Host / Device Communication through USB Synthésis

Copyright CIO Systèmes Embarqués, Selso LIBERADO 16/12/2020.

# Introduction

We present here all native/Androïd solution that were explored for Android to device communication through USB.

# Understand the USB concepts

All the presented methods are low-level APIs and expose the USB mechanisms with few abstraction.

So it is recommended to read some docs about the USB concept :

* The [official](https://www.usb.org/document-library/usb-20-specification) USB 2.0 specification (not read 😐 ).
* Keil made a good understanding [online doc](https://www.keil.com/pack/doc/mw/USB/html/_u_s_b__transfer__rates.html) with diagramms (maybe the faster link to read)
* So did [Beyond Logic](https://www.beyondlogic.org/usbnutshell/usb1.shtml)
* [Jan Axelson’site give](http://janaxelson.com/usb.htm) a lot resources (and this guy seems to be a USB Guru, author of USB Complete Book).
* The site [USB made simple](http://www.usbmadesimple.co.uk/index.html) presents all the USB basis.

# Developp Host to Device Transfers using the Android API

The Android API in Xamarin is almost the same as one found on official Androïd Java. It is recommend to consult google Android documentation :

* The [Guide](https://developer.android.com/guide/topics/connectivity/usb) will explain you the steps to detect and connect to device.
* The [API refence](https://developer.android.com/reference/android/hardware/usb/package-summary) for android.hardware.usb.

There are some example from [programcreek](https://www.programcreek.com/java-api-examples/?api=android.hardware.usb.UsbEndpoint) that show how to get your endpoints. I advice to not copy/paste them, they don’t iterate on all the interfaces, and don’t allow you to « bypass » the USB API understanding.

In my case it helped me to write a function that iterate over the interface in order to retreive bulk endpoints.

We extended the ISUBManager interface with two Read/Writes function, and implemented them for the [DroidPandaVcom](https://github.com/bizulk/AndroidFilaire/commit/334874e5d15eed7766f79488caab0361837c7fd3) class.

Theses functions were tested and are working.

So we added in the GUI a switch selector to select transfert méthode : dllCom / Android API.

Note that we did not implement all the dllCom protocol, and are still using it for frame formating and decoding.

Integrating this way means handling a big part of the dllCom protocol in the Androïd side, and this means a lot of work and maintenance, in particular we’ll need to redesign our customer library.

# Developp Host to device transfert with linux usbdev API

The advantage of this solution is that there is no need to embbed any  library, the linux Api is exposed natively. And it can use the file descriptor created by the Android API.

We expected it to work as someone reported that it got it [working](https://stackoverrun.com/fr/q/6070199).

Because the packet size where limited with control transfert, this type of tranfer not found when exploring the STM32 interfaces (done on the Androïd side) we used the bulk transfert method that seems more reliable for data transfers.

The usbdev\_fs Api is documented at [readthedocs](https://kernel.readthedocs.io/en/sphinx-samples/usb.html#the-usb-filesystem-usbfs).

There are some more documentation to read :

* [Howto](http://www.rennes.supelec.fr/ren/fi/elec/docs/usb/usb-logiciel.html) in french for linux,
* Some more kernel docs about [ioctls](https://01.org/linuxgraphics/gfx-docs/drm/ioctl/index.html) and related macro,

# Developp Host to device transfert with libusb

Libusb allows developping user-space driver. It is multi-platform, interacts with the linux usbfs API.

Some [has adapted the libusb](https://stackoverrun.com/fr/q/1907272) for the Android requirements : permission must be retreived from the android system and then the fd passed to libusb, but the mainstream repo does not handle it.

So we downloaded the [repository](https://gitlab.com/madresistor/libusb/-/tree/android/libusb) and build it visual studio ( external makefile project). Note that was really easy as visual Studio found by himself the mk files.

I then built a [dllCom device](https://github.com/bizulk/AndroidFilaire/commit/9ee860c9882ef799a4a2d4416154ecd3a7097bc8) for it, adjusting the project settings to include headers, and the prebuilt-library.

I based device connection implementation from the [repo guide](https://gitlab.com/madresistor/libusb/-/blob/android/README), then use the API for transferts basing on this [example](https://github.com/tytouf/libusb-cdc-example/blob/master/cdc_example.c). Note that this example does « not clean » things :

* Endpoint are hardcoded,
* Context is not retrieved,
* Magic number are used whereas the libusb defines constant and enums,
* The control transfers for device configuration is not justified,
* …

This [howto](https://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/) is a little better as it give some explanation, but this it does not show how to retreive the right endpoint.

For the libusb API understanding : we used the [official site documentation](https://libusb.info/) (doxygen).

Infortunately we only managed to write to device, the read transfers always fail with timeout. We cannot blame the device side implementation as it is working with the Android API implementation.

Also note that there is a book « [unboxing android USB](https://www.amazon.fr/Unboxing-Android-USB-approach-examples/dp/1430262087)» that present a libusb example, presenting the Cypress USB to Serial Device, but it actually requires to root the device and apply permissions on the dev/usb filesystem.

# Developp Host to device transfert using a Proxy on TCP

This is not a solution at the same level as others, it is based on the fact that we manage to handle communication using the Android USB API.

As we also manager to open socket in the native library, we developped a proxy server on the Android Side that will handle all USB transactions.

So we run threaded TCP server, that waits for the native client to send requests.

Each client Read/Write is expressed as a request, the Proxy does know nothing about the « device /host » application protocol.

Task Creation in Xamarin is based on the online [Microsoft doc](https://docs.microsoft.com/fr-fr/dotnet/api/system.threading.tasks.task.run?view=xamarinandroid-7.1).

The TCP Server is based on the online doc for the [TCPListener](https://docs.microsoft.com/fr-fr/dotnet/api/system.net.sockets.tcplistener?view=net-5.0)

We used the network stream class for handling data exchange with the [TCPClient](https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.networkstream?view=xamarinandroid-7.1).

We did not develop using [async mechanism](https://codereview.stackexchange.com/questions/24758/tcp-async-socket-server-client-communication), as we’re need some more documentation to do things this way.

Because Network stream handle bytes, and we need some structure interpretation for handling client requests, there was a high risque of overhead.

The common way to serialize seems to be using the [BinaryFormatter with the MemoryStream](https://stackoverflow.com/questions/14727791/sending-custom-structure-over-the-network-serializationexception) class.

There is a [blog note](https://ladeak.wordpress.com/2018/12/28/struct-serialization-with-spans/) that shows several way to thandle serialization with C#.

But we managed to do it with the Swig generator, that does a simple cast between the resquest header and char array.

For the client Native code, we simply code socket common functions, basing on examples from [developpez.com](https://broux.developpez.com/articles/c/sockets/#LII-A-2-b), [geekstuff](https://www.thegeekstuff.com/2011/12/c-socket-programming/),