Quick Start Guide:

Blue-Link™ Mobile RP1210 Drivers for Android and iOS

Document Overview

This document describes the basic setup and operation for using the Blue-Link™ Mobile RP1210 API on Android and iOS. The document is divided into the following sections:

Android Overview

- Step 1: Android Load Library from Java
- Step 2: Android RP1210 API Setup
- Step 3: Copy INI Files to Data Path
- Step 4: Set Data Path in RP1210 Drivers IOCTL 0x102
- Step 5: Android Discovery IOCTL 0x100
- Step 6: Bluetooth Connect on Android

iOS Overview

- Step 1: iOS RP1210 API Setup
- Step 2: iOS Discovery IOCTL 0x100
- Step 3: Bluetooth Connect on iOS

Firmware Update Overview

- Step 1: Firmware Status IOCTL 0x20004
- Step 2: Check Adapter Firmware Version
- Step 3: Web Service Firmware
- Step 4: Firmware Update IOCTL 0x20005
- Automatic Firmware Update (optional)

RP1210 Logging Overview

Android Overview

Android requires the Java application to load the library, nblr32.so, from Java. Then uncompress and copy the INI files from the package to a data directory accessible by the RP1210 drivers. After that find Bluetooth devices to connect with and make a connection.

NOTE: The Android drivers require that the libc++_shared.so library that is included in the SDK be installed on the device in the same folder as the nblr32.so library.

In practice, the RP1210 program accessing RP1210 should not run on the main thread (GUI thread). Otherwise the responsiveness of the application will suffer and Android may decide to kill the application.

Step 1: Android Load Library from Java

The following call must be made from Java:

```
System.loadLibrary("nblr32");
```

so that the Java_OnLoad() function is called and the Java virtual machine pointer is copied in the nblr32.so RP1210 library: Otherwise, Bluetooth® will not function.

Step 2: Android RP1210 API Setup

The RP1210 API is resolved from the dynamic library using the following code sample. The API functions are typedefed in rp1210_base.h and in Rp1210Test.cpp (or native-lib.cpp in Android Studio).

Two additional API calls, NexiqSetBluetoothAddress and NexiqSetDataPath, are also used to set the Bluetooth MAC address and to set the uncompressed INI files location respectively.

```
RP1210ConnectProc RP1210_ClientConnect = 0;
RP1210DisconnectProc RP1210_ClientDisconnect = 0;
RP1210ReadMessageProc RP1210_ReadMessage = 0;
RP1210SendMessageProc RP1210_SendMessage = 0;
RP1210SendCommandProc RP1210_SendCommand = 0;
RP1210ReadDetailedVersionProc RP1210_ReadDetailedVersion = 0;
RP1210ReadSerialNumberProc RP1210_ReadSerialNumber = 0;
RP1210GetErrorMsgProc RP1210_GetErrorMsg = 0;
RP1210GetLastErrorMsgProc RP1210_GetLastErrorMsg = 0;
RP1210GetHardwareStatusProc RP1210_GetHardwareStatus = 0;
RP1210IoctlProc RP1210_Ioctl = 0;
RP1210ReadVersionProc RP1210_ReadVersion = 0;

void nblr32_SetupAPI(void)
{
   void * g_dll_handle = dlopen("libnblr32.so",RTLD_GLOBAL|RTLD_NOW);

   RP1210_ClientConnect = (RP1210ConnectProc) dlsym(g_dll_handle,"RP1210_ClientConnect");
   RP1210_ClientDisconnect = (RP1210DisconnectProcdlsym(g_dll_handle,"RP1210_ClientDisconnect");
   RP1210_ReadMessage = (RP1210ReadMessageProc)dlsym(g_dll_handle,"RP1210_ReadMessage");
   RP1210_SendMessage = (RP1210SendMessageProc)dlsym(g_dll_handle,"RP1210_SendMessage");
   RP1210_SendCommand = (RP1210SendCommandProc)dlsym(g_dll_handle,"RP1210_SendCommand");
   RP1210_ReadDetailedVersion = (RP1210ReadDetailedVersionProc)
        dlsym(g_dll_handle,"RP1210_ReadDetailedVersion");
   RP1210_ReadSerialNumber = (RP1210ReadSerialNumberProc)
        dlsym(g_dll_handle,"RP1210_ReadSerialNumber");
   RP1210_GetErrorMsg = (RP1210GetErrorMsgProc)dlsym(g_dll_handle,"RP1210_GetErrorMsg");
   RP1210_GetLastErrorMsg = (RP1210GetLastErrorMsgProc)
        dlsym(g_dll_handle,"RP1210_GetLastErrorMsg");
   RP1210_GetHardwareStatus = (RP1210GetHardwareStatusProc)
        dlsym(g_dll_handle,"RP1210_GetHardwareStatus");
```

```
   RP1210_Ioctl = (RP1210IoctlProc) dlsym(g_dll_handle,"RP1210_Ioctl");
   RP1210_ReadVersion = (RP1210ReadVersionProc)dlsym(g_dll_handle,"RP1210_ReadVersion");
}
```

Step 3: Copy INI Files to Data Path

The files nblmap.ini, nblr32_internal.ini, and nblr32.ini should be placed in the assets/Files folder of the Android project. Then the files need to be uncompressed and copied to the data path so that the native C++ libraries can access them with standard file io calls.

Use the following code from a fragment that has access to the Context via getActivity():

```
Context ctx = getActivity();
try {
        String[] files = ctx.getAssets().list("Files");

        for(int i = 0; i < files.length; i++)
        {
                CopyFileToDataPath(files[i]);
        }
} catch (IOException e) {
        e.printStackTrace();
}

private void CopyFileToDataPath(String fname)
{
        // copy compressed file from apk assets folder to private
        // data folder for read/write in C/C++
        InputStream input;
        try {
                String path_fname = "Files/" + fname;
                input = ctx.getAssets().open(path_fname);

                int size = input.available();
                byte[] buffer = new byte[size];
                input.read(buffer);
                input.close();

                FileOutputStream outputStream;
                try{
                        outputStream = ctx.openFileOutput(fname, Context.MODE_PRIVATE);
                        outputStream.write(buffer);
                        outputStream.close();
                }
                catch (Exception e){
                        e.printStackTrace();
                }
        } catch (IOException e)
        {
                e.printStackTrace();
        }
}
```

Step 4: Set Data Path in RP1210 Drivers 0x102

From a Java Context get the data path as:

```
Context ctx = getActivity();
String data_path_str = ctx.getFilesDir().getPath()
```

Pass this Java string to native code via a native call with parameter jstring str_path. Then convert this string to a UTF8 string, fill in the SDATA_PATH struct and then call RP1210_Ioctl()

```
typedef struct _SDATA_PATH
{
    unsigned char *DataPath;
} SDATA_PATH;

...
```

```
    SDATA_PATH s_data_path;

    const char* temp_path = pEnv->GetStringUTFChars(str_path, NULL);
    strcpy(log_path,temp_path);

    strcpy((char*)data_path,temp_path);
    s_data_path.DataPath = data_path;

    RP1210_Ioctl(0,IOCTL_SET_DATA_PATH,&s_data_path,0);
```

After the RP1210 drivers have access to the uncompressed INI files and know their location, the application can make calls using the RP1210 API. Generally the RP1210_Ioctl with IOCTL_DISCOVERY will be used first. Afterwards, the RP1210_ClientConnect call will be made after the MAC address has been set for Bluetooth.

Step 5: Android Discovery IOCTL 0x100

```
#define PACKED __attribute__ ((packed, aligned (1)))

#define RP1210_DISCOVERY_MAX_ENTRY_LENGTH  (256)

struct Rp1210Discovery
{
     unsigned int    scan_interval_sec;
     unsigned int    max_entries;
} PACKED;
typedef struct Rp1210Discovery RP1210_DISCOVERY;

struct RP1210DiscoveryResult
{
    unsigned int    num_entries_found;
    char            buf[0];        // allocated buffer start
} PACKED;
typedef struct RP1210DiscoveryResult RP1210_DISCOVERY_RESULT;


void TestDiscovery(void)
{
        RP1210_DISCOVERY discovery;
        discovery.scan_interval_sec = 5;      // 5 second Wi-Fi scan interval
        discovery.max_entries = 10;           // scan for up to 10 devices

        // allocate space for returned resulting structure
        RP1210_DISCOVERY_RESULT *discovery_result = (RP1210_DISCOVERY_RESULT *)
                                                    malloc(sizeof(unsigned int) +
                            RP1210_DISCOVERY_MAX_ENTRY_LENGTH * discovery.max_entries);

        ret_val = RP1210_Ioctl(0, IOCTL_DISCOVERY, &discovery, discovery_result);

       for (int k = 0; k < discovery_result->num_entries_found; k++)
       {
            char *p = discovery_result->buf;
            p += k * RP1210_DISCOVERY_MAX_ENTRY_LENGTH;
#ifdef ANDROID
            __android_log_print(ANDROID_LOG_INFO,"Example_App","%d. = %s",k, p);
#else
            printf("%d. %s\n", k, p);
#endif
```

```
        }
}
```

Results for Android:

```
        0. BL_1,1,0,00:07:80:1D:33:FD
        1. BL_202,202,0,00:07:80:1D:22:45
```

Results are formatted as "*unique_id,serial_number,status_byte,discovery_string*"

- unique_id = unique identifier for product.
- serial_number = manufacturer assigned serial number for the product.
- status_byte = 0 if available or 1 if busy.
- discovery_string = parameter passed in to protocol connect string to connect to Blue-Link™ Mini.

Step 6: Bluetooth Connect on Android:

The Bluetooth results above will only be for previously paired devices. To pair with a device, go to the Android Bluetooth settings screen and select a discovered Blue-Link™ to pair to it.

```
    short n_ret;
    n_ret = RP1210_ClientConnect(0, 2, "J1708:DiscoveryString=00:07:80:1D:22:45", 32768, 32768, 0);
```

iOS Overview

Unlike Android, iOS does not compress the INI files; therefore, the drivers just need a copy of them. This is handled in the drivers as long as the INI files are located in the package as shown in the example application in the "Supporting Files" folder.

Libraries libnexiq_rp1210.a (iOS RP1210 library) and library libstdc++.6.tbd (c++ standard libary) need to be linked with the application. libnexiq_rp1210.a is built against libstdc++.6.

The application accessing RP1210 library functions must not run on the main thread (GUI thread), otherwise there will be a GUI responsiveness penalty.

The iOS RP1210API is setup following the example code given later on. After that, Bluetooth and Wi-Fi devices to connect with can be discovered and a connection to a Blue-Link™ can be made. In the Bluetooth case, if no DiscoveryString is given, the first available connected Blue-Link™ will be the one connected to.

Step 1: iOS RP1210 API Setup

The RP1210 API is resolved from the statically linked library using the following code sample. The API function prototypes and typedefs are in rp1210_base.hpp.

```
RP1210ConnectProc RP1210_ClientConnect = 0;
RP1210DisconnectProc RP1210_ClientDisconnect = 0;
RP1210ReadMessageProc RP1210_ReadMessage = 0;
RP1210SendMessageProc RP1210_SendMessage = 0;
RP1210SendCommandProc RP1210_SendCommand = 0;
RP1210ReadDetailedVersionProc RP1210_ReadDetailedVersion = 0;
RP1210ReadSerialNumberProc RP1210_ReadSerialNumber = 0;
RP1210GetErrorMsgProc RP1210_GetErrorMsg = 0;
RP1210GetLastErrorMsgProc RP1210_GetLastErrorMsg = 0;
RP1210GetHardwareStatusProc RP1210_GetHardwareStatus = 0;
RP1210IoctlProc RP1210_Ioctl = 0;
RP1210ReadVersionProc RP1210_ReadVersion = 0;

void nblr32_SetupAPI(void)
{
    RP1210_ClientConnect = nblr32_RP1210_ClientConnect;
    RP1210_ClientDisconnect = nblr32_RP1210_ClientDisconnect;
    RP1210_ReadMessage = nblr32_RP1210_ReadMessage;
    RP1210_SendMessage = nblr32_RP1210_SendMessage;
    RP1210_SendCommand = nblr32_RP1210_SendCommand;
    RP1210_ReadDetailedVersion = nblr32_RP1210_ReadDetailedVersion;
    RP1210_ReadSerialNumber = nblr32_RP1210_ReadSerialNumber;
    RP1210_GetErrorMsg = nblr32_RP1210_GetErrorMsg;
    RP1210_GetLastErrorMsg = nblr32_RP1210_GetLastErrorMsg;
    RP1210_GetHardwareStatus = nblr32_RP1210_GetHardwareStatus;
    RP1210_Ioctl = nblr32_RP1210_Ioctl;
    RP1210_ReadVersion = nblr32_RP1210_ReadVersion;
}
```

Step 2: iOS Discovery IOCTL 0x100

Following the example in section "Android Discovery IOCTL 0x100", the following will be returned on an iOS device.

Results for iOS:

```
0. BL_1,1,0, 1
1. BL_202,202,0,202
```

The discovery string parameters is typically serial number of the paired Blue-Link™ device.

Step 3: Bluetooth Connect on iOS:

The Bluetooth results above will only be of Blue-Link™ devices that are currently connected to the iOS device via the iOS Bluetooth Settings screen. This is also where pairing takes place for new Blue-Link™ Mini's that are unknown to the iOS device.

```
short n_ret;
n_ret = RP1210_ClientConnect(0, 2, "J1708:DiscoveryString=202", 32768, 32768, 0);
```

An alternative method for connecting to an Blue-Link™ using Bluetooth on iOS is to not specify the DiscoveryString. This will connect to the first Blue-Link™ connected via Bluetooth in the iOS Bluetooth Settings screen.

```
short n_ret;
n_ret = RP1210_ClientConnect(0, 2, "J1708", 32768, 32768, 0);
```

Firmware Update Overview

In earlier versions of the Android RP1210 drivers, when a firmware update for the adapter was available, the drivers would run the update during the RP1210_ClientConnect call. With the newer Firmware Update IOCTL command, the application is able to control when (or if) to perform a firmware update. In addition, the latest firmware is available for download through a web service, so the application does not necessarily have to be rebuilt to distribute the updated firmware file. If the original automatic update is still desired, this can be accomplished by including a diagnostic INI file in the application assets (See Automatic Firmware Update below).

Note the blmini.bin firmware file needs to be included in the application's assets.

Step 1: Firmware Status IOCTL 0x20004

RP1210_Ioctl ID 0x20004 is used for determining if there is a firmware update available for the adapter. This command checks the version of the firmware on the adapter against the firmware file that is in the folder specified in Nexiq_SetDataPath(). This command does not return any version numbers.

```
nIoctId     = IOCTL_FW_STATUS
pInput      = NULL
pOutput     = Pointer to RP1210_FW_STATUS struct as defined below

struct RP1210FwStatus
{
    unsigned int    status;
    unsigned int    file_size;
    unsigned int    file_completed;
} PACKED;
typedef struct RP1210FwStatus RP1210_FW_STATUS

Where possible status values are:
#define FW_STATUS_QUERY                     (0)
#define FW_STATUS_UPGRADE_IN_PROGRESS       (1)
#define FW_STATUS_UPGRADE_FINISHED          (2)
#define FW_STATUS_DEVICE_IN_SYNC            (3)  // The firmware versions match
#define FW_STATUS_DEVICE_NEWER              (4)  // The firmware is newer on the Miniadapter
#define FW_STATUS_DEVICE_OLDER              (5)  // A firmware update is available
#define FW_STATUS_UPDATE_NOT_SUPPORTED      (6)  // Non-Mini adapters not supported
#define FW_STATUS_UPGRADE_FAILED            (7)
```

Step 2: Check Adapter Firmware Version

The firmware version that is currently on the adapter can be checked by calling RP1210_ReadDetailedVersion:

```
short __declspec (dll export) WINAPI RP1210_ReadDetailedVersion
(
        short nClientID,
        char *fpchAPIVersionInfo,
        char *fpchDLLVersionInfo,
        char *fpchFWVersionInfo
);
```

The returned values are as follows:

```
        fpchAPIVersionInfo = RP1210 API version (currently "3.0")
        fpchDLLVersionInfo = RP1210 driver version on mobile device
        fpchFWVersionInfo = Firmware version on adapter
```

Step 3: Web Service Firmware

A web service is available where the latest released firmware for the adapter can be downloaded:

Production: https://prolinkiq.nexiq.com/WebServices/Device.asmx
Beta: https://prolinkiqbeta.nexiq.com/WebServices/Device.asmx

Operation: *GetFirmwareVersion*
Input: *ProductId*

The *ProductId* can be retrieved from the drivers using RP1210_Ioctl command IOCTL_PRODUCT_ID (0x20006):

```
int product_id = -1;
RP1210_Ioctl(clientId, IOCTL_PRODUCT_ID, NULL, &product_id);
```

GetFirmwareVersion will return a JSON string containing the version of the firmare as well as a link to the firmware file. For example:

```
{"Link":"http://snapondev.edgesuite.net/Cloud/Dongle/blmini/v1_38/blmini.bin","Version":"01.000038"}
```

The firmware file version on the web service can then be checked against the version on the adapter (see RP1210_ReadDetailedVersion above) to determine if a newer firmware is available. The firmware file should be downloaded to the folder specified using RP1210_Ioctl command IOCTL_SET_DATA_PATH.

Step 4: Firmware Update IOCTL 0x20005

RP1210_Ioctl ID 0x20005 will command the drivers to update the firmware on the adapter to the firmware file that is located in the folder specified using RP1210_Ioctl command IOCTL_SET_DATA_PATH. The update is made regardless of firmware version on the device, so this can also be used to force a firmware update if needed. This RP1210_Ioctl command will block until the firmware update has completed, so this call should be run in a separate thread.

```
nIoctId = IOCTL_PERFORM_FW_UPDATE
pInput = NULL
pOutput = NULL
```

During the update process IOCTL_FW_STATUS can be called to check the status of the firmware update (see Firmware Status IOCTL 0x20004 above). It is recommended that the status not be checked less than every 250 ms.

Automatic Firmware Update (optional)

If it is preferred to let the drivers automatically update the firmware during the RP1210_ClientConnect call (this will increase connection time if there is an update), then the BLMDiag.INI file needs to be installed in the folder specified using RP1210_Ioctl command IOCTL_SET_DATA_PATH and should have the following entry:

```
[BinLoad]
VerifyOnConnect=TRUE
```

Furthermore, if VerifyOnConnect is being used, a firmware update will not be performed if the firmware version on the adapter is greater than or equal to version in the firmware file. The following entry can be used to force a firmware update during RP1210_ClientConnect regardless of version:

```
[BinLoad]
Auto=TRUE
```

RP1210 Logging Overview

RP1210 logging can be enabled via the SET LOG LEVEL IOCTL 0x101.

```
typedef struct _SLOG_LEVEL
{
    unsigned long DebugLevel;
    unsigned char *DebugFilePath;
    unsigned long DebugFilePathLength;
    unsigned long DebugMode;
    unsigned long DebugFileSize;
} SLOG_LEVEL;

SLOG_LEVEL slog;

//set up slog
...


short ret_val = RP1210_Ioctl(0, IOCTL_SET_LOG_LEVEL, &slog, NULL);
```

Different logging options can be set up by configuring following the RP1227 specification.

On Android, the logging will be written to the LogCat debug window with the tag “RP1210_LOG.” On iOS the logging will be written to the debug window.

A log file named nblr32.log will be written to the DebugFilePath specified.

On Android, the file can be retrieved with Android Studio and Device File Explorer. The log file for the SDK example is stored in /data/data/**application_name**/files/, where **application_name** is com.nexiq.rp1210test for the Android Studio example and com.nexiq.rp1210example for the Xamarin example.

To retrieve the file from an iOS device with MacOS Catalina or later, conect the device to the Mac via USB and click on the device under Locations in Finder. Then select the disclosure triangle to the left of the iOS app name to show the nblr32.log file. Copy the file to the Mac to view it.