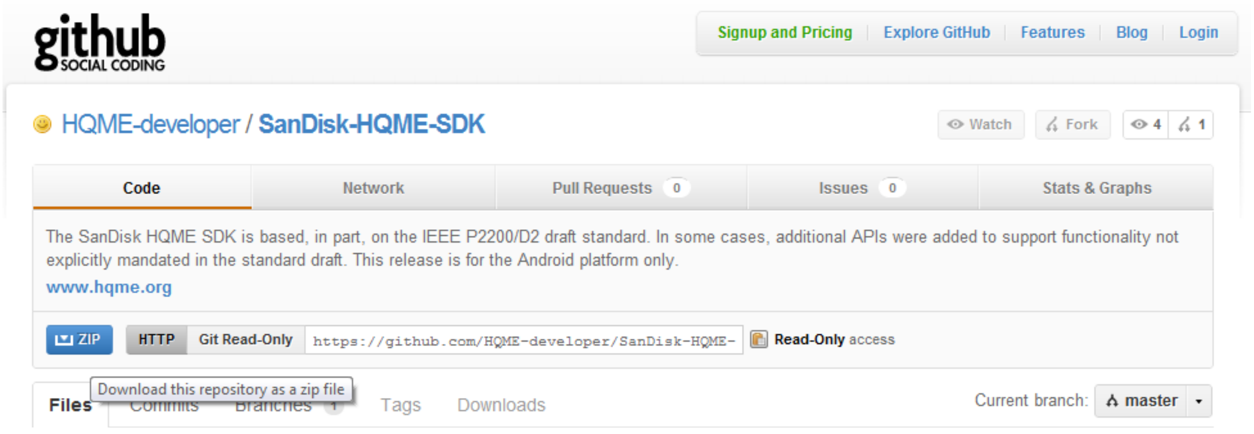
HQME Developer Day – SDK Getting Started

1. Retrieve the open source HQME SDK from the github.com repository

1.1. Browse to www.github.com

1.2. Search for "**hqme**"

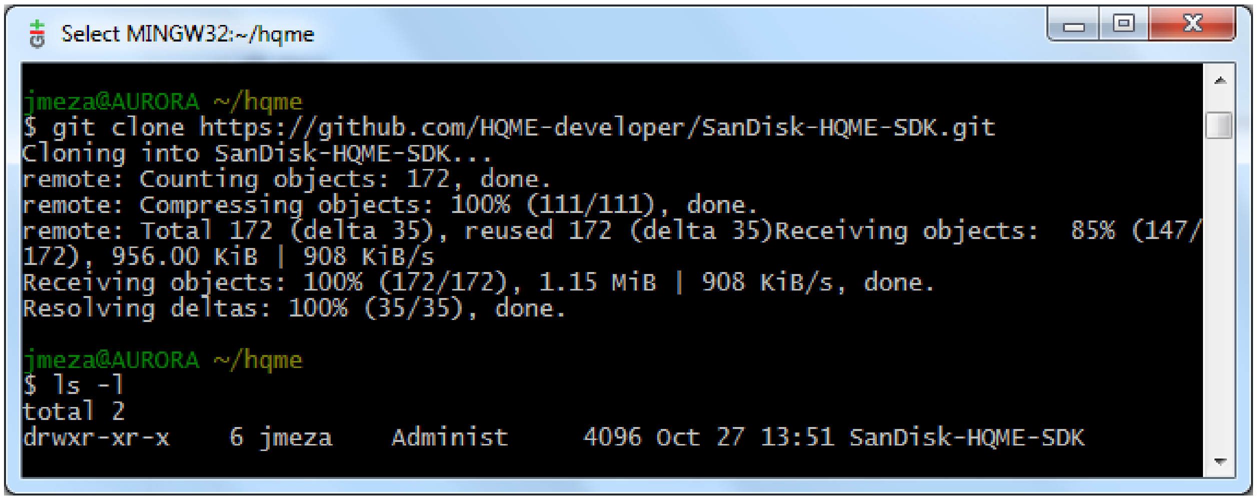
1.3. Download as Zipfile



The SanDisk HQME SDK is based, in part, on the IEEE P2200/D2 draft standard. In some cases, additional APIs were added to support functionality not explicitly mandated in the standard draft. This release is for the Android platform only.
www.hqme.org

https://github.com/HQME-developer/SanDisk-HQME-

1.4. OR clone the git repository:



```
jmeza@AURORA ~/hqme
$ git clone https://github.com/HQME-developer/SanDisk-HQME-SDK.git
Cloning into SanDisk-HQME-SDK...
remote: Counting objects: 172, done.
remote: Compressing objects: 100% (111/111), done.
remote: Total 172 (delta 35), reused 172 (delta 35)Receiving objects:  85% (147/
172), 956.00 KiB | 908 KiB/s
Receiving objects: 100% (172/172), 1.15 MiB | 908 KiB/s, done.
Resolving deltas: 100% (35/35), done.

jmeza@AURORA ~/hqme
$ ls -l
total 2
drwxr-xr-x    6 jmeza    Administ      4096 Oct 27 13:51 SanDisk-HQME-SDK
```

2. Review SDK contents

There are several documents to help get your started using the SDK. In the root SDK directory there is the **License** document and **Release Notes** (README.md - can open file with a text editor) document.

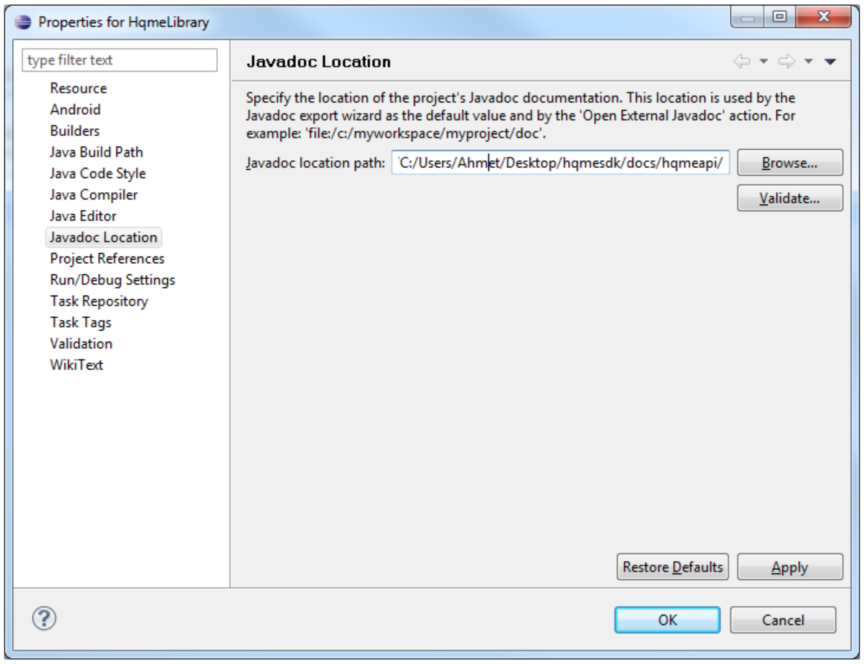
In the **docs** directory, there are two files, a zip file named **hqmeapi.zip** which contains the javadocs for the various SDK classes and a getting started document entitled, **HQME SDK Quick Start Guide.pdf.**

2.1.HQME SDK Javadocs

Extract the hqmeapi.zip file in the docs directory. The index.html file can be opened with any browser and is the starting point to discovering the details about particular classes, methods and data members of interest.

2.1.1. Integrating HQME SDK Javadocs into Eclipse

Eclipse provides a way to integrate your project's javadocs into the IDE for contextual information of the documented classes, methods and members. Integrating the newly extracted javadocs into eclipse is straight forward and easy.



Specify the location of the project's Javadoc documentation. This location is used by the Javadoc export wizard as the default value and by the 'Open External Javadoc' action. For example: 'file:/c:/myworkspace/myproject/doc'.

Javadoc location path: C:/Users/Ahmet/Desktop/hqmesdk/docs/hqmeapi/

a) Project Properties

Right click your project in the Project browser or use the Project->Properties menu to navigate to your properties dialog.

b) Set Javadoc Location in the Project Properties dialog

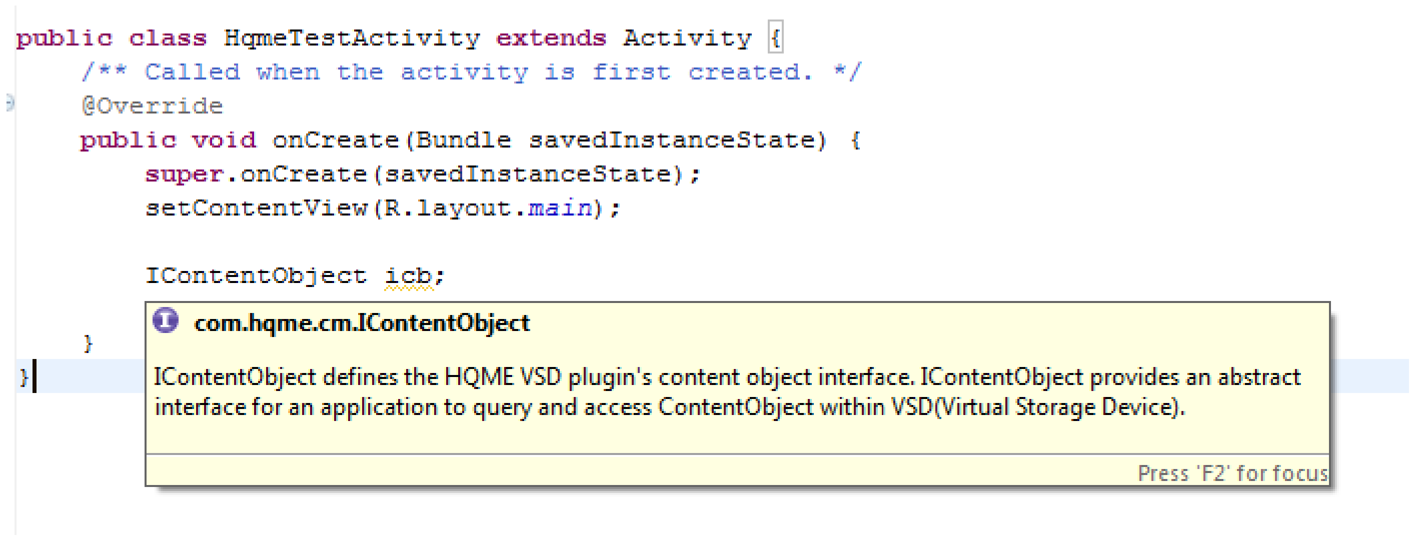
On the left hand side there is a property named, "**Javadoc Location**." When you select this property, the pane on the right changes to enable you to enter the directory location of the javadocs you just unzipped.

c) Browse to javadoc location

Using the browse button, a dialog appears which allows you to navigate to the directory location where you have unzipped your HQME SDK javadoc files. Select the appropriate directory and click OK. You can press the validate button to ensure that the appropriate directory has been selected. Click Apply and OK and you're done.

d) Eclipse integrated Javadocs in action

Now when hovering over classes, methods and data members, if documentation exists within the referenced javadocs, contextual information will be displayed to provide some additional information about the object of interest.



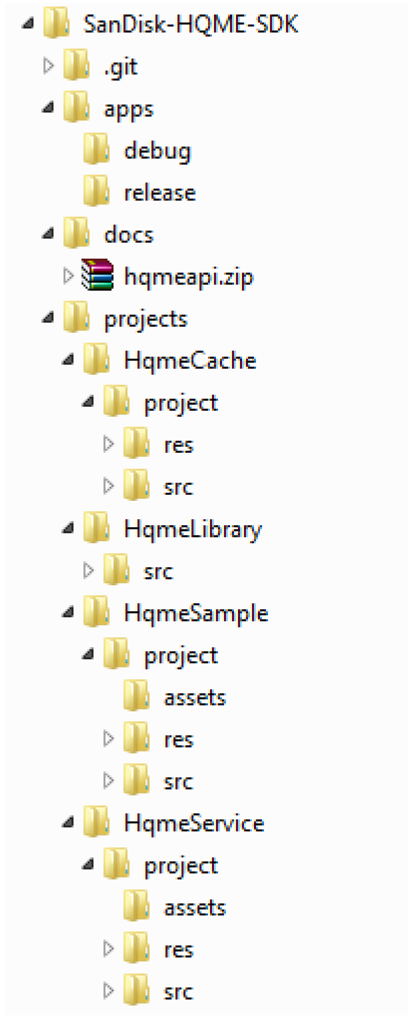
```
public class HqmeTestActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        IContentObject icb;
    }
}
```

com.hqme.cm.IContentObject

IContentObject defines the HQME VSD plugin's content object interface. IContentObject provides an abstract interface for an application to query and access ContentObject within VSD(Virtual Storage Device).

2.2.SDK directory structure

The directory structure is illustrated in the image below and at the root directory consists of three subdirectories, namely apps, docs and projects.



```
SanDisk-HQME-SDK
    .git
    apps
        debug
        release
    docs
        hqmeapi.zip
    projects
        HqmeCache
            project
                res
                src
        HqmeLibrary
            src
        HqmeSample
            project
                assets
                res
                src
        HqmeService
            project
                assets
                res
                src
```

The **apps** directory contains pre-compiled binaries compiled for Android SDK version 2.2.x (minimum SDK 8, Froyo). Both debug and release builds are available.

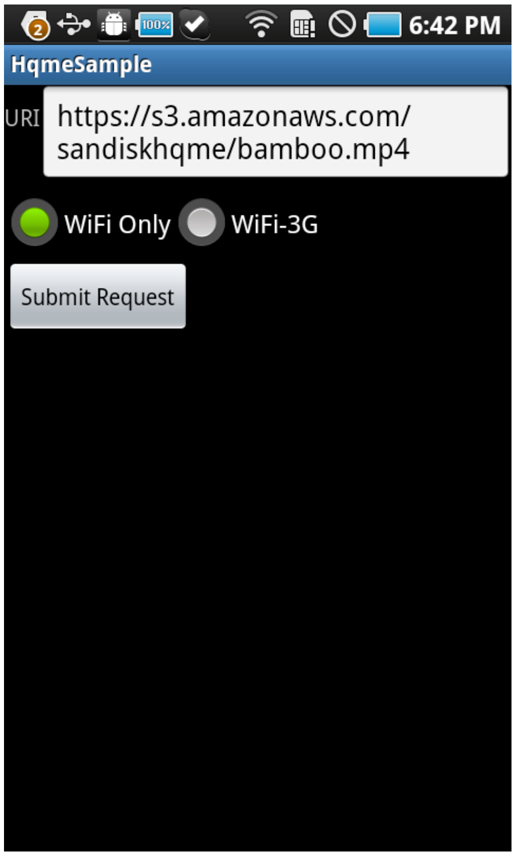
As described previously, the **docs** directory contains the javadocs files (zipped) and a Quick Start guide to help you on your way to using the SDK.

The **projects** directory contains four subdirectories:

- HqmeCache – A service which implements an interface for accessing cached content once retrieved.
- HqmeLibrary – A shared library between all projects, including your project which defines the interfaces to the HQME services and shared class definitions.
- HqmeSample – A sample application which utilizes the SDK by including the HqmeLibrary.
- HqmeService – A service which implements an interface for submitting requests and managing previously submitted requests.

3. (Optional) Install the pre-built binaries

The Quick Start guide describes how to install the pre-built, so will not be duplicated here, however, it might be a good step just to make sure that the HQME Services can be started and utilized by the calling application.



https://s3.amazonaws.com/sandiskhqme/bamboo.mp4

The application provides a simple illustration of how QueueRequests are created and submitted to the HQMEService, the callbacks received upon various events from both the HQMEService and HQMECache Services, and how to retrieve the cached content once a QueueRequest has completed.

To illustrate a simple policy, the user has an option to select under what network conditions the content should be downloaded, when connected via Wi-Fi Only or when connected either via Wi-Fi or 3G.

The URI textbox has a sample URI. You should supply a known URI of where some content resides. Upon completion, the content is immediately played back (video or audio).

4. Creating a new project to interface with the SDK

4.1. Create HQMELibrary project

The Quick Start guide uses the term import, however, we are not importing an existing project but rather creating a new Android project from existing source. This can be done for each of the directories in the projects directory.

The example given here is for creating the HQMELibrary Android project within the eclipse development environment.

4.1.1. Create HQMELibrary project

A. File->New Android Project

Select Android Project from the New Project Wizard and select Next.

B. New Android Project

Enter a project name, I used “HqmeLibrary” and select “Create Project From Existing Source.” Then browse to the directory containing the HqmeLibrary project (*sdk root*\projects\HqmeLibrary) and select Next.

C. Select Build Target

Choose an SDK target for your project; I used Android 2.2 and select Next.

D. Application Info

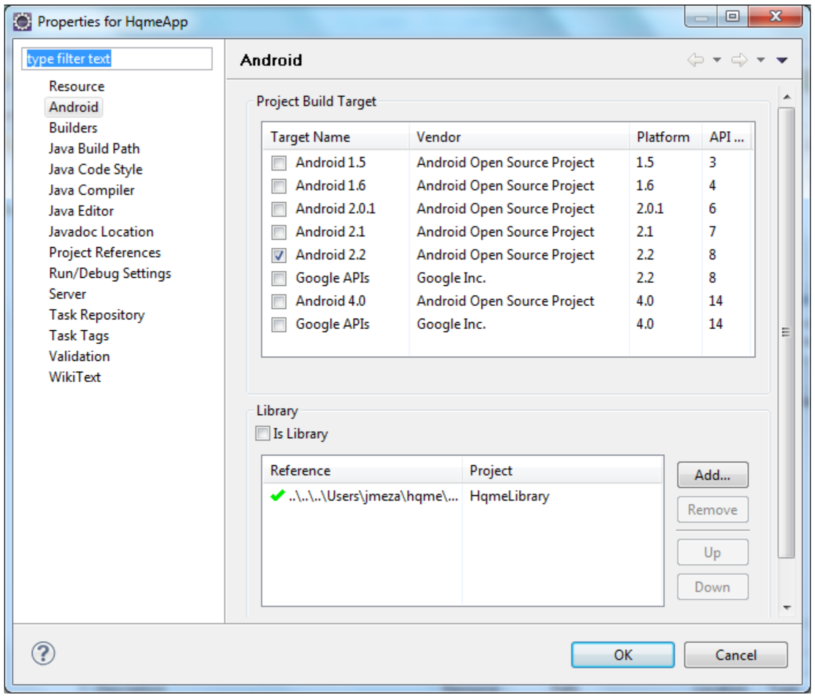
Change the Application Name and Package Name as needed and select Finish.

4.2.Create Your Android project

In a similar fashion, create your Android application as needed. In order to utilize the SDK, you will need to add the HQME SDK Library to your application as illustrated below:

A. Project properties

Right click on your project and select Properties. The Properties dialog for your project should come up which indicates your selected SDK version. In the lower part of the dialog you can add library dependencies. Select Add and another dialog opens up which should have the HqmeLibrary project we created previously as an option to select. Select the HqmeLibrary and click OK. The library should now appear in the textbox of your Properties Dialog indicating the newly added library as illustrated below.



Android

Project Build Target

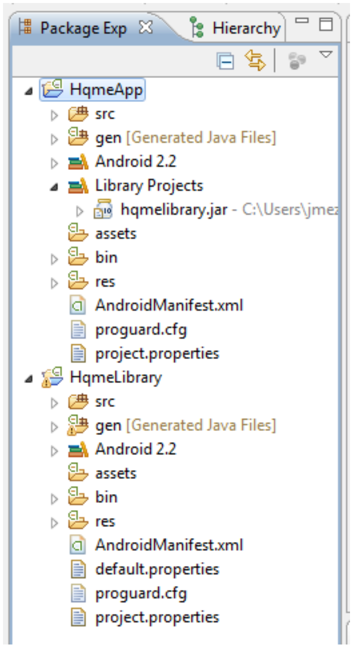
	Target Name	Vendor	Platform	API ...
☐	Android 1.5	Android Open Source Project	1.5	3
☐	Android 1.6	Android Open Source Project	1.6	4
☐	Android 2.0.1	Android Open Source Project	2.0.1	6
☐	Android 2.1	Android Open Source Project	2.1	7
☑	Android 2.2	Android Open Source Project	2.2	8
☐	Google APIs	Google Inc.	2.2	8
☐	Android 4.0	Android Open Source Project	4.0	14
☐	Google APIs	Google Inc.	4.0	14

Library

☐ Is Library

Reference	Project
✓ ..\..\..\Users\jmeza\hqme\...	HqmeLibrary

If everything is successful, your project explorer should look like this:



```
HqmeApp
  src
  gen [Generated Java Files]
  Android 2.2
  Library Projects
    hqmelibrary.jar - C:\Users\jmez
  assets
  bin
  res
  AndroidManifest.xml
  proguard.cfg
  project.properties
HqmeLibrary
  src
  gen [Generated Java Files]
  Android 2.2
  assets
  bin
  res
  AndroidManifest.xml
  default.properties
  proguard.cfg
  project.properties
```

5. Using the SDK from your application

5.1.Binding to the HQME Services

Your application can bind to the HQME Services by calling bindService(). When it does, it must provide an implementation of ServiceConnection, which monitors the connection with the service.

```
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);
    …

    }
```

In the sample application, several data members are created in the Activity to indicate if the connection to the HQME Services is active, these are sIsActive_RequestManager and sIsActive_ContentManager. These are set in the implementation of the ServiceConnections (mRequestManagerConnection and mStorageManagerConnection) when the connections are created or disconnected.

For more information on services, you can refer to the Google Android documentation at:
http://developer.android.com/guide/topics/fundamentals/bound-services.html

5.2. ServiceConnection implementations

```
private ServiceConnection mRequestManagerConnection = new ServiceConnection() {
   public void onServiceConnected(ComponentName className, IBinder service) {
   // This is called when the connection with the service has been established,
   // giving us the service object we can use to interact with the service. We
   // are communicating with our service through an IDL interface, so get a
   // client-side representation of that from the raw service object.

   sRequestManagerProxy = IRequestManager.Stub.asInterface(service);
   sIsActive_RequestManager = true;

   try {
       sRequestManagerProxy.registerCallback(requestManagerCallbackProxy);
       } catch (RemoteException fault) {
       Log.d(sTag, "onServiceConnected", fault);
       }

   getHandler().sendMessage(getHandler().obtainMessage(
      QUEUE_REQUEST_MGR_CONNECTED, 0, 0));

   }…
```

The call to getHandler is part of the sample application implementation such that the event handler is notified when the connection to the HQMEService is established.
ServiceConnection implementation continued…

```
public void onServiceDisconnected(ComponentName className) {
   sRequestManagerProxy = null;
   sIsActive_RequestManager = false;

   };
}
```

5.3.CallBack implementations (optional)

Your application will most likely want to listen to the various callback events from the HQME Services. In the sample application, the callbacks simply collect the information and then send a message to the consolidated event handler.

```
/**
 * This implementation can be used to receive callbacks from the RequestManager
 * service.
 */
private IRequestManagerCallback requestManagerCallbackProxy = new
   IRequestManagerCallback.Stub() {

   public void handleEvent(ReqEvents reqEvent) throws RemoteException {

      Message msg = getHandler().obtainMessage(REQ_CALLBACK_MSG);
      Bundle statusBundle = new Bundle();

      statusBundle.putParcelable("reqEvent", reqEvent);
      msg.setData(statusBundle);
      getHandler().sendMessage(msg);
      }

    };
```

5.4.Submit QueueRequest

There are two ways to build a QueueRequest, programmatically or passing in an XML document to the createQueueRequestXML method:

```
      IQueueRequest createQueueRequestXml(in String queueRequestXml);
```

The sample application uses the XML method, so the following illustrates an alternative method.

```
public void issueRequest()
   {
   try {
     IQueueRequest qr = getProxy().createQueueRequest();
     qr.setProperty("REQPROP_SOURCE_URI", "http://www.numbis.com/warPeace.txt");
     qr.setProperty("REQPROP_STORE_NAME", "WarAndPeace.txt");
     qr.setProperty("REQPROP_TYPE", "text/plain");
     qr.setProperty("REQPROP_TOTAL_LENGTH","3223627");
     qr.setProperty("VENDOR_TEST","Hello, Joe!");
     } catch (RemoteException e) {
     e.printStackTrace();
     }
    }
```

Accessing Cached Content

If your application registered for callbacks from the HqmeCache Service, a callback is issued when the when content on the VSD has changed. Once the cached content is available, your application can retrieve the content for playback. In this example, the content is
streamed:

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

         // New content is available. Find it using the VSD APIs
         int[] storageIds = getPluginProxy().getStorageIds(null);
         if (storageIds != null) {
            for (int i = 0; i < storageIds.length; i++) {
              IVSD store = getPluginProxy().getStorage(storageIds[i]);
              if (store != null) {
                 // Get a ContentObject reference to the newly available object.
                 IContentObject targetObject = store.getObject(path);
                 if (targetObject != null)
                    {
                    //Get the straming URI and playback using embedded player.
                    String streamingUri = targetObject.getStreamingUri();

                    sVideoView.setVideoPath(streamingUri);
                    sVideoView.start();
                    break;
                    }
                 }
            }
         }
     }
 } catch (Exception e1) {
    Log.d(sTag, "handleMessage, NEW_CONTENT", e1);
}
```