Oracle® Health Sciences mHealth Connector Cloud Service

Integrating the Client Library with a Mobile Application

Release 1.4.0.1

F25623-01

December 2019

Contents

Introduction

This document outlines the steps to integrate the Oracle Health Sciences mHealth Connector Cloud Service Client Library (Client Library) with a mobile application. Mobile application developers can use this library to send data collected by a mobile application to mHealth Connector Cloud Service (mHealth Connector). Instructions for both iOS and Android applications follow.

If you are interested in connecting to mHealth Connector, please contact your Oracle sales representative or email Oracle Health Sciences at: healthsciences_ww_grp@oracle.com.

You can also visit the Oracle Health Sciences website at:
https://www.oracle.com/industries/health-sciences/index.html

> **Note:** This library does not have the capability to capture clinical data directly from a medical device.

Prerequisites

1. An mHealth Connector study must be created and activated. Please contact your assigned project coordinator to receive the mHealth Connector URL and login credentials.
2. Before using a specific mobile device with the Client Library, the device data model must be created in mHealth Connector. Work with your project coordinator to define the device data model.

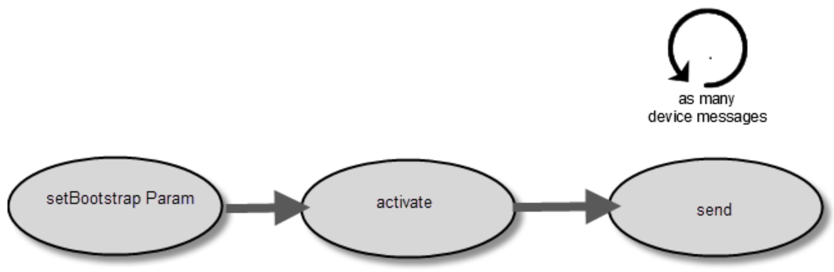
A device data model establishes a template for the messages sent from a mobile application to mHealth Connector. Each message type defined in the device model is identified by a unique URN (Uniform Resource Name). Data or Alert messages sent to mHealth Connector must include this URN.

3. Once the device models are created, a virtual device instance must be created for each subject using a mobile application that directly communicates with the mHealth platform. Typically, this registration is performed when a subject is enrolled in a clinical study. The registration process returns an access code and a one-time passcode. To securely connect with the mHealth platform, the mobile application needs to be bootstrapped with the access code and the one-time password as part of the subject onboarding process.

API Flow

Figure 1 shows the steps and order in which device data models are added.

Figure 1 mHealth Connector Cloud Service Client Library API flow



Working with the iOS Version of the mHealth Connector Cloud Service Client Library

This section describes importing the Client Library to an iOS-based application and the APIs to use to send data collected from a mobile application to mHealth Connector. For Android-based applications, see Working with the Android Version of the mHealth Connector Cloud Service Client Library.

Technical Requirements

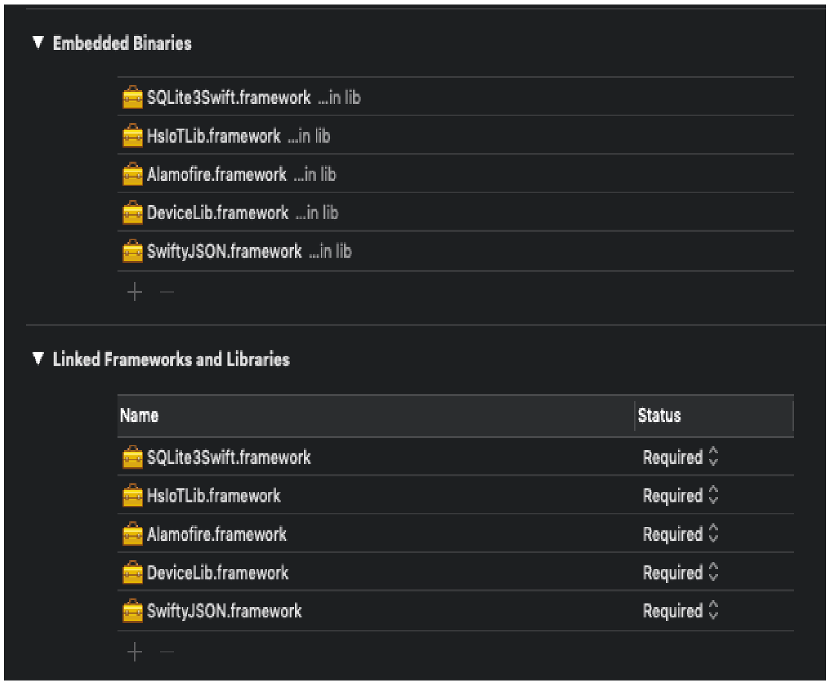
The following section assumes that Xcode 10.x is installed and ready to use. The Client Library supports iOS 12.4 and higher.

Import the mHealth Connector Cloud Service Client to the iOS Application

1. Open the Xcode project and navigate to the project target.
2. Copy the **lib** directory with the mHealth Framework files to the root project directory.
3. From the General tab, add all *.framework files in the Framework library ZIP file as embedded binaries.

Figure 2 shows adding framework files as embedded binaries from Targets—> General tab.

*Figure 2 Add mHealth *.framework files as embedded binaries*



▼ Embedded Binaries

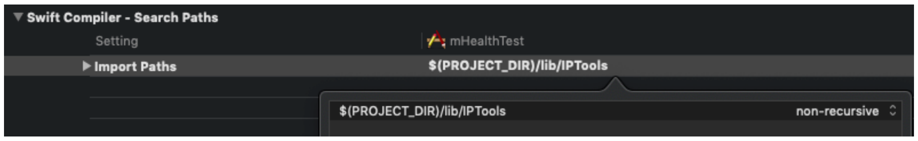
- SQLite3Swift.framework ...in lib
- HsIoTLib.framework ...in lib
- Alamofire.framework ...in lib
- DeviceLib.framework ...in lib
- SwiftyJSON.framework ...in lib

▼ Linked Frameworks and Libraries

Name	Status
SQLite3Swift.framework	Required
HsIoTLib.framework	Required
Alamofire.framework	Required
DeviceLib.framework	Required
SwiftyJSON.framework	Required

4. Click **Build Settings** and enter FRAMEWORK_SEARCH_PATHS in the **Search** field.
5. Double-click the **FRAMEWORK_SEARCH_PATHS** field, then click **+** and enter a new row with the value, $(PROJECT_DIR)/lib.
6. In the **Search** field, replace FRAMEWORK_SEARCH_PATHS with SWIFT_INCLUDE and press Enter.
7. Double-click the **Import Paths** field, then click **+**, enter a new row with the value $(PROJECT_DIR)/lib/IPTools, and press Enter (see Figure 3).

Figure 3 Enter $(PROJCT_DIR/lib/IPTools in the Import Paths field



▼ Swift Compiler - Search Paths

Setting	mHealthTest
▶ Import Paths	$(PROJECT_DIR)/lib/IPTools

$(PROJECT_DIR)/lib/IPTools non-recursive

8. In the **Search** field, replace SWIFT_INCLUDE with SWIFT_VERSION and press Enter.
9. As the **Swift Language Version,** select Swift 5.0.

Use the mHealth Connector Cloud Service Client Library API

To initialize the library and send the data to mHealth Connector, the mobile application must call the Client Library API methods in the order listed below.

Initialization

Set the device bootstrap parameters required to onboard a device on the mHealth platform.

Once a virtual device is registered in the mHealth platform for a subject, a unique client access code and one-time passcode is created for the device to enable physical device onboarding. These two parameters need to be set to call the **setBootstrapParams** method on the **HsmHealthDeviceManager** instance.

Method:

HsmHealthDeviceManager.shared.setBootstrapParam(url: String, accessId: String?, oneTimePassword: String?)

Arguments:

- **Url—**mHealth cloud device bootstrap webservice URL.
- **accessId—**Unique access code created when a virtual device is created in the mHealth application for a specific subject and an eSource.
- **OneTimePassword—**One-time password provided for the device onboarding process.

Return type: Void

Activation

A device cannot send or receive data until activated in the mHealth Connector with its virtual device model. After setting the bootstrap parameters, a device can be activated by calling **activate** method on the **HsmHealthDeviceManager** instance. The activation process downloads device activation parameters from mHealth Connector and activates the device in the cloud, establishing a secure communication channel between the application running on the physical device and mHealth Connector.

Method:

HsmHealthDeviceManager.shared.activate(callback: (Bool, HsError?) --> Void) --> Void)

Arguments:

- **callback—**This method returns the status of the activation process. On successful activation of the device, the Boolean value is set to true and HsError is nil. On error conditions, an HsError instance is returned with cause of error. Call the **send** API only when this callback indicates that a device has been activated successfully.

Return type: Void

Example:

```
let hsDeviceMgr = HsmHealthDeviceManager.shared
hsDeviceMgr.activate { (activated, hsError) in
    if activated {
       //send message
    } else {
       // error
    }
}
```

Data Transmission

To send data from your mobile application to mHealth Connector, use the **send** method in the **HsmHealthDeviceManager** class.

The data must be in JSON format (key-value pairs) and conform to the message structure and message format for a message URN defined in a device model. When called, this method will transmit the message to mHealth Connector.

Any message not containing the key attributes listed below will result in a message format error.

- **KEY_EVENT_DATETIME_UTC—** Event timestamp in yyyy-mm-ddThh:mi:ssZ format (ISO 8601 datetime format).
- **KEY_TRIALID** —Subject/trial Id value set via device bootstrap process.
- **KEY_UID—**Surrogate patient/subject Id value set via device bootstrap process.

Method:

HsmHealthDeviceManager.shared.send(_ msg:String?, msgFormat:String?,

msgType: HsMessageType = HsMessageType.Data,

msgReliability: HsMessageReliability =

HsMessageReliability.guaranteed_delivery,

callback: @escaping (Bool, String?)->Void)

Arguments:

- **msg—**JSON payload as string.
- **msgFormat** —Message URN. This value must match the message URN specified in the device data model created in mHealth Connector for this study.
- **msgType** —Message type can be Data or Alert type. Default type is Data.
- **msgReliability** —Value can be guaranteed_delivery or best_effort. By default, the library caches the messages until it is successfully transmitted to mHealth Connector. Message reliability is guaranteed only on activated devices.
- **Callback—**The caller receives status of the message transfer via this method. The Boolean value, True, indicates successful transfer of the message to mHealth Connector.

If the message is missing key attributes, as indicated above, or does not conform to JSON syntax, the result string includes the error description. A nil value for the result string indicates that the message was sent successfully to mHealth Connector.

Return: Void

Example: The example shows a JSON message for a pulse oximeter medical device.

- **msg:**

```
"{\"KEY_PULSE\":511,\"KEY_SP02\":\"127\",\"KEY_UID\":\"Myuid\",
\"KEY_DID\":\"pulseoxy\",\"KEY_SN\":\"pulseoxy\",
\"KEY_EVENT_DATETIME_UTC\":2018-09-30T22:01:30Z\",\"KEY_TRIALID\":\"mhealth\"}"
```

- **msgFormat:**

```
"urn:oraclehs:iot:device:data:hsgbupulseoxystrm3:mhealth"

let hsDeviceMgr = HsmHealthDeviceManager.shared
hsDeviceMgr.activate {(activated, hsError) in

  if activated {
                   let msgFormat =
            "urn:oraclehs:iot:device:data:hsgbupulseoxystrm3:attributes"
                     hsDeviceMgr.send(msg,
                                   msgFormat: msgFormat,
                            callback: { (sent, error) in
  if sent {
     print ("Message sent successfully")

  } else {
     print ("Message sent error \(error)")
  }

        } else {
           // error
        }
  }
```

For each device model defined in the mHealth Connector study container, a unique data/alert message URN is created for each message/data sent from a device application to the mHealth Connector. The one included here is the message URN.

Once the **send** method is called for a message, the Client Library will attempt to send the message to mHealth Connector. If the Client Library is not able to make an internet connection, **send** will return an error message in the **callback** method.

Uploading a Large Object

If the message payload needs to include a large object, such as an image or video file, upload the object calling the **uploadObject** method of the **HsmHealthDeviceManager** class. Once the file is uploaded, a URI of the object in mHealth Connector is returned. Insert this URI as the value of the data/alert message attribute of the message.

Method:

HsmHealthDeviceManager.shared.uploadObject(inputFile: URL, objectName: String?, contentType: String?, callback: (HsExternalObject?, HsError?))

Arguments:

- **inputFile—**Image or video file URL on the device.
- **objectName—**Name of the object if it needs to be different than the file name in the inputFile argument. Otherwise, set this value to nil.
- **contentType—**Mime-type of the file. If this value is nil, the mHealth platform will assign an appropriate value.
- **callback—**This method returns the status of the upload. If **HsExternalObject** is not null, the **externalObjectURL** property of this object returns the uploaded file URL in mHealth Connector. If the upload fails, the HsError object returns error description and potential cause of error.

Return type: Void

Example:

```
let hsDeviceMgr = HsmHealthDeviceManager.shared
hsDeviceMgr.uploadObject(inputFile: imgFileUrl,
                objectName: nil, contentType: nil) { (hsObject, hsError)
            in
if hsObject != nil {
                print("Object Url:\(hsObject?.externalObjectURL)")
}
   }
```

Enabling Device Action

mHealth Connector's message processing engine, or an enterprise application registered with mHealth Connector as a data target, can analyze the message, and send a text message back to the source (mobile application). This feature is called *device action*. To enable device application:

1. Device model[s] must be configured with device action named *feedback*. An Oracle representative working on the project will help you configure the device model.
2. The mobile application must register a delegate/callback with mHealth Client Library to receive feedback messages from the mHealth Connector. Within the delegate/callback, message received can be manifested as a local notification or to drive an application workflow.

Method:

HsmHealthDeviceManager.shared.onAction(actionHandler: HsActionHandler)

Arguments:

- **actionHandler—**Implementation of HsActionHandler interface.

Return type: Void

Troubleshooting

- To activate a new virtual device, it must be created in mHealth Connector following the subject registration process.
- The library activates a device for all device models created in the mHealth study container at the activation time. If the device model URN is changed after the device activation, the library will not detect that automatically. Therefore, a new virtual device is created in mHealth Connector and that device must be activated so that new devices models in the mHealth study container are visible to the mobile application.
- If the bootstrap URL is changed after activating the device, your mobile application may not be able to send the messages to mHealth Connector.

Working with the Android Version of the mHealth Connector Cloud Service Client Library

This section describes importing the mHealth Connector Cloud Service Client Library to an Android-based application and the APIs to use to send data collected from a medical device to mHealth Connector Cloud Service. For iOS-based applications, see Working with the iOS Version of the mHealth Connector Cloud Service Client Library.

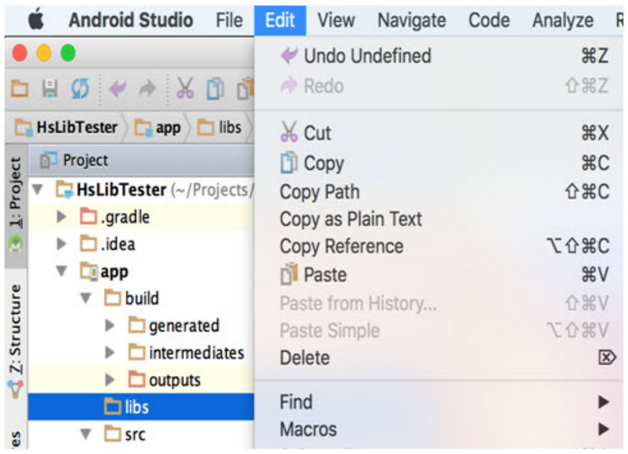
Technical Requirements

Android Studio 3.x and Android SDK Platform-tools 29.x or above must be installed. The Client Library is supported in Android OS versions 7.1 and above.

Import the mHealth Connector Cloud Service Client to the Android Application

1. Unzip the downloaded mHealth Android Client Library package to a local directory on the development machine.
2. Open Android Studio and create/open your mobile application project.
3. Add the file *hsiotlib.aar* to the project *libs* directory, located under the app folder. To do this, copy the file to the clipboard and, from the Edit menu, select Paste (see Figure 4).

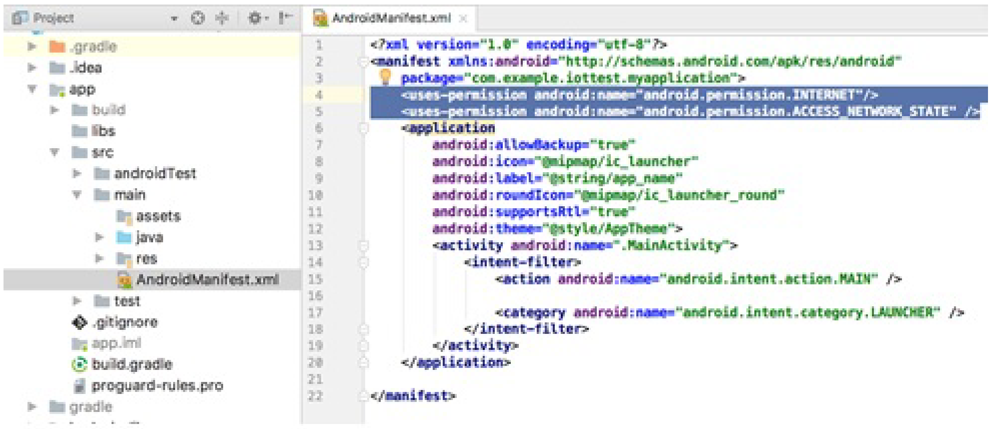
Figure 4 Add the hsiotlib.aar file to the app folder



4. Include the application permissions required to use the library.
   - `android.permission.INTERNET`
   - `android.permission.ACCESS_NETWORK_STATE`
5. To add Internet Permission to the app, open the file, *AndroidManifest.xml*, in the *app/src/main* folder.
6. Add the following lines to this file before the <Application tag. See Figure 5.

```
<uses-permission androd:name="android.permission.INTERNET"/>
<uses-permission androd:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Figure 5 Add permissions to the app



Project

- .gradle
- .idea
- app
  - build
  - libs
  - src
    - androidTest
    - main
      - assets
      - java
      - res
      - AndroidManifest.xml
    - test
  - .gitignore
  - app.iml
  - build.gradle
  - proguard-rules.pro
- gradle

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.iottest.myapplication">
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Using the mHealth Connector Cloud Service Client Library API

To initialize the Client Library and send the data to mHealth Connector, the mobile device must call Client Library API methods in the order listed below.

Initialization

Set the device bootstrap parameters required to onboard a device on the mHealth platform.

Once a virtual device is registered in the mHealth platform for a subject, a unique client access code and one-time passcode is created for the device to enable physical device onboarding. These two parameters need to be set to call the **setBootstrapParams** method on the **HsmHealthDeviceManager** instance.

Method:

HsmHealthDeviceManager.getSharedInstance().deviceBootstrap(final String url, final String clientAccessId, final String oneTimePassword, final Context context)

Arguments:

- **dthUrl—**mHealth cloud device bootstrap webservice URL.
- **clientAccessID—**Unique access code created when a virtual device is created in the mHealth application for a specific subject and an eSource.
- **oneTimePassword—**One-time password provided for the device onboarding process.
- **context—**Android application context.

Return type: Void

Example:

```
iotAccessManager.setBootstrapParams("https://mhealth.oraclecloud.com/deviceBoot
strap", "clientId", "otp", getApplicationContext() );
```

Activation

A device cannot send or receive data until activated in mHealth Connector with its virtual device model. After setting the bootstrap parameters, a device can be activated by calling the **activate** method on the **HsmHealthDeviceManager** instance. The activation process downloads device activation parameters from mHealth Connector and activates the device in the cloud, establishing a secure communication channel between the application running on the physical device and mHealth Connector.

Method:

HsmHealthDeviceManager.getSharedInstance().activate(HsActivationCallback callback) throws HsError

Arguments:

- **callback**—This method returns the status of the activation process. On successful activation of the device, the Boolean value is set to true and HsError is nil. On error conditions, an HsError instance is returned with cause of error. Call the **send** API only when this callback indicates that a device has been activated successfully.

Return type: Void

Example:

```
try{
HsmHealthDeviceManager.gerSharedInstance().activate(new HsActivationCallback() {
    @Override
    public void status(boolean success, HsError error) {
        if (success) {
    // send data
        } else {
        //print error
        }
    }
});

} catch (HsError e) {

}
```

Data Transmission

To send data from your mobile application to mHealth Connector, use the **send** method in the **HsmHealthDeviceManager** class.

The data must be in JSON format (key-value pairs) and conform to the message structure and message format for a message URN defined in a device model. When called, this method will transmit the message to mHealth Connector.

Any message not containing the key attributes listed below will result in a message format error.

- **KEY_EVENT_DATETIME_UTC—** Event timestamp in yyyy-mm-ddThh:mi:ssZ format (ISO 8601 datetime format).
- **KEY_TRIALID —**Trial/study Id value set via device bootstrap process.
- **KEY_UID—**Surrogate patient /subject Id value set via device bootstrap process.

Method:

HsmHealthDeviceManager.getSharedInstance().send(string jsonMsg, string dataFormat, HsMessageType hsMessageType, HsMessageReliability hsMessageReliability, HsDispatchCallback callback) throws HsError

HsmHealthDeviceManager.getSharedInstance().send(string jsonMsg, string dataFormat, HsDispatchCallback callback) throws HsError

Arguments:

- **jsonMsg**—JSON payload as string.
- **dataFormat**—Message URN. This value must match the message URN specified in the device data model created in mHealth Connector.
- **hsMessageType**—Message type. Can be Data or Alert. In the overloaded method where this value is not set, message type will be set as Data.
- **hsMessageReliability**—Value can be guaranteed_delivery or best_effort. Message reliability is guaranteed only on activated devices. In the overloaded method, where this value is not set, the library will use the guaranteed_delivery value.
- **callback**—The caller will get status of the message transfer via this method. Implement the **messageStatus** method to receive the status.

Return: Void

Exception: HSError

Example: The example shows a JSON message for a pulse oximeter medical device.

- **dataMsg:**

```
"{\"KEY_PULSE\":73,\"KEY_SP02\":95,\"KEY_UID\":\"Myuid\",
\"KEY_DID\":\"pulseoxy\",\"KEY_SN\":\"pulseoxy\",
\"KEY_EVENT_DATETIME_UTC\":"2017-11-01T12:02:01Z",\"KEY_TRIALID\":\"mhealth\"}"
```

- **dataFormat:**

```
"urn:oraclehs:iot:device:data:hsgbupulseoxystrm3:mhealth"
```

- **Code Snippet:**

```
HsmHealthDeviceManager.getSharedInstance().send(
     dataMsg,
     dataFormat,
     new HsDispatchCallback() {
         @Override
         public void messageStatus(boolean isSuccess, String msg)   {
             if(isSuccess) {
                            // send success
             } else {
               /error
                     }
         }
     });
```

Uploading a Large Object

If the message payload needs to include a large object, such as an image or video file, upload the object calling the **uploadObject** method of the **HsmHealthDeviceManager** class. Once the file is uploaded, a URI of the object in mHealth Connector is returned. Insert this URI as the value of the data/alert message attribute of the message.

Method:

HsmHealthDeviceManager.getSharedInstance().uploadObject(File inputFile, String objectName, HsUploadCallback callback)

Arguments:

- **inputFile**—Image or video file URL on the device.
- **objectName**—Name of the object.
- **callback**—Returns the status of the file uploaded and the URL of the object in mHealth Connector. Implement the **status** method to receive the status. If the file was uploaded successfully, the **HsExternalObject** instance will have the object URL. Once the object is uploaded, the caller should set the URL in the data/alert message explicitly.

Return: Void

Exception: HSError

Example:

```
HsmHealthDeviceManager.getSharedInstance().uploadObject(file, "objectName", new
HsUploadCallback() {
    @Override
    public void status(HsExternalObject externalObject, HsError error) {
        if (error == null && externalObject != null) {
            String objectUrl = externalObject.getObjectURL();
      //set object URL in the message
            JSONObject jsonMsg = json.put("ImgUrl", objectUrl);
            //send message
            try {
               iotAccessManager.send(
                   jsonMsg.toString(),
                   dataMsgFormat,
                   new HsDispatchCallback() {
                     @Override
                     public void messageStatus(boolean isSuccess, String msg)
                      {sendCallback(isSuccess, msg);
                     }
                   });
            }catch (HsError hsError) {
                Log.e(TAG, hsError.getMessage());
            }
        } else {
            if (error != null) {
                Log.e(TAG, error.getMessage());
            }
        }
    }
});
```

Enabling Device Action

mHealth Connector's message processing engine, or an enterprise application registered with mHealth Connector as a data target, can analyze the message, and send a text message back to the source (mobile application). This feature is called *device action*. To enable device application:

1. Device model[s] must be configured with device action named *feedback*. An Oracle representative working on the project will help you configure the device model.

2. The mobile application must register a delegate/callback with mHealth Client Library to receive feedback messages from the mHealth Connector. Within the delegate/callback, message received can be manifested as a local notification or to drive an application workflow.

Method:

HsmHealthDeviceManager. getSharedInstance().onAction(HsActionHandler actionHandler)

Arguments:

- **actionHandler—**Implementation of HsActionHandler interface.

Return type: Void

Troubleshooting

- To activate a new virtual device, it must be created in mHealth Connector following the subject registration process.
- The library activates a device for all device models created in the mHealth study container at the activation time. If the device model URN is changed after the device activation, the library will not detect that automatically. Therefore, a new virtual device is created in mHealth Connector and that device must be activated so that new devices models in the mHealth study container are visible to the mobile application.
- If the bootstrap URL is changed after activating the device, your mobile application may not be able to send the messages to mHealth Connector.

Secure Development Guidelines

This section provides guidelines and recommendations for using and developing mobile applications using the mHealth Connector Cloud Service Client Library API published by the mHealth Connector Cloud Service platform. The following are recommended practices for secure usage of this API.

Access Control

mHealth Connector Cloud Service assumes that the invoker has the permissions to access every requested resource and representation or entity returned. Therefore, call mHealth Connector Cloud Service Client Library methods after authenticating the mobile application user.

Securely Storing mHealth Connector Cloud Service Bootstrap Parameters

The mobile application must ensure that all mHealth Connector Cloud Service bootstrap parameters are stored securely (on the device or any back-end service) by the mobile application.

Data Transmission Security

The mobile application must use a secure communication channel (TLS 1.2 or above) for communication with back-end services.

Using Known Vulnerable Components

The mHealth Connector Cloud Service Client Library is constantly updated with the latest security fixes and patches. Oracle recommends that developers using the mHealth Connector Cloud Service Client Library do the same.

Documentation Accessibility

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

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Integrating the Client Library with a Mobile Application, Release 1.4.0.1
Part Number F25623-01