mobitv
CONNECTED MEDIA SOLUTIONS

CONFIDENTIAL INFORMATION: Do not disclose

MobiTV Connect SDK Developer Guide

v. 1.0

October 27, 2014

Contents

1 Foreword

1.1 About this Document

This document describes steps to implement MobiTV's Connect SDK for iOS and Android. This SDK is used to build iOS and Android mobile applications as well as a media player application which is loaded onto the MobiTV Connect device. The mobile application and media player application communicate with each other over the Controller and Receiver SDK.

1.2 About MobiTV, Inc.

MobiTV is a global leader in delivering live and on-demand video to any screen, connecting media reliably and securely anytime, anywhere, on any device. The company's end-to-end platform delivers a true TV everywhere experience that helps service providers reduce time to market and control costs associated with the deployment and operation of high concurrency, multiple platform services. MobiTV's connected media solutions solve the complexity of delivering video across networks, operating systems and devices while managing associated rights. The company efficiently optimizes for network conditions to deliver multiscreen media services that center on empowering the viewer. MobiTV's connected media solutions are tailored for IPTV operators, as well as mobile TV and over-the-top customers including AT&T, Deutsche Telekom, Sprint, T-Mobile, US Cellular, and Verizon among others.

1.3 SDK License Agreement

Use of this Software (the "SDK") is subject to execution of an **SDK LICENSE AGREEMENT** between **MobiTV, Inc.**, a Delaware corporation with principal offices at 6425 Christie Ave., 5th Floor, Emeryville, CA 94608 ("MobiTV") and the user ("Developer"). The terms of the License Agreement are as follows:

1. Use of the SDK and Software License. Subject to the terms and conditions of this Agreement, MobiTV grants Developer a personal, non-sublicensable, non-transferable, nonexclusive, royalty-free license to use the SDK in accordance with the documentation supplied by MobiTV solely for the purpose of utilizing the SDK to integrate Developer's proprietary application for use with the Android-based HDMI device that converts any television with an HDMI input into a smart TV (the "MobiTV Connect"). The SDK is licensed to Developer and not sold. Except as expressly set forth in Section 3 below, MobiTV shall at all times retain all title to and ownership of the SDK and all copies thereof. Developer agrees to use the SDK only for the Purpose, and not to reproduce or modify the SDK or any portion thereof. Developer shall not rent, sell, lease or otherwise transfer the SDK or any part thereof or use it for the benefit of a third party. Developer shall not remove, disable, circumvent or otherwise create or implement any workaround

to any copy protection, rights management or security features in or protecting the SDK. Developer shall not reverse assemble, reverse compile or reverse engineer the SDK, or otherwise attempt to discover any SDK source code or underlying Confidential Information (as defined in Section 2 below). Developer will not remove, obscure, or alter any proprietary rights notices (including copyright notices) contained on or within the SDK. Developer agrees to protect the SDK against any unauthorized or unlawful use or copying, using efforts at least consistent with the practices and procedures under which it protects its own most valuable proprietary information and materials. Without limiting the generality of the foregoing, except with respect to the Purpose, Developer shall not utilize the SDK in connection with the development or maintenance of any product or application or permit any third party to do so.

2. Confidentiality; Ownership. Developer acknowledges that, in the course of exercising the license under this Agreement, it may obtain proprietary and/or confidential information relating to the SDK, MobiTV Connect and/or MobiTV ("Confidential Information"). Such Confidential Information shall belong solely to MobiTV and includes, without limitation, the SDK, its features and mode of operation, trade secrets, know-how, inventions (whether or not patentable), techniques, processes, programs, ideas, algorithms, schematics, testing procedures, software design and architecture, computer code, internal documentation, design and function specifications, product requirements and other technical, business, product, marketing and financial information, plans and data. Confidential Information may not be shared with any third party or used for any purpose except the Purpose without the prior written consent of MobiTV. Developer will not export or re-export or allow the export or re-export of the SDK or any Confidential Information in violation of any applicable export law, restriction or regulation of the United States or any foreign agency or authority. Each party shall comply with all such law, restrictions and regulations. Developer hereby assigns to MobiTV any invention, work of authorship, mask work, idea, information, feedback or know-how (whether or not patentable) related to the SDK that is conceived, learned or reduced to practice in the course of exercising its license rights hereunder and intellectual property and industrial rights of any sort with respect thereto, all of which is MobiTV's Confidential Information. Developer agrees to take any action reasonably requested by MobiTV to evidence, perfect, obtain, maintain, enforce or defend the foregoing.

3. Open Source Software. Portions of the SDK may contain or be derived from software made available pursuant to a license that (i) permits access to the source code, or human readable form of the software code; (ii) permits free modification and redistribution of the software; and (iii) applies the same terms generally to all licensees, without discrimination based on field of use or technology platform ("Open Source Software License"). The terms and conditions governing the use of such portions of the SDK may be subject to the terms and conditions of the applicable Open Source Software License. To the extent that the terms and conditions of any such Open source Software License prohibit the application of the terms and conditions of this Agreement or any part hereof, then such software shall be subject to such Open Source Software License and this Agreement or portions of this Agreement (as applicable) shall not apply.

4. Warranty Disclaimer. The parties acknowledge that the SDK is provided by MobiTV "AS IS". MobiTV does not represent or warrant that the SDK will be uninterrupted or error free or meet any technical standard. MOBITV DISCLAIMS ALL WARRANTIES RELATING TO THE SDK, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES AGAINST INFRINGEMENT OF THIRD PARTY RIGHTS, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

5. Limitation of Liability. MOBITV SHALL NOT BE RESPONSIBLE OR LIABLE WITH RESPECT TO ANY SUBJECT MATTER OF THIS AGREEMENT UNDER ANY CONTRACT, NEGLIGENCE, STRICT LIABILITY OR OTHER THEORY (A) FOR LOSS OR INACCURACY OF DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, SERVICES OR TECHNOLOGY, OR (B) FOR ANY INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES INCLUDING, WITHOUT LIMITATION, LOSS OF REVENUE AND LOSS OF PROFITS. MOBITV SHALL NOT BE REPONSIBLE FOR ANY MATTER BEYOND ITS REASONABLE CONTROL.

6. Assignment. Developer shall not assign or transfer any of the rights or obligations arising under this Agreement. Any such attempted assignment or transfer shall be void and without effect. MobiTV may assign or transfer this Agreement without Developer's consent.

7. Indemnification. Developer agrees to indemnify, defend and hold harmless MobiTV and its officers, directors, employees, agents, affiliates, successors and assigns from and against any and all losses, damages, liabilities, deficiencies, claims, actions, judgments, settlements, interest, awards, penalties, fines, costs, or expenses of whatever kind, including attorneys' fees, arising from or relating to Developer's use or misuse of the SDK or Developer's breach of this Agreement.

8. Term and Termination. The term of this Agreement shall be one (1) year from the Effective Date, provided that the parties may mutually agree in writing (which may be via email) to extend the term for an additional period. This Agreement may be terminated by MobiTV, with or without cause immediately upon notice of any breach by Developer of the provisions of this Agreement. Upon termination, except for copies of the SDK which have already been integrated with the Developer application and distributed, the license granted hereunder will terminate and Developer shall promptly return the SDK, together with any and all documents, notes and other materials regarding the SDK and all Confidential Information and all copies and extracts of any of the foregoing to MobiTV.

9. Miscellaneous. The parties' rights and obligations pursuant to Sections 2 through and including 9 shall survive any termination or expiration of this Agreement. This Agreement constitutes the entire agreement between the parties pertaining to the subject matter hereof. Nothing in this Agreement shall be deemed or construed to confer

any rights of third party beneficiary on any person. Any waiver or failure to enforce any provision of this Agreement on one occasion will not be deemed to be a waiver of any provision on any other occasion. MobiTV may amend this Agreement by written notice to Developer. Any other amendment, waiver or modification of any provision of this Agreement will be effective only if in writing and signed by the parties. If any provision of this Agreement to be unenforceable, that provision of the Agreement will be enforced to the maximum extent permissible so as to affect the intent of the parties, and the remainder of this Agreement will continue in full force and effect. Nothing in this Agreement shall be construed to create a joint venture, partnership, employer-employee relationship, or agency relationship between the parties. This Agreement will for all purposes be governed by and interpreted in accordance with the laws of the State of California without giving effect to any conflict of laws principles that require the application of the laws of a different state. Any dispute arising out of or relating to this Agreement may be commenced in a state or federal court in Alameda County, California, and each party irrevocably submits to the jurisdiction and venue of such courts.

2 MobiTV Connect Overview

2.1 MobiTV Connect

The MobiTV Connect is an Android-based HDMI device that converts any TV with an HDMI input into a Smart TV. The Connect device is WiFi enabled, with a small footprint. Consumers are empowered to watch their favorite content and play mobile games on their TV. The MobiTV Connect device employs mobile phones and tablets for content discovery and remote control of the viewing experience on the TV screen.

The MobiTV Connect delivers an ecosystem for operators, content providers, and application developers to leverage their mobile assets and distribute content and services to the central point of in-home entertainment - the TV screen.

MobiTV provides a Controller SDK for mobile devices. Both iOS and Android application developers can enable their mobile applications to interoperate with the MobiTV Connect device. The SDK employs a lightweight integration model that uses JSON-RPC to initiate commands and receive responses from the MobiTV Connect device. It also supports binary data. Using the SDK, application developers control the end-user experience on the mobile device and on the TV Screen. The SDK has built-in support for sending commands to view content on the TV – including Play, Pause, Seek, Volume control, Closed Captions, and more.

A mobile application that uses the Controller SDK requires a corresponding small-footprint application (the media player) installed on the MobiTV Connect device to act on commands and provide responses. MobiTV provides a Receiver SDK for these small-footprint applications. The Receiver SDK is available for Android (the Connect hardware is an Android device). MobiTV also provides a sample Receiver application that uses the Receiver SDK to accept commands to play content on the TV and report playback progress back to the Controller application.

The mobile application and the MobiTV Connect device must be connected to the same WiFi network for them to be able to communicate with each other.

Applications are deployed over the air to the MobiTV Connect device in the market. Upgrades are triggered by MobiTV's cloud service.

2.2 Media Playback and Protection with the MobiTV Connect

The MobiTV Connect supports H.264, AAC encoded adaptive streaming via HTTP Live Streaming (HLS v4) with AES encryption. HTTP 1.1 progressive download playback is also supported.

MobiTV recognizes the need to protect content on the device. The MobiTV Connect supports Widevine DRM with Level 1 security as well as MobiTV's own DRM solution.

3 MobiTV Connect SDK Overview

The MobiTV Connect SDK facilitates communication and control of applications running on the MobiTV Connect device using a mobile application.

3.1 Key Components of the MobiTV Connect Ecosystem

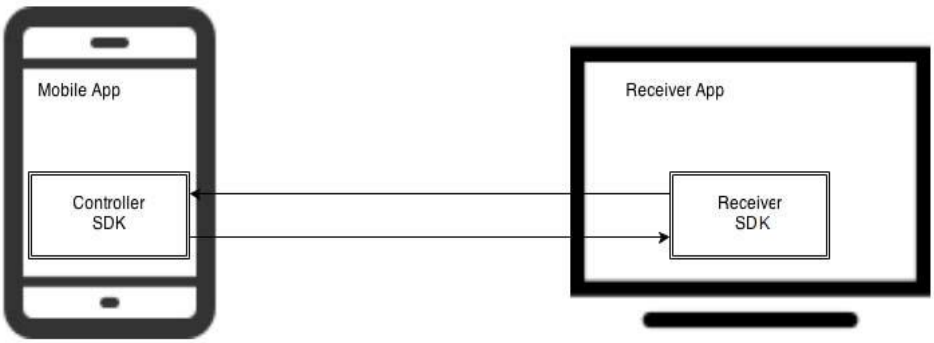
The MobiTV Connect ecosystem includes the following key components:

- Mobile App – Application running on an iOS or Android mobile device allowing user interaction. The Mobile App contains the Controller SDK and is also referred to as the Controller App.
- Receiver App – The Receiver application is an Android application running on the MobiTV Connect device.
- MobiTV Connect Controller SDK - A software library with both iOS and Android versions that is part of the Mobile App. It provides the ability to launch and control an application running on the MobiTV Connect device.
- MobiTV Connect Receiver SDK – A software library (Android only) included in the Receiver App that provides functionality to receive control commands from and send events to the Controller SDK.

3.2 Deployment Scenario

For this scenario, both the mobile device and the MobiTV Connect device are on the same subnet, such as a home WiFi network.

The following diagram depicts a typical deployment:



4 Connect Device WiFi Setup

Before you start building controller and receiver application, the MobiTV Connect device should be setup on your WiFi network for internet connectivity.

The steps to setup the MobiTV Connect device on your network are:

1. Download and install the .apk (for Android) for the WiFi Setup Mobile Application on the mobile device. The file is available on the Connect Partner site.
2. Establish a power supply and computer connection.
3. Check that the Firmware version is correct. If needed, update the Firmware following the instructions included here.
4. Start the WiFi Setup Mobile Application and complete a few simple steps to establish communication between the mobile device and the Connect device.

4.1 Download and Install the WiFi Setup Mobile Application

The WiFi Setup app is required on the mobile device to set up the Connect device on your WiFi network. The WiFi Setup mobile application is currently available for Android.

1. Download the .apk from the Connect Partners site.
2. Install on an Android mobile device.

4.2 Establish Power Supply and Computer Connection

Before starting the installation, the Connect device requires a power source, a keyboard, monitor, and a connection to a computer.

1. Use the micro USB port to power the Connect device.
2. Use a USB splitter to connect both power and USB keyboard to the Connect device.
3. Plug in the Connect device to a TV HDMI input port, or a computer monitor. If the monitor does not have an HDMI input port, use a DVI male to HDMI female adapter.
4. Connect the Connect device to the computer using a standard USB port.

4.3 Optional Firmware Update

The Connect SDK release may require an optional update of the Firmware that is installed on the Connect device. If a Firmware update is required, follow these steps:

Download the following two files from the Connect Partners site to your computer:

Note: To perform this step, the Connect device must have a power source and be connected to your computer as described in *Establish Power Supply and Computer Connection*.

- The zip file containing the update. The zip file naming convention follows this format:

`<board_name>-ota-<carrier_name>-<version_number-eng.zip`

Where the version_number format is: `major.minor.build.YYYYMMDDHHMM`

For example, `g35emmc-ota-carrier_name-0.5.2.201409241414-eng.zip.`

- The text file named "`command`" which contains the path to the update file.

Follow these steps to update the firmware:

1. Copy the files to a directory on your computer. Note that the file "`command`" contains the name of the update package (`update.zip`) and can be used unchanged for all firmware updates. The "`command`" file has one line with this exact text:

   `--update_package=/cache/update.zip`

2. Connect the Connect device to your computer USB port using the USB cable (for power and adb access). Insert the Connect Device into the HDMI port on the TV/monitor. Power on the TV.

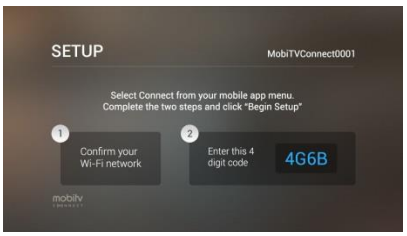
3. Push the firmware package to the Connect device using the following commands:

```
> adb remount

> adb shell

root@android:> mkdir /cache/recovery  (create the dir if it does not exist)

root@android:> exit

> adb push g35emmc-ota-carrier_name-0.5.2.201409241414-eng.zip
/cache/update.zip
> adb push command /cache/recovery
> adb reboot recovery
```

4. The firmware should now be updated and an automatic update occurs.

Note: The files that you pushed to the cache will be deleted after the update is completed.

After the Connect device is rebooted with the updated firmware, you will see the following **SETUP** screen for the WiFi Setup Mobile Application:



4.4 Setup WiFi

1. Check that the MobiTV Connect device is inserted into the HDMI port, with a power source. Also, power on the TV.

2. Check that the mobile device is connected to the WiFi network on which it will be running.

Important: The MobiTV Connect device requires a 2.4 GHz network. If the router supports dual band and the 2.4GHz and 5GHz bands have different SSIDs, ensure that the mobile device is connected to the 2.4GHz SSID at the time of setup. If the 2.4GHz and 5GHz bands have the identical SSID, then setup may continue without a reconnection.

3. Click the MobiTV Connect icon to start the WiFi Setup Mobile Application on the mobile device.

4. Select Easy Install. Your network name will appear. Enter the password for the network, or leave it blank if not password-protected.

5. Enter the 4-digit verification code displayed on the TV.

6. Single click the "Begin Setup" button to automatically connect the MobiTV Connect device to both the WiFi network and the mobile device.

7. After the connection is established, you may choose one of two options:

    - Rename your MobiTV Connect – Enter a friendly name for the Connect device
    - Browse Content – TV Displays the Ready screen

4.5 Verify Firmware Update

Follow these steps to use the Reference App to view the installed firmware version:

1. Launch the Reference App on the Connect device using adb shell:
    - `adb shell am start -n com.mobitv.dongle.refrence/com.mobitv.dongle.reference.MainMenu` *(this will display the Reference App menu of options)*
2. Use adb shell to input key codes to navigate to the option Build View:
    - `adb shell input keyevent KEYCODE_DPAD_DOWN` *(repeat as many times as needed to highlight option Build View)*
    - `adb shell input keyevent KEYCODE_DPAD_CENTER` *(to select the option Build View)*
3. Verify that the Firmware build listed is the one you installed:
    - `adb shell input keyevent KEYCODE_HOME` *(to return to main screen)*

5 Building and Installing the Android ReceiverSampleApp

5.1 Shell Script Included to Build Sample Apps

Use Gradle to build the sample apps from source. We have included a shell script named *gradlew* (and a bat file) to automate the build process. This script downloads Gradle 1.9 and then builds the apps. SDK jar files and documentation are included in the zip file for the SDK.

5.2 Building ReceiverSampleApp for Android

The ReceiverSampleApp is delivered in the zip file `mobiconnect-sdk-<x.x.x>-android-receiver.zip` where <x.x.x> indicates the version number. Unzip the SDK file and run the following commands:

```
> cd ~/mobiconnect-sdk-<x.x.x>-android-receiver/ReceiverSampleApp

> ./gradlew build
```

This will build the following package:

```
~/mobiconnect-sdk-<x.x.x>-android-
receiver/ReceiverSampleApp/build/apk/ReceiverSampleApp-debug-unaligned.apk
```

5.3 Installing the ReceiverSampleApp on the Connect Device

The MobiTV Connect Device comes with two port layouts.

- Unit with two ports: a regular USB and Micro USB port. Use a USB cable to go directly to the computer and use the Micro port for power only.
- Unit with one port: a single Micro USB port. Use a splitter cable for power and keyboard.

The ReceiverSampleApp is an Android application.

Follow these steps to install the ReceiverSampleApp onto the Connect Device:

1. Use the USB cable or splitter cable to connect the Connect Device to your computer.

2. On your computer, download the zip file from the partner wiki: https://mbtv-cnfl.atlassian.net/wiki/display/Connect/MobiTV+Connect

3. Unzip the package.

4. Using the ADB command, install the following APKs on the Connect device, where <x.x.x> represents the version number of the release:

   ```
   adb install ~/mobiconnect-sdk-<x.x.x>-android-receiver
   /ReceiverSampleApp/build/apk/ReceiverSampleApp-debug-unaligned.apk
   ```

   Note: If you have a previous version installed, first enter the uninstall command: `adb uninstall com.mobitv.connect.services.example` (this is the ReceiverSampleApp package name)

5. Disconnect the Connect device from the computer.

6 Building and Installing the Android ControllerSampleApp

6.1 Shell Script Included to Build Sample Apps

Use Gradle to build the sample apps from source. We have included a shell script named *gradlew* (and a bat file) to automate the build process. This script downloads Gradle 1.9 and then builds the apps. SDK jar files and documentation are included in the zip file for the SDK.

6.2 Building ControllerSampleApp

The ControllerSampleApp is delivered in the zip file `mobiconnect-sdk-<x.x.x>-android-controller.zip` where <x.x.x> indicates the version number. Unzip the SDK file and run the following commands:

```
> cd ~/mobiconnect-sdk-<x.x.x>-android-controller/ControllerSampleApp/

> ./gradlew build
```

This will build the following package:

```
~/mobiconnect-sdk-<x.x.x>-android-
controller/ControllerSampleApp/build/apk/ControllerSampleApp-debug-
unaligned.apk
```

6.3 Installing the Android ControllerSampleApp on an Android Device

The Android ControllerSampleApp is delivered as part of the zip file `mobiconnect-sdk-<x.x.x>-android-controller.zip`, where <x.x.x> represents the version number of the release/

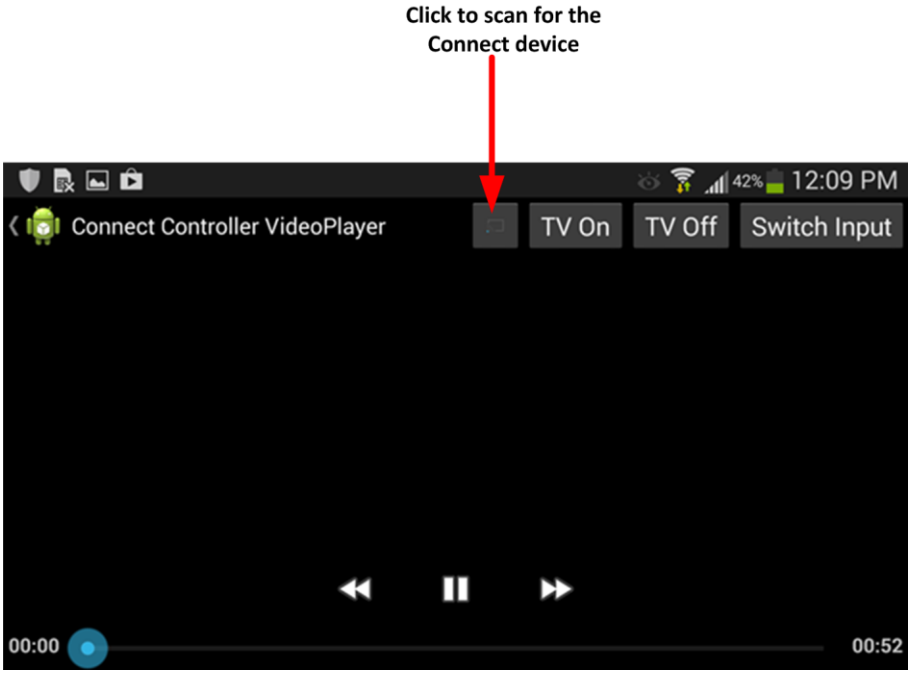
Enter the following command to install the Android ControllerSampleApp:

```
adb install mobiconnect-sdk-<x.x.x>-android-controller
/ControllerSampleApp/build/apk/ControllerSampleApp-debug-unaligned.apk
```

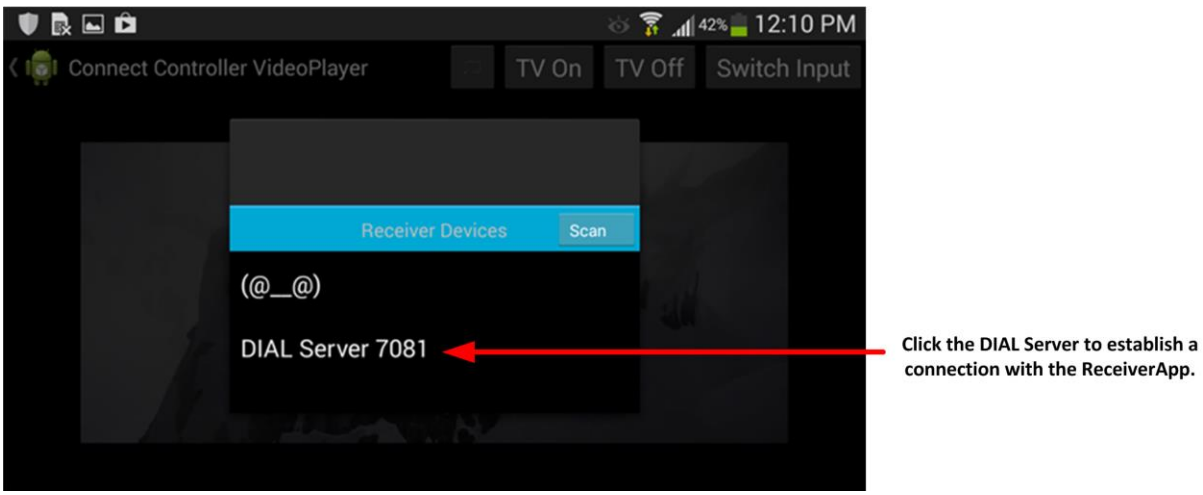
6.4 Using the Android ControllerSampleApp

1. Launch the ContollerSampleApp on the mobile device by clicking the **Connect Controller** icon.

2. Click the **Connect** icon to scan for the Connect device when it appears as shown below. It is next to the **TV On** icon.



3. Click the name of the DIAL Server to establish a connection.



4. Video will play on TV.

7 Building and Installing the iOS ControllerSampleApp

The MobiTV Connect iOS SDK software distribution includes sample apps that demonstrate the usage of the API. These are delivered as XCode projects in the *samples* directory. You can open these apps in XCode, review the source code, build them with your Enterprise or Individual Code Signing Identity and Provisioning Profile. The sample apps require iOS 7.0 or greater.

7.1 Using the iOS ControllerSampleApp

7.1.1 Configure the WiFi Network for the Mobile Device

Connect the mobile device to the same WiFi network as the Connect Device.

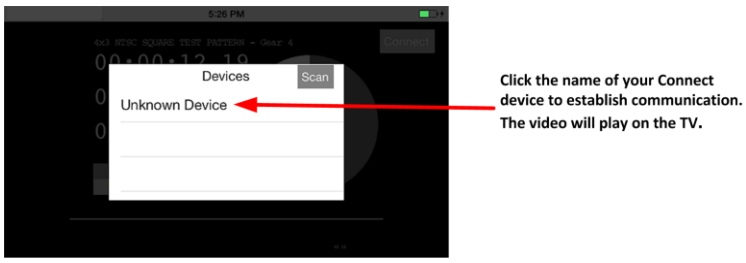
7.1.2 Connecting to the Connect Device

After the ControllerSampleApp is installed on the iOS device, the next step is to establish communication with the MobiTV Connect device.

Note: The MobiTV Connect device must be properly inserted into the HDMI port on the TV and working over the same WiFi network as the iOS device.

1. Connect the iPhone (or other iOS device) to the same WiFi network as the MobiTV Connect device.
2. Click the **ControllerSampleApp** icon on the iPhone to start the ControllerSampleApp. A video screen will appear with a **Connect** button.
3. Click the **Connect** button to establish communication with the Connect device.

4. Select the Connect Device from the Devices list.



When the video is playing on the TV, the mobile phone becomes a soft remote control.

7.2 Playing Media on the TV

1. Connect the Connect device to the TV HDMI-In port.
2. Start the ControllerSampleApp on the mobile device. It will play media, start device discovery in the background and enable the Connect icon in the media player when a device is detected.
3. Click on the Connect icon to start playing media on the Connect Device. Perform Pause, Forward, Rewind, and Seek actions on the mobile device and see action reflected on the TV. Media is paused on the mobile device, but the seek position is being updated using Notifications received from the Connect device.

8 Writing an Android Receiver Application

Any application running on a MobiTV Connect device needs to integrate with the Android Receiver SDK to receive commands from a Controller application. This section contains sample code for integrating an application with the Android Receiver SDK.

The Android Receiver SDK allows an application to perform the following functions:

- Establish a connection with one or more Controllers
- Receive requests/notifications from connected Controllers
- Send notifications to connected Controllers
- Turn the TV On/Off and switch the input source to HDMI

8.1 Setting up a Receiver Project

To use the Android Receiver SDK, it must be added to your project. Follow these steps to add the SDK to your project:

1. Add the SDK library **.jar** file to your application project.
2. Add the appropriate permissions to the manifest for the Receiver application.
3. Register your Receiver application with the DIAL service.

8.1.1 Add the SDK Library to Your Project

In the SDK **libs** folder you will find a **receiver-sdk-<*version*>.jar** file. Place the receiver SDK jar in your receiver application's directory, typically under a **libs** subfolder.

For example, if your receiver application's project is located at *~/my_name/projects/myreceiverapp*, you would place the controller SDK .jar file at *~/my_name/projects/myreceiverapp/libs*.

Next, add the library dependency to your project. If you are using Android Studio as your development environment, you can do this by adding the following code to the **dependencies** block of your project's *build.gradle* file:

```
dependencies {

    compile fileTree('libs')

}
```

This will place the jars in the libs folder into your classpath and will cause your application's APK to include the jar code.

8.1.2 Receiver Application Manifest Updates

For the Receiver application side, the following updates must be in the manifest:

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

8.1.3 Registering an Android Connect Receiver Application with the DIAL Service

The Connect SDK uses the DIAL protocol from Netflix and others to discover and launch the Receiver application. DIAL enables mobile devices (“2nd screen”) to send content to a television (“1st screen”).

DIAL is a “simple protocol for Discovery And Launch” that enables 2nd screen applications to discover and launch 1st screen applications on 1st screen devices.” From the DIAL, Discovery And Launch protocol specification, version 1.64, © Netflix. The DIAL specification is available at: http://www.dial-multiscreen.org/dial-protocol-specification

In order to launch over the DIAL protocol, the Receiver application must be registered with the DIAL service.

Two configuration steps are required for a Receiver application to be discovered and launched by the DIAL service:

1. Assign the application a unique name in the *AndroidManifest.xml* file
2. Inform the DIAL Service which activity to launch

To register the Receiver application with the DIAL service, assign a unique name for the application in the *AndroidManifest.xml* application manifest file.

To launch the Receiver application over the DIAL protocol, assign the activity an intent filter marked with `action MAIN/category LAUNCHER`.

8.1.4 Assigning the Application a Name in the Receiver Manifest

When the application package is installed on the Receiver device, the application is addressed by the controller using a unique name.

Add the following metadata to the *AndroidManifest.xml* file, inside the `<application>` tag:

```
<meta-data android:name="com.mobitv.receiverapp"
android:value="com.vendorname.appname">
```

Replace the value `"com.vendorname.appname"` with a unique name to identify the application in fully qualified reverse-domain name notation.

8.1.5 Informing the DIAL Service to Launch an Activity in the Receiver Manifest

In the *AndroidManifest.xml* file, assign the activity an intent filter marked with `action MAIN/category LAUNCHER`. Following is an example:

```
<activity android:name="com.vendorname.packagename" android:label="My App" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

If there is more than one activity in the manifest, the DIAL service launches the last activity found with the above intent filter.

8.1.6 Example Receiver Manifest

Following is an example of the *AndroidManifest.xml* for the Receiver for a "Hello World" application:

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"package="com.mycompany.He
lloWorld">
    <application>
```

```
        <meta-data
android:name="com.mobitv.receiverapp"android:value="com.mycompany.HelloWorld">
<!-- this is the unique application name-->
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
        <activity
            android:name="com.mycompany.helloworld.HelloActivity"
            android:label="Hello World App">
            <intent-filter> <!--this is the LAUNCHER activity intent filter-->
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

8.2 Setup for Accepting Connections

Once the application is started, it needs to start the Receiver with a Listener to accept incoming controller connections.

```
Receiver.start(this, mConnectionListener);
```

8.2.1 Create a Connection Listener

Register a **ConnectionListener** to receive notifications of new incoming connections from Controllers or of Controllers disconnecting. For example:

```
private ConnectionListener mConnectionListener = new ConnectionListener() {
       @Override
       public void onControllerConnected(Controller controller) {
              // connection to the controller is established
       }

       @Override
       public void onControllerDisconnected(Controller controller) {
              // connection to the controller is lost
       }
};
```

8.3 Receiving Incoming Messages

- **JSON-RPC Format**

Once the connection is established, the application can receive messages from the Controller by setting a **RequestListener**. For example:

```
private RequestListener mRequestListener = new RequestListener() {
       @Override
```

```
	public JSONRPCResponse onRequestReceived(final Controller controller,
	                                  final JSONRPCRequest jsonrpcRequest) {
		runOnUiThread(new Runnable() {
			@Override
			public void run() {
				mContentView.append(controller.getName());
				mContentView.append(jsonrpcRequest.getParam("message").
				toString());
			}
		});
		return null;
	}

	@Override
	public void onNotificationReceived(final Controller controller,
	                           final JSONRPCNotification jsonrpcNotification) {
		runOnUiThread(new Runnable() {
			@Override
			public void run() {
				mContentView.append(controller.getName());
				mContentView.append(jsonrpcNotification.getParam
				("message").toString().concat("\n"));
			}
		});
	}
};
Receiver.setRequestListener(mRequestListener);
```

- **Binary Format**

To receive binary messages, set a **BinaryMessageListener**:

```
private BinaryMessageListener mBinaryMessageListener = new BinaryMessageListener()
{
	@Override
	public void onBinaryMessageReceived(Controller controller, ByteBuffer
byteBuffer) {
		// process binary data in byteBuffer
	}
}
Receiver.setBinaryMessageListener(mBinaryMessageListener);
```

8.4 Sending Messages to Connected Controllers

8.4.1 Broadcasting Messages to All Connected Controllers

- **JSON-RPC Format**

JSON-RPC is the required format for messages that are broadcast to all connected controllers.

To broadcast a JSON-RPC notification, call **Receiver.broadcastNotification().**

Following is an example:

```
// example of how to send a notification

JSONRPCNotification notification = new
JSONRPCNotification("com.mycompany.examplemethod");

notification.setParam("param1", "value1");

Receiver.broadcastNotification(notification);
```

8.4.2 Unicasting Messages to Individual Controllers

Messages may also be addressed individually (unicast) to a specific Controller device.

- **JSON-RPC Format**

Both JSON-RPC Requests and JSON-RPC Notifications may be sent to individual controllers. When a controller receives a JSON-RPC Request, a response will be returned. For notifications, no response will be sent.

To send requests, call **sendRequest()** on the **Controller** object for the device you wish to send the request to. Pass a **JSONRPCCallback** object to receive the response to the request.

```
JSONRPCRequest request = new JSONRPCRequest("com.mycompany.examplerequest");
request.setParam("param1", "value1");
try {
    // controller here is an instance of the Controller class, previously
    // received in the onControllerConnected() method of the registered
    // connection listener
    controller.sendRequest(request, new JSONRPCCallback() {
        @Override
        public void onCallback(JSONRPCResponse response) {
            // response contains the response to the request
            // obtain the response result by calling getResult()
            if(response.isSuccess()) {
                Object result = response.getResult();
                // ... do something with the result ...
            } else if(response.isError()) {
                // error will contain JSON describing the error
                JSONObject error = response.getError();
            }
        }
    });
} catch (IOException exception) {
    // controller has disconnected if this exception is thrown
}
```

To send notifications, call **sendNotification()** on the **Controller** object:

```
JSONRPCNotification notification = new
JSONRPCNotification("com.mycompany.examplemethod");

request.setParam("param1", "value1");
try {
    // controller here is an instance of the Controller class, previously
    // received in the onControllerConnected() method of the registered
    // connection listener
    controller.sendNotification(notification);
    // notifications are one way, so no response will be received
} catch (IOException exception) {
    // controller has disconnected if this exception is thrown
}
```

- **Binary Format**

Binary messages consisting of byte arrays may be sent to Controller devices. Use **sendBinaryData()** on the **Controller** object to do this:

```
byte[] bytes = {0xa, 0xb, 0xc, 0xd, 0xe, 0xf};
try {
    controller.sendBinaryData(bytes);
} catch (IOException e) {
    // controller has disconnected if this exception is thrown
}
```

8.5 Using the HDMI Class to Control TV

Android Connect Receiver SDK provides a class "HDMI" which contains the following methods:

- `HDMI.isReceiverPluggedin()` - Determines HDMI plugin (HPD) status of the receiver device.
- `HDMI.switchInputToReceiver()` - Sends a CEC command to tell the TV to switch HDMI input to the receiver device.
- `HDMI.TVOff()` - Sends a CEC command to turn the TV off.
- `HDMI.TVOn()` - Sends a CEC command to turn the TV on.
- `HDMI.getAudioCapabilities()` - Queries the audio capabilities of the connected device.

9 Writing an iOS Controller Application

This section contains sample code for integrating the iOS Controller SDK in a mobile application. Developers that intend to control applications running on a MobiTV Connect device must integrate the iOS Controller SDK into their mobile applications for iOS 7.0 or higher.

The Connect SDK allows an application to perform the following functions:

- Discover MobiTV Connect devices connected on a WiFi network
- Launch and connect the Receiver application running on those devices
- Send/receive messages from connected applications

9.1 Setting up a Connect Project for iOS

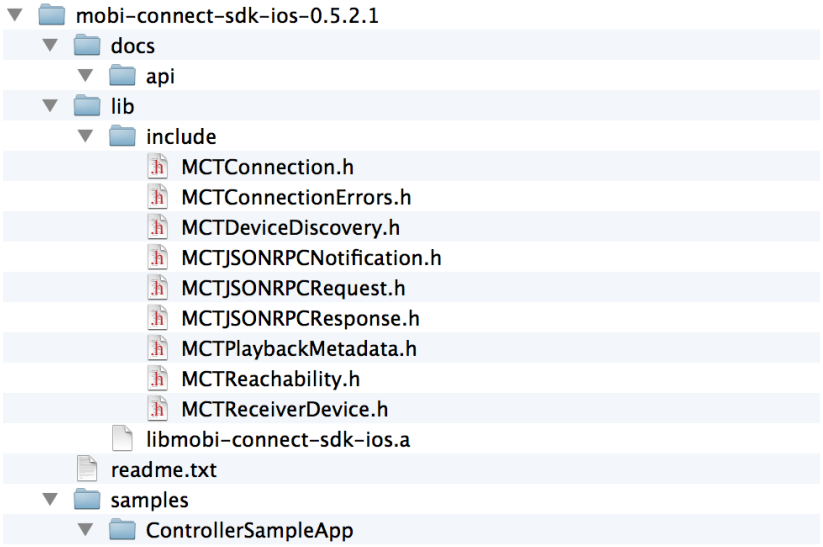
9.1.1 Requirements and Archive Structure

The minimum requirements for projects that use the MobiTV Connect iOS SDK are XCode 5.0 and iOS 7.0. The Deployment Target SDK must be iOS 7.0 or greater.

The MobiTV Connect iOS SDK is distributed as a zip file that includes the following items:

1. The SDK packaged as a universal static library with support for arm64, armv7, armv7s, i386 and x86_64 architectures
2. Objective-C header files that define the public API
3. Documentation including the User Guide and API documentation in appledoc format
4. Sample Apps that demonstrate usage of the API

A typical unzipped archive for a MobiTV Connect iOS SDK release will have the following structure:



9.1.2 Adding the iOS SDK to Your Project

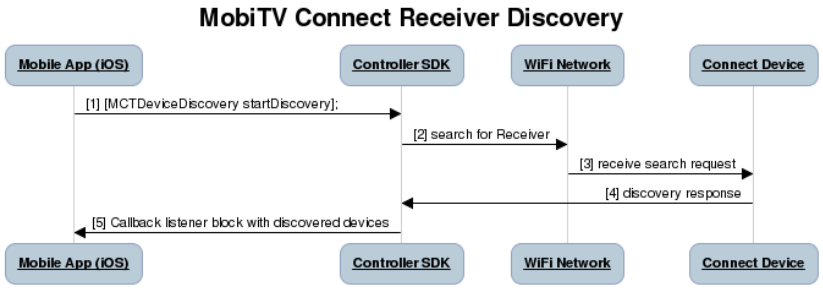
To use the MobiTV Connect iOS SDK, it must be added to your project. Follow these steps to add the SDK to your project:

1. Copy the contents of the *lib* directory, which includes the static library *libmobi-connect-sdk-ios.a* and the Objective-C header files to the project.
2. Link the following frameworks to the project:
    a. CFNetwork.framework
    b. SystemConfiguration.framework
    c. libicucore.dylib
    d. libxml2.dylib
    e. libcrypto_iOS.a
3. In build setting for the application targets, add the value `-lxml2` to the build setting *Other Linker Flags*.

To upgrade the version of the MobiTV Connect iOS SDK in an existing project, delete the static library and header files that were copied in Step 1 and copy the new static library and header files. Note that upgrades can add, as well as remove, header files.

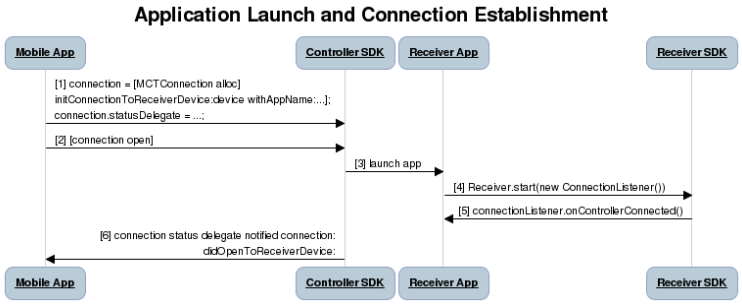
9.2 Call Sequences

9.2.1 Discover Receiver Device



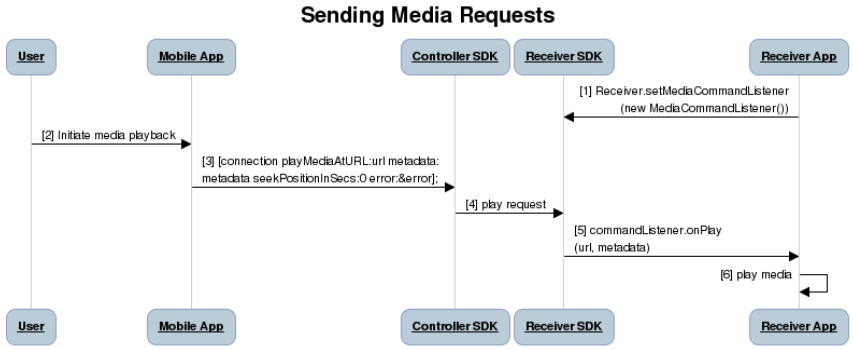
1. To begin device discovery, the iOS Application invokes the **startDiscovery** method. This method takes a callback block which will be invoked when devices have been found.
2. The Controller SDK then initiates a search on the local wireless network for one or more receiver devices.
3. One or more MobiTV Connect Receiver devices receive the search request.
4. The Connect device(s) send a discovery response.
5. The Controller SDK notifies the callback block that one or more devices have been found.

9.2.2 Establish Connection and Launch Application



1. The Mobile Application creates a **MCTConnection** instance for a given Receiver device and sets up a status delegate to be notified of connection status.
2. To initiate the connection to the Receiver, the Mobile Application calls **open** on the connection.
3. The Controller SDK launches the application on the receiver device.
4. Once started, the application on the Receiver side calls **Receiver.start()** with a connection listener. This establishes a connection endpoint for the controller application to communicate with.
5. The Receiver SDK notifies the receiver side **ConnectionListener** that a controller has connected.
6. Finally, once the connection has been established, the connection's status delegate is notified. Communication with the Receiver application can then begin.

9.2.3 Send Media Requests



1. In order to receive media events, the Receiver application must first register a **MediaCommandListener** by calling **Receiver.setMediaCommandListener()**.
2. The user of the mobile application interacts to start playing media on her Receiver device.
3. The application invokes the **playMediaAtURL:** API on the connection object.
4. The request is issued over the connection to the Receiver application where it is intercepted by the Receiver SDK.
5. The playback information is passed up to the Receiver application's **MediaCommandListener.**
6. Now the Receiver application has the information it needs to play media.

Note: Other commands such as `play/setPlaybackSpeed/seek`, etc. are handled in much the same way. There are entry points in the connection interface in the Controller SDK for each of these. The command is always handled on the Receiver side by passing the information to a **MediaCommandListener.**

9.3 Discovering MobiTV Connect Devices on WiFi Network

9.3.1 Search for Devices

The application must set a listener block, which will be invoked once the Controller finishes searching for all connected Connect devices. This listener will receive updates when new devices join or existing devices disconnect from the WiFi network. For example:

```
[MCTDeviceDiscovery startDiscoveryWithPollFrequency:5 listenerBlock:^(NSSet
*devices) {
      // NSSet of MCTReceiverDevice instances representing devices discovered
}];
```

9.3.2 Stop Device Discovery

The application can stop device discovery by calling the **stopDiscovery** method, for example:

```
[MCTDeviceDiscovery stopDiscovery];
```

9.3.3 Recommendations for Usage

The device discovery process involves heavy use of the WiFi radio, and thus is battery intensive. We recommend performing device discovery only when necessary.

9.4 Launching and Connecting to a Receiver Application

Once the Controller application has selected a device, use the following code to start a corresponding Receiver application:

```
/*
 Create controller instance with the user-selected device and by providing the
application identifier and controller name.
 */

MCTConnection *controller = [[MCTConnection alloc]
initConnectionToReceiverDevice:device withAppName:@"com.mobitv.connect.example"
controllerName:@"My Controller Device"];

// Set the delegates to the class who will be receiving connection/disconnection
events and messages from the receiver (see the section "Setting up Connection
Status and Request Delegates")
controller.statusDelegate = self;
controller.requestDelegate = self;

// Open the connection
[controller open];
```

9.4.1 Secure Connection

The SDK offers the option of encrypting data exchanged over the connection. To use the encrypted connection, use the following initializer:

```
MCTConnection *controller = [[MCTConnection alloc]
initSecureConnectionToReceiverDevice:device
withAppName:@"com.mobitv.connect.example" controllerName:@"My Controller Device"];
```

9.5 Setting up Connection Status and Request Delegates

A class should conform to **MCTConnectionStatusDelegate** in order to be notified whenever there is a change in the connection. This class will be set as the **statusDelegate** on the connection.

Additionally, a class should conform to **MCTConnectionRequestDelegate** in order to receive JSON-RPC and binary messages from a Receiver.

The methods that will be invoked to notify connection status are as follows:

```
//Successfully opened a connection

-(void) connection:(MCTConnection*)connection
didOpenToReceiverDevice:(MCTReceiverDevice*)device {

}

//Successfully closed a connection

-(void) connection:(MCTConnection*)connection
didCloseOnReceiverDevice:(MCTReceiverDevice*)device {

}

//Connection closed with an error

-(void) connection:(MCTConnection *)connection
didCloseOnReceiverDevice:(MCTReceiverDevice*)device withError:(NSError*)error {

}
```

NOTE: Once either the **connection:didCloseOnReceiverDevice:** or **connection:didCloseOnReceiverDevice:withError:** methods are called, the connection is invalid.

9.5.1 Sending JSON-RPC and Binary Messages

- **JSON-RPC Format**

Once a connection to the Receiver is established, a Controller can send messages on any open connection in JSON-RPC format. Messages in JSON-RPC format and are constructed using the **MCTJSONRPCRequest** and **MCTJSONRPCNotification** classes.

JSON-RPC method names may be anything you desire, although we recommend using reverse-DNS style namespace names. Following is an example:

```
// Construct a message by creating a JSONRPCRequest/Notification for the method
that is implemented on the receiver
MCTJSONRPCRequest * request = [[MCTJSONRPCRequest alloc]
initWithMethodName:@"com.mycompany.examplemethod"];

// Set parameters that the method expects either by passing a dictionary or the
array

//Example for passing as an Array
[request setPositionalParams:paramsArray];
//Example for passing as a Dictionary
[request setNamedParams:paramsDictionary];

// Send this request/notification using an established connection
[controller sendRequest:request andCallbackWithResponse:^(MCTJSONRPCResponse*
```

```
response){
        // Response callback block
}];
```

- **Binary Format**

Once a connection to a Receiver application has been established, binary data can be sent on the connection.

To do this, pass the binary data as an instance of NSData. Example:

```
 char bytes[] = { 0xd, 0xe, 0xa, 0xd, 0xb, 0xe, 0xe, 0xf }; // this is
test data to be encapsulated in an NSData instance

 [controller sendBinaryData:[NSData dataWithBytes:bytes
length:sizeof(bytes)]];
```

9.5.2 Receiving JSON-RPC and Binary Messages

- **JSON-RPC**

The request delegate methods **handleRequest:** and **handleNotification:** will be invoked when JSON-RPC requests and notifications are received, respectively.

For JSON-RPC Requests, be sure to return a response:

```
-(MCTJSONRPCResponse*) handleRequest:(MCTJSONRPCRequest*)request
{
      // ...
      // process the request
      // ...
      // and return a response
      return [request createResponseWithResult:@"OK"];
}
```

- **Binary Messages**

When binary messages are received, the request delegate method **handleBinaryData:** will be called with an **NSData** instance containing the binary data.

```
-(void) handleBinaryData:(NSData*)data
{
      // process the binary data here
}
```

9.6 Media Interface

The Controller SDK provides some of the APIs for controlling media on the Receiver. For example, the code below is used to instruct the Receiver application to play video:

```
NSError* error;

if(![mController playMediaAtURL:url metadata:playbackMetadata
seekPositionInSecs:position error:&error]) {

       // if FALSE is returned then the command did not complete successfully.

       // in that case, "error" will have the error details.

}
```

which invokes the following interface:

```
-(BOOL) playMediaAtURL:(NSURL*)url metadata:(MCTPlaybackMetadata*)metadata
seekPositionInSecs:(int)position error:(NSError**)outError
```

Parameters:

- *url*: URL to playback
- *metadata*: Metadata of the program being played.
- *position*: Seek position from where the playback should begin.
- *outError*: Address of an error object.

Return Type: A *boolean*, true in case of success, and in case of failure, returns false and error object is populated.

Declared In: MCTConnection.h

10 Writing an Android Controller Application

This section contains sample code for integrating the Android Controller SDK in a mobile application. Developers that intend to control applications running on a MobiTV Connect device must integrate the Controller SDK into their mobile applications.

Note: Refer to the *README.md* file included with the sample application included in the Connect SDK software distribution for the most current instructions.

The Connect SDK allows an application to perform the following functions:

- Discover MobiTV Connect devices connected on a WiFi network
- Launch and connect the "receiver" application running on those devices
- Send/receive messages from connected applications

10.1 Setting up a Controller Project for Android

To use the Android SDK, it must be added to your project. Follow these steps to add the SDK to your project:

1. Add the SDK library **.jar** file to your application project.
2. Add the appropriate permissions to the manifest for the Controller application.

10.1.1 Add the SDK Library to Your Project

In the SDK **libs** folder you will find a **controller-sdk-<*version*>.jar** file. Place the controller SDK jar in your controller application's directory, typically under a **libs** subfolder.

For example, if your controller application's project is located at *~/my_name/projects/mycontrollerapp*, you would place the controller SDK .jar file at *~/my_name/projects/mycontrollerapp/libs*.

Next, add the library dependency to your project. If you are using Android Studio as your development environment, you can do this by adding the following code to the **dependencies** block of your project's *build.gradle* file:

```
dependencies {

    compile fileTree('libs')

}
```

This will place the jars in the libs folder into your classpath and will cause your application's APK to include the jar code inside of it.

10.1.2 Controller Application Manifest Updates

To successfully discover MobiTV Connect Receiver devices and start and communicate with Receiver applications, the application needs certain permissions in its manifest.

Add the following lines to your Controller application's *AndroidManifest.xml*, if they are not already there:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission
android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
```

10.1.3 Example Controller Manifest

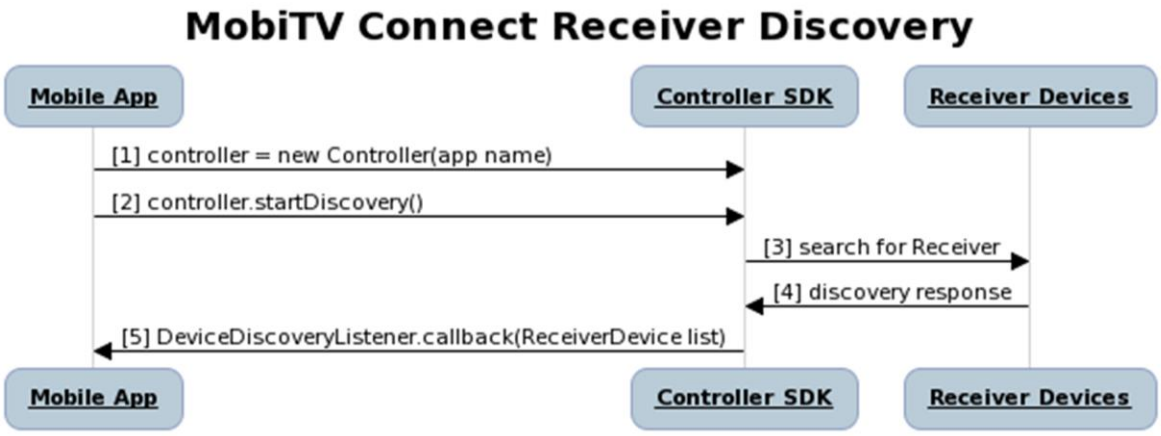
Following is an example of the *AndroidManifest.xml* for the Controller for a "Hello World" application:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mycompany.HelloWorldController" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
    <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"
/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name">
        <activity
            android:name="MainActivity"
            android:configChanges="orientation|screenSize"
```

```
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

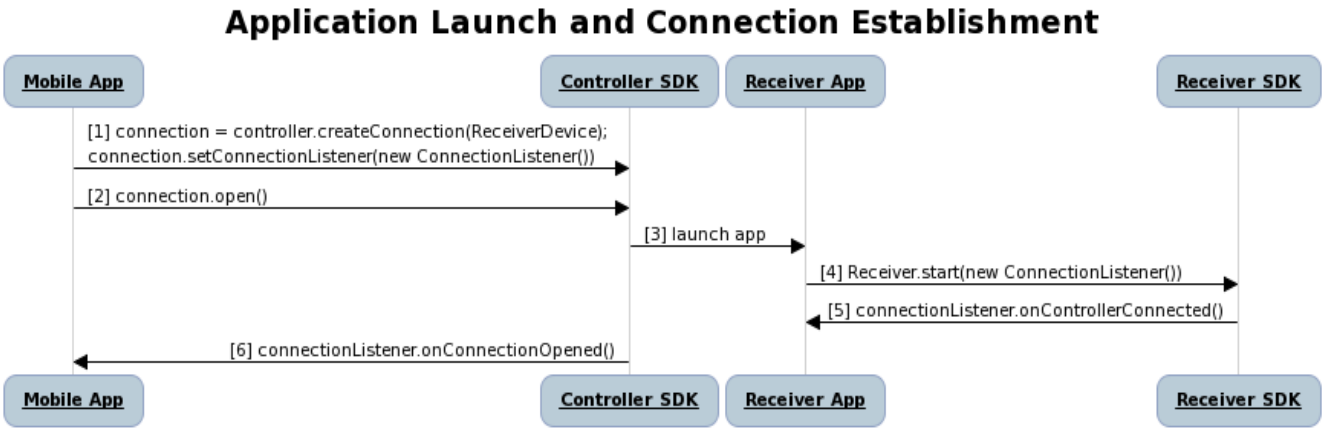
10.2 Call Sequences

10.2.1 Discover Receiver Device



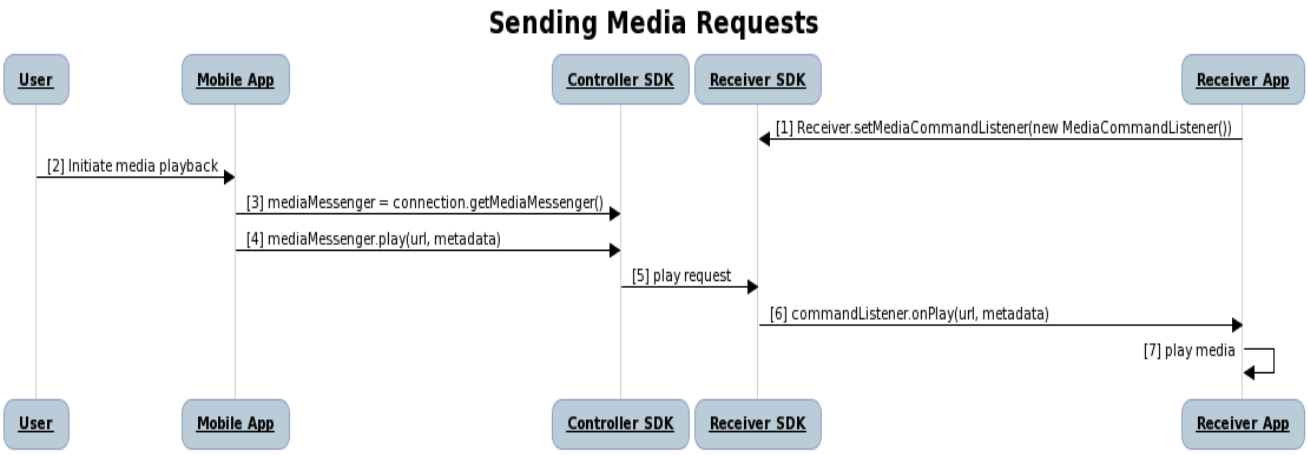
1. The Mobile Application begins by instantiating a **Controller** object with the specified app name. This object manages connections to one or more receiver devices.
2. The Mobile Application then invokes **startDiscovery()** on the controller. This method takes a **DeviceDiscoveryListener** which will be invoked when devices have been found.
3. The Controller SDK then initiates a search on the local wireless network for one or more receiver devices.
4. One or more Receiver devices respond to the search request.
5. The Controller SDK notifies the application's **DeviceDiscoveryListener** that one or more devices have been found.

10.2.2 Establish Connection and Launch Application



1. The Mobile Application creates a **Connection** instance for a given Receiver device and sets a **ConnectionListener** to be notified of connection status.
2. To initiate the connection to the receiver, the Mobile Application calls **open()** on the connection.
3. The Controller SDK launches the application on the receiver device.
4. Once started, the application on the Receiver side calls **Receiver.start()** with a connection listener. This establishes a connection endpoint for the controller application to communicate with.
5. The Receiver SDK notifies the receiver side **ConnectionListener** that a controller has connected.
6. Finally, once the connection has been established, the Mobile Application's **ConnectionListener** is notified. Communication with the Receiver application can then begin.

10.2.3 Send Media Requests



1. In order to receive media events, the Receiver application must first register a **MediaCommandListener** by calling **Receiver.setMediaCommandListener()**.
2. The user of the mobile application interacts to start playing media on her Receiver device.
3. The application requests a **MediaMessenger** instance on the connection with the Receiver by calling **getMediaMessenger().**
4. The application then invokes the **MediaMessenger play**() API with the program URL and metadata.
5. The request is issued over the connection to the receiver application where it is intercepted by the Receiver SDK.
6. The playback information is passed up to the Receiver application's **MediaCommandListener.**
7. Now the Receiver application has the information it needs to play media.

Other commands such as `play/setPlaybackSpeed/seek`, etc. are handled in much the same way. There are entry points in the **MediaMessenger** interface in the Controller SDK for each of these. The command is always handled on the Receiver side by passing the information to a **MediaCommandListener.**

10.2.4 Managing the TV from the Controller side

The Receiver SDK supports commands to control the TV using HDMI CEC (Consumer Electronic Control) and HPD (Hotplug Detection) state. These commands are available in the HDMI class (see *Using the HDMI Class to Control TV*).

To initiate the commands from the Controller side, send a custom JSON-RPC Notification over the connection. When the Receiver's request listener receives the notification, perform the HDMI operation.

10.3 Discovering MobiTV Connect Devices on WiFi Network

10.3.1 Create a Controller

The application must create a Controller to use the features of this SDK. The application must provide an application context and an identifier of the application installed on the Connect Receiver device. For example:

```
mController = new Controller(getApplication(),"com.mobitv.connect.example");
```

10.3.2 Set a Listener and Search for Devices

The application must set a listener which will be invoked once the Controller finishes searching for all connected Connect devices. This listener will receive updates when new devices join or existing devices disconnect from the WiFi network. For example:

```
mController.startDiscovery(mDeviceDiscoveryListener);
```

10.3.3 Stop Device Discovery

The application can stop device discovery by calling the **stopDiscovery** method, for example:

```
mController.stopDiscovery();
```

10.3.4 Recommendations for Usage

- Although the SDK allows the creation of multiple Controllers, it is recommended to use a single controller shared across multiple activities.
- The discovery process makes heavy use of the wifi radio and should be started only when necessary, such as when presenting a discovery UI to a user.

10.4 Setting up the Discovery Listener

Applications must set a **DeviceDiscoveryListener**. This callback is invoked every time there is a new device added or an existing device removed from the network. To stop getting discovery updates, call the **stopDiscovery** method on the Controller.

```
private Controller.DeviceDiscoveryListener mDeviceDiscoveryListener = new
Controller.DeviceDiscoveryListener() {
```

```
	@Override
	public void callback(final List<ReceiverDevicedevices) {

		runOnUiThread(new Runnable() {
			@Override
			public void run() {
				// application specific code to handle UI changes
			}
		});
	}
};
mController.startDiscovery(mDeviceDiscoveryListener);
```

Once the initial search is complete or there is an update to devices connected on network, this callback is invoked with a list of **ReceiverDevice**.

10.5 Launching and Connecting to a Receiver Application

Once application has selected a device that it wants to connect with, use the following code to start a corresponding receiver application:

```
// device selected by user
final Connection connection = mController.createConnection(device);

ConnectionListener connectionListener = new ConnectionListener() {
	@Override
	public void onConnectionOpened() {
		MainActivity.this.runOnUiThread(new Runnable() {
			@Override
			public void run() {
				Toast.makeText(MainActivity.this, "Connection
				Established", Toast.LENGTH_SHORT);
				// Connection can only be used once onConnectionOpened
				// call back is received
				mMessageConnection = connection;
			}
		});
	}

	@Override
	public void onConnectionError(final int errorCode,
				final String errorMessage) {
		MainActivity.this.runOnUiThread(new Runnable() {
			@Override
			public void run() {
				Toast.makeText(MainActivity.this, "Connection Error - "
				+ errorMessage, Toast.LENGTH_SHORT);
			}
		});
	}

	@Override
	public void onConnectionClosed() {
```

```
                MainActivity.this.runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                        Toast.makeText(MainActivity.this, "Connection Terminated",
                        Toast.LENGTH_SHORT);
                        mMessageConnection = null;
                        }
                });
        }
};

// set a connection listener to get connection updates
connection.setConnectionListener(connectionListener);
connection.open();
```

10.5.1 Secure Connection

The SDK offers the option of encrypting data exchanged on the connection. To use the encrypted form of connection, create the connection using the **createSecureConnection** method:

```
// device selected by user
final Connection connection = mController.createSecureConnection(device);
```

10.6 Setting up a Connection Listener

The application has the option to be notified when a connection to a receiver is established. Set a **ConnectionListener** to receive notifications as shown in the example above.

NOTE: Once the **onConnectionClosed**() method is called, the connection is invalid.

10.6.1 Sending JSON-RPC and Binary Messages

- **JSON-RPC Format**

Once a connection to a Receiver is established, a controller can send messages on any open connection, as shown in the following example:

```
        // Construct a message by creating a JSONRPCRequest/Notification
        // for the method that is implemented on the receiver
        JSONRPCNotification message = new JSONRPCNotification("Post");
        // Set parameters and their values, that the method expects
        message.setParam("message", text);
        // Send this request/notification using an established connection
        mMessageConnection.sendNotification(message);
```

- **Binary Format**

Once a connection to a Receiver application has been established, binary data can be sent on the connection.

To do this, send the binary data as a raw byte array to the **sendBinaryData()** method of the connection instance:

```
byte[] bytes = {0xa, 0xb, 0xc, 0xd, 0xe, 0xf};
mMessageConnection.sendBinaryData(bytes);
```

10.6.2 Receiving JSON-RPC and Binary Messages

- **JSON-RPC Format**

To receive JSON-RPC Requests and Notifications on a connection, set a request listener:

```
mConnection.setRequestListener(new RequestListener() {
    @Override
    public JSONRPCResponse onRequestReceived(JSONRPCRequest request) {
        // ...
        // handle JSON-RPC requests here
        // ...
        // return a response
        return request.createResponse("OK");
    }

    @Override
    public void onJSONRPCNotificationReceived(JSONRPCNotification notification) {
        // handle JSON-RPC notifications here
    }
});
```

- **Binary Format**

To receive binary data, set a binary message listener:

```
mConnection.setBinaryMessageListener(new BinaryMessageListener() {
    @Override
    public void onBinaryMessageReceived(ReceiverDevice receiverDevice, ByteBuffer
byteBuffer) {
        // handle binary data in the byteBuffer here
    }
});
```