iWRAP Host Controller Library

The iWRAP library is a generic implementation of parser for the AT-style command interface used by Bluegiga's iWRAP firmware that runs on their classic Bluetooth modules (WT11i, WT12, WT32/WT32i, and WT41). The core of the library is a byte-by-byte parser routine which processes all data coming from iWRAP, performs basic state tracking, and triggers event callbacks for all known events generated by the firmware. It also includes a few routines for encoding and decoding iWRAP MUX frames, which makes it far easier to use the powerful MUX mode that allows simultaneous multiple connection management.

This means the module will generate output like *this*:

```
RING 0 a8:7b:39:c3:ca:99 3 HFP
HFP 0 BSRF 49
HFP 0 NETWORK "Verizon"
NO CARRIER 0 ERROR 113 HCI_ERROR_OETC_USER
```

…and all you have to do is feed the incoming serial data into the parser, which will trigger callbacks you can define (or not) in your code like *this*:

```
iwrap_evt_ring(uint8_t link_id, const iwrap_address_t *ad
dress, uint8_t channel, const char *profile);
iwrap_evt_hfp(uint8_t link_id, const char *type, const ch
```

```
ar *detail);
iwrap_evt_no_carrier(uint8_t link_id, uint16_t error_code
, const char *message);
```

Although there are examples for a few different platforms, the main part of the library is written to be as platform-agnostic as possible using pure ANSI C. There are no host-specific input or output routines like **printf()** or **Serial.write()**, but instead only data processing and callbacks, along with a few helper functions. All you need to do is add a little magic specific to your host to implement the UART hardware interfacing required, and then just write event handlers for whatever parts of iWRAP's functionality you want to harness.

Not all event/response callbacks are currently implemented yet in the `iwrap_parse()` function. These will be added over time, or you can add them yourself if necessary.

=== PLEASE READ THE "IMPORTANT NOTES" SECTION AT THE END OF THIS DOCUMENT, ESPECIALLY THE PART CONCERNING MUX MODE ===

Implementation Basics

It doesn't take a lot of effort to start using iWRAP on a Bluegiga module, which is one of the reasons why they are so awesome. There are three parts to using iWRAP: the hardware (module), the firmware (iWRAP) and the software (host/MCU). Note that iWRAP uses an AT-

style command interface, which means you have the option to use it with serial terminals or just about anything you might want to.

Module Hardware

1. Use iWRAP-capable Bluegiga module (WT11i, WT12, WT32i, or WT41)
2. Connect RXD, TXD, and GND pins to host device; signals are 3.3v UART
3. Connect RTS and CTS pins to host device for hardware flow control ***(OPTIONAL)***
4. See selected module's datasheet for other connection details specific to your application

Pin assignments for certain hosts are included in the host-specific subfolders in this repository (for example, the Arduino README file includes notes about the Arduino Uno, Arduino Pro Mini, Arduino Leonardo, Arduino Mega, Teensy 2.0, and Teensy++ 2.0).

Module Firmware

1. Update iWRAP firmware image to the latest v5.0.2 release build using SerialDFU and Windows PC ***(RECOMMENDED, download from here)***
2. Apply initial config/profile settings via serial terminal (e.g. Realterm or PuTTY)
3. Apply custom config for optimal usage with this library *(see*

note below!)

4. Set config bits with: `SET CONTROL CONFIG 3400 0040 70A1`
5. Enable MUX mode using: `SET CONTROL MUX 1`

Host Software (MCU/PC)

1. Add `iWRAP.c` and `iWRAP.h` to your host project (some platforms use `iWRAP.cpp` instead of `iWRAP.c`)
2. Write UART output function and assign to `iwrap_output()` function pointer
3. Implement UART input routine so all data is sent to `iwrap_parse()` function
4. Create and assign handler functions for desired response/event callbacks
5. Copy pre-written stub callbacks from `iWRAP_stubs.h` ***(OPTIONAL)***
6. Disable functions in `iWRAP.h` which you don't need, to reduce flash usage ***(OPTIONAL)***

You can see a few ready-to-go examples in the repository, at least one of which will probably give you a good starting point to work from.

Important Notes

SET CONTROL CONFIG bits

iWRAP has standard behavior that works pretty well, but there are

some tweaks that you can apply which change this behavior slightly in ways that help improve MCU parsing ability and a few other things. The iWRAP User Guide contains details on this setting, but the recommended 3-word value of `3400 0040 30A1` enables the following behavior:

- CB bit 0 - RSSI shown in partial inquiry results
- CB bit 5 - Bluetooth (MAC) address shown in CONNECT event
- CB bit 7 - PAIR event shown after pairing from remote device
- CB bit 12 - Old pairings are overwritten during pairing if max paircount is reached
- CB bit 13 - CLOCK event shown after RING or CONNECT events
- CB bit 14 - Optimize UART for low latency instead of throughput
- OB1 bit 5 - “OK.” response shown after every command completes
- OB2 bit 10 - RSSI shown in final inquiry results
- OB2 bit 12 - CR/LF not automatically added to data sent with ECHO commands
- OB2 bit 13 - Events are not printed while in DATA mode

MUX mode considerations

Enabling MUX mode in iWRAP will cause all incoming and outgoing data to be wrapped in binary MUX frames (starting with 0xBF), as described in the iWRAP User Guide. This setting persists across resets and will make manual communication through a serial terminal **very**

difficult. MUX mode is fantastic for microcontrollers, but not so much for humans. If you need to disable MUX mode after enabling it, you can use Realterm's "Send" tab and "Send Numbers" button with the following string of hex codes:

```
$BF $FF $00 $11 $53 $45 $54 $20 $43 $4f $4e $54 $52 $4f $
4c $20 $4d $55 $58 $20 $30 $00
```

It is also possible to disable this with PSTool and BCSP+UART or the SPI interface, though this is generally more complicated than using UART.
