E20 Kit2 Demo

This demonstration kit showcases the following products:

- SNAP Connect E20 (using DHCP on the Ethernet Port and the SNAP radio)
- SN173 Prototyping board with SM220 module

The kit (or assembled parts) is upgradeable to the demonstration application. Simply power up the E20 and load the software onto the E20 in the snap user directory. You must edit the SN173_Demo_Server.py file to change the SN173_Addr to your SN173 MAC address. Execute the script as sudo.

```
sudo python SN173_Demo_Server.py
```

Download and install Portal (found elsewhere). Next copy the SN173DemoLedBtns.py to your Portal/snappyImages directory. Connect the SN173 to your PC via the mini-USB port. Now you can connect Portal to the SN173 as a bridge node and download the script (SN173DemoLedBtns.py). Use a PC or mobile device to connect to the E20's IP address in your browser (Note you will need to be on the same network):

To find your E20's IP address consult the E20 User's Guide for login information and issue the command:

```
Ifconfig
```

to locate the IP address assigned to your Ethernet Port.

Open a web browser, and point to the E20's URL: http://[E20_IP_Addr]

The web page will display a picture SN173 board. You can click the buttons in the web page to enable the LEDs 1-4 or you can click the buttons on the SN173. The two interfaces remain in sync because the SN173 sends a message on each button press (physical or virtual) that the gateway hears and updates the status of the LEDs. The SN173 also sends a message every 10 seconds to notify the gateway of the LED status.

Exploring the Demo

Full source code for this example is available on Github here: https://github.com/synapse-wireless/demo-kits

The Synapse Portal IDE will allow complete embedded module development, as well as wireless sniffer capability – download latest version here: https://forums.synapse-wireless.com/showthread.php?t=9

The web application is a basic python program built with high-performance libraries, Tornado and SNAP Connect. The javascript/html is kept deliberately simple for ease of understanding, although it showcases a low-latency websockets technique. This can be easily extended to REST interfaces, and other web/backend approaches to fit application requirements.