Getting Started: Building for Android

TracNav

- **Getting Started**
  - **Preparation**
    - Get the source code
    - Disk Space Requirements
    - Build Preparation
  - **Build for Desktop**
    - Windows
    - Linux
    - MacOS X
    - Python
  - **Build for Mobile**
    - iOS: Apple iPhone, iPad, and iPod Touch
    - Android
    - BlackBerry 10 (BB10)
    - Windows Mobile
    - Windows Phone 8.x and UWP
    - **Symbian...**
  - Build for Other
  - **Next: Using the libraries**
    - Running pjsip Applications
    - Building Application using PJSIP with GNU Tools
    - Video User's Guide (2.0 onwards)
- **See Also**
  - Installing OpenSSL on Windows
  - Using Subversion
  - Visual Studio Build Configurations
  - Using Eclipse with PJSIP
  - S60 3rd Edition devices

Requirements

- Besides the ?Android SDK, you will also need the ?Android NDK.
- Optional if you want to build and and run the sample applications (i.e: pjsua2 and pjsua):
  - ?SWIG
  - ~~Eclipse with ?ADT Plugin~~ (deprecated since 2.6)
  - ?Android Studio
  - Telnet application to interact with PJSUA command line. If you are not familiar with telnet, please find a tutorial suitable for your development platform.

Build Preparation

1. Get the source code from repository, if you haven't already.
2. Set your config_site.h to the following:

```
/* Activate Android specific settings in the 'config_site_sample.h' */
#define PJ_CONFIG_ANDROID 1
#include <pj/config_site_sample.h>
```

Building PJSIP

Just run:

```
$ cd /path/to/your/pjsip/dir
$ export ANDROID_NDK_ROOT=/path_to_android_ndk_dir
$ ./configure-android
$ make dep && make clean && make
```

Notes:

- It will build `armeabi` target, to build for other targets such as `arm64-v8a, armeabi-v7a, x86`, instead of just '`./configure-android`', specify the target arch in `TARGET_ABI` and run it with `--use-ndk-cflags`, for example:

  ```
  TARGET_ABI=armeabi-v7a ./configure-android --use-ndk-cflags
  ```

  Also you should adjust ?Application.mk and ?library packaging path (see also #1803).
- The `./configure-android` is a wrapper that calls the standard `./configure` script with settings suitable for Android target. Standard `./configure` options should be applicable to this script too. Please check `./configure-android --help` for more info.
- Other customizations are similar to what is explained in Building with GNU page.

Video Support

Features

Video on Android will be supported since PJSIP version 2.4. It has the following features:

- native capture
- native OpenGL ES 2.0 renderer (requires Android 2.2 (API level 8) or higher).
- H.264 codec (via OpenH264 library, see below)

Requirements

OpenH264 (recommended)

Provides video codec H.264, alternatively you can use ffmpeg (together with libx264).

1. Follow the instructions in ticket #1947 (or ticket #1758 if you use PJSIP before version 2.6).
2. Copy all library .so files into your Android application project directory, for example:

```
cp /Users/me/openh264/android/*.so /Users/me/pjproject-2.0/pjsip-apps/src/swig/java/android/libs/arme
```

libyuv (recommended)

Provides format conversion and video manipulation, alternatively you can use ffmpeg. If you are using 2.5.5 or newer, libyuv should be built and enabled automatically, see ticket #1937 for more info. If you are using 2.5.1 or older:

1. Follow the instructions in ticket #1776
2. Copy all library .so files into your Android application project directory, for example:

```
cp /Users/me/libyuv-android/*.so /Users/me/pjproject-2.0/pjsip-apps/src/swig/java/android/libs/armea
```

ffmpeg (optional)

Provides format conversion and video manipulation as well as video codecs: H.264 (together with libx264) and H263P/H263-1998.

1. Follow the instructions from the web on how to build ffmpeg for android. We followed the instructions provided ?here and successfully built with Android NDK r10.
2. Copy all library .so files into your Android application project directory, for example:

```
cp /Users/me/src/ffmpeg-2.5/android/arm/lib/*.so /Users/me/pjproject-2.0/pjsip-apps/src/swig/java/android/li
```

Configuring

To enable video, append this into `config_site.h`:

```
#define PJMEDIA_HAS_VIDEO 1
```

Specify third-party video libraries when invoking `./configure-android`, e.g:

```
$ ./configure-android --with-openh264=/Users/me/openh264/android
```

Make sure openh264 is detected by `./configure-android`:

```
...
Using OpenH264 prefix... /Users/me/openh264/android
checking OpenH264 availability... ok
...
```

Note: if you use PJSIP before version 2.6, you need to specify external libyuv via the configure script param `--with-libyuv`, check ticket #1776 for more info.

Adding Video Capture Device to Your Application

Copy the java part of PJSIP Android capture device to the application's source directory:

```
cp pjmedia/src/pjmedia-videodev/android/PjCamera*.java [your_app]/src/org/pjsip/
```

Using Video API

Please check Video User's Guide.

Video capture orientation support

To send video in the proper orientation (i.e. head always up regardless of the device orientation), application needs to do the following:

1. Setup the application to get orientation change notification (by adding `android:configChanges="orientation|keyboardHidden|screenSize"` in the application manifest file and override the callback `onConfigurationChanged()`).
2. Inside the callback, call PJSUA2 API `VidDevManager.setCaptureOrient()` to set the video device to the correct orientation.

For sample usage, please refer to pjsua2 sample app. Ticket #1861 explains this feature in detail.

OpenSSL Support

1. Build OpenSSL (tested with OpenSSL 1.0.2a) for Android. The instruction provided here is specifically for arm64. For other architectures, modify accordingly. Please visit ?this page for reference and some examples. Note: change the NDK path below.

```
.2a

ur_android_ndk_path]/android-ndk-r10d

ls/make-standalone-toolchain.sh --platform=android-21 --toolchain=aarch64-linux-android-4.9 --install-dir=`pwd`
IN_PATH=`pwd`/android-toolchain-arm64/bin
rch64-linux-android
LCHAIN_BASENAME=${TOOLCHAIN_PATH}/${TOOL}
_TOOLCHAIN_BASENAME-gcc
K_TOOLCHAIN_BASENAME-g++
CXX}
_TOOLCHAIN_BASENAME-ld
_TOOLCHAIN_BASENAME-ar
$NDK_TOOLCHAIN_BASENAME-ranlib
NDK_TOOLCHAIN_BASENAME-strip
AGS=
NK=
S=" ${ARCH_FLAGS} -fpic -ffunction-sections -funwind-tables -fstack-protector -fno-strict-aliasing -finline-lim
S=" ${ARCH_FLAGS} -fpic -ffunction-sections -funwind-tables -fstack-protector -fno-strict-aliasing -finline-lim
" ${ARCH_FLAGS} -fpic -ffunction-sections -funwind-tables -fstack-protector -fno-strict-aliasing -finline-limit
=" ${ARCH_LINK} "

droid
```

   Then copy the libraries into lib folder:

   ```
   mkdir lib
   cp lib*.a lib/
   ```

2. Specify OpenSSL location when running `configure-android`, for example (with Bash): (change the openssl path folder)

```
TARGET_ABI=arm64-v8a ./configure-android --use-ndk-cflags --with-ssl=[your_openssl_path]/openssl-1.0
```

And check that OpenSSL is detected by the configure script:

```
...
checking for OpenSSL installations..
checking openssl/ssl.h usability... yes
checking openssl/ssl.h presence... no
aconfigure: WARNING: openssl/ssl.h: accepted by the compiler, rejected by the preprocessor!
aconfigure: WARNING: openssl/ssl.h: proceeding with the compiler's result
checking for openssl/ssl.h... yes
checking for ERR_load_BIO_strings in -lcrypto... yes
checking for SSL_library_init in -lssl... yes
OpenSSL library found, SSL support enabled
...
```

3. Build the libraries:

```
make dep && make
```

If you encounter linking errors, you need to add this in user.mak:

```
export LIBS += "-ldl -lz"
```

Trying our sample application and creating your own

Setting up the target device

To run or debug application (such as the sample applications below), first we need to setup the target device:

- using virtual device: ?http://developer.android.com/tools/devices/index.html
- using real device: ?http://developer.android.com/tools/device.html

Building and running pjsua2 sample application

A sample application using ?pjsua2 API with SWIG Java binding, is located under `pjsip-apps/src/swig/java/android`. It is not built by default, and you need ?SWIG to build it.

Follow these steps to build pjsua2 sample application:

1. Make sure SWIG is in the build environment PATH.
2. Run `make` from directory `$PJDIR/pjsip-apps/src/swig` (note that the Android NDK root should be in the PATH), e.g:

```
$ cd /path/to/your/pjsip/dir
$ cd pjsip-apps/src/swig
$ make
```

This step should produce:

- native library `libpjsua2.so` in `pjsip-apps/src/swig/java/android/app/src/main/jniLibs/armeabi`

        ◊ note: if you are building for other target ABI, you'll need to manually move `libpjsua2.so` to the appropriate target ABI directory, e.g: `jniLibs/armeabi-v7a`, please check ?here for target ABI directory names.
    - pjsua2 Java interface (a lot of `.java` files) in `pjsip-apps/src/swig/java/android/app/src/main/java/org/pjsip/pjsua2`
3. Make sure any library dependencies are copied to `pjsip-apps/src/swig/java/android/app/src/main/jniLibs/armeabi` (or the appropriate target ABI directory), e.g: `libopenh264.so` for video support.
4. Open pjsua2 app project in Android Studio, it is located in `pjsip-apps/src/swig/java/android`.
5. Run it.

Log output

The pjsua2 sample application will write log messages to **LogCat** window.

Creating your own application

For developing Android application, you should use ?pjsua2 API whose Java interface available via SWIG Java binding.

1. First, build `pjproject` libraries as described above.
2. Also build `pjsua2 sample application` as described above, this step is required to generate the pjsua2 Java interface and the native library.
3. Create Android application outside the PJSIP sources for your project.
4. Get pjsua2 Java interface and native library from pjsua2 sample application:
    1. Copy pjsua2 Java interface files from `pjsip-apps/src/swig/java/android/app/src/main/java` to your project's `app/src/main/java` folder, e.g:

```
$ cd $YOUR_PROJECT_DIR/app/src/main/java
$ cp -r $PJSIP_DIR/pjsip-apps/src/swig/java/android/app/src/main/java .

# Cleanup excess pjsua2 application sources.
$ rm -r org/pjsip/pjsua2/app
```

    2. Copy native library `libpjsua2.so` from `pjsip-apps/src/swig/java/android/app/src/main/jniLibs` to your project's `app/src/main/jniLibs` folder:

```
$ cd $YOUR_PROJECT_DIR/app/src/main/jniLibs
$ cp -r $PJSIP_DIR/{pjsip-apps/src/swig/java/android/app/src/main/jniLibs .
```

5. Start writing your application, please check ?pjsua2 docs for reference.

Pjsua sample application with telnet interface

There is also the usual ?pjsua with telnet command line user interface, which is located under `pjsip-apps/src/pjsua/android`. It is not built by default and you need ?SWIG to build it. Application flow and user interface are handled mainly in the native level, so it doesn't use pjsua2 API with Java interface.

Follow these steps to build pjsua:

1. Make sure SWIG is in the build environment PATH.
    - Alternatively, update SWIG path in `$PJDIR/pjsip-apps/src/pjsua/android/jni/Makefile` file.
2. Run `make` from directory `$PJDIR/pjsip-apps/src/pjsua/android/jni` (note that the Android NDK root should be in the PATH), e.g:

```
$ cd /path/to/your/pjsip/dir
$ cd pjsip-apps/src/pjsua/android/jni
$ make
```

3. Open pjsua2 app project in Android Studio, it is located in `pjsip-apps/src/pjsua/android`.
4. Run it.
5. You will see telnet instructions on the device's screen. Telnet to this address to operate the application. See PJSUA CLI Manual for command reference.

Important Issue(s) when Developing Android Apps

Unable to Make or Receive Call (Problem with sending and receiving large (INVITE) requests over TCP)

The issue is documented in #1488. The solution is to try using port other than 5060 in *both* client and server, and/or reducing the SIP message size by following our FAQ here.

Garbage Collector May Crash Your App (Pjsua2 API)

Please check this pjsua2 book page about ?problems with GC.

OpenSLES audio device deadlock upon shutdown

As reported in ?Android NDK forum, when shutting down OpenSLES sound device backend, it may block forever:

```
tf/IBufferQueue.c:57: pthread 0x5fce71c0 (tid 6670) sees object 0x5fcd0080 was locked by pthread 0x5f3a2cb0 (t
```

Currently, the only workaround is to use PJSIP's Android JNI sound device instead (one way to do this is by defining PJMEDIA_AUDIO_DEV_HAS_ANDROID_JNI to 1 and PJMEDIA_AUDIO_DEV_HAS_OPENSL to 0).

Bad audio recording quality on some devices

Reported that audio quality recorded on the microphone is bad and the speed is twice what it should be, it only happens on some devices. It could be fixed by setting audio mode via `AudioManager` to `MODE_IN_COMMUNICATION` in the application, e.g:

```
AudioManager am = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
int original_mode = am.getMode();

/* Set audio mode before using audio device, for example before making/answering a SIP call */
am.setMode(AudioManager.MODE_IN_COMMUNICATION);
...
/* Restore back to the original mode after finished with audio device */
```

```
am.setMode(original_mode);
```

Build failure or configure-android error with "--use-ndk-cflags" using android-ndk-r13 or later

Release notes to r13 stated that NDK_TOOLCHAIN_VERSION now defaults to Clang, resulting error on configure-android which expects gcc:

```
configure-android error: compiler not found, please check environment settings (TARGET_ABI, etc)
```

It can also cause PJSIP to fail to build on android-ndk-r15.

As a workaround, you can specify NDK_TOOLCHAIN_VERSION (to 4.9) to use gcc.

```
NDK_TOOLCHAIN_VERSION=4.9 TARGET_ABI=armeabi-v7a ./configure-android --use-ndk-cflags
```

UnsatisfiedLinkError exception with "cannot locate 'rand'" message

This may occur when running pjsua2 sample app:

```
Exception Ljava/lang/UnsatisfiedLinkError; thrown while initializing Lorg/pjsip/pjsua2/app/MyApp;
...
java.lang.UnsatisfiedLinkError: Cannot load library: reloc_library[1285]:    37 cannot locate 'rand'...
```

As described ?here, this happens if you've built your native components with the android-21 target, but are trying to run it on a device with an older Android version, so re-run configure with APP_PLATFORM set to lower platform version, e.g:

```
APP_PLATFORM=android-19 ./configure-android
```

UnsatisfiedLinkError exception with "Native method not found: org.pjsip.pjsua2.pjsua2JNI.swig_module_init" message

The reason might be:

- The Java interface files and/or the native library wasn't copied to the appropriate folder. Please have a look at here
- You built the lib using newer APP_PLATFORM on an older device. Please have a look at here

A review of Android audio output latency

For a review of Android audio output latency, please have a look at ticket #1841.

Other Android projects

Also have a look at the following PJSIP Android project:

- ?csipsimple project, an Android port of pjsip.