MediaTek SDK for Android Developer's Guide

Version: 1.0
Release date: September 8, 2014

Specifications are subject to change without notice.

Table of contents

1 Introduction 5

2 MediaTek technology for Android Phones 5

2.1 HotKnot wireless data transfer 5

2.2 Multiple SIM 6

2.3 Multimedia manipulation 6

2.3.1 Portrait enhancements 7

2.3.2 Image Transformations 7

2.3.3 HD audio recording and file tagging 8

2.4 Multimedia capture 8

2.4.1 Gesture detection 8

2.4.2 Multi Angle View (MAV) image capture 8

2.4.3 Camcorder audio and video settings 9

3 Developing for MediaTek powered devices 9

3.1 MediaTek SDK for Android 9

3.2 Reference Android Phones 9

4 Getting started 10

4.1 Prerequisites 10

4.2 Downloading 10

4.3 Installation 11

4.3.1 Installing the API package 11

4.3.2 Install the tools package 11

4.4 Start developing with the MediaTek APIs 11

4.4.1 MediaTek feature APIs 12

4.5 MediaTek Compatibility API 13

4.6 Configuring and using the MediaTek emulator....16
4.6.1 Replace the standard emulator....16
4.6.2 Create an Android Virtual Device....17
4.6.3 Additional configuration and tools....20
4.7 MediaTek Emulator supported APIs....21
4.8 MediaTek DDMS Guide....21
4.8.1 Running the MediaTek DDMS....22
4.8.2 Simulating calls from either SIM card....22
4.8.3 Simulate SMS being sent from either SIM card....23
4.8.4 Simulate headset changes....23
4.8.5 Simulating power states....24
4.8.6 Simulated sensor states....25
4.8.7 Simulated SD Card states....26
4.9 Other tools....26
4.9.1 Intel® Hardware Accelerated Execution Manager....26
5 API Guides....27
5.1 API Level and Compatibility....27
5.2 Using HotKnot (com.mediatek.hotknot)....28
5.2.1 Creating a HotKnot sender application....28
5.2.2 Create a HotKnot receiver application....29
5.3 Using multi-SIM (com.mediatek.telephony)....30
5.3.1 TelephonyManagerEx Class....30
5.3.2 SmsManagerEx Class....30
5.4 Using the MAV capture (com.mediatek.hardware)....32
5.4.1 CameraEx Class....32

Lists of tables and figures

Table 1 Suggested values for AVD settings....19
Table 2 MediaTek API support in the SDK....21
Table 3 MediaTek API support by level....28

Figure 1 The interaction between the TX and RX grids on separate screens....5
Figure 2 Before (left) and after (right) processing with portrait enhancements....7
Figure 3 Extracting the content of the API ZIP to the Android SDK....11
Figure 4 Coding with the MediaTek APIs....12
Figure 5 Creating a libs folder for the Compatibility API....13
Figure 6 Copying the Compatibility API's jar to the project....13
Figure 7 The Compatibility API JAR file in the app project....14
Figure 8 The unpacked Compatibility API JAR showing the available APIs....14
Figure 9 The built project's APK file in the bin folder....15
Figure 10 Replacing the standard emulator file with those for the MediaTek emulator....16
Figure 11 The Android Virtual Device Manager....17
Figure 12 Settings for the MediaTek emulator AVD....18
Figure 13 Launching the MediaTek emulator....19
Figure 14 The MediaTek emulator showing the dual-SIM indicator....20
Figure 15 Simulate calls with Call1 and Call2....22
Figure 16 Simulating the sending of an SMS with Send1 and Send 2....23
Figure 17 Various headset simulations are provided....23
Figure 18 Various power simulation features are available....24
Figure 19 Various states can be set for sensors....25
Figure 20 Options for simulating an SD card being inserted and removed....26
Figure 21 Windows command to query the presence of HAXM....26
Figure 22 Linux command to query the presence of HAXM....27
Figure 23 The support for APIs across levels....27

1 Introduction

This Developer's Guide provides an introduction to the MediaTek SDK for Android. It provides all the information you need to install and use the SDK, as well as an introduction to the unique technology available in Android devices built on MediaTek chipsets.

A complementary API Reference is provided on the MediaTek Labs site that detailed documentation of APIs available in the SDK.

2 MediaTek technology for Android Phones

MediaTek chips are at the heart of a growing family of Android devices. Our chips bring multi-SIM, file transfer (with HotKnot), imaging, video, and gesture control features to Android phones. This section provides you with an introduction to those technologies.

2.1 HotKnot wireless data transfer

Hotknot is an innovative data and file transfer mechanism invented by MediaTek. It utilizes the physics of a capacitive touch screen to enable the exchange of data between two devices.

The technology takes advantage of the characteristics of a capacitive touch screen, such that the transmitting grid in the screen of one phone can interact with the receiver grid in a second when they are in close proximity (less than 1 centimeter). This process is shown in Figure 1.

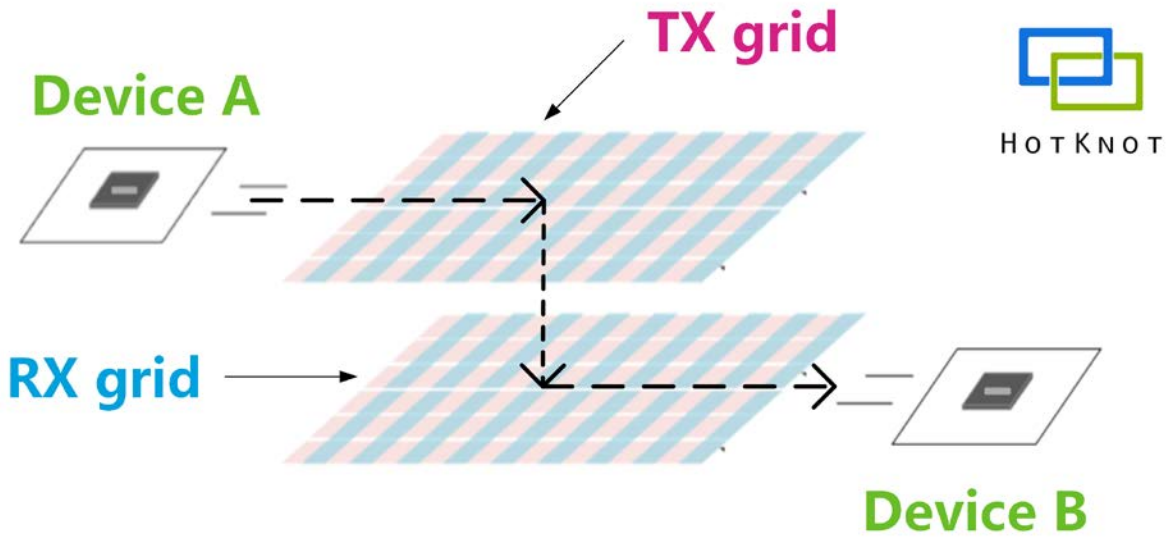

Figure 1 The interaction between the TX and RX grids on separate screens

The appeal of this technology to device manufacturers is that, unlike NFC where they have to add radio functionality and antenna to their BOM, HotKnot works with the phone's screen a component that is already in their BOM.

For user HotKnot offers similar tap and send/share data exchange features to NFC. Unlike NFC, however, there is much less danger of snooping or hacking as the devices need to be in very close proximity.

The MediaTek API for HotKnot (`com.mediatek.hotknot`) enable you to make full use of this technology by detecting the presence of a HotKnot enabled device touching the user's phone, setting up a connection between the two phone, and transferring data between the phones.

With Hotknot you can do pretty much anything you would otherwise do with NFC for phone to phone data transfer, such as:

- Exchange web addresses, contact data, photos, and videos
- Exchange game play moves
- Exchange any discrete data between app
- Exchange device capability data to enable Bluetooth pairing, WiFi connectivity, or other service hookup
- Facilitate mobile payments

2.2 Multiple SIM

Multi-SIM technology enables devices to use 2 or more SIM cards to access the mobile network. This enables users to make use of multiple mobile subscriptions from one phone. Typical examples might include:

- one SIM for a personal subscription and another for a business or an employer's subscription.
- regular travelers may have a SIM for their home network and another for a local network in the country they are visiting.
- taking advantage of different subscription packages among different operators; one SIM may offer cheap calls, another cheap data.

The MediaTek APIs (`com.mediatek.telephony`) allows you to get information about the available SIMs/subscriptions and listen for changes or activity on the SIM cards, such as call or data activity or connected cell details. In addition, the API enables apps to create SMS messages and send them using a specific SIM card.

2.3 Multimedia manipulation

Using hardware optimization MediaTek offers you a number of image and audio processing enhancements that can offer you apps both improved performance and unique functionality.

2.3.1 Portrait enhancements

Much like the tools used in commercial studios to enhance portraits for magazine and advertising, these image effects enable you to create apps that offer users the ability to enhance any portrait style images they capture. The range of effect provided include:

- skin smoothing
- skin tone adjustments
- facial 'slimming'
- eye enlargement.

Figure 2 Before (left) and after (right) processing with portrait enhancements

The Image editing effects (`com.mediatek.effect`) API enables you to take full advantage of these features in image editing apps.

2.3.2 Image Transformations

MediaTek chipsets provide hardware accelerated support for a collection of image editing tools that can:

- Flip images horizontally or vertically
- Rotate them by 90, 180 or 270 degrees clockwise
- Crop the image
- Apply dither to the image
- Adjust the encoding quality for JPEG images
- Adjust the sharpness of the image

The APIs (`com.mediatek.imagetransform`) enable you to make full use of these features and offer more responsive image editing apps.

2.3.3 HD audio recording and file tagging

MediaTek Android chipsets include features to enable HD audio recording and audio file tagging.

For users HD audio offers better recording quality. The technology provides for 48 KHz stereo recording with support from noise reduction, dynamic range compression, and stereo enhancement technology.

In your audio apps, the API enables the HD audio recording mode to be set, and recording paused and restarted. The HD recording modes available adjust the level of noise reduction applied to the recording:

- Normal (noise reduction off), for situations where the acoustic characteristics of the environment need to be captured, for example when recording music.
- Indoor (medium level of noise reduction), for recordings where the main sound source is close by, for example when recording a meeting or voice memo.
- Outdoor (low level of noise reduction), for situations where there are higher levels of background noise or the sound source is some distance away, such as the speaker in a lecture as well as recording outdoors.

In addition, the API provides for setting audio file tags for album and artist information. This means that audio recorded from your app will be index correctly in the device's music player – for example the user could create a 'Jack's birthday' album and record 'Happy Birthday to Jack', 'Jack opening his presents', and more, then have instant access to the recordings from their music player.

The APIs (`com.mediatek.media`) enable you to make full use of the HD recording settings and file tagging in your audio apps.

2.4 Multimedia capture

2.4.1 Gesture detection

Devices with MediaTek chipsets provide the ability for apps to watch a device's camera image stream and detect the open palm and victory (V) hand gestures.

Using this feature apps can initiate actions based on these gestures, such as taking a picture or starting or stopping a process. For example a timing application could start a stopwatch on the open palm gesture and stop it when the user indicated they have finished with the victory gesture.

The APIs (`com.mediatek.gesture`) enable you to watch for and detect the gestures in your apps.

2.4.2 Multi Angle View (MAV) image capture

Using this technology users capture images from around an entire object or person. The recorded images are stored in a Camera and Imaging Products Association Multi-Picture Format file, along with information on the angle at which each image was captured (provided by the device's accelerometer).

Using these images an app can create 3D models or other renditions of the object or person photographed, this opens up possibilities for making use of emerging technologies such as 3D printing.

The APIs (`com.mediatek.hardware`) enable you to add MAV file recording to your app. You are then free to add additional processing of the image file to your apps.

2.4.3 Camcorder audio and video settings

This feature enables your apps to access the details of the front or back camcorders setting, including:

- for video: output file video codec format, bit rate, frame rate, and resolution.
- for audio: audio codec format, bit rate, sample rate, and the number of audio channels for recording.

The APIs (`com.mediatek.camcorder`) enable you to make full use of the HD recording settings and file tagging in your audio apps.

3 Developing for MediaTek powered devices

This section provides an overview to developing and testing apps that make use of the features of Android devices built around MediaTek chipsets.

3.1 MediaTek SDK for Android

To enable the development of apps, the MediaTek SDK for Android is based on reference implementations of Android devices powered by MediaTek chipsets. It's delivered as a set of extensions that you drop into the Google Android SDK. The SDK provides:

- libraries for the MediaTek APIs, which you import into your code.
- x86 MediaTek emulator based, which provides features for testing the telephony and multi-SIM features of your apps. In addition to all the usual Android emulator features, the emulator offers a vibrator indicator and hot-swappable SD card.
- a customized version of the Dalvik Debug Monitor Server (DDMS), which adds support for testing telephony features, such as plugging and unplugging a headset. The MediaTek DDMS can also be used to send telephony commands to test call and SMS features.

3.2 Reference Android Phones

The MediaTek emulator provide for the testing of telephony and multi-SIM features. For apps that make use foo the MediaTek features, all testing will have to be done on a device. An up to date list of devices suitable for testing is provided in the Reference Android Phones page on the MediaTek Labs website.

4 Getting started

This section provides you with a detailed guide to installing, configuring, and using the MediaTek SDK for Android.

The instructions provided in this section are based on installing and using the SDK on Microsoft Windows. Unless otherwise noted, these instructions apply also to the use of the SDK on Apple Mac and Linux based computers.

The MediaTek emulator and customized DDMS are currently not available for Apple Mac.

4.1 Prerequisites

Before installing the MediaTek SDK for Android you should have an installation of the Android SDK, including the required supporting software. For more information see Get the Android SDK on the Android Developer website

4.2 Downloading

The latest version of the MediaTek SDK for Android package can be obtained from the MediaTek Labs website here. Within the packages there are:

- `mediatek_android_sdk_api-<api_package_version>.zip` — the MediaTek API package. Containing the MediaTek's SDK API libraries and MediaTek Emulator components for Microsoft Windows and Linux.
- `mediatek_android_sdk_tools-<tools_package_version>.zip` — the MediaTek Tools package. Includes a customized version of DDMS for Microsoft Windows and Linux versions.
- `Release-<sdk_version>.txt` — the versions release notes providing a summary of changes made in this release.

4.3 Installation

This section covers the procedures for installing the SDK and tools packages.

4.3.1 Installing the API package

To install the MediaTek Android SDK API package:

1 Extract the content of the `mediatek_android_sdk_api-<api_package_version>.zip` file to a new folder within the `android sdk/add-ons` folder on your computer, such as `sdk\add-ons\mtk_sdk_api_addon-19.3`, as shown in Figure 3.

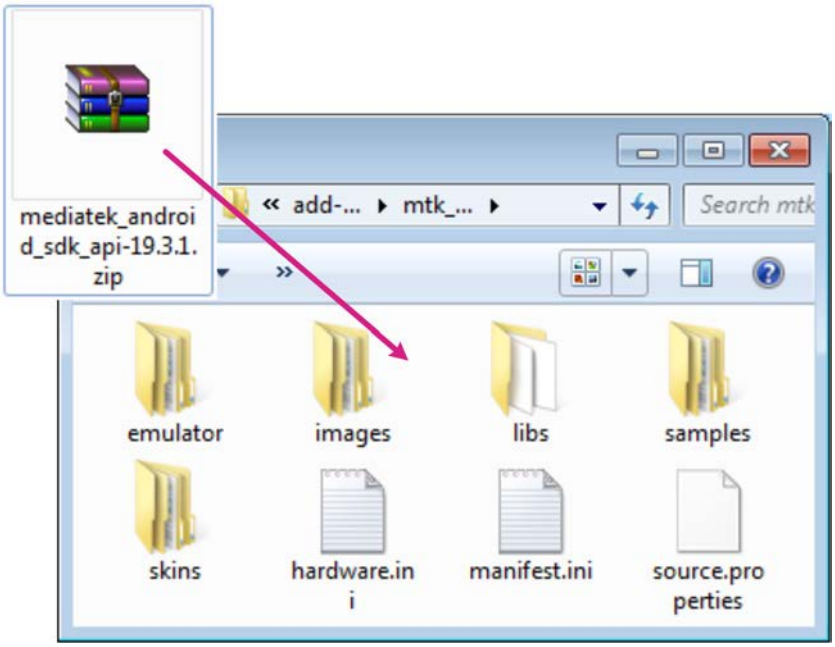

Figure 3 Extracting the content of the API ZIP to the Android SDK

4.3.2 Install the tools package

To install the MediaTek Android SDK tools package (customized DDMS):

1 Extract `mediatek_android_sdk_tools<tools_package_version>.zip` to a convenient folder on your local disk.

4.4 Start developing with the MediaTek APIs

You can now start coding with the MediaTek APIs.

4.4.1 MediaTek feature APIs

To do this select **MediaTek Add-On (MediaTek Inc.)** as the project target. You will then be able to cope with the APIs, as shown in Figure 4.

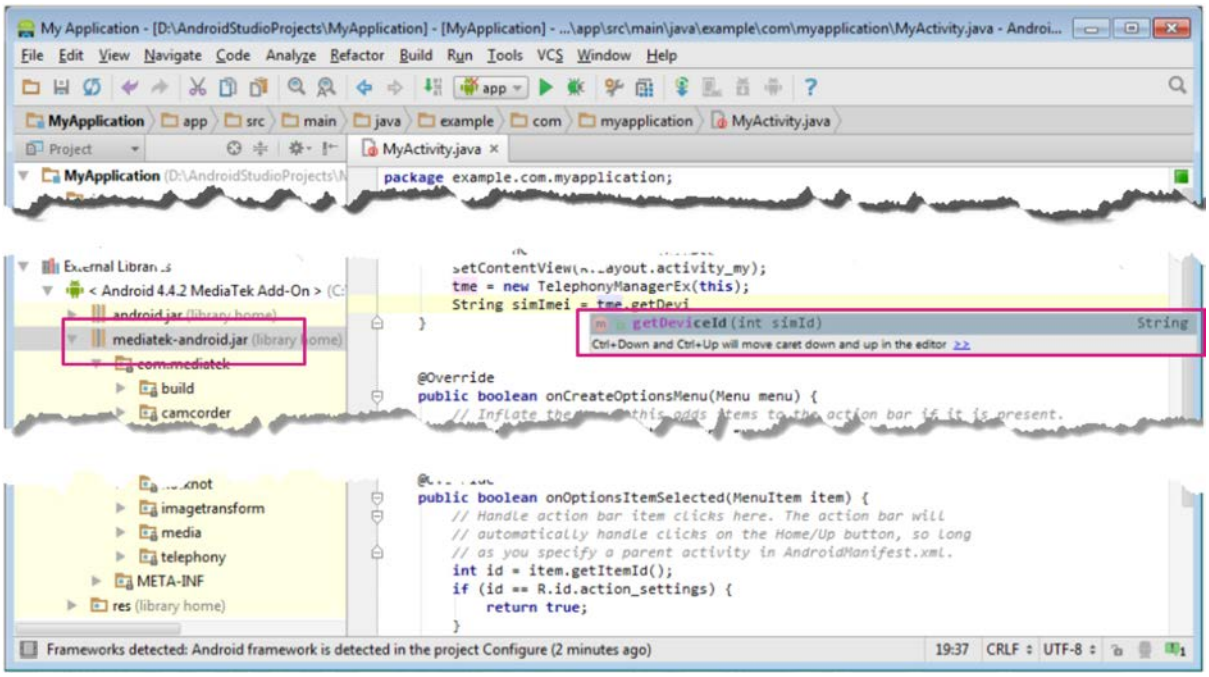

Figure 4 Coding with the MediaTek APIs

4.5 MediaTek Compatibility API

As you'll want you apps to be able to run on both Android devices with MediaTek chipsets and those based on other chipsets, the Compatibility API is provided to enable your apps to test for the presence of the MediaTek APIs. To make use of this API, do the following:

1 In your project create a `libs` folder, as shown in Figure 5.

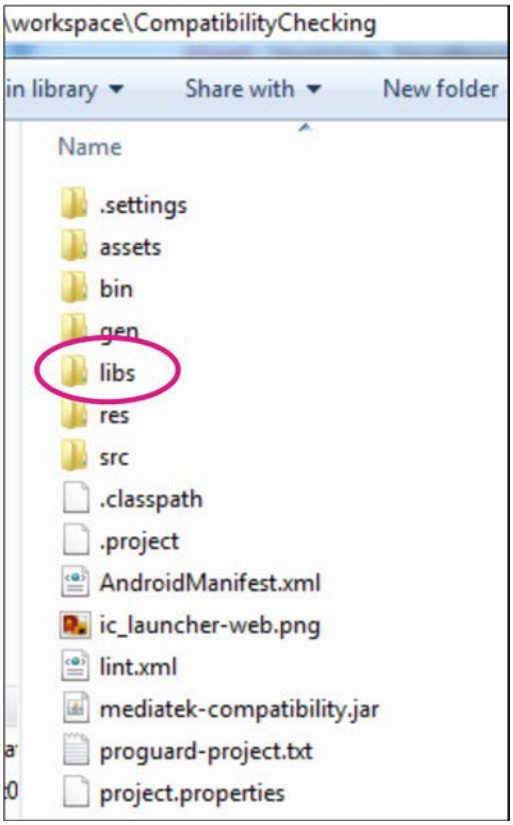

Figure 5 Creating a libs folder for the Compatibility API

2 Copy `mediatek-compatibility.jar` from the Android SDK's plug-in folder `mediatek_android_sdk_api-<api_package_version>\libs` to the `libs` folder in your project, as shown in Figure 6.

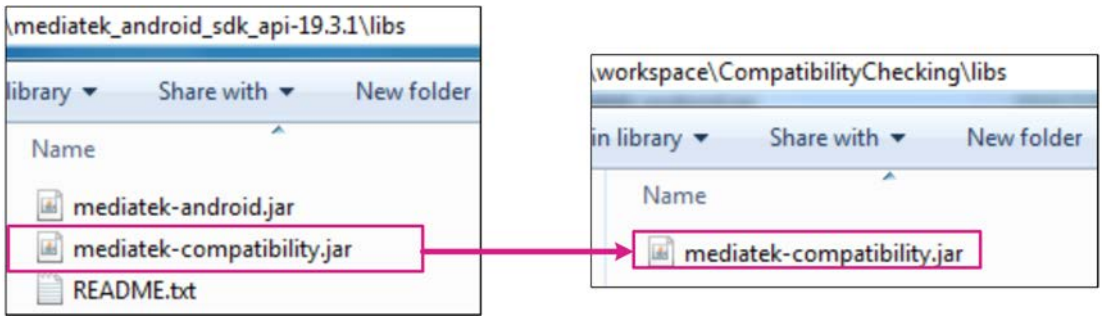

Figure 6 Copying the Compatibility API's jar to the project

3 Refresh the project to check that the JAR file is found in the `libs` folder, see Figure 7.

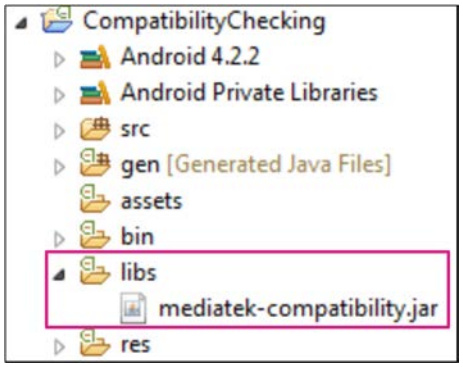

Figure 7 The Compatibility API JAR file in the app project

4 You can now expand the `mediatek-compatibility.jar` to view the APIs, as shown in Figure 8.

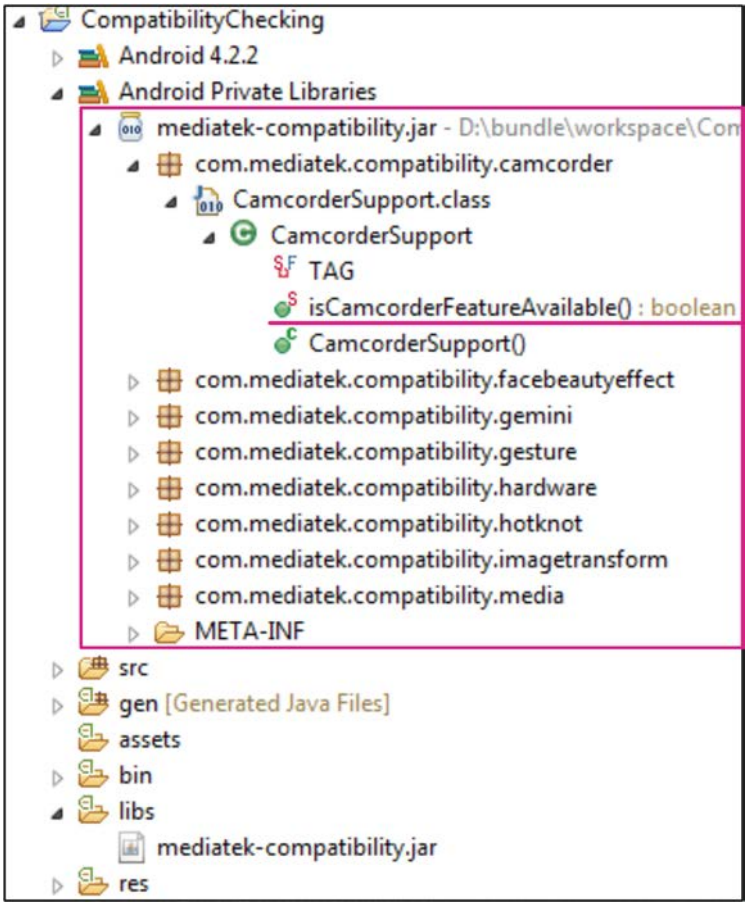

Figure 8 The unpacked Compatibility API JAR showing the available APIs

5 Write your compatibility check code using API, for example:

```
If (com.mediatek.compatibility.camcorder.CamcorderSupport.\
    isCamcorderFeatureAvailable(this)==true){
    // body
}
```

6 Build the project to create the apps APK file, which will be found in the `bin` folder as usual, as shown in Figure 9.

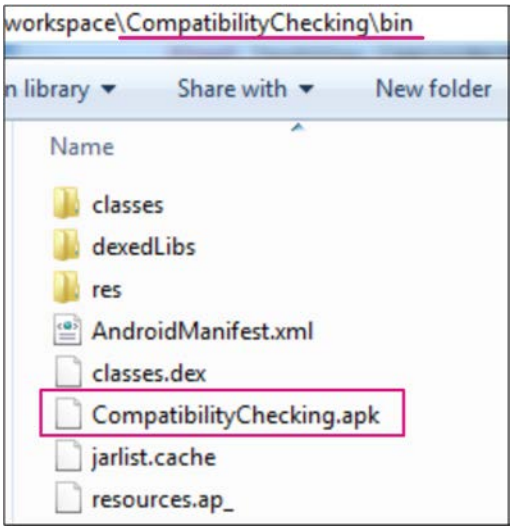

Figure 9 The built project's APK file in the bin folder

4.6 Configuring and using the MediaTek emulator

This section explains how to configure the MediaTek emulator and use it to test your apps with telephony and multi-SIM features.

The MediaTek emulator and customized DDMS are currently not available for Apple Mac.

4.6.1 Replace the standard emulator

To use the MediaTek emulator the first step is to replace the standard Android emulator executables.

You may wish to take a copy of the original emulator files so you can restore them later.

1 Copy the MediaTek emulator files from `android-sdk/add-ons/mtk_sdk_addon-<tools_package_version>/emulator/<OS version>` replacing the corresponding files in `android-sdk/tools`, as shown in Figure 10.

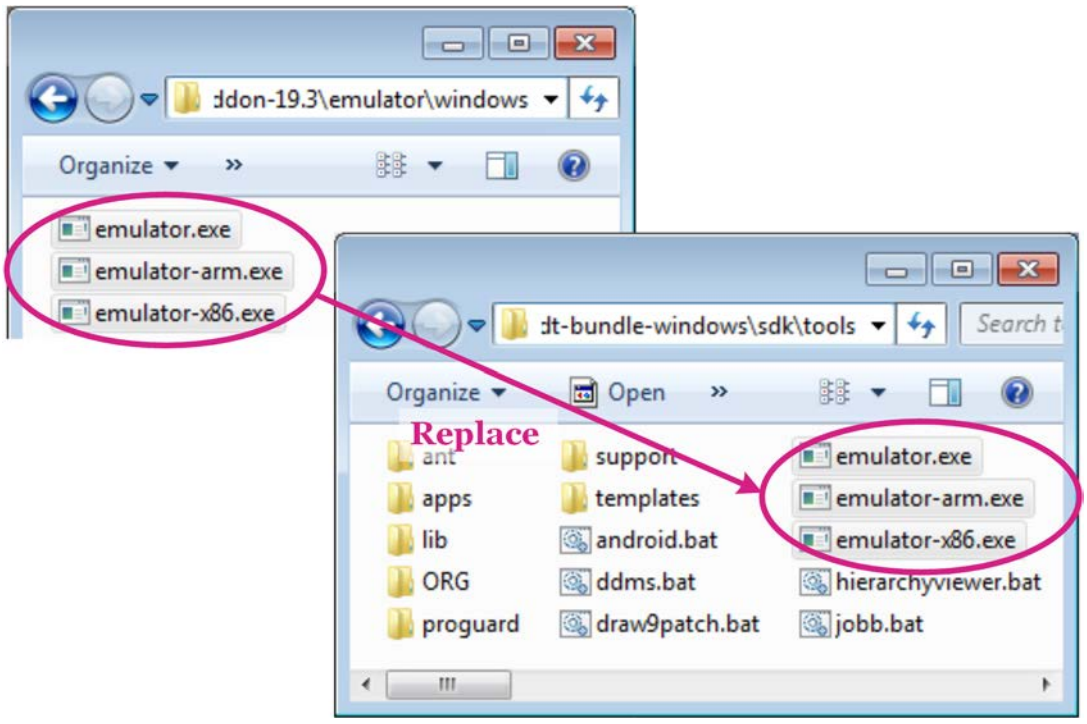

Figure 10 Replacing the standard emulator file with those for the MediaTek emulator

4.6.2 Create an Android Virtual Device

To enable the Eclipse IDE to run and debug your software in the MediaTek emulator you first need to create an Android Virtual Device (AVD). The AVD provides a model of a device by defining hardware and software options to be emulated. You do this from within the Eclipse IDE as follows

1 On the **Window** menu click **AVD Manager**.

2 In Android Virtual Device Manager, see Figure 11, click **New**.

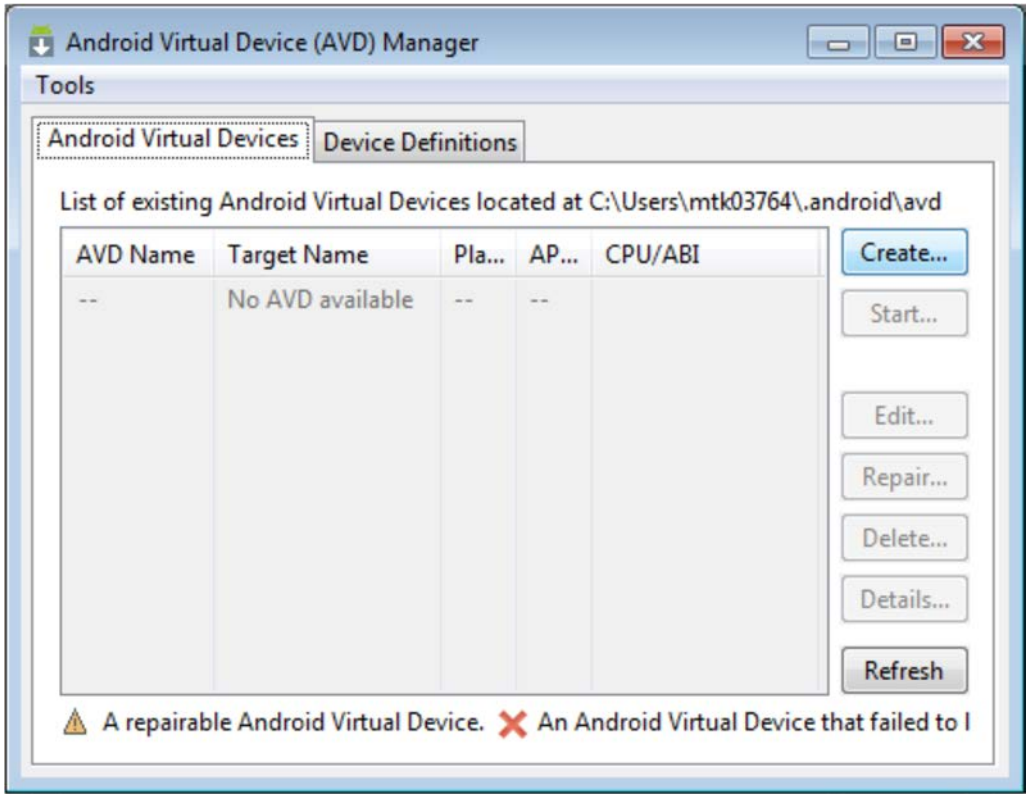

Figure 11 The Android Virtual Device Manager

3 In the **Create new Android Virtual Device (AVD)** dialog, see Figure 12, define the **AVD Name**, **Device**, **Target**, and other features — recommended values are provided in Table 1.

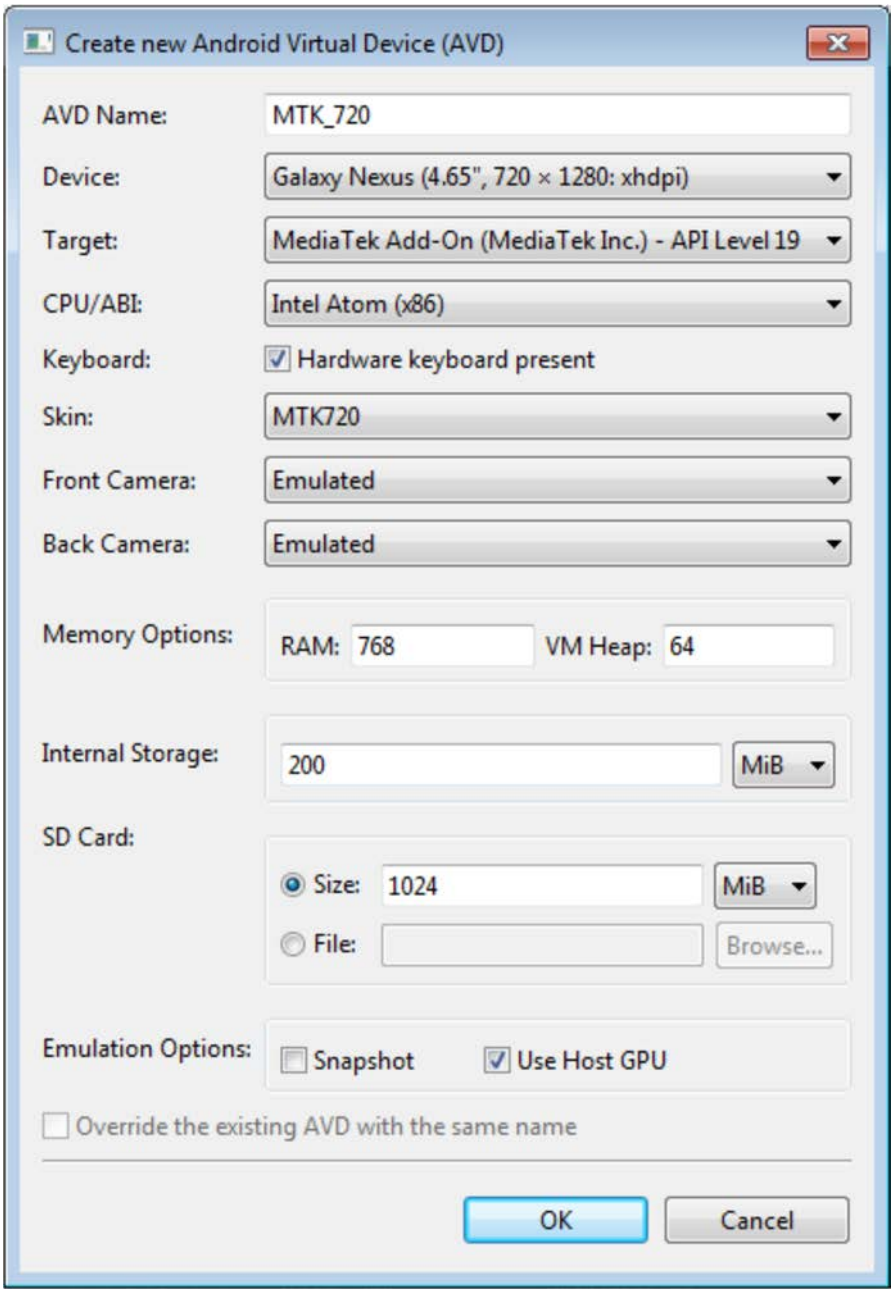

Figure 12 Settings for the MediaTek emulator AVD

	Windows 7	Windows 7	Linux Ubuntu	Linux Ubuntu
Resolution	720	1080	720	1080
Device	Galaxy Nexus (4.65", 720x1280)	Nexus 5 (4.95",1080x1920)	Galaxy Nexus (4.65", 720x1280)	Nexus 5 (4.95",1080x1920)
Target	MediaTek Add-On (MediaTek Inc.) - API Level 19	MediaTek Add-On (MediaTek Inc.) - API Level 19	MediaTek Add-On (MediaTek Inc.) - API Level 19	MediaTek Add-On (MediaTek Inc.) - API Level 19
CPU/ABI	Intel Atom (x86)	Intel Atom (x86)	Intel Atom (x86)	Intel Atom (x86)
Keyboard	Yes	Yes	Yes	Yes
Skin	MTK720	MTK1080	MTK720	MTK1080
Front Camera	Emulated	Emulated	Emulated	Emulated
Back Camera	Emulated	Emulated	Emulated	Emulated
RAM	512	768	1024	2048
VM Heap	64	64	64	64
Internal Storage	200	200	200	200
SD Card	256	256	256	256
Snapshot	No	No	No	No
Use Host GPU	Yes	Yes	Yes	Yes

Table 1 Suggested values for AVD settings

4 When you have entered the AVD settings, click **OK**.

5 In the Android Virtual Device Manager, click **Start** to launch the MediaTek emulator.

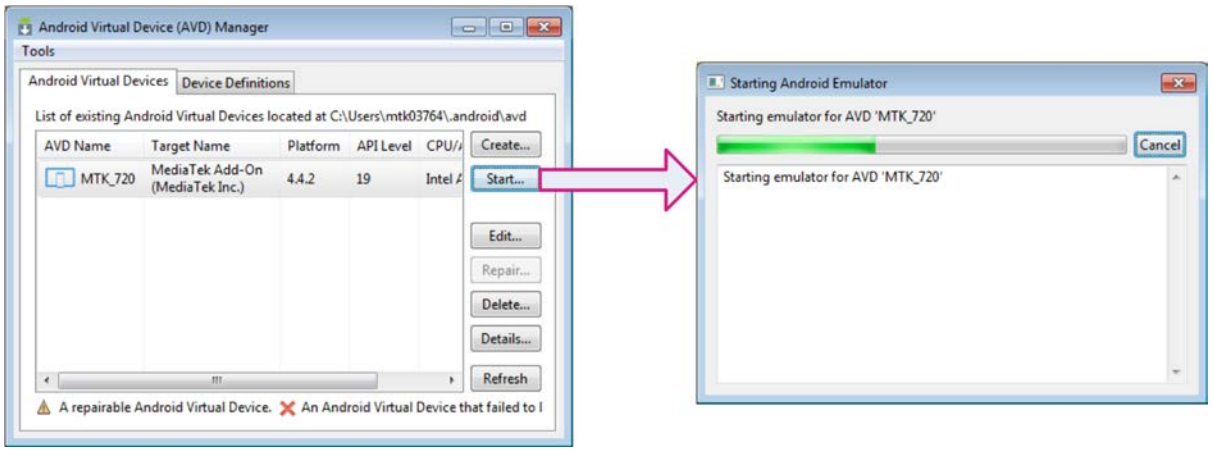

Figure 13 Launching the MediaTek emulator

6 After the emulator starts, you'll see there are two SIM indications in the status bar. If you can't see the icon, close the emulator, check that you have replaced the emulator executable files as described in step 2 of Installation, and once done restart the emulator.

Figure 14 The MediaTek emulator showing the dual-SIM indicator

The emulator is now set up.

The emulator is based on Android SDK Tools Revision 17 and as such includes the experimental support for the GPU on host computers. Please refer to the Configuring Graphics Acceleration notes on the Android Developer website.

4.6.3 Additional configuration and tools

You may find that the emulator performance can be improved by using Intel® HAXM. For details please refer to the section Intel® Hardware Accelerated Execution Manager.

4.7 MediaTek Emulator supported APIs

The MediaTek emulator doesn't provide support for testing all of the available APIs; only the SDK level version checking and multi-SIM (Telephony / SMS) APIs can be tested in MediaTek emulator, as shown in Table 2.

Name	Class	SDK Level	Can run on emulator
MediaTek SDK	SdkVersion	1	Y
Telephony	TelephonyManagerEx	1	Y
SMS	SmsManagerEx	1	Y
Camera MAV	CameraEx	1	N
Audio/Video	MediaRecorderEx	1	N
	CamcorderProfileEx		
HotKnot	HotKnotAdapter	2	N
	HotKnotMessage		
Gesture	Gesture	3	N
ImageTransform	ImageTransformFactory	3	N
Portrait enhancements	Effect	3	N
	EffectFactory		

Table 2 MediaTek API support in the SDK

4.8 MediaTek DDMS Guide

The MediaTek Android SDK offers a customized version of the Dalvik Debug Monitor Server (DDMS). The customizations provide simulations for:

- Calls from either SIM1 or SIM2
- SMS sending from SIM 1 or SIM2
- Headset changes
- Power states
- Sensor values
- SD card insertion and removal

This section explains how to run DDMS and use its features to control aspects of the MediaTek emulator.

4.8.1 Running the MediaTek DDMS

To launch the customized DDMS, open the `ddms.jar` file from the folder in which it was installed.

4.8.2 Simulating calls from either SIM card

To simulate a call from SIM1 and SIM2, under **Emulator Controls** click **Call1** or **Call2** as shown in Figure 15.

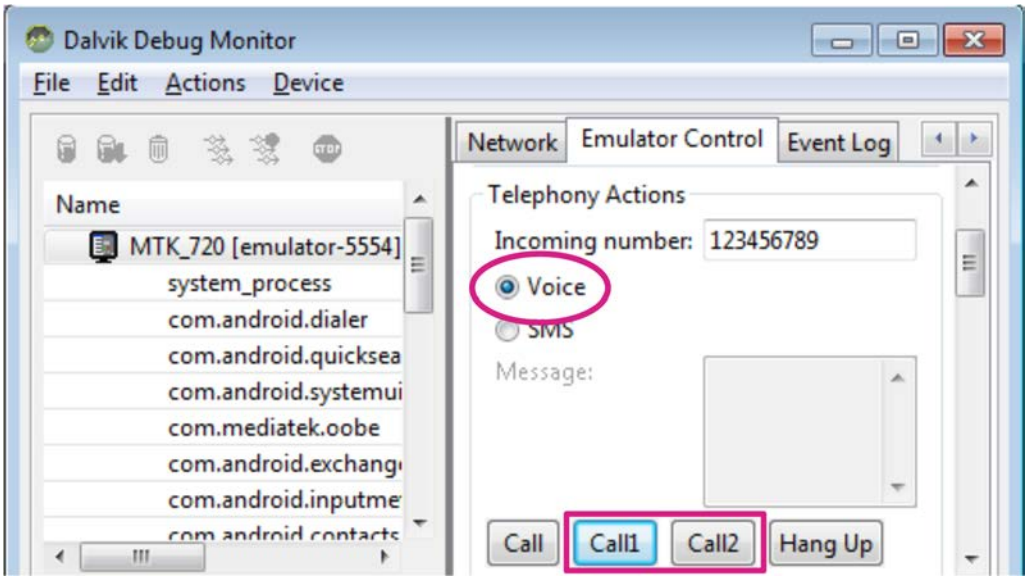

Figure 15 Simulate calls with Call1 and Call2

4.8.3 Simulate SMS being sent from either SIM card

To simulate am SMS being sent from SIM1 or SIM2, under **Emulator Controls** click **Send1** or **Send 2** as shown in Figure 16.

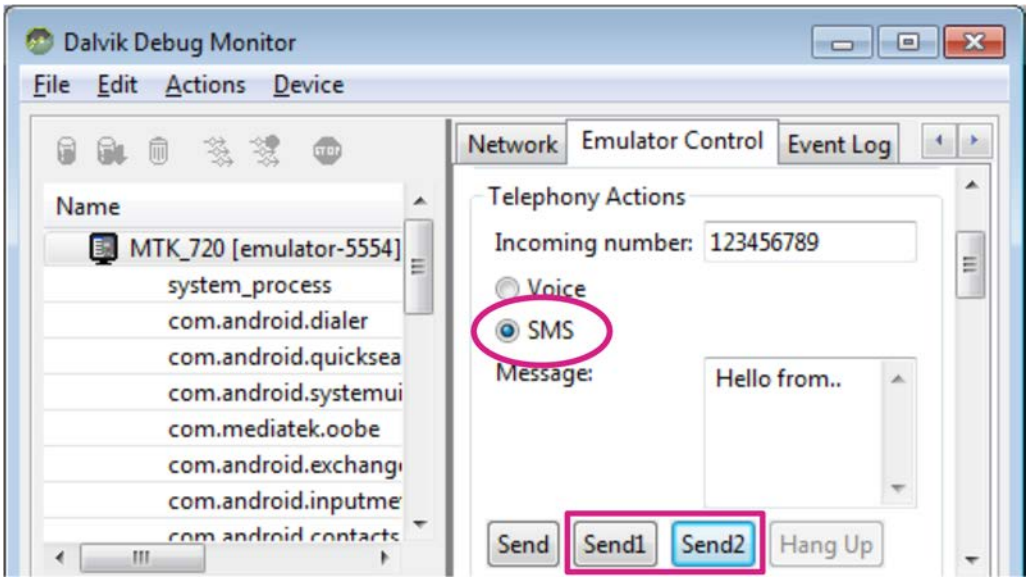

Figure 16 Simulating the sending of an SMS with Send1 and Send 2

4.8.4 Simulate headset changes

As shown in Figure 17, under **Emulator Controls** then **Headset** the following headset state changes can be simulated:

- removal of a headset, by clicking **Unplug headset.**
- insertion of a headset, by clicking **Plug headset.**
- insertion of a headset with microphone, by clicking **Plug headset (with microphone).**

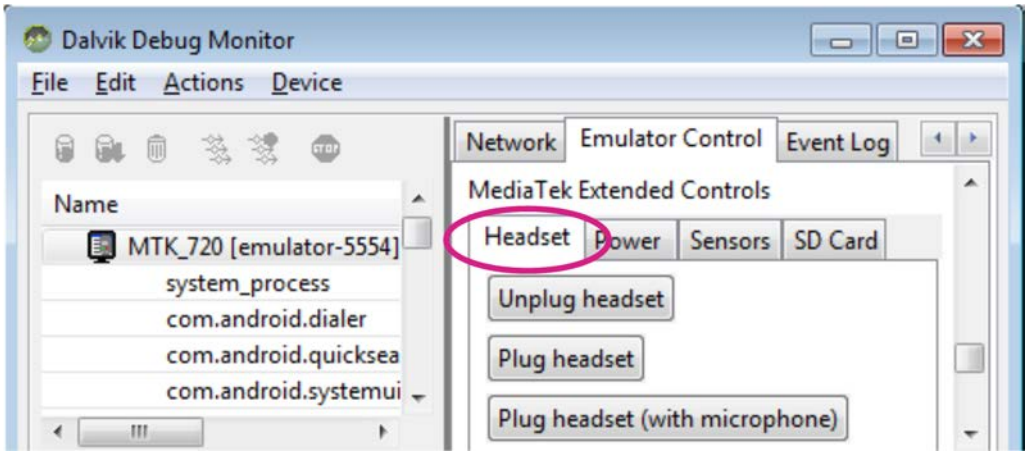

Figure 17 Various headset simulations are provided

4.8.5 Simulating power states

Under **Emulator Controls** then **Power**, as shown in Figure 18, tools to simulate the following are provided:

- defining a level of battery **Capacity** and applying this value to the emulator by clicking **Set**.
- clicking **Unknown**, **Charging**, **Discharging**, **Not charging**, or **Full** to simulate battery status.
- clicking **Unknown**, **Good**, **Overheat**, **Dead**, **Over Voltage**, or **Unspecified Failure** to simulate battery health.
- clicking **Online** or **Offlin**e to simulate whether a charger is connected or not.

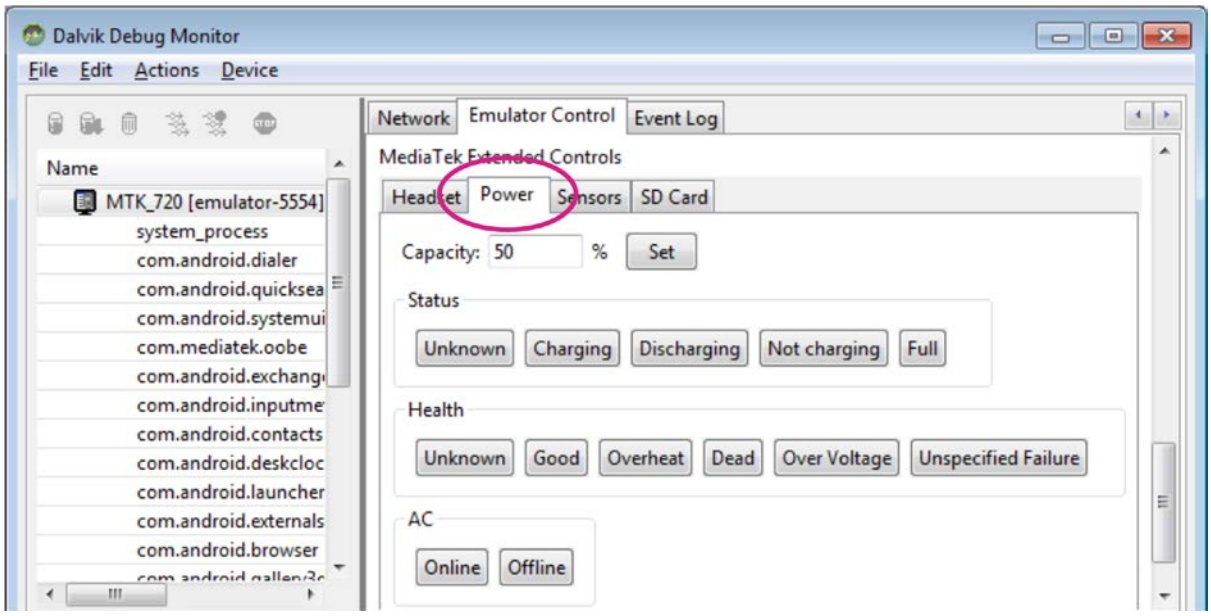

Figure 18 Various power simulation features are available

4.8.6 Simulated sensor states

Under **Emulator Controls** then **Sensor**, as shown in Figure 19, the following values can be set for sensors:

- Accelerometer: **X**, **Y**, and **Z**.
- Orientation Sensor: **Azimuth**, **Pitch**, and **Roll**.
- Magnetic Field Sensor: **X**, **Y**, and **Z**.
- Temperature Sensor: temperature.

Once value have been entered the sensor value in the emulator are updated by clicking **Set**.

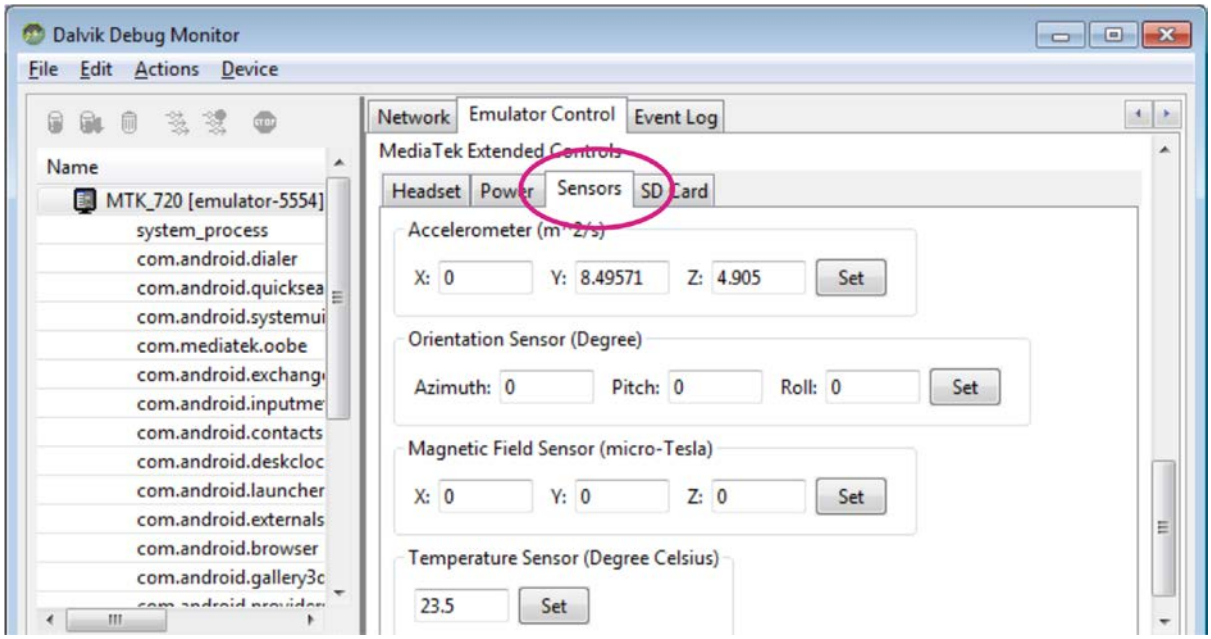

Figure 19 Various states can be set for sensors

4.8.7 Simulated SD Card states

Under **Emulator Controls** then **SD Card**, as shown in Figure 20, the presence or absence of an SD card can be simulated by clicking **Insert SD Card** or **Remove SD Card**.

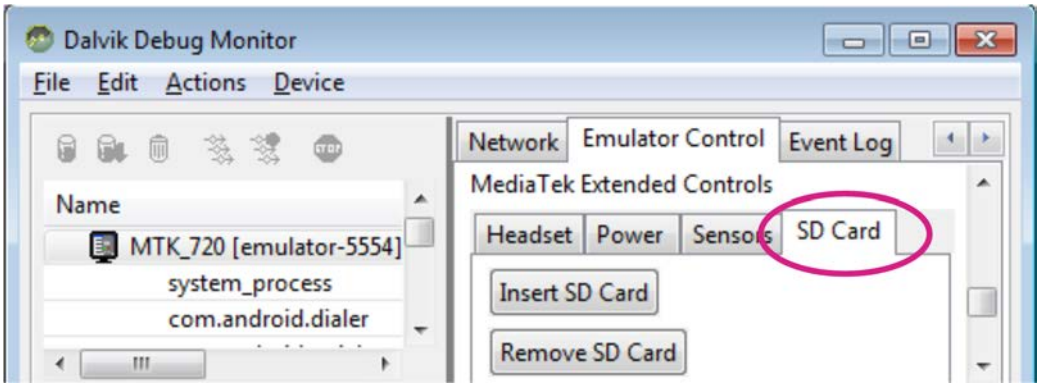

Figure 20 Options for simulating an SD card being inserted and removed

4.9 Other tools

This section described other Android SDK tools that you might find useful when working with the MediaTek emulator.

4.9.1 Intel® Hardware Accelerated Execution Manager

The Intel® Hardware Accelerated Execution Manager (HAXM) can be used to optimize the performance of your MediaTek emulator. You can download this tool from the Intel website here and follow the installation instructions provided.

You can check whether you have Intel® HAXM installed as follows:

- In Windows at the command prompt enter `sc query intelhaxm` as shown in Figure 21.

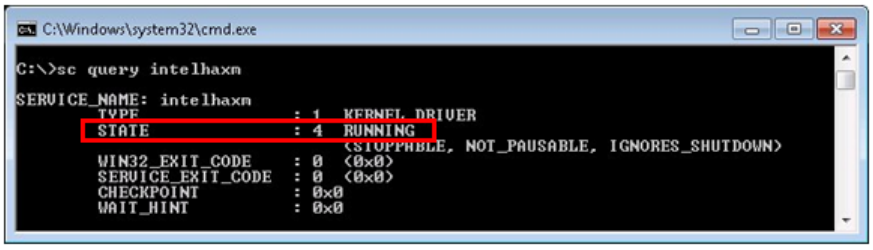

Figure 21 Windows command to query the presence of HAXM

- In Linux in the terminal windows enter `kvm-ok` as shown in Figure 22.

```
$kvm-ok
INFO: Your CPU supports KVM extensions
INFO: /dev/kvm exists
KVM acceleration can be used
```

Figure 22 Linux command to query the presence of HAXM

If the results shown above aren't obtained, then Intel HAXM isn't installed.

5 API Guides

The section provides information about the API level and a guide to using three of the key APIs:

- HotKnot (`com.mediatek.hotknot`)
- Multi-SIM (`com.mediatek.telephony`)
- MAV (`com.mediatek.hardware`)

5.1 API Level and Compatibility

As with the Android SDK APIs, the MediaTek SDK APIs are designed to be backward compatible, so that your apps should always work on the version it was built for and later versions (MediaTek SDK API Level) of MediaTek Android platforms.

Each supported MediaTek device comes with a MediaTek SDK API ate level 1, 2, or 3. As shown in Figure 23, an application written using Level 1 can run on the devices with MediaTek SDK API Level 1 and higher.

While this is the first developer release of the APIs, you may still encounter devices supporting levels 1 and 2 as the APIs were included in earlier releases of the version of Android made available to device manufacturers.

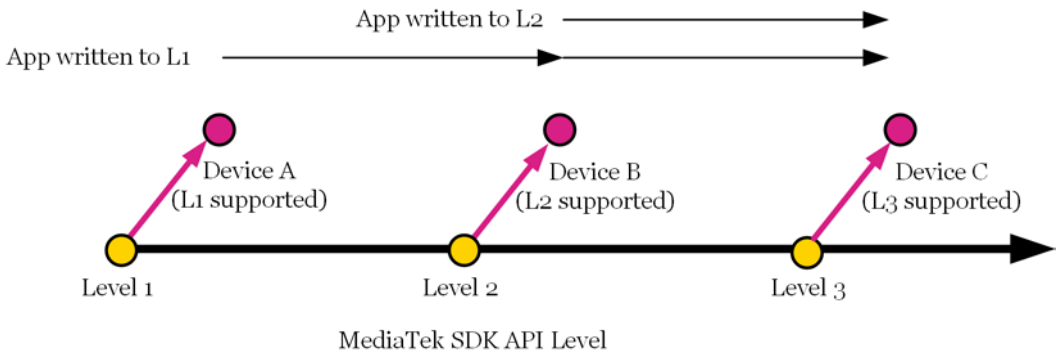

Figure 23 The support for APIs across levels

The APIs supported in levels 1, 2, and 3 are shown in Table 3'

Package	Level 1	Level 2	Level 3
com.mediatek.build	X	X	X
com.mediatek.telephony	X	X	X
com.mediatek.hardware	X	X	X
com.mediatek.media	X	X	X
com.mediatek.camcorder	X	X	X
com.mediatek.hotknot		X	X
com.mediatek.effect			X
com.mediatek.imagetransform			X
com.mediatek.gesture			X

Table 3 MediaTek API support by level

5.2 Using HotKnot (com.mediatek.hotknot)

HotKnot is a near field communication technology that uses the features of capacitive touch screens to enable the transfer of files and data between devices. The operating range is typically 0 ~ 1 cm with a data rate of 1.43Kbps~7.88Kbps offering half-duplex serial transfer.

5.2.1 Creating a HotKnot sender application

To create an application that sends data using HotKnot:

1 Create an Android application project with activities

2 In the `AndroidManifest.xml` add:

```
<uses-permission android:name="android.permission.HOTKNOT"/>
```

3 Initialize Hotknot API and check the device is HotKnot enabled

a Get the HotKnotAdapter

```
HotKnotAdapter hotKnotAdapter = HotKnotAdapter.getDefaultAdapter(activity);
```

b Check that HotKnot is enabled

```
if(hotKnotAdapter.isEnable()) {
    Log.d(TAG,"HotKnot is enable");
}
else {
   hotKnotAdapter.enable();
}
```

c Setup the callback to detect when HotKnot transfer is complete

```
hotKnotAdapter.setOnHotKnotCompleteCallback {
        new HotKnotAdapter.OnHotKnotCompleteCallback() {
                public void onHotKnotComplete(int reason) {
                        Log.d(TAG, “onHotKnotComplete reason:” + reason);
                }
        }, activity);
```

4 Prepare the message to send:

o Case 1: Send URIs

```
URI[] uri = {uri1, uri2};
hotKnotAdapter.setHotKnotBeamUris(uris, activity);
```

o Case 2: Send Messages

```
String mimeType = “binary/com.mediatek.demo”;
byte[] payload = “Hello”.getBytes();
HotKnotMessage message = new HotKnotMessage(mimeType, payload);
hotKnotAdapter.setHotKnotMessage(message, activity);
```

5 Place the screens of the HotKnot devices (Sender and Receiver devices) together

5.2.2 Create a HotKnot receiver application

To create an application that sends data using HotKnot:

1 Create an Android application project with activities

2 In `AndroidManifest.xml` add:

```
<uses-permission android:name=”android.permission.HOTKNOT”/>
```

3 Add HotKnot message filter rules:

```
<activity
       android:name=”.DemoActivity”
       android:label=”@string/app_name”>
      <intent-filter>
              <action android:name=”com.mediatek.hotknot.action.MESSAGE_DISCOVERED” />
              <data android:mimeType=”binary/com.mediatek.demo” />
              <category android:name=”android.intent.category.DEFAULT” />
      </intent-filter>
</activity>
```

4 Place the screens of the HotKnot devices (Sender and Receiver devices) together

5 Get and parse HotKnot message until the target activity has been started

```
@Override
protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);

        Intent intent = getIntent();
        If(HotKnotAdapter.ACTION_MESSAGE_DISCOVERED.equals(intent.getAction()) {

        }
        else {
                Log.w(TAG, “Unknown intent: “ + intent);
                finish();
                return;
        }

        // Get HotKnot message
        String mimeType = intent.getType();
        HotKnotMessage hotKnotMessage =
intent.getByteArrayExtra(HotKnotAdapter.EXTRA_DATA);

        if(“binary/com.mediatek.demo”.equalsIgnoreCase(mimeType)) {
                String message = new String(hotKnotMessage);
                 if(“hello”.equals(message)) {
                        // Correct
                 }
         }
```

5.3 Using multi-SIM (com.mediatek.telephony)

There are two classes that provide the multi-SIM functionality in the MediaTek platform: `SmsManagerEx` and `TelephonyManagerEx`.

5.3.1 TelephonyManagerEx Class

The `TelephonyManagerEx` class provides access to information about the telephony services on a device, particularly for a multiple SIM cards equipped device. Applications can use the methods in this class to determine telephony services and states, as well as access some types of subscriber information. Applications can also register a listener to receive notification of telephony state changes.

Example: If the apps want to know the call state on specific SIM, the following code can be used:

```
TelephonyManagerEx.getDefault().getCallState(simID);
```

`simID`: 0 is for SIM1 phone call state; 1 is for SIM2 phone call state.

5.3.2 SmsManagerEx Class

The `SmsManagerEx` class is an extension of `SmsManager`. The extension's functions primarily support multi-SIM features, with additional variable `slotId` and `simId`. If you want to develop applications using the multi-SIM features for SMS features, you can use the SDK emulator to test these features.

Here is the code for sending an SMS with message text "Test words":

```
String text = "Test words";/* The words want to send */
```

1 Divide the words to multi-parts, because the size of one SMS is limited.

```
// divide a long SMS to multi-parts
messages = SmsManagerEx.getDefault().divideMessage(text, codingType);
int messageCount = messages.size();
```

2 To get information about send finished or delivery finished, create `deliveryIntents` and `sentIntents`, on every part of the message that is sent or delivered. The intent will broadcast out.

```
/*
* Every part can add a deliveryIntent and a sentIntent, After delivery
* or sent finished, the intent will broadcast out.
*/

ArrayList<PendingIntent> deliveryIntents =  new
ArrayList<PendingIntent>(messageCount);
ArrayList<PendingIntent> sentIntents = new ArrayList<PendingIntent>(messageCount);
for (int i = 0; i < messageCount; i++) {
    // only the last part of one SMS need deliveryReport
    if (mRequestDeliveryReport && (i == (messageCount - 1))) {
      Intent intent = new Intent("sms_test_delivery_intent");
      deliveryIntents.add(PendingIntent.getBroadcast(mContext, i, intent, 0));
    } else {
      deliveryIntents.add(null);
    }
    Intent intent = new Intent("sms_test_intent_sent_a_part");
    // normally we need to know all parts of the message are sent.
    if (i == messageCount - 1) {
      intent.putExtra("sms_test_key_is_last_part", true);
    }
    sentIntents.add(PendingIntent.getBroadcast(mContext, i, intent, 0));
    /// @}
}
```

3 Now, send the SMS to the target address, `scAddr` should be null unless you need specify a service center.

```
// Send this multi-part text to target.
try {
  SmsManagerEx.getDefault().sendMultipartTextMessageWithEncodingType(
    destAddr/* targeAddress */, scAddr/* serviceCenter */, messages, codingType,
    sentIntents, deliveryIntents, slotId);
  } catch (Exception ex) {
    throw new MmsException("SmsMessageSender.sendMessage: caught " + ex +
        " from SmsManager.sendTextMessage()");
  }
```

5.4 Using the MAV capture (com.mediatek.hardware)

There is one class that provides MAV functionality on the MediaTek platform: `CameraEx`.

5.4.1 CameraEx Class

The `CameraEx` class is used to start and stop Multi Angle View (MAV) image capture. The workflow for your application to take a MAV photo is similar to taking a normal picture, that is calling `startMav` instead of `takePicture`, and calling `stopMAV` after the capture number reaches the number of images set when starting MAV capture. In addition, you can stop MAV during the capture process, and then set the parameter `isMerge` to false to discard the capture.

Here is the code for recording a MAV:

1 Implement the interface `CameraEx.MavCallback`

```
private final class MavFrameCallback implements MAVCallback {

  public void onFrame(byte[] jpegData) {
    // To Do
  }
}
```

2 Add a Listener for the data callback.

```
mCamera.getCameraDevice().setMAVCallback(getMavCallback());
```

3 Start the capture of a number of images into a MAV.

```
mCamera.getCameraDevice().startMAV(NUM_MAV_CAPTURE);
```

`NUM_MAV_CAPTURE`: Total number of images to include in the MAV

4 Stop the MAV capture.

a If the number of individual imaged capture reached set total, `setmavcallback` to `null`.

```
If (isMerge) {
    mCamera.getCameraDevice().setMAVCallback(null);
}
```

b Otherwise, make sure to use the same camera device before calling `stopMAV`.

```
if (holder.isSameCameraDevice(mCamera.getCameraDevice(),mCamera.getCameraId())) {
    // means that hw was shutdown
    // and no need to call stop anymore.
    mCamera.getCameraDevice().stopMAV(isMerge ? 1:0);
} else {
    Log.w(TAG, “doStop device is release?”);
}
```