

Android Development with OpenCV

This tutorial has been created to help you use OpenCV library within your Android project.

This guide was written with Windows 7 in mind, though it should work with any other OS supported by OpenCV4Android SDK.

This tutorial assumes you have the following installed and configured:

- JDK
- Android SDK and NDK
- Eclipse IDE
- ADT and CDT plugins for Eclipse

If you need help with anything of the above, you may refer to our [Introduction into Android Development](#) guide.

This tutorial also assumes you have OpenCV4Android SDK already installed on your development machine and OpenCV Manager on your testing device correspondingly. If you need help with any of these, you may consult our [OpenCV4Android SDK](#) tutorial.

If you encounter any error after thoroughly following these steps, feel free to contact us via [OpenCV4Android](#) discussion group or OpenCV [Q&A forum](#). We'll do our best to help you out.

Using OpenCV Library Within Your Android Project

In this section we will explain how to make some existing project to use OpenCV. Starting with 2.4.2 release for Android, *OpenCV Manager* is used to provide apps with the best available version of OpenCV. You can get more information here: [Android OpenCV Manager](#) and in these [slides](#).

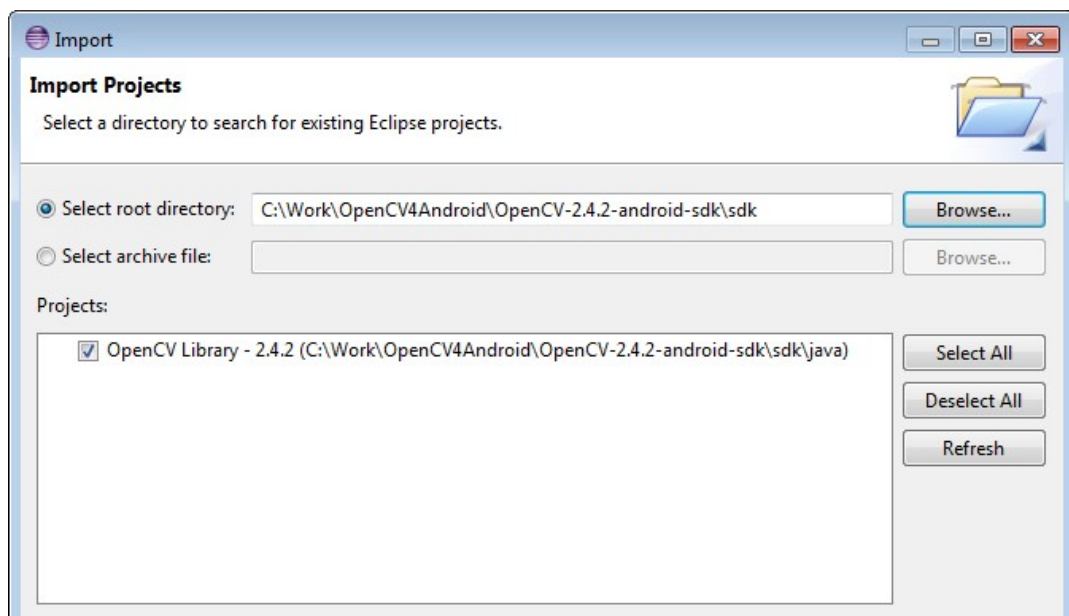
Java

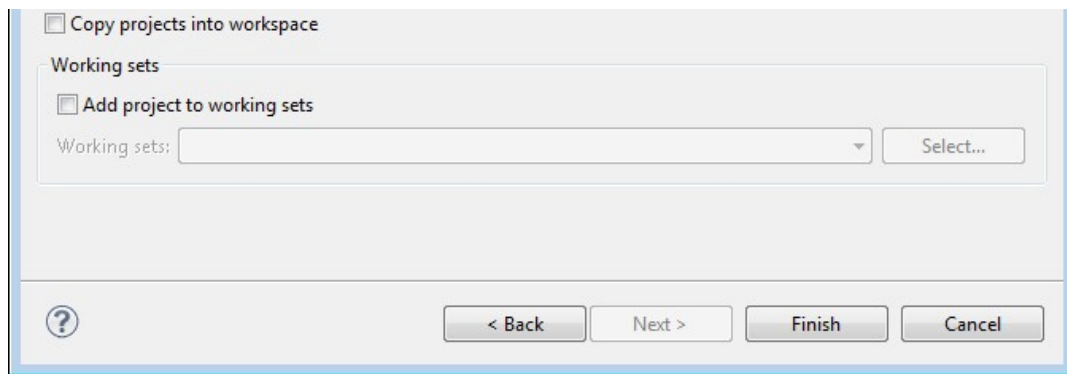
Application Development with Async Initialization

Using async initialization is a **recommended** way for application development. It uses the OpenCV Manager to access OpenCV libraries externally installed in the target system.

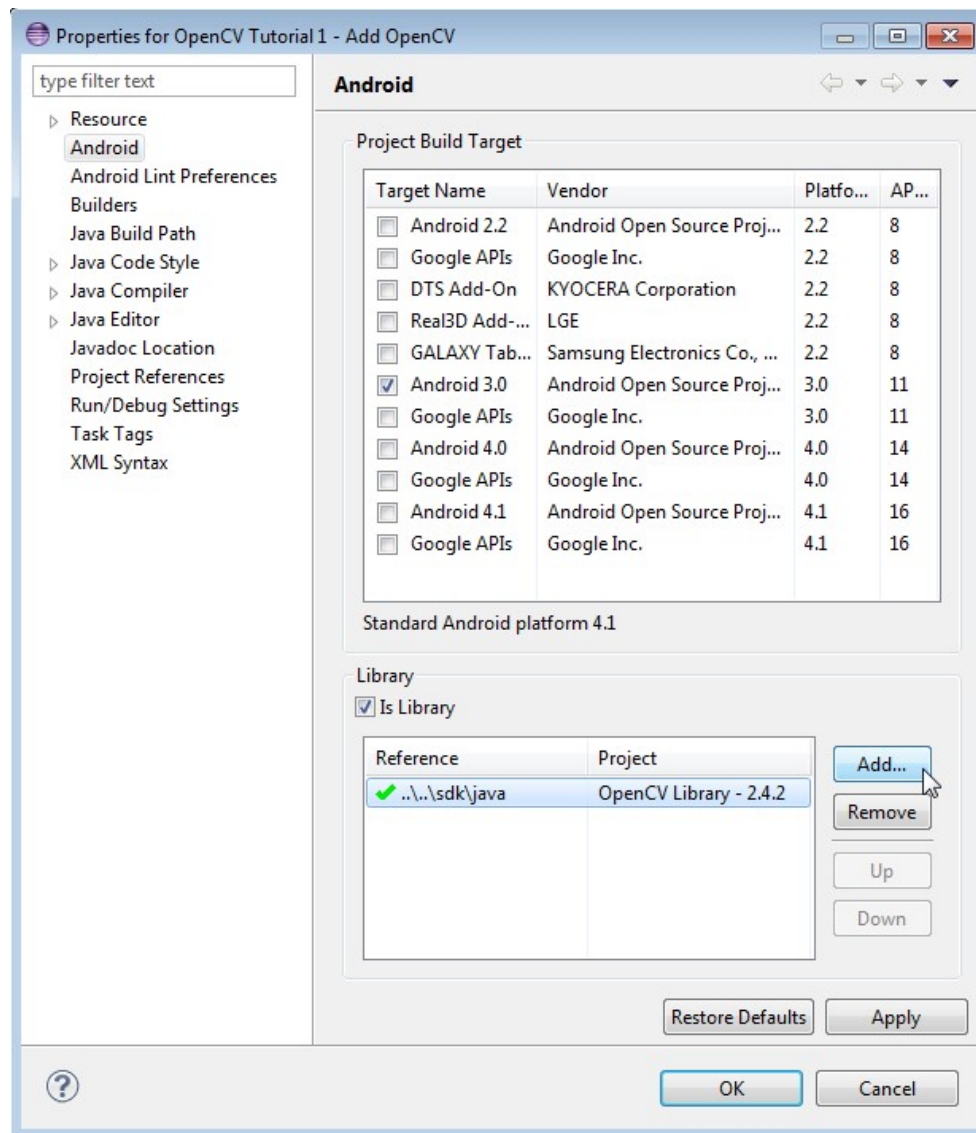
1. Add OpenCV library project to your workspace. Use menu *File -> Import -> Existing project in your workspace*.

Press *Browse* button and locate OpenCV4Android SDK (`OpenCV-2.4.13-android-sdk/sdk`).





2. In application project add a reference to the OpenCV Java SDK in *Project -> Properties -> Android -> Library -> Add select OpenCV Library - 2.4.13.*



In most cases OpenCV Manager may be installed automatically from Google Play. For the case, when Google Play is not available, i.e. emulator, developer board, etc, you can install it manually using adb tool. See [How to select the proper version of OpenCV Manager](#) for details.

There is a very base code snippet implementing the async initialization. It shows basic principles. See the “15-puzzle” OpenCV sample for details.

```

1 public class Sample1Java extends Activity implements CvCameraViewListener {
2
3     private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
4         @Override
5         public void onManagerConnected(int status) {
6

```

```

7         switch (status) {
8             case LoaderCallbackInterface.SUCCESS:
9             {
10                 Log.i(TAG, "OpenCV loaded successfully");
11                 mOpenCvCameraView.enableView();
12             } break;
13             default:
14             {
15                 super.onManagerConnected(status);
16             } break;
17         }
18     };
19
20     @Override
21     public void onResume()
22     {
23         super.onResume();
24         OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_6, this, mLoaderCallback);
25     }
26
27     ...
28 }

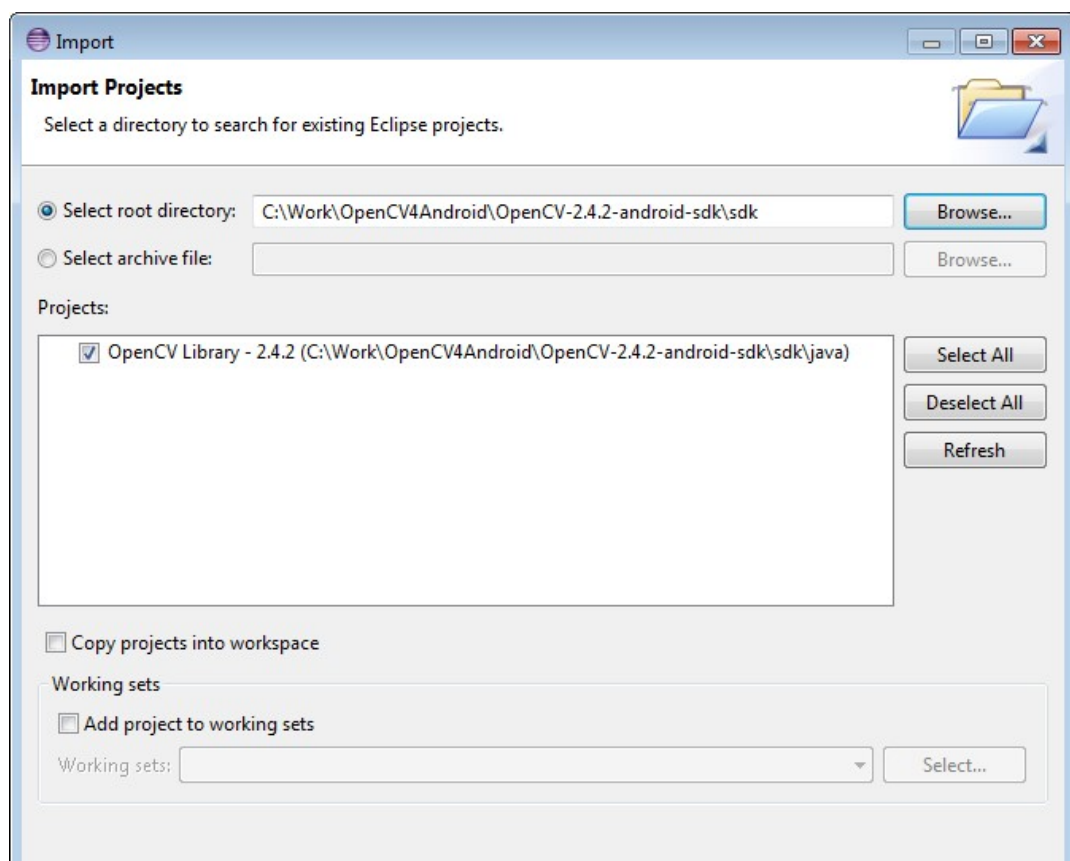
```

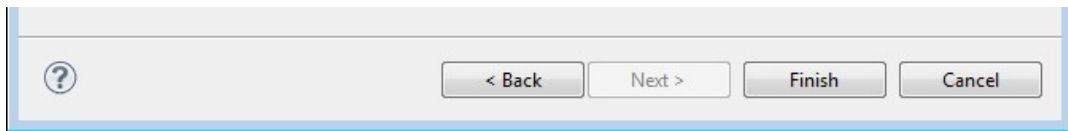
It this case application works with OpenCV Manager in asynchronous fashion. `onManagerConnected` callback will be called in UI thread, when initialization finishes. Please note, that it is not allowed to use OpenCV calls or load OpenCV-dependent native libs before invoking this callback. Load your own native libraries that depend on OpenCV after the successful OpenCV initialization. Default `BaseLoaderCallback` implementation treat application context as `Activity` and calls `Activity.finish()` method to exit in case of initialization failure. To override this behavior you need to override `finish()` method of `BaseLoaderCallback` class and implement your own finalization method.

Application Development with Static Initialization

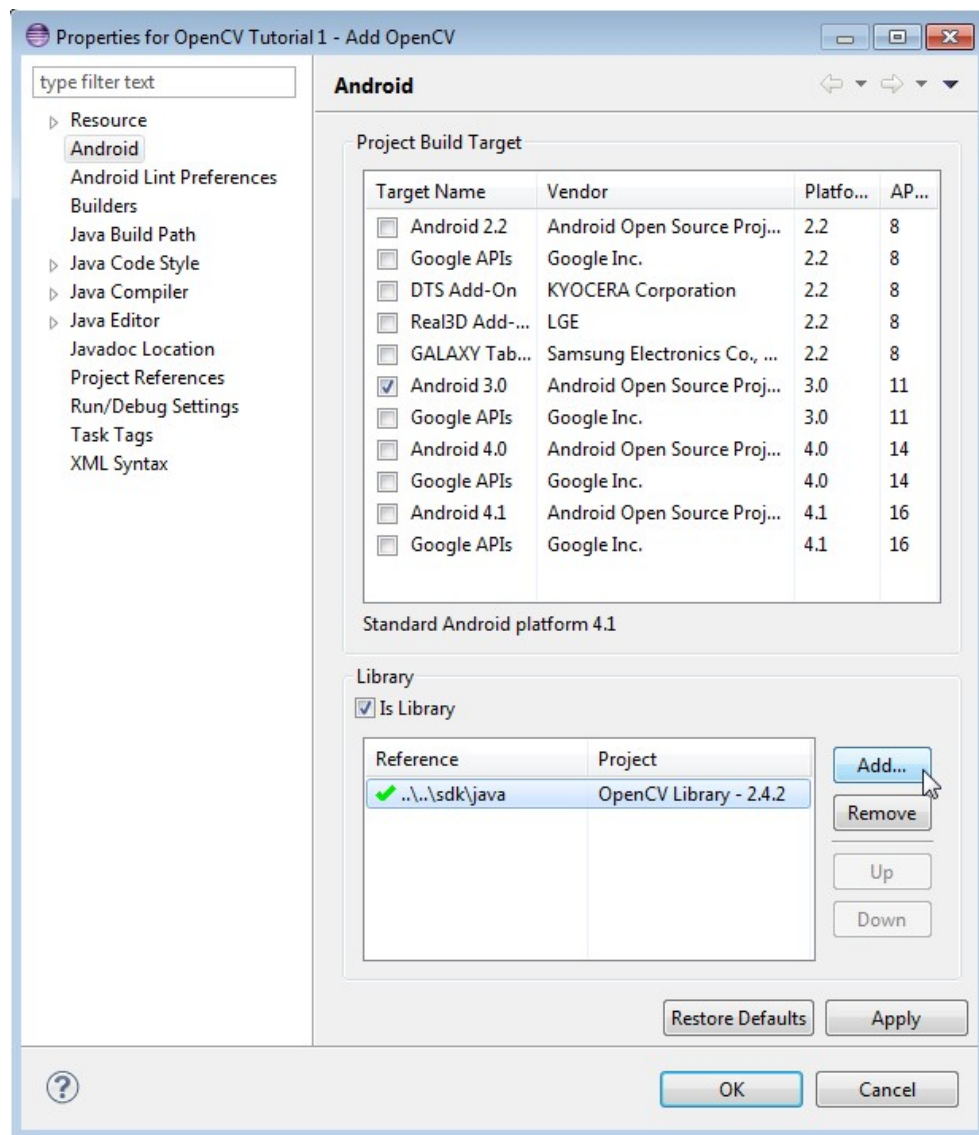
According to this approach all OpenCV binaries are included into your application package. It is designed mostly for development purposes. This approach is deprecated for the production code, release package is recommended to communicate with OpenCV Manager via the async initialization described above.

1. Add the OpenCV library project to your workspace the same way as for the async initialization above. Use menu *File -> Import -> Existing project in your workspace*, press *Browse* button and select OpenCV SDK path (OpenCV-2.4.13-android-sdk/sdk).





- In the application project add a reference to the OpenCV4Android SDK in *Project* -> *Properties* -> *Android* -> *Library* -> *Add* select OpenCV Library - 2.4.13;



- If your application project **doesn't have a JNI part**, just copy the corresponding OpenCV native libs from `<OpenCV-2.4.13-android-sdk>/sdk/native/libs/<target_arch>` to your project directory to folder `libs/<target_arch>`.

In case of the application project **with a JNI part**, instead of manual libraries copying you need to modify your `Android.mk` file: add the following two code lines after the `"include $(CLEAR_VARS)"` and before `"include path_to_OpenCV-2.4.13-android-sdk/sdk/native/jni/OpenCV.mk"`

```
1  OPENCV_CAMERA_MODULES:=on
2  OPENCV_INSTALL_MODULES:=on
```

The result should look like the following:

```
1  include $(CLEAR_VARS)
2
3  # OpenCV
4  OPENCV_CAMERA_MODULES:=on
5  OPENCV_INSTALL_MODULES:=on
6  include ../../sdk/native/jni/OpenCV.mk
```

After that the OpenCV libraries will be copied to your application `libs` folder during the JNI build.

Eclipse will automatically include all the libraries from the `libs` folder to the application package (APK).

- The last step of enabling OpenCV in your application is Java initialization code before calling OpenCV API. It can be done, for example, in the static section of the `Activity` class:

```

1  static {
2      if (!OpenCVLoader.initDebug()) {
3          // Handle initialization error
4      }
5  }

```

If your application includes other OpenCV-dependent native libraries you should load them **after** OpenCV initialization:

```

1  static {
2      if (!OpenCVLoader.initDebug()) {
3          // Handle initialization error
4      } else {
5          System.loadLibrary("my_jni_lib1");
6          System.loadLibrary("my_jni_lib2");
7      }
8  }

```

Native/C++

To build your own Android application, using OpenCV as native part, the following steps should be taken:

- You can use an environment variable to specify the location of OpenCV package or just hardcode absolute or relative path in the `jni/Android.mk` of your projects.
- The file `jni/Android.mk` should be written for the current application using the common rules for this file.

For detailed information see the Android NDK documentation from the Android NDK archive, in the file `<path_where_NDK_is_placed>/docs/ANDROID-MK.html`.

- The following line:

```
include C:\Work\OpenCV4Android\OpenCV-2.4.13-android-sdk\sdk\native\jni\OpenCV.mk
```

Should be inserted into the `jni/Android.mk` file **after** this line:

```
include $(CLEAR_VARS)
```

- Several variables can be used to customize OpenCV stuff, but you **don't need** to use them when your application uses the *async initialization* via the *OpenCV Manager* API (It is not recommended since OpenCV 2.4.13).

Note: These variables should be set **before** the `"include ../OpenCV.mk"` line:

```
OPENCV_INSTALL_MODULES:=on
```

Copies necessary OpenCV dynamic libs to the project `libs` folder in order to include them into the APK.

```
OPENCV_CAMERA_MODULES:=off
```

Skip native OpenCV camera related libs copying to the project `libs` folder.

```
OPENCV_LIB_TYPE:=STATIC
```

Perform static linking with OpenCV. By default dynamic link is used and the project JNI lib depends on

```
libopencv_java.so.
```

5. The file `Application.mk` should exist and should contain lines:

```
APP_STL := gnustdl_static
APP_CPPFLAGS := -frtti -fexceptions
```

Also, the line like this one:

```
APP_ABI := armeabi-v7a
```

Should specify the application target platforms.

In some cases a linkage error (like "In function 'cv::toUtf16(std::basic_string<...>... undefined reference to 'mbstowcs'") happens when building an application JNI library, depending on OpenCV. The following line in the `Application.mk` usually fixes it:

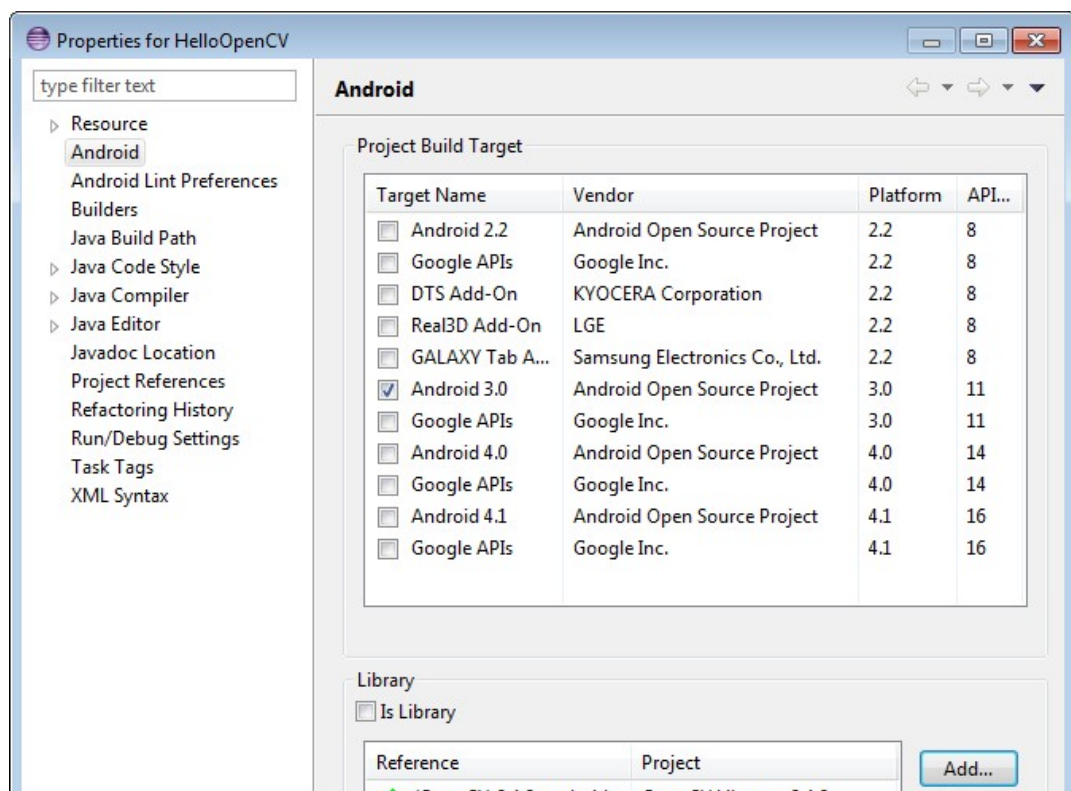
```
APP_PLATFORM := android-9
```

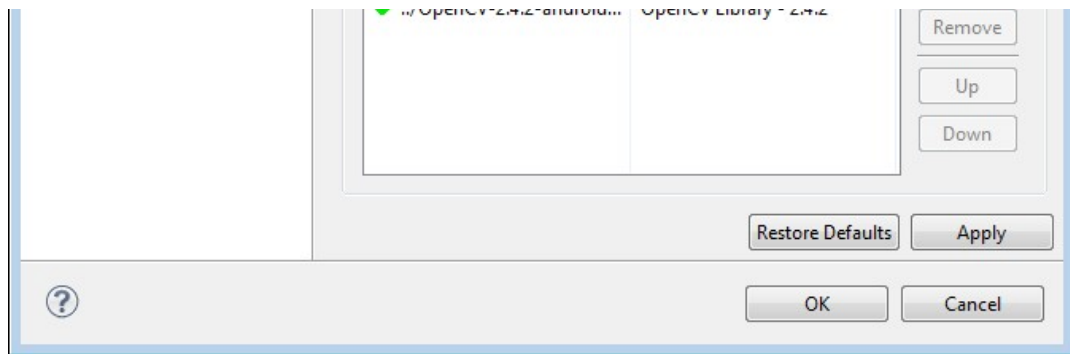
6. Either use [manual](#) `ndk-build` invocation or [setup Eclipse CDT Builder](#) to build native JNI lib before (re)building the Java part and creating an APK.

Hello OpenCV Sample

Here are basic steps to guide you through the process of creating a simple OpenCV-centric application. It will be capable of accessing camera output, processing it and displaying the result.

1. Open Eclipse IDE, create a new clean workspace, create a new Android project *File* › *New* › *Android Project*
2. Set name, target, package and `minSDKVersion` accordingly. The minimal SDK version for build with OpenCV4Android SDK is 11. Minimal device API Level (for application manifest) is 8.
3. Allow Eclipse to create default activity. Lets name the activity `HelloOpenCvActivity`.
4. Choose Blank Activity with full screen layout. Lets name the layout `HelloOpenCvLayout`.
5. Import OpenCV library project to your workspace.
6. Reference OpenCV library within your project properties.





7. Edit your layout file as xml file and pass the following layout there:

```

1  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      xmlns:opencv="http://schemas.android.com/apk/res-auto"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent" >
6
7      <org.opencv.android.JavaCameraView
8          android:layout_width="fill_parent"
9          android:layout_height="fill_parent"
10         android:visibility="gone"
11         android:id="@+id/HelloOpenCvView"
12         opencv:show_fps="true"
13         opencv:camera_id="any" />
14
15  </LinearLayout>

```

8. Add the following permissions to the AndroidManifest.xml file:

```

1  </application>
2
3  <uses-permission android:name="android.permission.CAMERA"/>
4
5  <uses-feature android:name="android.hardware.camera" android:required="false"/>
6  <uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
7  <uses-feature android:name="android.hardware.camera.front" android:required="false"/>
8  <uses-feature android:name="android.hardware.camera.front.autofocus" android:required="false"/>

```

9. Set application theme in AndroidManifest.xml to hide title and system buttons.

```

1  <application
2      android:icon="@drawable/icon"
3      android:label="@string/app_name"
4      android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >

```

10. Add OpenCV library initialization to your activity. Fix errors by adding required imports.

```

1  private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
2      @Override
3      public void onManagerConnected(int status) {
4          switch (status) {
5              case LoaderCallbackInterface.SUCCESS:
6              {
7                  Log.i(TAG, "OpenCV loaded successfully");
8                  mOpenCvCameraView.enableView();
9              } break;
10             default:
11             {
12                 super.onManagerConnected(status);
13             } break;
14         }
15     }
16 };
17
18 @Override

```

```

19 public void onResume()
20 {
21     super.onResume();
22     OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_2_4_6, this, mLoaderCallback);
23 }

```

11. Defines that your activity implements `CvCameraViewListener2` interface and fix activity related errors by defining missed methods. For this activity define `onCreate`, `onDestroy` and `onPause` and implement them according code snippet below. Fix errors by adding required imports.

```

1 private CameraBridgeViewBase mOpenCvCameraView;
2
3 @Override
4 public void onCreate(Bundle savedInstanceState) {
5     Log.i(TAG, "called onCreate");
6     super.onCreate(savedInstanceState);
7     getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
8     setContentView(R.layout.HelloOpenCvLayout);
9     mOpenCvCameraView = (CameraBridgeViewBase) findViewById(R.id.HelloOpenCvView);
10    mOpenCvCameraView.setVisibility(SurfaceView.VISIBLE);
11    mOpenCvCameraView.setCvCameraViewListener(this);
12 }
13
14 @Override
15 public void onPause()
16 {
17     super.onPause();
18     if (mOpenCvCameraView != null)
19         mOpenCvCameraView.disableView();
20 }
21
22 public void onDestroy() {
23     super.onDestroy();
24     if (mOpenCvCameraView != null)
25         mOpenCvCameraView.disableView();
26 }
27
28 public void onCameraViewStarted(int width, int height) {
29 }
30
31 public void onCameraViewStopped() {
32 }
33
34 public Mat onCameraFrame(CvCameraViewFrame inputFrame) {
35     return inputFrame.rgba();
36 }

```

12. Run your application on device or emulator.

Lets discuss some most important steps. Every Android application with UI must implement Activity and View. By the first steps we create blank activity and default view layout. The simplest OpenCV-centric application must implement OpenCV initialization, create its own view to show preview from camera and implements `CvCameraViewListener2` interface to get frames from camera and process it.

First of all we create our application view using xml layout. Our layout consists of the only one full screen component of class `org.opencv.android.JavaCameraView`. This class is implemented inside OpenCV library. It is inherited from `CameraBridgeViewBase`, that extends `SurfaceView` and uses standard Android camera API. Alternatively you can use `org.opencv.android.NativeCameraView` class, that implements the same interface, but uses `VideoCapture` class as camera access back-end. `opencv:show_fps="true"` and `opencv:camera_id="any"` options enable FPS message and allow to use any camera on device. Application tries to use back camera first.

After creating layout we need to implement `Activity` class. OpenCV initialization process has been already discussed above. In this sample we use asynchronous initialization. Implementation of `CvCameraViewListener2` interface allows you to add processing steps after frame grabbing from camera and before its rendering on screen. The most important function is `onCameraFrame`. It is callback function and it is called on retrieving frame from camera. The callback input is object of `CvCameraViewFrame` class that represents frame from camera.

Note: Do not save or use `CvCameraViewFrame` object out of `onCameraFrame` callback. This object does not have its own state and its behavior out of callback is unpredictable!

It has `rgba()` and `gray()` methods that allows to get frame as RGBA and one channel gray scale `Mat` respectively. It expects that `onCameraFrame` function returns RGBA frame that will be drawn on the screen.

Help and Feedback

You did not find what you were looking for?

- Ask a question on the **Q&A forum**.
- If you think something is missing or wrong in the documentation, please file a **bug report**.