ChirpVision SDK for Android

CHIRPVISION

DELIVERING LIVE, INTERACTIVE
EVENT CASTING TO SMART PHONES

Getting Started with ChirpVision SDK for Android

ChirpVision SDK for Android is the easiest way to integrate your Android app with ChirpVision's platform. The SDK provides support for live streaming video, DVR functionalities, Highlights tickers*, Advertising thru banners*, Social media interaction* and a white label reloadable skin style that can be setup per event.

*Available as separate add-on packages.

This guide shows you how to get started developing with the ChirpVision SDK for Android. The screenshots in this guide are from an Apple OS X system, but the steps for installing on a Windows computer are almost identical. Any differences in the instructions are highlighted.

1. Install the Prerequisites

Eclipse is the most popular choice for developing Android apps. 'Eclipse Classic', found on the downloads page, is sufficient for Android development. This guide assumes you are using v4.2 (Juno) or later with a default install. You also need Java v1.6 or later.

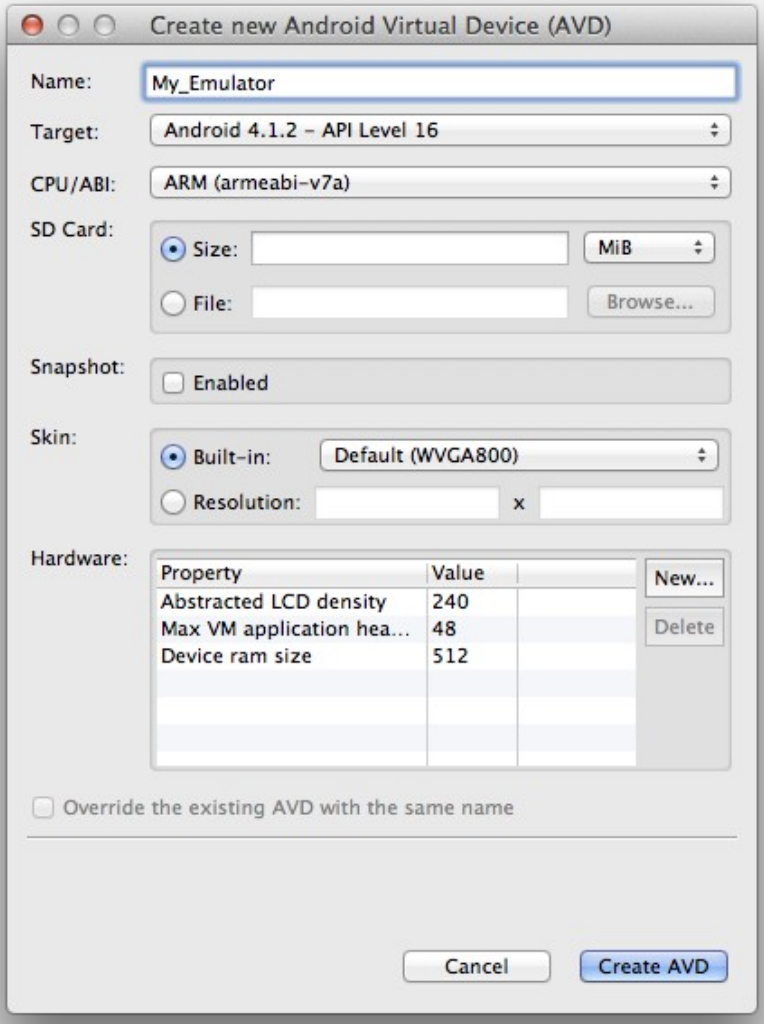
Once you have Eclipse, download the Android SDK, available from the Android Developers site, and install it.

Also make sure you follow the instructions to install the Android Developer Tools (ADT) Plugin. You may be asked to restart Eclipse to complete the installation.

You'll need to have the Android 3.0 (API 11) components installed in order to use ChirpVision SDK.

You should now see two new icons in your Eclipse toolbar and entries in its 'Window' menu to launch the Android SDK Manager and AVD Manager (for device emulators).

Even if you have a real Android device to develop with, it is very useful to have an Android emulator configured. In the newly-installed 'AVD Manager', create a new AVD, and leave simple, default settings for it:



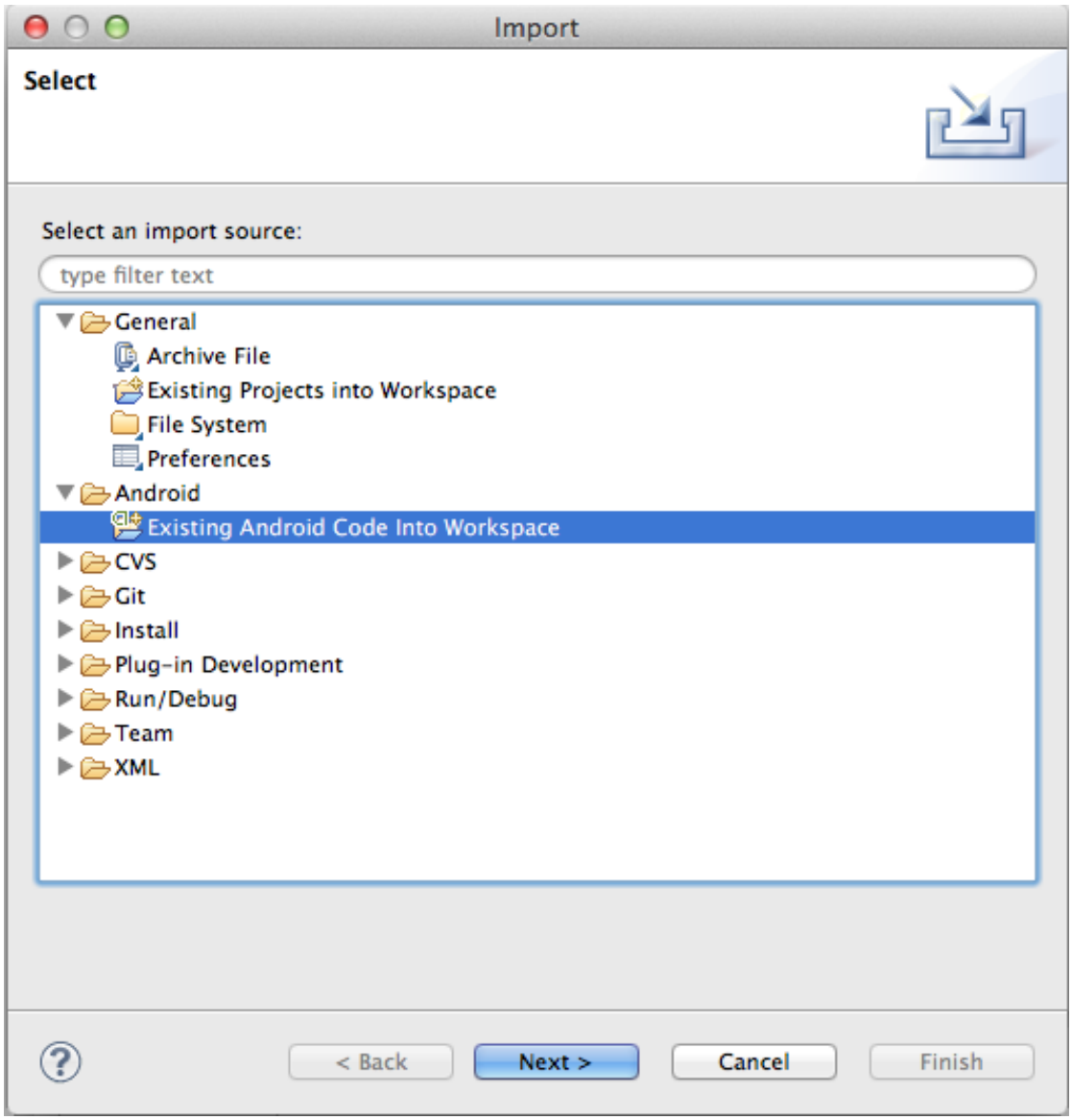
You can now start the emulator either directly from the AVD Manager tool or do it later when you run your first Android project in Eclipse.

2. Reference the SDK in your project

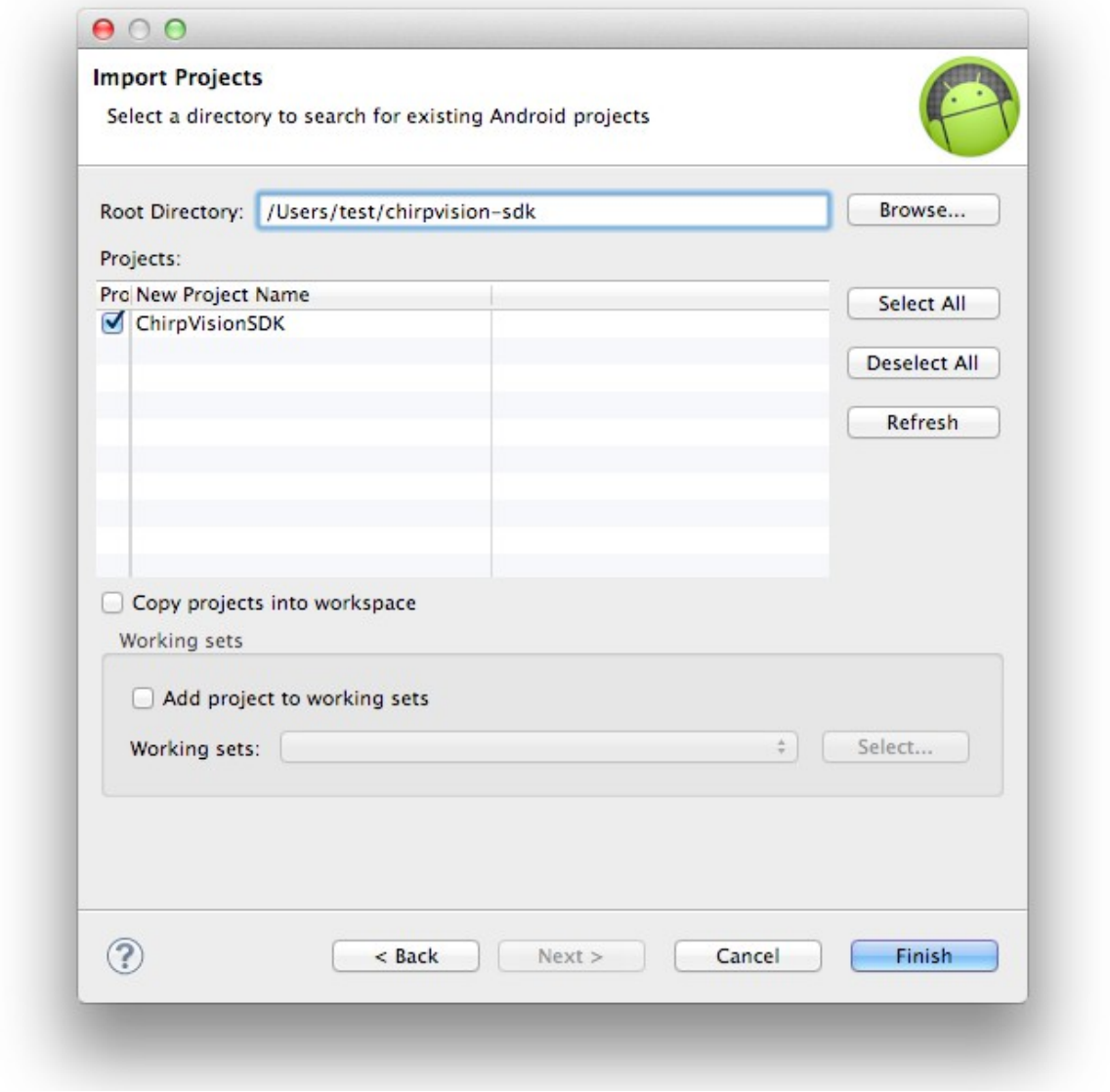
After downloading and extracting the SDK ZIP file. the resulting folder, chirpvision-sdk, contains the core SDK itself. Note the location of this folder.

There is also a sample project at the chirpvision-sample folder. You may import these projects into a clean workspace all at once. In particular, make sure there is no existing chirpvision-sdk project in your workspace. If one exists, it is possibly referencing an older ChirpVision SDK. You should either delete it or change your workspace.

To import the SDK library project and the samples, with the new SDK, go to Eclipse's 'File' > 'Import' menu, and select 'Android' / 'Existing Android Code into Workspace':



Browse to select the root of your SDK folder, chirpvision-sdk. The SDK should appear in the list as 'ChirpVisionSDK':

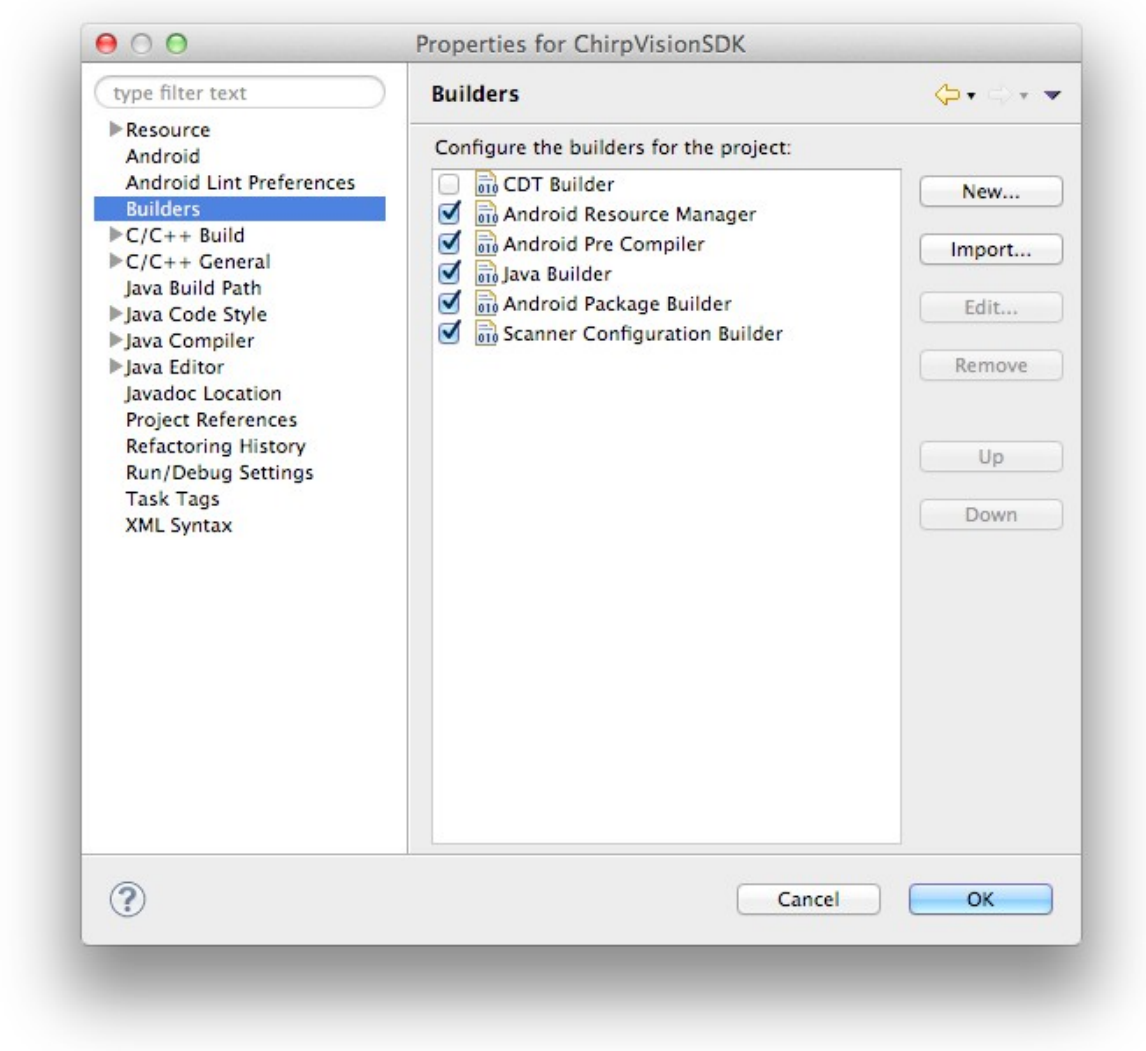


You *must* unselect the 'Copy projects into workspace' option so that each of the sample projects retains a correct reference to the neighboring SDK. However, this means that Eclipse creates a link to the project in the SDK installation, rather than making a copy of it.

(This is useful if you expect to update the SDK in-place in the future and have your workspace's library project update accordingly. However, if you choose to edit the SDK samples in the future, you may wish to subsequently re-import them with a copy so as not to affect the original versions). Click 'Finish'.

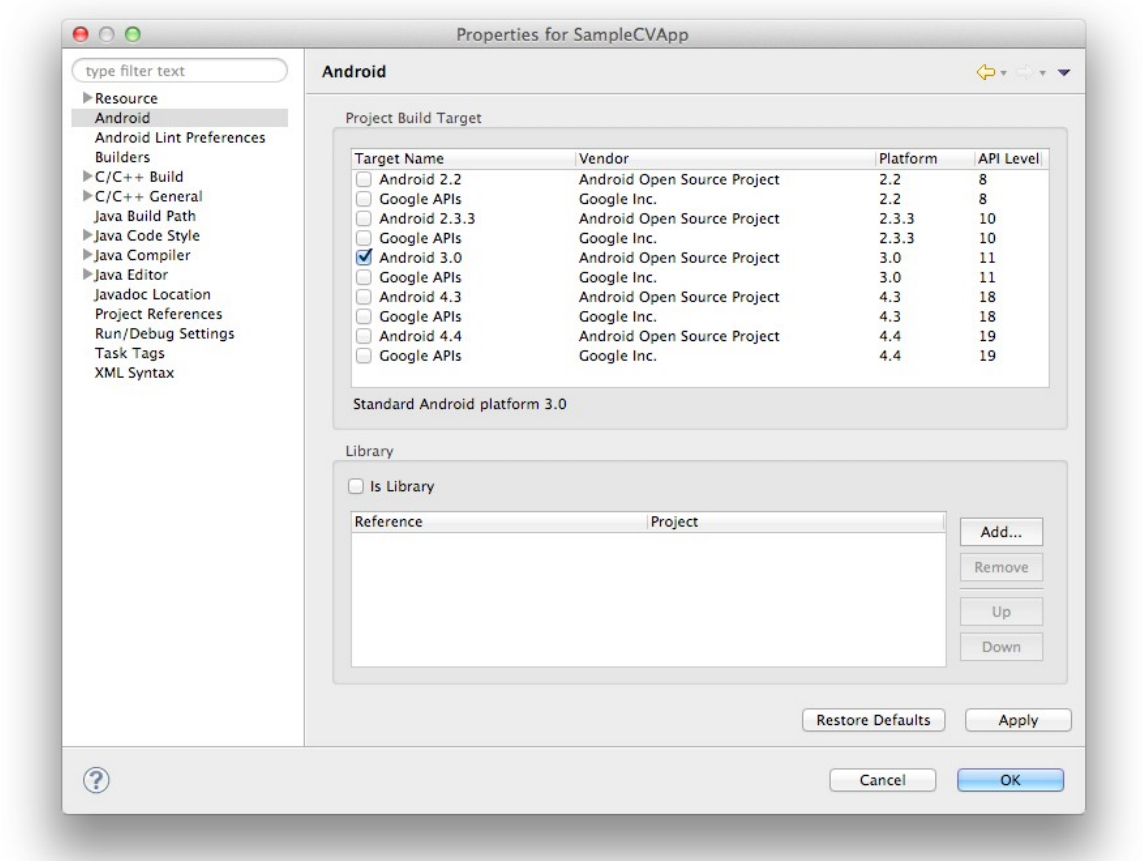
After the project is imported to your workspace, open ChirpVisionSDK project's

properties and go to 'Builders', make sure the CDT Builder is unchecked as follows:



You are now ready to integrate ChirpVisionSDK into your app.

To integrate ChirpVisionSDK into your project, you must first link the project library to your project by selecting your project's property and going to Android:



Click on 'Add' and select ChirpVisionSDK project, then click 'OK' and your project should be now referencing ChirpVisionSDK and ready to build.

3. Modify your AndroidManifest.xml file

In order for the integration to work, the following user-permissions are needed on your application's AndroidManifest.xml:

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

```
    <uses-permission
android:name="android.permission.CHANGE_NETWORK_STATE" />
     <uses-permission
android:name="android.permission.WAKE_LOCK"/>
     <uses-permission android:name="android.permission.VIBRATE"/>
     <uses-permission
android:name="android.permission.READ_PHONE_STATE"/>
     <uses-permission
android:name="android.permission.INTERNET"/>
```

Also, ChirpVision's WiFiListener receiver needs to be registered as follows:

```
<receiver android:name="com.chirpvision.listeners.WiFiListener"
                        android:enabled="true" >
    <intent-filter>
        <action
android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
```

4. Initializing ChirpVision in your Application.java

ChirpVision setup is required at the application initialization / termination process, this can be achieved by invoking ChirpVision.init() and ChirpVision.terminate() methods on your application's onCreate() and onTerminate() methods respectively as shown bellow:

```
import android.app.Application;
import com.chirpvision.ChirpVision;

public class ChirpVisionSampleApp extends Application {

     @Override
     public void onCreate() {
          super.onCreate();

          ChirpVision.init(getBaseContext());
     }

     @Override
     public void onTerminate() {
          ChirpVision.terminate();

          super.onTerminate();
     }
}
```

5. Integrating ChirpVision's Live Stream into your app

ChirpVision's Live streaming feature can be integrated to your app using ChirpVision SDK in a few different ways, depending on your application needs to have more control or less control over the multicast stream, a high level ChirpvisionActivity is provided, as well as a lower level StreamBridge interface to handle the stream functionalities, both approaches are covered in this section.

5.1 Integration using ChirpVisionActivity

The easiest and most straightforward way to integrate the live streaming to your app is to use ChirpvisionActity, which handles most of the live streaming, as well as the DVR functionality for supported multi-core devices and dynamic UI skin loading to cover different UI schemes. To use ChirpvisionActity, you need to declare it on your AndroidManifest.xml file as follows:

```
<activity
android:name="com.chirpvision.activity.ChirpVisionActivity"
          android:screenOrientation="landscape"
android:launchMode="singleInstance"

android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
             android:configChanges="orientation|keyboardHidden">
</activity>
```

And from another activity, call:

```
            Intent cvIntent = new Intent(SampleActivity.this,
ChirpVisionActivity.class);

            startActivity(cvIntent);
```

At this point, your app should be able to stream videos using ChirpVision's stream format.

5.2 Integration using StreamBridge

For those applications that require a little more control over the stream, ChirpVisionSDK provides a StreamBridge interface which provides methods to handle DVR functions as well as initialize, open, reopen and close the stream.

To handle the StreamBridge, a StreamBridgeFactory is also provided, which allows for the creation and handling of a StreamBridge. Before getting hold of a

StreamBridge, its factory must be initialized passing an implementation of AudioVideoCallback interface, which provides methods for handling synchronized audio and video frames. It is strongly recommended that the implementation of an AudioVideoCallback interface takes place on an Activity or Fragment, so that the frames can be set on their respective artifacts. Follows an example of the initialization of a StreamBridgeFactory from an Activity that implements AuioVideoCallback:

```
public class VideoActivity extends Activity implements
AudioVideoCallback {

      @Override
      protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            try {
                  StreamBridgeFactory.init(this);
            } catch (Exception e) {
                  …
            }
                                                                  }
...
}
```

The StreamBridgeFactory should be initialized once per VM and once initialized, a StreamBridge instance can be obtained by calling StreamBridgeFactory.getStreamBridge().

Another important step on the StreamBridge initialization process involves setting the host context, which is done by calling StreamBridge's SetHostContext() method passing down a Bitmap as parameter, as follows:

```
      Bitmap bitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.RGB_565);
      StreamBridgeFactory.getStreamBridge().SetHostContext(this,
bitmap);
```

The AudioVideoCallback implementation will provide methods to handle the audio and video frames received from the StreamBridge. The onDrawFrame() method receives a Bitmap as a parameter which can be set to a SurfaceView as follows:

```
public void onDrawFrame(Bitmap bitmap, int frame_channel, int
channel_locked, int fps, int frameWidth, int frameHeight) {
      try {
            Canvas canvas = mSurfaceHolder.lockCanvas();
            if (canvas != null) {
                Paint paint = new Paint();
                canvas.drawBitmap(bitmap, null,
                        new Rect(0, 0, mSurfaceDrawWidth,
mSurfaceDrawHeight), paint);

                mSurfaceHolder.unlockCanvasAndPost(canvas);

             }
      } catch (Exception e) {/*handle exception*/}
}
```

The onPlayAudio() method receives a byte array containing the audio frame to be played and can be implemented as follows:

```
public void onPlayAudio(byte[] audioBuffer, int length) {
      if (mTrack.getPlayState() != AudioTrack.PLAYSTATE_PLAYING)

          mTrack.play(); //mTrack is an AudioTrack

       if (audioBuffer != null)
           mTrack.write(audioBuffer, 0, length);
}
```

6. Setting up multiple channels - config.plist

ChirpVision SDK is bundled with a default pre-defined configuration file named config.plist, this file should be located at the application's *assets* folder and is responsible for setting up most of the SDK's configuration, including multiple channels configuration, registered authorized WiFi SSID names, social pre-defined share messages/hashtags and dynamic skin load.

If using ChirpvisionActivity to integrate with the SDK, any change made to the config file and rebuilt into the application should be loaded by the Activity and the UI should be updated with the viewable channels that were setup, the ChirpvisionActivity has the limitation of having a maximum of 4 viewable channels loaded.

The configuration file is essentially a XML file and it comes pre-configured on the SDK bundle for one single channel, if more channels are required, you can set them up under the channel_urls tag as shown bellow:

```
<key>channel_urls</key>
<array>
      <string>239.0.0.1:5004</string>
      <string>239.0.0.2:5004</string>
</array>
<key>channel_flags</key>
<array>
      <integer>0</integer>
      <integer>0</integer>
</array>
```

The channel_urls element defines the URL(s) that will be listened to for the single/multiple defined multicast channel(s), the above sample defines 2 video channels listening on both multicast stream addresses 239.0.0.1 and 239.0.0.2 both listening on port 5004. Note that the URL does not include the protocol and that a maximum of 4 viewable channels can be defined.

The channel_flags element defines the configuration for each channel and matches the channel url array by the index, so each index on this array should correspond to the specific channel url. Flags are defined at SDK's ChannelFlags enum and are defined as follows:

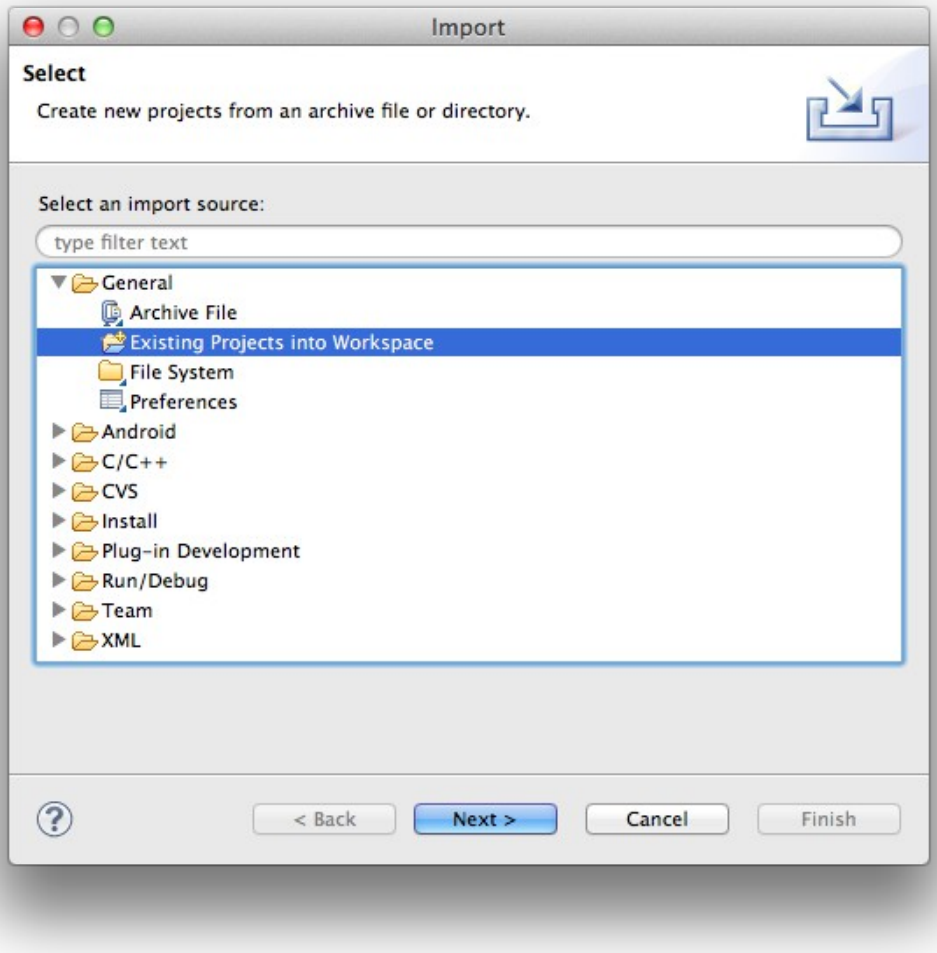
AUDIO_CH(16)	Lower 4 bits used to encode audio source (Channel number) for this video channel.
NOREC(32)	Channel may only be played live (pause, rev, ff, skip are disabled).
HIGHLIGHT(64)	Channel will contain embedded highlight marker commands.
CMD_CH(128)	Channel URL is the 'global' command channel for this site (can only be set on one channel and the channel must be listed after all A/V channels in the channel list).
PING_ADR(256)	Channel URL is the ping data server for this site (can only be set on one channel and the channel must be listed after all A/V channels in the channel list)
UNI_ADR(512)	Channel URL is unicast stream.

PRI_CH(1024)	Channel URL is a priority channel that should be received and decoded, even in single_channel_receiver mode.

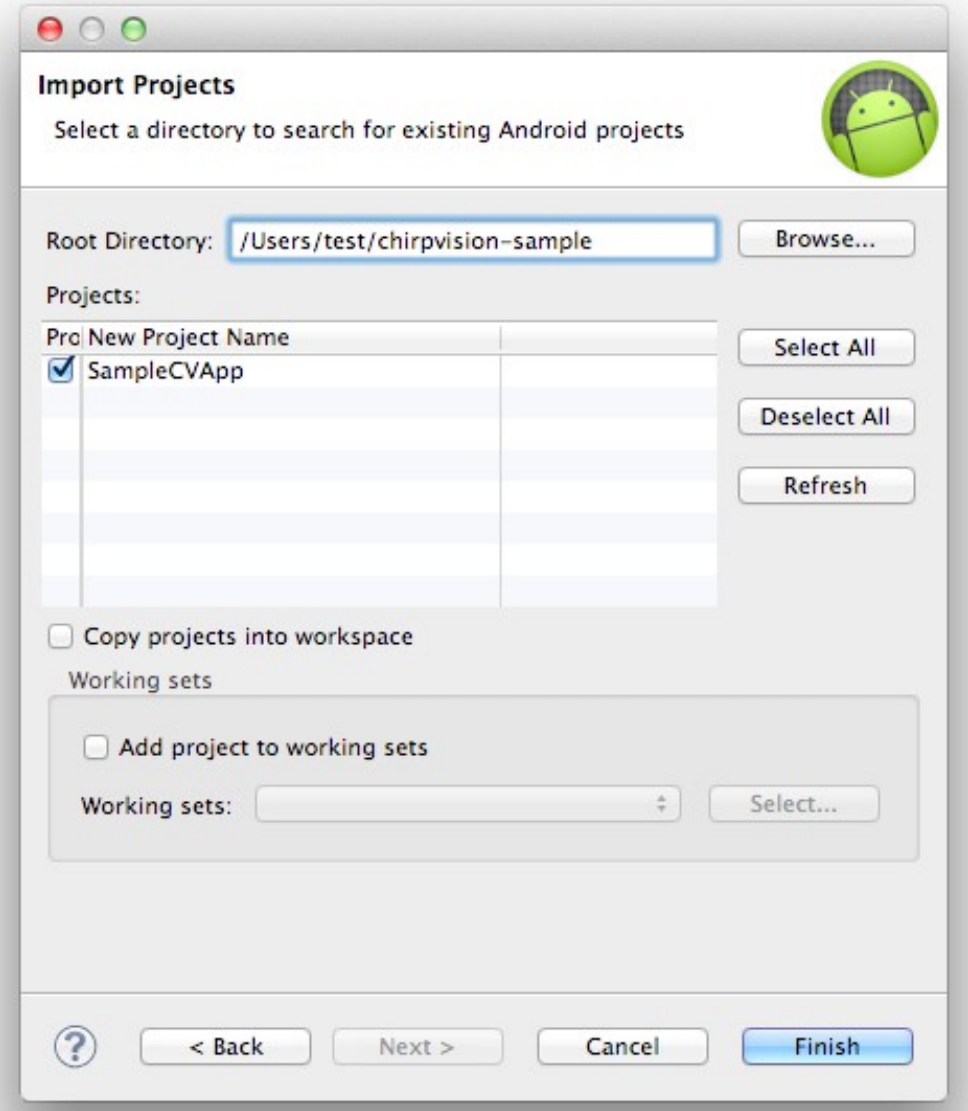
7. Sample Project

The SDK bundle comes with a Sample ChirpVision project, located at chirpvision-sample folder, which serves as a reference for those interested in understanding the integration of the ChirpVision SDK into an Android project.

To import the sample project, with the new SDK, go to Eclipse's 'File' > 'Import' menu, and select 'General' / 'Existing Projects into Workspace':



Next, click on 'Browse…' and select for the chirpvision-sample folder, the 'SampleCV' project should be displayed and selected as follows:

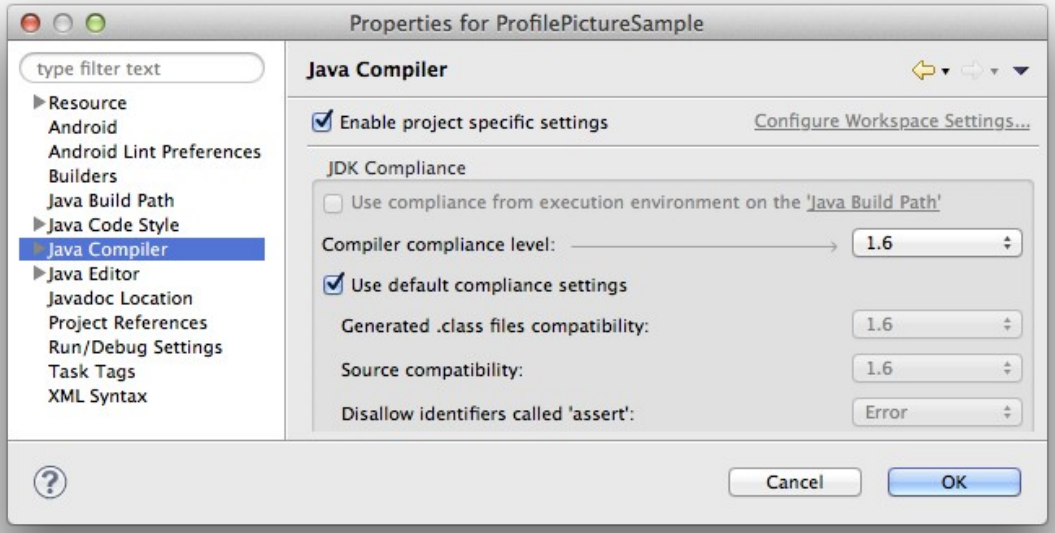


Click 'Finish', the project is now imported into your workspace.

8. Troubleshooting

If the sample project is showing errors, simply use 'Project' > 'Clean' in Eclipse to refresh all the project's states.

It is also possible that your Eclipse compiler level has not matched that required by Android. If you see errors like "Android requires compiler compliance level 5.0 or 6.0.", simply make sure the project (or Eclipse) properties mandate v1.6 in the Java Compiler section.



Finally, if you're having trouble getting Eclipse to import the SDK at all, try importing it as an Android project. To do this, in the initial step when you go to Eclipse's 'File' > 'Import' menu, instead of selecting 'General' / 'Existing Projects into Workspace', expand the 'Android' section and choose 'Existing Android Code into Workspace':

Support

Contact us for technical support at

Email : support@chirpvision.com

Phone: 855.552.4477 , M-F 8-5 Pacific Standard Time