Introduction

Speex is a popular and patent-free speech audio codec for developed by the Xiph.Org Foundation. It is a reasonable choice for speech processing applications; however, Android does not currently support encoding and decoding this format. The Speex library is written in C, so the only possibility to implement it on Android is using JNI.

A much easier way is to use JSpeex, a Speex Java port. The disadvantages of this solution are that it uses an old codec version (1.0.3 in JSpeex while 1.2rc1 is available) and poorer performance, which may be inadequate for applications encoding large files or many files.

Therefore you may be interested in implementing Speex in native form through the Java Native Interface for your Android application. In this article we will go through the process of native compilation of Speex library, implementing a simple wrapper for the encoder and creating a sample Android application capturing sound and encoding it.

Environment configuration

In order to use JNI in Android, we will need Android NDK. It is a set of tools you can use to build Android application components in native code. You can download it from Android Developers site:

http://developer.android.com/sdk/ndk/index.html
After unpacking it, the is an **ndk-build** script in the main directory. This script automatically builds the process by the NDK. It should be run from the root directory of your project to build or rebuild the native library. If you want to learn more about the NDK itself, take a look to the documentation:
http://developer.android.com/sdk/ndk/overview.html

Preparing libraries

In addition to libspeex, you will need libogg to put an encoded stream into the Ogg container. You can download these libraries from the following websites:

http://www.speex.org/downloads/
http://xiph.org/downloads/
When the download has finished, we can build an initial version of our shared library (without a functional wrapper). In order to do so, create a directory called **jni** in your project's root. Then copy the required files as follows (you can skip all Makefiles):

```
/libspeex/*             ? /jni/libspeex/
/include/speex/*        ? /jni/include/speex/
/src/wav_io.h   ? /jni/include/speex
/src/*          ? /jni/libogg/
/include/ogg/*  ? /jni/include/ogg
```

Edit two files manually, since we are not using the standard compilation method with autoconf.

In <project_root></project_root>/jni/include/speex/speex_config_types.h.in replace all:

```
@SIZE16@ with short
@SIZE32@ with int
```

Then remove the .in suffix from the file name.
In <project_root></project_root>/jni/include/ogg/config_types.h.in replace:

```
@INCLUDE_INTTYPES_H@    with 1
```

```
@INCLUDE_STDINT_H@      with 1
@INCLUDE_SYS_TYPES_H@   with 1
@SIZE16@        with short
@USIZE16@       with unsigned short
@SIZE32@        with int
@USIZE32@       with unsigned int
@SIZE64@        with ogg_int64_t
```

Then remove the .in suffix from the file name.

Compilation with NDK

Create your own Makefile for the NDK and include all libspeex and libogg source files and the folder structure. The file should be named Android.mk and placed in the jni directory. Your file should look like this:

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := libspeex
LOCAL_CFLAGS = -DFIXED_POINT -DUSE_KISS_FFT -DEXPORT="" -UHAVE_CONFIG_H
LOCAL_C_INCLUDES := $(LOCAL_PATH)/include

LOCAL_SRC_FILES :=  \
./libspeex/bits.c \
./libspeex/buffer.c \
./libspeex/cb_search.c \
./libspeex/exc_10_16_table.c \
./libspeex/exc_10_32_table.c \
./libspeex/exc_20_32_table.c \
./libspeex/exc_5_256_table.c \
./libspeex/exc_5_64_table.c \
./libspeex/exc_8_128_table.c \
./libspeex/fftwrap.c \
./libspeex/filterbank.c \
./libspeex/filters.c \
./libspeex/gain_table.c \
./libspeex/gain_table_lbr.c \
./libspeex/hexc_10_32_table.c \
./libspeex/hexc_table.c \
./libspeex/high_lsp_tables.c \
./libspeex/jitter.c \
./libspeex/kiss_fft.c \
./libspeex/kiss_fftr.c \
./libspeex/lpc.c \
```

```
./libspeex/lsp.c \
./libspeex/lsp_tables_nb.c \
./libspeex/ltp.c \
./libspeex/mdf.c \
./libspeex/modes.c \
./libspeex/modes_wb.c \
./libspeex/nb_celp.c \
./libspeex/preprocess.c \
./libspeex/quant_lsp.c \
./libspeex/resample.c \
./libspeex/sb_celp.c \
./libspeex/scal.c \
./libspeex/smallft.c \
./libspeex/speex.c \
./libspeex/speex_callbacks.c \
./libspeex/speex_header.c \
./libspeex/stereo.c \
./libspeex/vbr.c \
./libspeex/vq.c \
./libspeex/window.c \
./libogg/bitwise.c \
./libogg/framing.c


include $(BUILD_SHARED_LIBRARY)
```

Now you should be able to build the shared library. Go to the jni folder and call the ndk-build script from there. A proper compilation will give you an output like this:

```
Compile thumb  : speex <= bits.c
Compile thumb  : speex <= buffer.c
Compile thumb  : speex <= cb_search.c
Compile thumb  : speex <= exc_10_16_table.c
Compile thumb  : speex <= exc_10_32_table.c
Compile thumb  : speex <= exc_20_32_table.c
Compile thumb  : speex <= exc_5_256_table.c
Compile thumb  : speex <= exc_5_64_table.c
Compile thumb  : speex <= exc_8_128_table.c
Compile thumb  : speex <= fftwrap.c
Compile thumb  : speex <= filterbank.c
Compile thumb  : speex <= filters.c
Compile thumb  : speex <= gain_table.c
Compile thumb  : speex <= gain_table_lbr.c
Compile thumb  : speex <= hexc_10_32_table.c
Compile thumb  : speex <= hexc_table.c
Compile thumb  : speex <= high_lsp_tables.c
```

```
Compile thumb  : speex <= jitter.c
Compile thumb  : speex <= kiss_fft.c
Compile thumb  : speex <= kiss_fftr.c
Compile thumb  : speex <= lpc.c
Compile thumb  : speex <= lsp.c
Compile thumb  : speex <= lsp_tables_nb.c
Compile thumb  : speex <= ltp.c
Compile thumb  : speex <= mdf.c
Compile thumb  : speex <= modes.c
Compile thumb  : speex <= modes_wb.c
Compile thumb  : speex <= nb_celp.c
Compile thumb  : speex <= preprocess.c
Compile thumb  : speex <= quant_lsp.c
Compile thumb  : speex <= resample.c
Compile thumb  : speex <= sb_celp.c
Compile thumb  : speex <= scal.c
Compile thumb  : speex <= smallft.c
Compile thumb  : speex <= speex.c
Compile thumb  : speex <= speex_callbacks.c
Compile thumb  : speex <= speex_header.c
Compile thumb  : speex <= stereo.c
Compile thumb  : speex <= vbr.c
Compile thumb  : speex <= vq.c
Compile thumb  : speex <= window.c
Compile thumb  : speex <= bitwise.c
Compile thumb  : speex <= framing.c
SharedLibrary  : libspeex.so
Install        : libspeex.so => libs/armeabi/libspeex.so
```

You should now have two new folders in your project's root: libs and obj.

Creating a wrapper

Create a wrapper, which will be the interface for the future Android application to use on our recently compiled Speex library. Create a wrapper.c file in the jni folder and include it in your Android.mk file in LOCAL_SRC_FILES variable:

```
LOCAL_SRC_FILES :=  \
./libspeex/bits.c \
[...]
./libogg/framing.c \
./wrapper.c
```

The wrapper should be adapted to your needs. Our sample implementation provides three functions:

- initEncoder – prepares the encoder to work in a given encoding mode and quality
- destroyEncoder – frees the allocated resources

- encodeFile – encodes source raw file to the target Speex file; the function receives paths to both of these files as parameters

If you don't have any experience with Speex programming, you should first refer to its manual:http://www.speex.org/docs/manual/speex-manual/node7.html
You may also want to get familiar with JNI specific functions:http://docs.oracle.com/javase/1.5.0/docs/guide/jni/spec/functions.html
Since this is a mobile application, the wrapper will encode only one signal channel (mono). Future application will encode signal at a 16 kHz sampling rate (wideband mode) and in default quality (which is 8 in 1-10 range).

The wrapper determines the sampling rate from the Speex mode passed to the init() function. Then it creates a header containing audio parameters, which will be written at the beginning of each encoded file. The encoded file will also contain a comment with the Speex version used. After writing this information, the encoder processes all frames in a loop.

When the wrapper is finished, rebuild the shared library with ndk-build. The output should look like this:

```
Compile thumb  : speex <= wrapper.c
SharedLibrary  : libspeex.so
Install  : libspeex.so => libs/armeabi/libspeex.so
```

Creating the Android Application

Since we have a functional wrapper for our encoder, we can use it in Java code. First, let our application record audio and access the SD card to store raw and encoded files by adding Android permissions: **RECORD_AUDIO** and**WRITE_EXTERNAL_STORAGE**.
The first step is to load our native library. Do it in a static block, so it is performed only once. Then declare the methods:

```
static {
         System.loadLibrary("speex");
}
private native void initEncoder(int mode, int quality);
private native void destroyEncoder();
private native int encodeFile(String sourcePath, String targetPath);
```

Now you can use them as normal. The sample application delivered with this article has a simple one-button layout to start or stop recording. The recorder and encoder are initialized at the beginning. When you press the start button, the application will begins capturing the sound and writes it to the buffered output. When you press the stop button, the sound capturing ends and the captured data is passed on to the encoder. When encoding is finished, you are notified with the encoded file name. When the application finishes, the recorder and encoder are released.

Running the Sample Application

The sample project in this article does not contain any third-party code. You will need to prepare the Speex and Ogg libraries as described in the “Preparing Libraries” paragraph and then compile them with the ndk-build script.

Ref:http://developer.samsung.com/android/technical-docs/Porting-and-Using-the-Speex-Library-in-Android-with-JNI