WOULD YOU LIKE SOME PI WITH YOUR CONNECT IQ?

This guest post was written by Jim Miller, a Connect IQ developer in Phoenix, AZ.

This sample shows how to interface Connect IQ (CIQ) 3.1 with a Raspberry Pi using Bluetooth Low Energy (BLE). While I have been creating CIQ apps for some time, this is the first time I've used a Raspberry Pi; please be forgiving with things that might have been done in an easier, better, or different way. I have tested so far with a Raspberry Pi Zero W as well as a Raspberry Pi 3b+, running the June/July 2019 version of Raspbian, and while earlier version/older models should work, I'm by no means a Pi expert, so probably can't help too much, but I'll note things you can try in the Setup.

To run the CIQpi app in the CIQ Simulator, you'll need the Nordic nRF52 Development Kit; see the Programmer's Guide in the 3.1 SDK for details. In the 3.1 SDK there are also two samples showing BLE used with a Nordic Thingy52, which is another place to see how BLE works with CIQ.

There are two very distinct parts of this sample:

1. A Connect IQ app (CIQpi) that can run in the CIQ 3.1 simulator or on a device with CIQ 3.1. You can download a zip of the project here; Just unzip it and import it into your workspace if you use Eclipse.
2. A service that runs on the Raspberry Pi. The Raspberry Pi project is available for download here. file. I'll go into how to set up the Pi later in this tutorial

This sample assumes a working understanding of CIQ, and how to run/edit/sideload/etc.

THE CIQ APP - CIQPI

When the CIQpi app is run on the wearable, it will first scan for a UUID advertised by the Pi end, connect to it and interact with it.

CIQpi has two main screens, with the first showing what I call the "base characteristics" - available memory, uptime, etc.

The second screen can be one of two things: data on 4 GPIO pins, or data from a Pimoroni Enviro pHAT (temperature, pressure, etc.). The default is GPIO and can be changed by pressing Menu when on the base characteristic screen.

The up/down keys move between screens the Start button disconnects from the Pi and restarts the scan. On the second screen, Menu allows you to flash the LEDs on the Enviro pHAT screen, or to control the outputs on the GPIO screen. There's also an option for the GPIO demo; see the `demo2*` variables in PiView.mc for the options.

By default, in the demo one input kind of simulates a Garage Door with an open/close contact, where when open, a counter starts to show how long it's been open, and with the Start button you can trigger one of the outputs to turn on for a couple seconds, simulating something like activating a relay to close the garage door. The other output just reflects the state of the other input.

The characteristics used for the base data and the Enviro data are usually just read based on a timer, but only when that screen is visible, so if you're on the GPIO screen and return to the base data, you'll see it "catch up". This data is seen in the app as a string with JSON data (e.g. `{"temp":"93"}`), while the write for one of the base characteristics is a UINT16, and the write to flash the enviro pHAT LEDs is a single byte (any byte value causes the flash).

For the GPIO characteristics, it's always a single byte, read or written (UINT8), and for the input characteristics also have a "notify" so that when a pin changes state the CIQ app is told about it within a second or so based on configuration.

In CIQpi, specific GPIO pins are referenced with names like `Out C1` and `In C2`. If you want to change the pins used, that needs to be done on the Pi - more about that later

SETTING UP THE RASPBERRY PI AND CIQ-BLE

Now we are ready to begin setting up the Raspberry Pi.

SETUP

I started with a blank SD card, started with noobs_3_0_1, installed the full Raspbian, then did the basic setup for things like location, password and Wi-Fi and let it check for updates. Other versions of Raspbian should work, but I'll leave that to others to discover.

The code runs using nodejs, so there are a couple things to check/install. Log in to the Pi and run the following commands:

```
node -v
```
(You'll probably see 8.11.1, but it could be higher)
```
npm -v
```
(You'll probably see "command not found")

You'll need npm, so you'll want to install it. This is where there may be a difference based on your exact model of the Raspberry Pi, but it works for both the Pi Zero W and the Pi 3b+. The difference may be using something like "7l" in places where it starts "6l". Also, you may want to try a version newer than 8.11.1, but that's what I used, as it matched the version of node that was already there.

```
cd ~
wget https://nodejs.org/dist/v8.11.1/node-v8.11.1-linux-
armv6l.tar.xz
tar xvf node-v8.11.1-linux-armv6l.tar.xz
cd node-v8.11.1-linux-armv6l/
sudo cp -R * /usr/local/
```

After this, running the commands again should result in:

```
node -v
```
(You'll probably still see 8.11.1, but it could be higher)
```
npm -v
```
(You'll probably see 5.60, but it could be higher)

INSTALLING THE CIQ BLE SOFTWARE

This will be in the ciq-ble.tar.xz file. The first step is to extract it. When doing so, the ciq-ble directory will be created with everything needed under that.

```
tar xvf ciq-ble.tar
cd ciq-ble
```

A couple of node modules are used in this sample. They should already be in the `node_modules` directory, but here's how to install them and links to some additional info.

PYTHON SHELL

This is used to execute Python scrips from within the nodejs files. To install:

```
npm install python-shell
```

[Link to Python Shell Info]

RPIO

This is used to interface with the GPIO pins from within the nodejs files. To install:

```
npm install rpio
```

[Link to RPIO Info]

PIMORONI ENVIO PHAT

You will need to install the Python library for this if you have one and plan to use it. See this link:

[Installing Libraries for the Pimoroni Enviro pHAT]

RUNCIQ

The sample has a file called runciq. It's a simple shell script that shows how to start things for the various options for starting the same code. The basic format of the command is:

```
sudo node ciq-<which>.js
```

(where <which> is base, gpio, or enviro)

HEADLESS RASPBERRIES

Let's cut the connection the app and the shell so it can run headless. The easiest way I've found to start things up at boot time is to edit the `/etc/rc.local` file and add:

```
cd (to the ciq-ble directory)
./runciq &"
```

In runciq, select the set of characteristics you want to use, and you can direct the output to a file if you want.

CHANGING THE NAME SEEN OVER BLE

In the CIQpi app, the name of the Pi is displayed at the top, and is obtained by way of `device.getName()` in the CIQpi app. If you'd like to change that, modify the `/etc/hostname` file on the Pi. Be sure to change the `/etc/hosts` file to adjust for the change to hostname.

Changing Options on the Pi

In the files with the code for the characteristics, there are a few variables that make it easy to change things.

The first are `logit` and `logItP` that control what's displayed on the console, with `logIt` being used for general read/writes and `logItP` for info on the "loops" used in some. By default, `logIt` is true and `logItP` false.

The second is "pin" and it only applies to the GPIO characteristic. For files that use RPIO, you can just change its value to use a different pin, while for those that use python, you need to change it in the associated .py file.

The third are sec and ms which define the delay in loops. **NOTE**: If you are using a Raspberry Pi Zero, you'll probably want to increase these values for the GPIO Input characteristics, especially if you see a delay on the GPIO Python outputs.

UUIDS

UUIDs are the addressing scheme for BLE services. UUIDs are used to specify what service/characteristic is being used. For this sample, I used this as the general UUID:

```
0f3df50f09c040a582089836c6040bXX
```

where XX is:

- 0x00-0x0f is for services
- 0x10-0x1f is for base characteristics
- 0x20-0x2f is for Enviro pHAT characteristics
- 0x30-0x3f is for GPIO characteristics

In CIQpi, these are defined in the `PiPM.mc` file and must match those used in the various JavaScript files on the Pi, and if you plan to do your own app, you'll want to start with your own base UUID.

Some may ask why all the characteristics are under one service instead of 3, and the answer is CIQ only allows at most 3 BLE profiles, and with 3 different services, that would already be maxed out. Also, it's simpler.

CHARACTERISTICS

XX	What	Properties	Data
		Base	
10	Avail Memory	READ	JSON
11	Uptime	READ	JSON
12	System Load	READ	JSON
13	Write Counter	READ, WRITE, *NOTIFY	UINT16 - Big Endian
		Enviro pHAT	
20	Temperature	READ	JSON
21	Pressure	READ	JSON
22	Light Level	READ	JSON
23	LED Control	WRITE	UINT8 (anything)
		GPIO	
30	python Output	READ, WRITE, *NOTIFY	UINT8 (0 or 1)
31	RPIO Output	READ, WRITE, *NOTIFY	UINT8 (0 or 1)
32	python Input	READ, NOTIFY	UINT8 (0 or 1)
33	RPIO Input	READ, NOTIFY	UINT8 (0 or 1)

**NOTIFY - for these, the Notification is sent as the result or a WRITE, so is very constant. Others notify when there's been a change to a GPIO pin.*

There's 3 different ways the characteristics interface with the Raspberry Pi shown in the sample.

- The Base characteristics rely primarily on calls to the OS except for write which just uses its own data
- The Enviro pHAT characteristics use a vendor provided python library
- The GPIO characteristics use python or a node module called RPIO to access the pins directly

PYTHON VS JAVASCRIPT/RPIO

One odd thing with python-shell, is that the scripts run after it's referenced in the JavaScript file. For example, with the Enviro pHAT characteristics, if the call to run the script was part of the read process, it would get the data from the last time the script was run. For the very first read, there would be no data, the second read would be the data from the first and so on. To get around this, there's an async function in each JavaScript that just keeps the data fresh, so a read can provide timely data. In the sample, the async functions for the Enviro pHAT characteristics run every 30 or 60 second. with this loop, it would be possible to add a notify for when the data changes, but I'm a bit lazy!

A similar loop is used in the GPIO input characteristics, but there the loop runs every 500ms (**The time for the loop will impact CPU usage! Watch out if you have a Pi Zero!)** looking for a change if notify is enabled ("Subscribe" as shown in the Pi console output). For the Python version of GPIO input, there's a second loop for the reason stated above. To keep the data "fresh".

Personally, for GPIO I find the RPIO approach a bit simpler, in that all the logic to deal with a pin is in a single file and not an external python script, and it's likely more efficient. With things like the Enviro pHAT, there's really no option, so python it is!

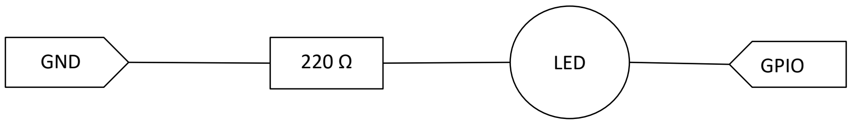
GPIO PIN USAGE

For this sample, two GPIO pins are used for input, and two for output. In each case, one uses a python script to control a pin, and for the other RPIO is used in the JavaScript file.

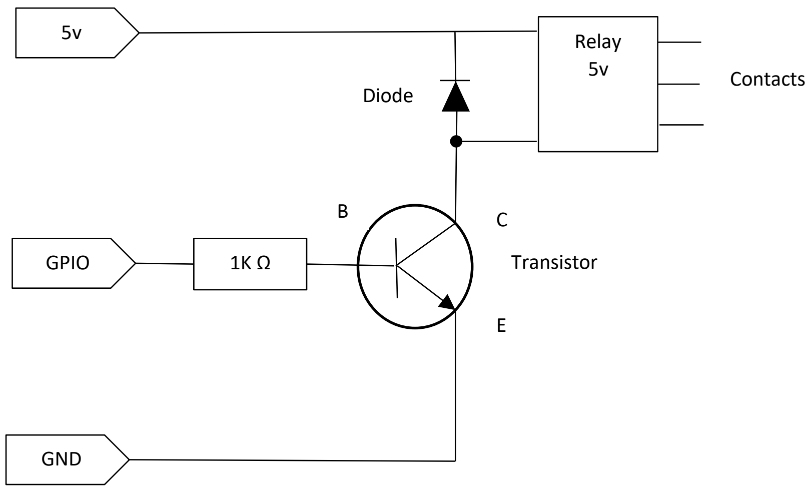
The defaults are:

GPIO Pin	Usage	CIQpi Reference
18	output using python	Out C1
23	output using RPIO	Out C2
19	input using python	In C1
13	input using RPIO	In C2

For output, it's simply a 220-ohm resistor with one end connected to ground, an LED cathode connected to the other end of the resistor, and the LED anode connected to the GPIO pin:

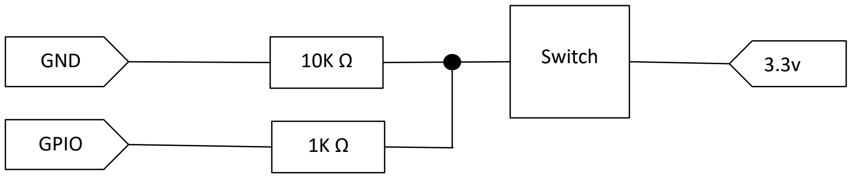


Instead of an LED, here's how to wire a 5v relay. Since the GPIO pin is 3.3v, a transistor is used as a switch, in this case a 2N2222 and the diode is a 1N914:



For input, I opted to use an external pull-down resistor vs the on-chip pull up/pull down. I feel it's a bit safer in a breadboard environment or when something else might have left the chip in output mode.

Here's how they're wired: A 10k ohm resistor is connected to ground, and one side of the switch, and the other side of the switch to 3.3v. Then, 1k ohm resistor is connected to the GPIO pin and the common point where the switch connects to the other resistor.



If you choose to do this differently (using on-chip pull up/down), you need to change the characteristics JavaScript file, and python script (if it uses python).

There's nothing special about why the 4 pins used in the sample were picked. You can use others, and this too means editing the characteristic's JavaScript and possibly it's python script(s). The files have a variable called `pin` and you just need to change that. Note: the BCM naming scheme is used in the code.

EVEN FURTHER

I've added a CIQ widget (piGd – zip of the project here) to demonstrate more of a "real world" example. It was based on the GPIO demo mode but is done as a standalone widget and shows a few things that can be done with UUIDs and characteristics.   For one thing, some things have more specific names in the code, so things for the door sensor are called `door` and not `gpioinc2`. It uses a different base UUID, so you can see how to set your own, which is something you'll want to do for a specific application. In the widget, you can see this in `PiGdPM.mc`. Note: The last two digits in the UUID should be "00", and it is divided in to two 16-digit values.

On the Pi, you need to start things a bit differently than with the CIQpi app, due to the difference in the UUID and characteristics. Use

```
    sudo node ciq-garage.js
```

To set the general UUID on the Pi, that's in `garageuuid.js` (there also `mainuuid.js` used with CIQpi). There, it's only the first 30 digits, and the last two are appended for the specific things.

All the characteristics used are in characteristics/garage directory. They are:

OUTPUT:

- **led** – this one isn't used by the widget but is there for informational purposes. This output is controlled by "door". The output goes high when the door is open, and low when it's closed. The widget isn't involved, so will happen if the widget is running or not
- **relay** – this is how you "push the button" to open/close the door. When it gets a 0, it turns the output off, 1, it turns it on, but if it's 2 or greater, it turns it on for "value" times 100 milliseconds; for example if it gets a 10 the output will be on for 1 second and the widget

doesn't need to be involved in the timing, etc. See "armed" as that's involved. The output won't change unless the "armed" input is high.

- **pulse** – This allows something - say a led or maybe a buzzer - to turn on and off several times. Two bytes are passed to it, with the first byte being the number of cycles, and the second byte being how long the output will be on and then off, which again, is multiplied by 100 to get milliseconds; so "5,3" will cycle 5 times, with the output on for 300ms then off for 300ms.

INPUT:

- **armed** – this is used by "relay" to see if the "push button" is armed or disarmed and would be an easy way to lock out the widget. Its state is available to the widget. Tie the associated GPIO pin high to keep it always armed.
- **door** – The sensor on the garage door. As stated before, it also controls the led output to show the state of the door.

Here's how things are used in the widget. When the widget starts, it scans and connects to the Pi (if in range). It then reads the state of the two inputs and turns on notify for both, so when either changes, it's reflected in the widget.

On the screen you'll see the state of the door, and if armed, a message that says, "Start to open/close". If not armed, "Widget Control Disabled".

If armed and you press start, a write is done to the relay, specifying the time to keep the relay closed, and a write is done to pulse to control that output.

JUST THE BEGINNING

This is just a basic sample of how a CIQ app can interact with a Raspberry Pi, and for a real app, other things can be done. For example, for an actual app, you may also want to investigate some form of security. A different pHAT/Hat could be used, or more different IO with the GPIO pins – more pins, analog data, or more complex characteristics that handle multiple pins, etc.

About the Author *Jim Miller is a Connect IQ developer in Arizona. "In early 2015, I had a Forerunner 15, and liked the GPS and step tracking. Then the original vívoactive was announced, and I pre-ordered it, and downloaded the CIQ 1.0.0 SDK the same week!" You can see his apps on the app store and find him on Instagram, his Connect IQ Facebook Page, or on the Connect IQ forums.*