MCS Utility public sample that allows application users to authenticate by their username and password, or as anonymous for public API access only.

The sample login screen (Authentication.amx in the view controller project) accesses the `authenticateUser` and `anonymousLogin` methods in the `MobileBackendDC` data control class located in the application controller project.

Example: Anonymous login

```
Boolean authenticationSuccess = Boolean.FALSE;
if (mobileBackend != null) {
  Authorization authorization = mobileBackend.getAuthorizationProvider();
  try {
     authorization.authenticateAsAnonymous();
     authenticationSuccess = Boolean.TRUE;
  } catch (ServiceProxyException e) {
     authenticationSuccess = Boolean.FALSE;
     if (e.isApplicationError()) {
       try {
          OracleMobileError mobileError =
               OracleMobileErrorHelper.getMobileErrorObject(e.getMessage());
          …
        } catch (JSONException f) {
  …
        }
     }else {

   …
 }
```

Debugging

The Oracle MAF public samples not only contain mafmcsutility.jar and the public sample that demonstrates how to work with the utility, but also the MAF MCS Utility source code in a JDeveloper workspace. The MAF MCS Utility source code is well commented and useful for any developer to understand what specific Service Proxy methods do and what they are for.

To ease development and debugging, you can add the MAF MCS Utility source code project to your application workspace. Figure 15 shows the dialog that opens when you choose File → Open in the JDeveloper menu.

Navigate to the mafmcsutility.jpr project file and open it. OK the dialog that informs you that any changes will be applied to the MAF MCS Utility workspace.

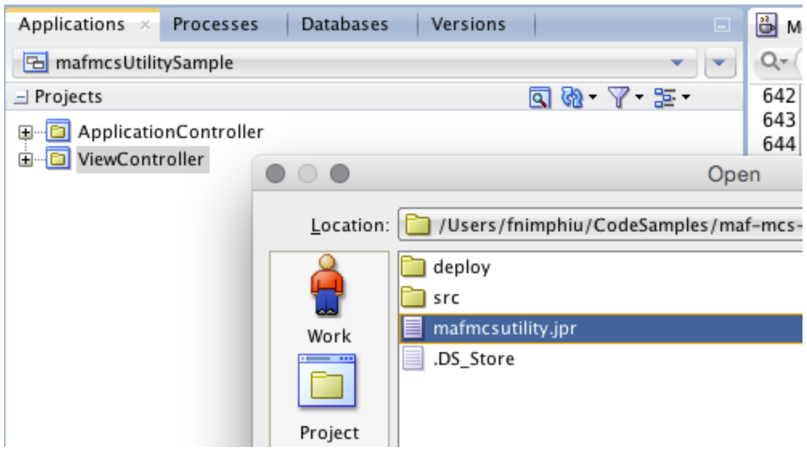


Figure 15: Adding the MAF MCS Utility project to a custom MAF application workspace

You then configure the MAF MCS Utility project as a dependency to the MAF ApplicationController or View Controller project. This allows you to easily navigate to specific utility classes and also to set break points in the MAF MCS Utility code.

At the end of your application development you then replace the MAF MCS Utility project dependency with a library reference to the mafmcsutility.jar file. Figure 16 shows the "mafmcsutility" project being added to the MAF MCS Utility public sample workspace.

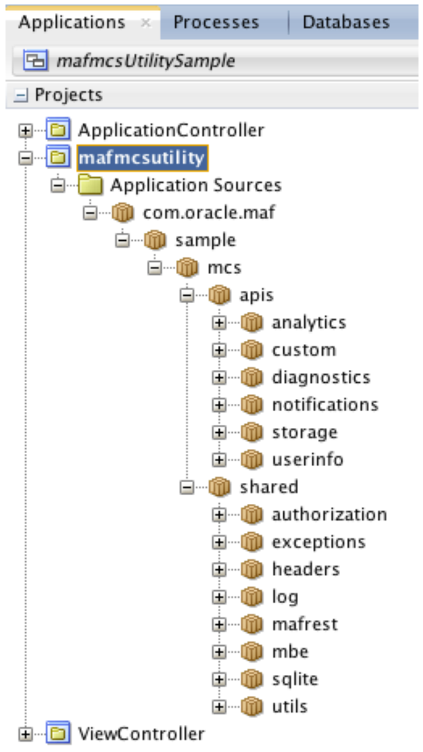


Figure 16: MAF MCS Utility project added to the public sample work space

Conclusion

This whitepaper provided MAF application developer with an overview of MAF MCS Utility, a public sample and library that simplifies Oracle MAF application access to Oracle Mobile Cloud Services (MCS). Though Oracle MAF MCS Utility is not called an SDK for Oracle MCS, it provides similar functionality and infrastructure.

The API documentation and code examples in this whitepaper, in addition to the Oracle MAF MCS Utility public sample application and the also provided MAF MCS Utility source code, enable MAF application developers to easily integrate calls to Oracle MCS public cloud in their custom mobile applications.

Oracle Mobile Application Framework – MAF MCS Utility Developer Guide
August 2015
Author: Frank Nimphius
Contributors: Chris Muir, Mireille Duroussaud, Lynn Munsiger, Gary Williams, Jeff Gallus