Mirics Limited.
SDR API Android Integration
Applications
Revision History

Revision	Release Date:	Reason for Change:	Originator
1.0	15th October 2013	Pre-Release 0.0.1	APC
1.1	20th February 2014	Added device removal code	APC
1.2	13th March 2015	Corrections	APC

Contents

1 Introduction

There are a few steps required to access the Mirics based USB module from an Android device. The USB module needs to be opened in Java and then the USB handle can be passed to the Mirics API to have control of the USB module. This document details this procedure.

2 Overview

There are 3 major parts to this process. The initialization code to register a callback, the Java code to call the demod initialization and the demod calls. Within each of these major blocks, there are various setup and make files that will be detailed here.

The prerequisite for this document is an understanding of how Android Java projects are constructed and also the appropriate Android SDK and NDK installs have been done and are configured. Within this document some of these directory locations will be referred to and defined here…

`${NDK_ROOT}` = Local directory where the Android NDK is installed in (e.g. `/home/user/android-ndk-r9`)
`${SYSROOT}` = Local directory where the Android NDK include and libs are for the given platform and architecture (e.g. `${NDK_ROOT}/platforms/android-9/arch-arm`)

3 JNI Setup & Build

3.1 Files

Within the Android Java project a `jni` directory needs to be created. In the `jni` directory go the following files (the files marked with * are example filenames)

`libmir_sdr_api.a` – Mirics static library
`mir_sdr.h` – Mirics include file
`initialization-jni.cpp`* - example of setup code for the JNI communication (see section 4.1)
`demod-jni.cpp`* - example of the code to use the Mirics SDR API (see section 4.2)
`demod-jni.h`* - example of the header file to accompany demod-jni.cpp (see section 4.3)
`Android.mk` – Makefile to build the library to be included within the Android Java project.

3.2 Android.mk makefile

A brief overview of the Android.mk file. The comments explain the section details…

```
# $(call my-dir) returns the local directory which is the jni directory
LOCAL_PATH := $(call my-dir)

# libmir_sdr_api.a – this section creates a version of the Mirics API to be used below
include $(CLEAR_VARS)

LOCAL_MODULE := mir_sdr_api-prebuilt
LOCAL_SRC_FILES := libmir_sdr_api.a
LOCAL_EXPORT_C_INCLUDES := $(call my-dir)

include $(PREBUILT_STATIC_LIBRARY)

include $(CLEAR_VARS)
```

```
# mirics-jni – this section uses the jni C++ source code to build the dynamic library
LOCAL_MODULE := mirics-jni
LOCAL_SRC_FILES := initialisation-jni.cpp demod-jni.cpp
LOCAL_C_INCLUDES := $(call my-dir)
LOCAL_LDLIBS := -L$(SYSROOT)/usr/lib
LOCAL_STATIC_LIBRARIES := mir_sdr_api-prebuilt

include $(BUILD_SHARED_LIBRARY)
```

3.3 JNI Build

`${NDK_ROOT}/ndk-build` – perform this build from the Android Java project directory.

This compiles the JNI C++ source code and the Mirics static library using the Android.mk makefile into a dynamic library that is included in the Android Java project.

3.4 Java Integration

The library built from the `ndk-build` command can be imported and used in the Java code. The example here extends the MainActivity class to open a handle to the USB device and pass it to the JNI code. Further JNI code then accesses the USB device directly.

Please note: there are comments within the code to help with the correct implementation. Please read them carefully.

```
package com.mirics.mirjavatest; // change for your own company/project

import java.util.HashMap;
import java.util.Iterator;
import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.app.AlertDialog;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.usb.UsbDevice;
import android.hardware.usb.UsbDeviceConnection;
import android.hardware.usb.UsbManager;

public class MainActivity extends Activity
{
    static final int CCMD_GET_FD    = 0;
    static final int CCMD_CLOSE_FD  = 1;
    static final int CCMD_NOT_FOUND = 2;
    static final int CCMD_REMOVED   = 3;

    private static final String ACTION_USB_PERMISSION = "com.mirics.mirjavatest.USB_PERMISSION";
    private static UsbManager usbMgr;
    private static PendingIntent mPermissionIntent;
    private static int fd = 0;
    private static UsbDeviceConnection connection;
    private static int connectionMade = 0;

    private Button initButton;

    public static Context mContext;
    public native static void InitialiseDemod();
    public native static void UninitialiseDemod();
    public native static void WaitDemodTerminated();
```

```
static
{
    System.loadLibrary("mirics-jni");
}
// this is the routine that is called to receive permission
// to open the device (in user mode)

private final BroadcastReceiver mRecvr = new BroadcastReceiver()
{
    public void onReceive(Context context, Intent intent)
    {
        String action = intent.getAction();
        if (ACTION_USB_PERMISSION.equals(action))
        {
            synchronized(this)
            {
                UsbDevice device = (UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA_DEVICE);
                If (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED, false))
                {
                    if (device != null)
                    {
                        connection = usbMgr.openDevice(device);
                        connectionMade = 1;
                        fd = connection.getFileDescriptor();
                    }
                }
                else
                {
                    new AlertDialog.Builder(MainActivity.this)
                    .setMessage("Permission nust be granted to use the Mirics USB device.\n
                                 Please restart the application and grant permission.")
                    .setTitle("Permission Not Granted!")
                    .setCancelable(false)
                    .setPositiveButton("OK", new DialogInterface.OnClickListener()
                    {
                        @Override
                        public void onClick(DialogInterface dialog, int which)
                        {
                            finish();
                        }
                    }).show();
                }
            }
        }
    }
};

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

    usbMgr = (UsbManager)getSystemService(Context.USB_SERVICE);
    if (usbMgr == null)
    {
        new AlertDialog.Builder(MainActivity.this)
        .setMessage("Cannot find USB Host Manager.")
        .setTitle("No USB HOST!")
        .setCancelable(false)
        .setPositiveButton("OK", new DialogInterface.OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
                finish();
            }
        }).show();
    }
    mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION_USB_PERMISSION), 0);
    IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
```

```
        registerReceiver(mRecvr, filter);

        setContentView(R.layout.activity_main);
        mContext = MainActivity.this;

        this.findAllViewsById();

        initButton.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                InitialiseDemod();
            }
        });
    }

    private void findAllViewsById()
    {
        initButton = (Button)findViewById(R.id.init_button);
    }

    // this is the java part of the callback that will request permission
    // to open the device (and then close it later)

    public static int rxCcmd(int cmd)
    {
        if (cmd == CCMD_GET_FD)
        {
            connectionMade = 0;
            HashMap<String, UsbDevice> deviceList = usbMgr.getDeviceList();
            Iterator<UsbDevice> deviceIterator = deviceList.values().iterator();
            while (deviceIterator.hasNext())
            {
                UsbDevice device = deviceIterator.next();
                if ((device.getVendorId() == 0x1df7) && (device.getProductId() == 0x2500))
                {
                    usbMgr.requestPermission(device, mPermissionIntent);
                    while (fd == 0)
                    {
                        try
                        {
                            Thread.sleep(1000);
                        }
                        catch (InterruptedException e)
                        {
                            e.printStackTrace();
                        }
                    }
                    connectionMade = 1;
                    return fd;
                }
            }
            return 0;
        }
        else if (cmd == CCMD_CLOSE_FD)
        {
            if (connectionMade == 1)
            {
                connection.close();
                connectionMade = 0;
                fd = 0;
                return 0;
            }
        }
        else if (cmd == CCMD_NOT_FOUND)
        {
            ((Activity)mContext).finish();
        }
        else if (cmd == CMD_REMOVED)
        {
            if (connectionMade == 1)
            {
                connection.close();
                connectionMade = 0;
                fd = 0;
                return 0;
            }
        UninitialiseDemod();
        ((Activity)mContext).finish();
        return 0;
    }

    @Override
    protected void onDestroy()
    {
        super.onDestroy();
        WaitDemodTerminated();
        unregisterReceiver(mRecvr);
    }
}
```

4 JNI Examples

4.1 initialization-jni.cpp

```
#include <jni.h>
#include <stdlib.h>
#include "mir_sdr.h"

JNIEnv *mEnv = NULL;
JavaVM *mVM = NULL;

// Main activity
jclass mActivityInstance;

// method signatures
jmethodID mRxCcmd;

// functions called by JNI

// Library init

extern "C" jint JNI_OnLoad(JavaVM *vm, void *reserved)
{
    JNIEnv *env = NULL;
    jint result = -1;

    if (vm->GetEnv((void **)&env, JNI_VERSION_1_4) != JNI_OK)
    {
        return result;
    }
    mEnv = env;
    mVM = vm;

    jclass cls = mEnv->FindClass("com/mirics/mirjavatest/MainActivity");
    cls = (jclass)(mEnv)->NewGlobalRef(cls);
    mActivityInstance = cls;
    mRxCcmd = mEnv->GetStaticMethodID(cls, "rxCcmd", "(I)I");

    if (!mRxCcmd)
    {
        return result;
    }
    return JNI_VERSION_1_4;
}

// NOTE: this is the callback routine from native to java

extern "C" int callSendCcmdMethod(mir_sdr_JavaReqT cmd)
{
    JNIEnv *tEnv = NULL;
    bool isAttached = false;

    if ((mVM)->GetEnv((void **)&tEnv, JNI_VERSION_1_4) < 0)
    {
        if ((mVM)->AttachCurrentThread(&tEnv, NULL) < 0)
        {
            return 0;
        }
        isAttached = true;
    }
    int res = tEnv->CallStaticIntMethod(mActivityInstance, mRxCcmd, (int)cmd);
    if (isAttached)
    {
        (mVM)->DetachCurrentThread();
    }
    return res;
}
```

4.2 demod-jni.cpp

This example shows a test to catch a surprise removal event.

```
#include <jni.h>
#include <pthread.h>
#include <unistd.h>
#include "mir_sdr.h"
#include "demod-jni.h"

extern "C" int callSendCcmdMethod(mir_sdr_JavaReqT cmd);

void *worker_thread_routine(void *arg);

int killthread = 0;
int threadactive = 0;

pthread_t initThread;

// NOTE: The name of the function needs to match the package name used in the Java code

JNIEXPORT void JNICALL Java_com_mirics_mirjavatest_MainActivity_InitialiseDemod(JNIEnv *env, jobject jobj)
{
    // NOTE: the callback (which happens from within mir_sdr_Init()) must be called
    // from a separate thread for the Broadcast Reciever to be able to run

    pthread_create(&initThread, NULL, worker_thread_routine, NULL);
    threadactive = 1;
}

JNIEXPORT void JNICALL Java_com_mirics_mirjavatest_MainActivity_UninitialiseDemod(JNIEnv *env, jobject
                                                                                  jobj)
{
    killthread = 1;
}

JNIEXPORT void JNICALL Java_com_mirics_mirjavatest_MainActivity_WaitDemodTerminated(JNIEnv *env, jobject
                                                                                    jobj)
{
    while(threadactive) usleep(20 * 1000); // wait 20ms at a time for thread to stop
    killthread = 0;
}

void *worker_thread_routine(void *arg)
{
    mir_sdr_ErrT err;
    int sampsPerPkt;
    int gr = 40;
    double fs = 2.048;
    double rf = 176.640;
    mir_sdr_Bw_MHzT bwType = mir_sdr_BW_1_536;
    mir_sdr_If_kHzT ifType = mir_sdr_IF_Zero;
    int i;
    short *xi;
    short *xq;
    unsigned int firstSampleNum;
    int grChanged;
    int rfChanged;
    int fsChanged;

    err = mir_sdr_SetJavaReqCallback(callSendCcmdMethod);

    err = mir_sdr_Init(gr, fs, rf, bwType, ifType, &sampsPerPkt);

    if (err)
    {
        threadactive = 0;
        return NULL;
    }
    xi = (short *)malloc(sampsPerPkt * sizeof(short));
    xq = (short *)malloc(sampsPerPkt * sizeof(short));

    usleep(1000 * 1000);    // 1000ms wait

    for (i = 0; i < 60000; i++)  // wait ~10s to allow surprise removal
    {
        mir_sdr_ReadPacket(xi, xq, &firstSampleNum, &grChanged, &rfChanged, &fsChanged);
        if (killthread) break;
    }

    mir_sdr_Uninit();
    free(xi);
    free(xq);
    threadactive = 0;
    return NULL;
}
```

4.3 demod-jni.h

```
// again the name of the function here should match as above

extern "C" {
   JNIEXPORT void JNICALL Java_com_mirics_mirjavatest_MainActivity_InitialiseDemod(JNIEnv *, jobject);
}
extern "C" {
   JNIEXPORT void JNICALL Java_com_mirics_mirjavatest_MainActivity_UninitialiseDemod(JNIEnv *, jobject);
}
extern "C" {
   JNIEXPORT void JNICALL Java_com_mirics_mirjavatest_MainActivity_WaitDemodTerminated(JNIEnv *, jobject);
}
```

For more information contact:

Mirics Limited

info@mirics.com
www.mirics.com

Legal Information