

HQME D2 SDK – Quick Start Guide

Contents

Introduction	1
SDK Architecture	1
Installing the SDK.....	2
Unzip the Package	3
Compile the Code	3
Install HQME Client and Run Sample.....	4
Integrating with the Library	5
Step by step sample	6
Connect to the HQME client	6
Schedule a download.....	7
Receive download notification.....	8
Use downloaded item.....	8

Introduction

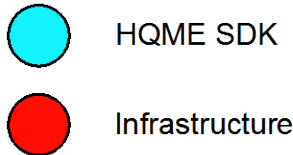
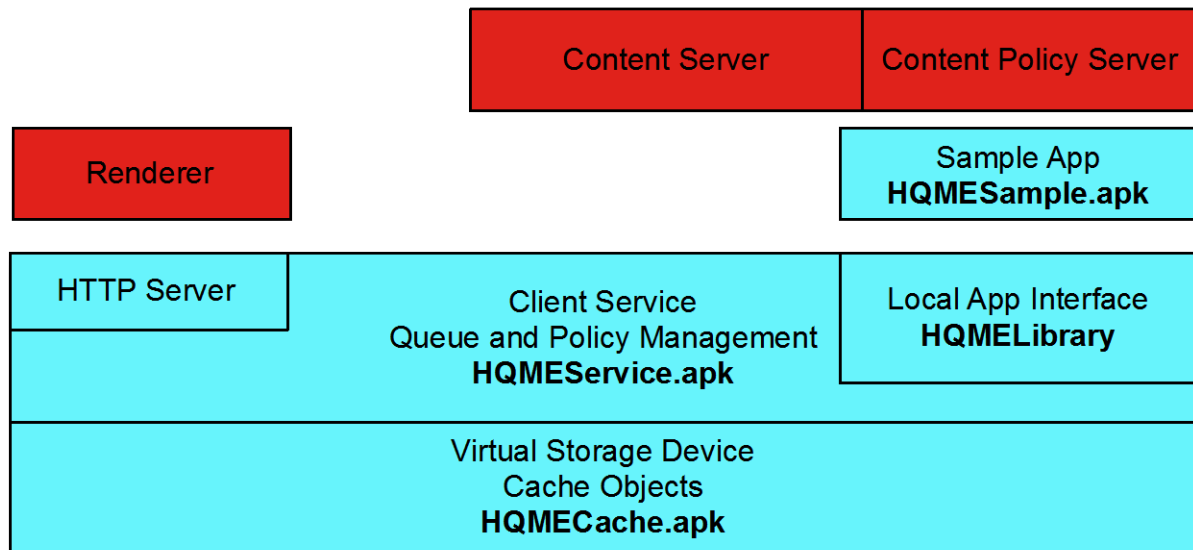
This guide is intended for experienced Android developers. It describes the process of integrating the HQME SDK into an Android handset, building the SDK source code, and integrating the HQME SDK Library into a project. The example illustrated in this document covers the basic use case of connecting to the HQME client, scheduling a download, receiving a notification once the download is completed and using the downloaded item. A fully working sample Android project is included in the SDK package and this guide should be used in conjunction with the sample project.

Only the essential parts of the code are included in this guide and in the sample project for the sake of brevity and they should not be used in production code as is.

SDK Architecture

The HQME D2 SDK provides a partial reference implementation of the proposed IEEE standard P2200 that defines a set of objects and method interfaces for queuing, downloading, caching and playing content from local storage. The SDK implements some of the features described in the P2200 draft standard on the Android platform.

The HQME SDK architecture is depicted below:



The HQMEService.apk package corresponds to the P2200 Client in the IEEE P2200 reference architecture. The HQMECache.apk package is a minimal Virtual Storage Device. HQME SDK supports multiple Virtual Storage Devices using a plug-in architecture. (The plug-in system includes additional functions for registration of new VSDs – these functions are not defined in the P2200 draft standard.)

Installing the SDK

This section describes how to install the HQME SDK services on an Android handset and integrate the SDK into an application.

Proceed through this document in the following order. When you are done, you will have the HQME Service installed on your device, you will be able to build and run the sample application, and you will have installed the sample project on your development system so you can begin integration.

1. Prepare development: If you have not already done so, install the following on your development system.
 - [Eclipse IDE for Java Developers](#)
 - [Android SDK](#)
 - [ADT Plugin for Eclipse](#) (includes USB driver for Android devices if required for Windows)
2. **Unzip** the HQME Service Integration for Android™ Software Development Kit package

3. **Compile** the HQME SDK applications (optional)
4. **Install** the HQME Service APK, HQME cache APK and sample application on the handset and run the sample.
5. **Import** sample project into Eclipse IDE.

Unzip the Package

Unzip the distribution file to your development system; be sure to maintain the same folder hierarchy. *SanDisk strongly recommends avoiding spaces in the full path to the un-zipped directory.* The following lists the folders and files in the SDK.

Folder Files	Description
projects/HqmeSample	Sample program package for Android
projects/HqmeLibrary	HQME Library (to be linked into calling applications)
projects/HqmeCache	Sample Virtual Storage Device (VSD)
projects/HqmeService	The HQME Service (P2200 Client) Android application.
docs	Document files (including this one)
apps	Pre-built APK files

Compile the Code

The HQME SDK supports two methods of compiling the code:

1. Using Eclipse. This method is ideal for continuing development.
2. Using Ant and Android command line utilities. This method allows automatic build integration.

To compile using Eclipse, create Eclipse projects for each of the directories under the projects/ directory. HqmeLibrary is a Library project and should be included as a library in the other projects.

Compiling code using Ant and Android command line utilities

These instructions apply to HqmeSample, HqmeCache and HqmeService projects. HqmeLibrary is a reference project and used by all the other projects as a library.

1. Update project/default.properties file to add a library reference to HqmeLibrary project. Add a new line as shown below:

android.library.reference.1=../HqmeLibrary/

2. Run android command line utility from the project directory to generate build.xml file. (Note: This utility is located in platform-tools folder of the android sdk installation.) Example for HqmeSample project is shown below, change the name for other projects:

android update project --target android-8 --name HqmeSample --path .

3. Run ant from the project directory to build. Ant should be already installed and defined in your system path. To build a debug version run

ant debug

4. To build and unsigned version first change AndroidManifest.xml to set android:debuggable flag to false then run ant for the release target. Finally, you can sign the binaries with your production keys using jarsigner.

Note: Replace the hqmetest keystore with your production keystore for release. The hqmetest keystore is not secure.

ant release

***jarsigner -keystore hqmetest.keystore -storepass hqmetest -keypass hqmetest
HqmeSample-unsigned.apk***

Install HQME Client and Run Sample

In this section you install the HQME Client, cache (VSD) and sample application on the handset and run the application.

Install Applications

All the files you need to run the sample are in the bin folder.

Connect the handset to your development system. Use the adb utility supplied with the Android SDK from the command line to first confirm that the handset is attached and then install the applications.

- Confirm the handset is attached:
`adb devices`
- Install 'HqmeService' on the device from the distribution:
`adb install HqmeService.apk`
- Install Hqme Default VSD on the device from the distribution:

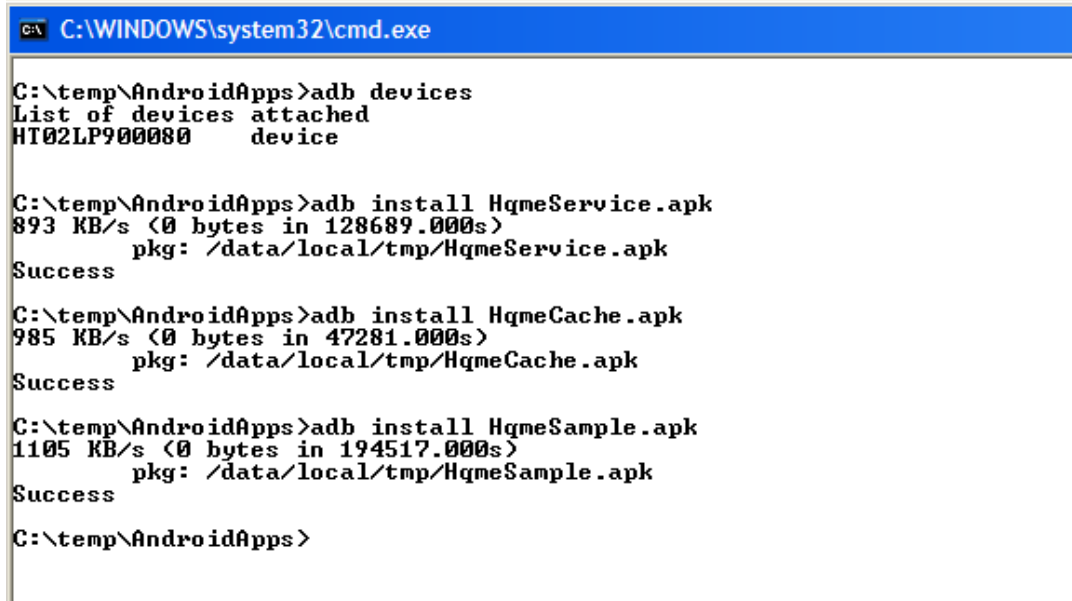
```
adb install HqmeCache.apk
```

- Install the 'HqmeSample' on the device

```
adb install HqmeSample.apk
```

The following figure illustrates the command sequence. Note that your device will have a different identification number.

Figure 1 – Client, VSD, and Sample Installation Commands



```
C:\WINDOWS\system32\cmd.exe

C:\temp\AndroidApps>adb devices
List of devices attached
HT02LP900080    device

C:\temp\AndroidApps>adb install HqmeService.apk
893 KB/s (0 bytes in 128689.000s)
pkg: /data/local/tmp/HqmeService.apk
Success

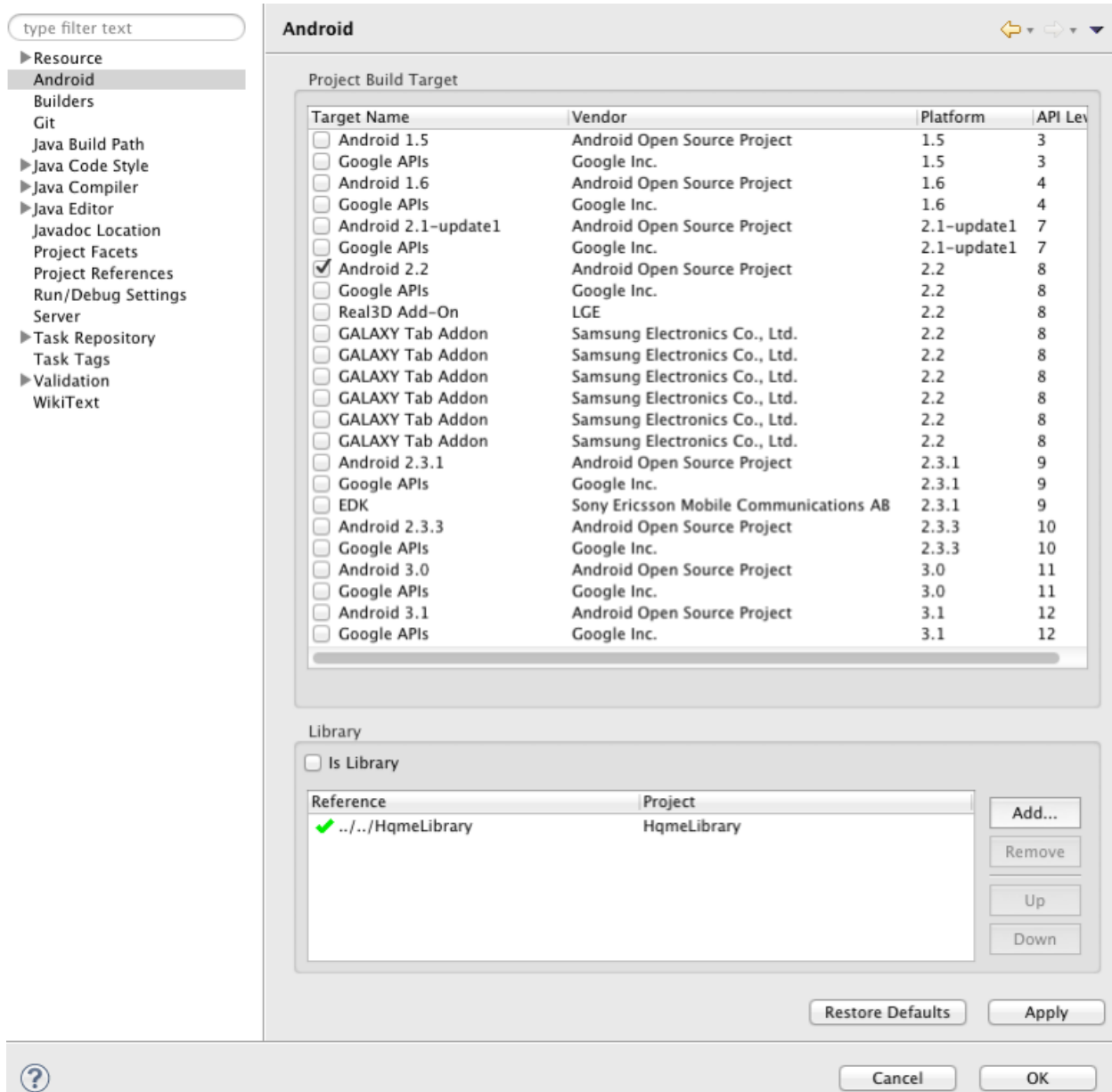
C:\temp\AndroidApps>adb install HqmeCache.apk
985 KB/s (0 bytes in 47281.000s)
pkg: /data/local/tmp/HqmeCache.apk
Success

C:\temp\AndroidApps>adb install HqmeSample.apk
1105 KB/s (0 bytes in 194517.000s)
pkg: /data/local/tmp/HqmeSample.apk
Success

C:\temp\AndroidApps>
```

Integrating with the Library

The HQME Library is a standard Android Library supporting Android 2.2 and above. To link to the library, insure that it is in your Eclipse workspace. Then, open the Android project configuration dialog and add the library using the Add button.



Step by step sample

(This illustrates code found in the HqmeSample project supplied with the SDK.)

Connect to the HQME client

1. Connect to RequestManager and ContentManager Services on your activities `onStart` function.

```
@Override
protected void onStart() {
    super.onStart();
    // Bind to the RequestManager service.
    if (sIsActive_RequestManager == false)
        bindService(new Intent(IRequestManager.class.getName()),
```

```

        mRequestManagerConnection, Context.BIND_AUTO_CREATE);

    // Bind to the ContentManager service
    if (sIsActive_ContentManager == false)
        bindService(new Intent(IStorageManager.class.getName()),
            mStorageManagerConnection, Context.BIND_AUTO_CREATE);
}

```

2. Sub class `ServiceConnection` and register for callbacks in `onServiceConnected` function.

```

private ServiceConnection mRequestManagerConnection = new
ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder
service) {
        sRequestManagerProxy.registerCallback(requestManagerCallbackProxy
        );
    }
};

```

Schedule a download

Sample code uses a template `QueueRequest` stored as an asset. This template policy has `#FIELDS#` that needs be replaced during run time. It defines two rules one for WiFi connection only and another one for WiFi or 3G with no roaming.

```

<QueueRequest name="tester">
    <Property key="REQPROP_SOURCE_URI">#URI#</Property>
    <Property key="REQPROP_STORE_NAME">#FILENAME#</Property>
    <Property key="REQPROP_TYPE">#MIMETYPE#</Property>
    <Property key="REQPROP_TOTAL_LENGTH">0</Property>
    <Property key="REQPROP_BROADCAST_INTENT">#RECEIVER#</Property>
    <Property key="REQPROP_POLICY">
        <Policy name="simplePolicy">
            <Download>//Rule[@name="#RULENAME#"]</Download>
            <Cache>true()</Cache>
            <Rule name="WiFiOnly">
                <Property key="RULE_CONNECTION_TYPE">LAN WAN</Property>
            </Rule>
            <Rule name="WiFi3GNoRoaming">
                <Property key="RULE_CONNECTION_TYPE">LAN WAN CELL3G</Property>
                <Property key="RULE_ROAMING">false</Property>
            </Rule>
        </Policy>
    </Property>
</QueueRequest>

```

3. Read template `QueueRequest` asset , fill in the fields and submit

```

public void submitRequest() {
    // get QueueRequest XML template from assets.
    String queueRequestXml = getQRTemplate();

    // Get URI from the UI.
    String uri = editURI.getText().toString();
}

```

```
// Guess extension and mime type from URI.
String extension = MimeTypeMap.getFileExtensionFromUrl(uri);
String mime =
MimeTypeMap.getSingleton().getMimeTypeFromExtension(extension);

// Replace fields
queueRequestXml = queueRequestXml.replace("#URI#", uri);
queueRequestXml = queueRequestXml.replace("#RECEIVER#",
getPackageName() + ".QR_COMPLETED");
queueRequestXml = queueRequestXml.replace("#FILENAME#", "sample_" +
Integer.toString(sRandomGenerator.nextInt(0x7fffffff)) + "." + extension);
queueRequestXml = queueRequestXml.replace("#MIMETYPE#", mime);
if (radioWiFiAnd3G.isChecked())
    queueRequestXml = queueRequestXml.replace("#RULENAME#",
"WiFi3GNoRoaming");
else
    queueRequestXml = queueRequestXml.replace("#RULENAME#",
"WiFiOnly");

// Submit QueueRequest
IQueueRequest qr =
sRequestManagerProxy.createQueueRequestXml(queueRequestXml);
long queueRequestId = sRequestManagerProxy.submitRequest(qr);
}
```

Receive download notification

4. Implement your `BroadcastReceiver`'s `onReceive` function to handle HQME events.

```
public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    if (action.equals(context.getPackageName() + ".QR_COMPLETED")) {
        long queueRequestId = intent.getLongExtra("index", -1);
        // Call the app-specific function to handle this event.
        updateStatus(queueRequestId);
    }
}
```

Use downloaded item

5. Get streaming URI for the recently downloaded item and launch it.

```
IQueueRequest qr = sRequestManagerProxy.getRequest(queueRequestId);
if (qr != null) {
    String path = qr.getProperty("REQPROP_STORE_NAME");

    int[] storageIds = sStorageManagerProxy.getStorageIds(null);
    if (storageIds != null) {
        for (int i = 0; i < storageIds.length; i++) {
            IVSD store = sStorageManagerProxy.getStorage(storageIds[i]);
            if (store != null) {
                IContentObject targetObject = store.getObject(path);
                if (targetObject != null) {
                    String streamingUri = targetObject.getStreamingUri();

                    // Launch URI with default handler
                }
            }
        }
    }
}
```



```
        final Intent intent = new  
Intent(Intent.ACTION_VIEW).setData(Uri.parse(streamingUri));  
        startActivity(intent);  
  
        break;  
    }  
    }  
    }  
    }  
}
```