Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

USER GUIDE

SNAP Connect E20

SNAP Enabled Gateway

Version 1.1 for

Firmware Versions 1.X and Higher

Doc# 116-031520-002-G000

6723 Odyssey Drive // Huntsville, AL 35806 // (877) 982-7888 // Synapse-Wireless.com

Disclaimers

Information contained in this Manual is provided in connection with Synapse products and services and is intended solely to assist its customers. Synapse reserves the right to make changes at any time and without notice. Synapse assumes no liability whatsoever for the contents of this Manual or the redistribution as permitted by the foregoing Limited License. The terms and conditions governing the sale or use of Synapse products is expressly contained in the Synapse's Terms and Condition for the sale of those respective products.

Synapse retains the right to make changes to any product specification at any time without notice or liability to prior users, contributors, or recipients of redistributed versions of this Manual. Errata should be checked on any product referenced.

Synapse and the Synapse logo are registered trademarks of Synapse. All other trademarks are the property of their owners. For further information on any Synapse product or service, contact us at:

Synapse Wireless, Inc.
6723 Odyssey Drive
Huntsville, Alabama 35806
256-852-7888
877-982-7888
256-924-7398 (fax)

www.synapse-wireless.com

License governing any code samples presented in this Manual

Table of Contents

1. Overview

Thanks for buying a Synapse Wireless E20 gateway device! This small-package computer bridges the gap between your SNAP-powered mesh network of sensors and controllers, and the rest of the world, through the Internet.

The SNAP Connect E20 combines a Synapse SM220 RF module and an embedded Linux-based computer to provide connectivity (Ethernet, cellular, Wi-Fi, serial) and site-aggregation capabilities to a diverse array of SNAP-powered IoT networks across industrial temperature ranges.

TCP/IP connections can even bridge remote devices running SNAP into one common network, an effective method for centralizing data storage, performing web-based analytics, and monitoring remote applications.

Powered by Freescale's i.MX6 processor, the E20 has ample computing power, reliable connectivity options, and a sturdy design that makes it the ideal network gateway for large-scale IoT deployments.

The E20 gateway pairs this i.MX6 with an SM220 RF module, a microprocessor with a 2.4 GHz data radio that allows the gateway to communicate with your entire SNAP-based network, expanding your command-and-control structure to extend beyond your installation site to anywhere in the world. The gateway not only extends your connectivity range (using cellular or TCP/IP connections), but provides a platform for data aggregation and for even more specialized control of nodes in its network.

The Linux Processor

An E20 gateway ships with Ubuntu 14.04, running a custom 3.10.17 Linux kernel based on the Freescale kernel. With 4 GB of flash space and 512 MB of RAM, the processor has enough power to establish complex control and communication environments for collecting data from your network, making decisions about how to respond, and forwarding data anywhere you want it.

Local USB 2.0 connectivity offers nearly unlimited possibilities for on-premises data warehousing. Integrated wired or wireless TCP/IP options and cellular connectivity permit the gateway to connect to any server you wish — or to be connected to as a remote server, either for configuring your network or for having a single gateway operate as a centralized data repository for a broader SNAP-powered network.

The SNAP-based RF Module

The SM220 module in the E20 gateway provides 2.4 MHz 802.15.4-based radio communications on the SNAP-powered network. The module's ATmega128RFA1 microprocessor provides enough computing power to intelligently handle the communication tasks for commanding your remote network nodes and for receiving their sensory feedback.

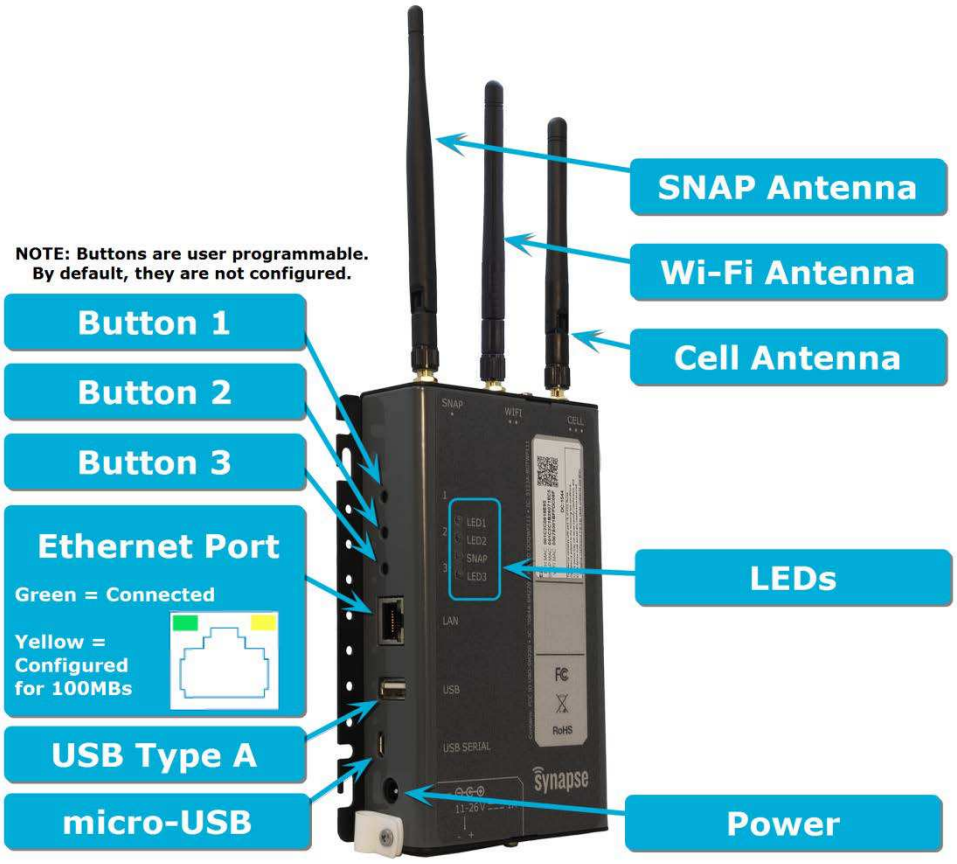
The Symbiosis

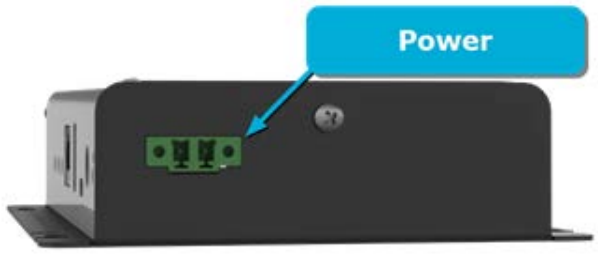
This radio communication is the heart of the SNAP-based mesh network. The E20, combining both the radio module and the power of the Linux computer, is the gateway between your remote nodes and the Internet (or network of your choice), controlling your devices from wherever you are and delivering your data from wherever it is generated to wherever you want it to be.

The software that makes this possible on the E20 gateway is SNAP Connect, a Python package you install and import into your own programs to provide complete control of your SNAP network. SNAP Connect speaks the language of your SNAP-powered nodes, sending and receiving messages and processing incoming and outgoing data to meet your needs. SNAP Connect makes it all work together, seamlessly.

The Device

Depending on configuration, the E20 will have one to three antennas, and connectors as shown below:





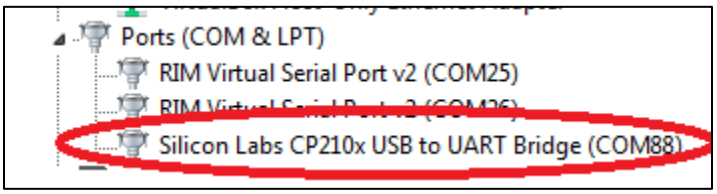
All E20s include a SNAP-powered module, and thus will come with an antenna for the connection labeled SNAP on the E20. Your E20 configuration may also include Wi-Fi and/or cellular connectivity options, and if so it will come with antennas appropriate for those communications. For hardware not included in your E20, there will not be an RP-SMA jack for an external antenna.

If your E20 needs more than one antenna, you can determine which antenna should go on which RP-SMA connector on the E20 based on the number of dots under the label on the E20 (one for SNAP, two for WIFI, three for CELL) matching the number of dots marked on the included antenna.

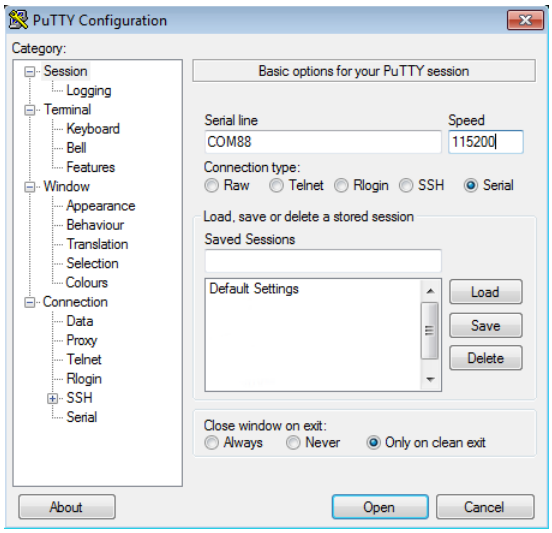
2. Getting Started

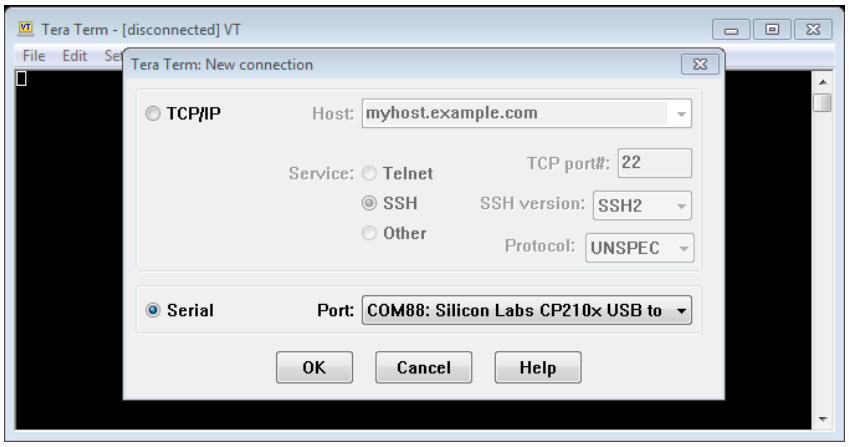
Adding an E20 gateway device to your SNAP network is easy, but as with adding any computer to any network, if you don't follow the right steps, you'll end up in the wrong place. These directions provide the steps for connecting to your E20 from either a Windows PC or a Linux PC, which we will refer to as your host PC. They assume that you have some familiarity with your host operating system. See your OS help files if you need assistance installing software or navigating applications.

1. Ensure that you have terminal emulation software installed on your host PC. Popular software for this purpose includes Tera Term, PuTTY, minicom, screen, or any of many others.
2. Connect the micro-USB port on your E20 to an open USB port on your host PC using a standard USB to micro-USB cable, such as might be used for charging a cell phone. This will provide the serial terminal connection needed to configure your E20.
3. Apply power to the E20.
   The device requires DC power from 11 to 26 volts, and you can use either the barrel connector on the "left" side, or the terminal-block connector on the "bottom" side. (See Powering the E20 below for power supply options.)
4. Find the serial port that your host PC has assigned to the E20 (over that USB connection)
   a. Windows:
      i. Check under "Ports" in the Device Manager.
      ii. Look for "Silicon Labs CP210x USB to UART Bridge (COMxx)" where the xx will indicate the serial port assigned (e.g., COM3, or COM88).



      iii. Connect using your preferred terminal application. Example screenshots of connecting via PuTTY and TeraTerm are provided below:





b. Linux:
   i. Before plugging in the E20's USB cable, check for ttyUSB connections in the /dev directory.
   ii. Plug in the E20 and look for a new ttyUSBx, where x indicates the USB connection assigned (e.g., ttyUSB0).

```
synapse@synapse ~ $
synapse@synapse ~ $ ls /dev/ttyUSB*
/dev/ttyUSB0
synapse@synapse ~ $
synapse@synapse ~ $
```

   If you had any other USB-serial devices plugged in, you may see more than just /dev/ttyUSB0, which is why you should check for the presence of these devices first.
   iii. You can use any of a number of serial terminal programs to connect to your serial port. Linux versions of PuTTy and TeraTerm exist and work the same way shown in the Windows examples above (other than the name of the port to which you are connecting. additionally, utilities such as cu or screen are available as well:
      1. With cu:
         To connect: `sudo cu -l /dev/ttyUSB0 -s 115200`
         To disconnect, at the command line type `~.` (tilde, period) and then press Enter.
      2. With screen:
         To connect: `sudo screen /dev/ttyUSB0 115200`
         To disconnect, press `Ctrl-A`, and then `\` (backslash).

If you find that your host PC cannot connect to the E20 over the USB connection, you may need to install the Silicon Labs CP210x USB to UART VCP drivers, available from http://www.silabs.com

5. Using your terminal emulator, connect to the E20 using the following serial port settings:
   a. 115200 baud
   b. 8 bits
   c. No parity
   d. 1 stop bit
   e. No flow control
6. Use your terminal emulation window to log in to the E20 gateway.
   a. Username: snap
   b. Password: synapse

NOTE: You must change your password the first time you log in. This prevents you from installing an E20 gateway with the default password set, which is pretty much the definition of a bad security idea. Ubuntu enforces some restrictions on what constitutes a valid password.

7. Make an Internet connection with your E20.
   The easiest way to do this is to make a wired connection to a host (i.e., router) that supports DHCP. (Most do, by default.) However, if you wish to configure your device for a static IP address or configure your Wi-Fi at this point, you may do so before making your Internet connection.
8. Install SNAP Connect, if necessary.
   The SNAP Connect software that enables the connection from your E20 device to the rest of your SNAP-powered network did not come preinstalled on E20 image versions before 1.0.8. Fortunately installation is easy, if you need to do it. In your terminal window, while connected to the Internet, execute the following command and check the image builder version[1]:
   ```
   cat e20_version
   ```
   This will tell you which version of the E20 firmware you have. If your version is earlier than 1.0.8, perform the following command:
   ```
   sudo -H pip install snapconnect -i https://update.synapse-wireless.com/pypi/
   ```
   Remember that Ubuntu Linux does not, by default, enable root as a user. The sudo command temporarily escalates your privileges to su, so the E20 will prompt you for your password.
9. Install PyCrypto, if necessary.
   The PyCrypto project is required for using AES128 encryption on your radio network. If you needed to install SNAP Connect, you also need to install PyCrypto. Installing PyCrypto is no more difficult than installing SNAP Connect:
   ```
   sudo pip install python-crypto
   ```

Your E20 is now ready to work with your SNAP-powered network. Your Python program, using the SNAP Connect library, interfaces with the SM220 module directly, and the rest of your nodes through that. You can also have full Internet access (either through a wired connection, Wi-Fi, or a point-to-point cellular connection).

Now it's up to you to do awesome things with your SNAP-powered network. You can find examples of other people's efforts on the Synapse Wireless repository at GitHub: https://github.com/synapse-wireless. The site includes sample projects for things like sending data collected by SNAP-powered nodes to cloud services, or an E20-hosted web server. Download the code there, or fork it for your own projects. Better yet, contribute to the code base for other users.

[1] Be aware that PDF files have been known to internally optimize text to control how it is stored and displayed. Copying and pasting from a PDF file can sometimes result in extra linefeeds or other whitespace being inserted into your pasted text. Before you paste any command in this document into your command prompt window, consider pasting into a text editor (such as Notepad or gedit) to confirm that the text is formatted the way it should be.

3. E20 Software Specifics

The E20 uses Canonical's Ubuntu 14.04, running a custom 3.10.17 Linux kernel based on the i.MX6 kernel by Freescale. There are many resources out there for learning about Ubuntu online, and the topic possibilities far exceed the scope of this manual. However there are a few details that warrant discussion.

Passwords and root Access

The default configuration for Ubuntu Linux is to have the root user disabled. This is a security precaution, as it means a hacker who comes across a connection to your device does not automatically know the login name of a user with full administrative rights to your device.

Instead, Ubuntu works with the `sudo` paradigm; when you need to perform a function that requires administrative access, you preface your command with `sudo` and then are prompted for your password (as a reminder that what you are doing could potentially affect the device's ability to function).

The default `snap` user on the E20 has `sudo` access, and thus can perform all administrative tasks on the device. If you wish, you can create your own user account on the device and grant it `sudo` access as well. Removing the `snap` user would then further reduce a hacker's knowledge of how to access the gateway.

If instead you would rather work with the root account, you can enable the account by assigning it a password:

```
sudo passwd root
```

Similarly, you can change the snap password with the same command:

```
sudo passwd snap
```

NOTE: No account can connect via SSH without a password, though connecting over a serial terminal session is possible for accounts with no password.

E20-Specific Software Packages

The E20 comes with several support packages installed, and additional ones are available via `apt` and `pip`.

NOTE: Before installing new packages, be sure to run `sudo apt-get update` to sync your E20 with the package servers so you will obtain the newest version. This action may take a few minutes, depending on your Internet connection speed.

First, upgrade your SNAP Connect and the encryption libraries necessary for AES128 communications using these commands:

```
sudo -H pip install snapconnect -i https://update.synapse-wireless.com/pypi/
sudo pip install -upgrade python-crypto
```

We also recommend you install a collection of utilities for administering the SM220:

```
sudo apt-get install e20-snap-utils
```

Finally, check for any updates to other E20-specific packages[2]:

```
sudo apt-get install e20-cell-helpers e20-leds e20-buttons e20-gpio-scripts e20-
network-help
```

[2] Remember that copying and pasting from PDF files can give unpredictable results. Try pasting into a text editor first to be sure that the complete command comes across as one line, and that there are not added characters in your pasted text. Then, copy from the text editor and paste into your command window. Or, type it into the command window directly.

These installations include the following packages, which are installed in /usr/local/bin except where noted otherwise:

e20-cell-helpers – a cell modem support package	Telit modem example scripts to reset, disable, enable the cell modem	callvz – Invokes PPPD to communicate with cell modem on Verizon power-cell-modem – Powers the cell modem enable-cell-modem – Enables the cell modem wake-cell-modem – Wakes the cell modem reset-cell-modem – Performs a hard reset of the cell modem
	Configuration and control files for cell modems (/etc/ppp/peers)	telit-att – pppd configuration file for ATT telit-att-chat – scripted AT commands issued to modem for AT&T telit-verizon – pppd configuration file for Verizon telit-verizon-disconnect – Disconnects modem from tower verizon-chat - scripted AT commands issued to modem for Verizon
e20-leds, e20-buttons packages – a simple LED and button control scripts package	led-1, led-2, led-3	Controls lighting for led 1, 2, and 3
	button-1, button-2, button-3	Reads button states
e20-gpio-scripts package - Initializes GPIO lines (/etc/rc2.d)	S30gpios	Startup script to initialize GPIO lines package
e20-snap-utils package – maintenance and support scripts for SM220 *This package depends on SNAP Connect*	wake_snap_node	wakes the SM220 (if it was sleeping)
	reset_snap_node	resets the SM220
	flash-bridge	performs maintenance on the SM220

4. E20 Physical Interface

The E20 includes three tri-color LEDs that you can control from your programs, plus three buttons you can monitor. Control scripts in the e20-leds and e20-buttons packages assist with controlling the LEDs and monitoring the buttons.

See the device diagram above for a map of which LED and which button is which.

E20 LEDs

Each of the three LEDs can be red, green, or amber. Each has a Bash script (provided by the `e20-leds` package) you can use to set the LED state:

- `sudo led-1 red`
- `sudo led-2 green`
- `sudo led-3 amber`
- `sudo led-1 off`

By default, all three of these LEDs will turn amber when the E20 is powered on and then turn off after the device boots.

Each of the three LEDs is controlled by a pair of GPIOs from the i.MX6 processor, with one controlling the red, one controlling the green, and the two of them together generating amber.

If you would rather control the LEDs using the GPIOs rather than the provided Bash scripts, these are the lines for each LED and color:

GPIO 40	LED-1	red
GPIO 41	LED-1	green
GPIO 42	LED-2	red
GPIO 43	LED-2	green
GPIO 44	LED-3	red
GPIO 45	LED-3	green

E20 Buttons

The three buttons on the left side of the E20 are fully user-accessible, too. You can monitor the button states at GPIOs 117, 118, and 119 for button 1, button 2, and button 3, respectively. The `e20-buttons` package provides Bash scripts that print the button status to STDIO and return the button status (as 1 for up or 0 for pressed):

- `button-1`
- `button-2`
- `button-3`

You can monitor the i.MX6 processor GPIOs directly rather than using the Bash scripts if you find that to be easier. Unlike the Bash scripts that set states on the E20, these three scripts do not require `sudo` access to run.

5. Working With the SM220

The E20 contains a Synapse Wireless SM220 surface-mount node, which it can access serially via serial ports `/dev/snap0` and `/dev/snap1` connecting to UART0 and UART1 on the module, respectively. By default, SNAP-powered modules communicate serially over UART1, so when making your SNAP Connect connection to the SM220, you should use `/dev/snap1` unless you have modified your SM220's default UART settings.

> **Remember:** SNAP Connect is not delivered on the E20. You install it as you configure your gateway.

For detailed instructions on SNAP Connect, please consult the SNAP Connect Python Package Manual, available from http://forums.synapse-wireless.com.

In addition to the serial connections, there are two GPIO pins from the E20 that are tied to lines on the SM220 for controlling and signaling.

- **GPIO 33:** Tied to GPIO_F1 on the SM220, you can use this pin as a signaling semaphore or to wake the SM220 when it is sleeping.
- **GPIO 34:** Tied to the Reset pin on the SM220, you can use this pin to reboot the module.

Waking the SM220

At times it may be helpful to have the SM220 in your E20 sleep, and then be woken by the E20's processor. If you have installed the recommended e20-snap-utils package, you can easily do this by defining GPIO_F1 on the SM220 as a wake pin, like this:

```
from synapse.pinWakeup import *
from synapse.platforms import *

@setHook(HOOK_STARTUP)
def onStartup():
    setPinDir(GPIO_F1, False)
    setPinPullup(GPIO_F1, True)
    wakeupOn(GPIO_F1, True, True)
```

Now, whether your SM220 is in a timed sleep or an untimed sleep, having the code on your E20 invoke this command will wake the SM220:

```
/usr/local/bin/wake-snap-node
```

This command invokes a Bash script to pull the E20s GPIO33 high, pause a second, and then pull the line low. The Bash script must be invoked as `sudo` or by a process invoked as `sudo`. You can examine the Bash script to see how the GPIO value is controlled for use in your own scripts, should you wish to use the pin as a one-bit signal to the SM220.

Resetting the SM220

Just as you can wake a sleeping SM220, there is a pin you can use to reset your module should you need to. (This is necessary, for example, when you reset factory parameters on the node.)

The e20-snap-utils package provides a script to assist with this, as well. Invoke this Bash script to briefly pull the reset pin low and then release it to high, resetting the node:

```
/usr/local/bin/reset-snap-node
```

While this is an important thing to be able to do, in most circumstances it will be less useful day-to-day. (If you need to reset your SM220 daily, you have code issues you need to address.)

Restoring Functionality to an Unresponsive SM220

The risk of having a module that provides many configuration options is that it expands the possibility of misconfiguring something making you lose contact with the module. The SM220 is like that; there are many things you can do to the module that could make it unresponsive, from setting an encryption key that you then forget, to putting a script on the device that sends the node to sleep with an invalid wake pin defined. (Remember that the E20 only has wake access on pin GPIO_F1.)

The Portal software from Synapse Wireless provides mechanisms for node recovery, but since you cannot make a serial connection from the SM220 in your E20 to Portal, that functionality needs to exist on the E20 as well. The e20-snap-utils package provides help here, too, with the `flash-bridge` Bash script, located in `/usr/local/bin/flash-bridge`.

If you find that your SM220 node is unresponsive or unreachable over the air or serially, the first suspect is typically the user script on the node. Many a programmer has accidentally specified the wrong wake pin or accidentally dropped a node into an endless loop. So, typically the first thing to try in node recovery is forcibly removing the SNAPpy script from your SM220:

```
sudo flash-bridge -e
```

This leaves your SM220's NV parameters untouched, but removes the existing SNAPpy script from the node. You can then load an appropriate script over the air or serially.

If this does not restore your access to the node, the most likely reason for your inability to communicate is mismatched configuration (NV) parameters on the node. This could be the result of different encryption keys or encryption types, misconfigured UARTs, differences in how many CRCs are expected, or some other configuration setting. The easiest thing to do next is to have the node default its NV parameters, which you can also do with `flash-bridge`:

```
sudo flash-bridge -nv
```

This clears the encryption settings (no key, no encryption), sets UART connections to their default settings (UART1, 38,400 baud, 8N1), and clears other settings to their default levels. (Refer to the SNAP Reference Manual for what the defaults are for your firmware version.)

Typically it is best to start with clearing the script in your node before resetting its parameters, because it is possible for the script to re-set (away from default values) parameters that you just reset (to default values).

> **REMEMBER:** Resetting your SM220 to its default settings does not magically mean that other devices can talk to it, over the air or serially. It does mean that you now know how to configure those devices to talk to it.
>
> If you have another radio device on channel 13 using network ID 0xABCD, you will have to set that device to channel 4, network ID 0x1C2C to talk to your defaulted SM220. You can then use that radio connection to move the E20's SM220 to your preferred network settings. Or, you could change those settings serially from your E20 — if your E20 is set to communicate serially the way that your SM220 is (considering encryption keys and types, serial rates, etc.).
>
> The point is: defaulting a device doesn't mean you have it where you want it, only that you now know where to go look for it.

Upgrading the SM220 Firmware

Synapse Wireless is always working to improve the experience with SNAP-powered networks, and that means new firmware every now and then. If you find that you want to upgrade the firmware in your SM220, you can do it over the air or you can do it serially from the E20. The flash-bridge command we've been using for clearing scripts and resetting parameters saves the day again:

```
sudo flash-bridge -i <imageName>
```

For this command, `<imageName>` refers to an absolute or relative path to a Synapse firmware image file, which will have the extension .sfi.

Loading new firmware erases the script previously in the node but does not change any NV parameters (unless the two firmware versions, old and new, have different default values for something).

The SM220-Controlled LED

The SM220 controls the tri-color LED labeled "SNAP" on the case via GPIO_A4 (green) and GPIO_A5 (red). (For amber, use both green and red.) This LED is only accessible via the SM220. It cannot be controlled by the E20's i.MX6 processor, except through calls to the SM220.

These two IO lines from the SM220 will light their respective colors when written high. This sample code demonstrates its use:

```
from synapse.platforms import *

@setHook(HOOK_STARTUP)
def onStartup():
    setPinDir(GPIO_A4, True)
    setPinDir(GPIO_A5, True)
    LED_off()

def LED_off():
    writePin(GPIO_A4, False)
    writePin(GPIO_A5, False)

def LED_green():
    writePin(GPIO_A4, True)
    writePin(GPIO_A5, False)

def LED_red():
    writePin(GPIO_A4, False)
    writePin(GPIO_A5, True)

def LED_amber():
    writePin(GPIO_A4, True)
    writePin(GPIO_A5, True)
```

This is the only LED controllable directly from the SM220. The other three LEDs are controlled from the E20's i.MX6 processor.

Controlling the E20 Processor from the SM220

Just as there are lines from the E20's i.MX6 processor to the SM220 as a wake pin and a reset, there are corresponding pins back to the i.MX6 from the SM220:

- **GPIO_F2:** Tied to GPIO 32 on the i.MX6, you can use this as a one-bit signal from the SM220 even when SNAP Connect software is not running on the E20.
- **GPIO_C4:** Tied to the system reset line on the E20, active low, you can use this to reboot the i.MX6 processor without interrupting service on the SM220.

The reset line will force a reboot of the E20's i.MX6 processor. This may cause the loss of unsaved data, and as with any uncontrolled shutdown of a computer it is not recommended that you use this often. It is intended only to recover an unresponsive E20. The following sample code demonstrates rebooting the i.MX6 processor from the SM220:

```
from synapse.platforms import *

@setHook(HOOK_STARTUP)
    setPinPullup(GPIO_C4, True)
    writePin(GPIO_C4, True)
    setPinDir(GPIO_C4, True)

def resetE20():
    pulsePin(GPIO_C4, 1, False)
```

In contrast, the GPIO_F2 pad on the SM220 can signal the GPIO 32 line on the i.MX6 without any danger to either system, and the signal can be processed or ignored by your E20 program as you choose. The following sample Bash script demonstrates how to watch for the pin change from your E20 perspective:

```
if [[ ! -d /sys/class/gpio/gpio32/ ]] ; then
    echo 32 > /sys/class/gpio/export
    echo in > /sys/class/gpio/gpio32/direction
fi

while [[ `cat /sys/class/gpio/gpio32/value` != 0 ]] ; do
    sleep 1
done

echo "Got interrupt!"
```

6. Accessing the MicroSD Slot

The E20 includes an on the board[3] microSD slot for reflashing your device to its factory state. You can also use a card in this slot as additional flash storage on your gateway if you need it.

The following instructions will work for ext4-, FAT32-, or exFAT-formatted cards. Ubuntu Linux 14.04 does not support exFAT by default, so you will first need to run the following command to install exFAT support:

```
sudo apt-get install exfat-fuse exfat-utils
```

To access a card in the microSD slot:

1. Insert the microSD card into the microSD card slot under the access cover on the rear of the E20:
   - Slide the microSD card carrier toward the bottom of the unit (away from the antenna end) about a sixteenth of an inch (1.5 mm).
   - Open the card carrier frame. It is hinged at the top (antenna end) edge.
   - Insert your microSD card, contacts first, with the contacts exposed.
   - Close the card carrier frame, and slide it toward the top of the unit to lock the card in place.
2. Create a mount point for the card. In this example, the directory will be named sdcard, and it will be in the /mnt directory. (If you have previously done this, you do not need to repeat it.)

   ```
   sudo mkdir /mnt/sdcard
   ```
3. Mount the card.
   a. For cards formatted with the ext4 file system:

      ```
      sudo mount -t ext4 /dev/mmcblk0 /mnt/sdcard
      ```

   b. For cards formatted with the FAT32 file system:

      ```
      sudo mount -t vfat /dev/mmcblk0 /mnt/sdcard
      ```

   c. For cards formatted with the exFAT file system:

      ```
      sudo mount -t exfat /dev/mmcblk0 /mnt/sdcard
      ```
4. You can confirm it is mounted by using the `mount` command and looking for an entry like the following (with the appropriate file system format):

   ```
   /dev/mmcblk0 on /mnt/sdcard type ext4 (rw)
   ```

[3] Early versions of the hardware may not include this feature. See the Factory Restore / Re-Flashing Your E20 section later in this document to determine whether your hardware has it.

7. Using the Cell Modem

The E20 currently supports the Telit DE910-DUAL on Verizon Wireless. Support for AT&T and others is forthcoming.

Activating the Telit Modem on the Verizon Network

Install the E20 Gateway at the location where it will reside during normal operation, then power it to ensure your cellular provider will be able to communicate with it during the activation process.

Setup

You will need to have the following information available to set up service.

Product Model Number	: Synapse E20
Product Manufacturer.	: Synapse
The modem MEID#	: (Unique number located on the E20 label. Highlighted in yellow.)
Type of Modem	: M2M (Note: This isn't a normal cell phone.)

You will also need to know:

- If you will be using PPP connections and if tethering needs to be added to your account options.
- Your data plan usage requirements.
- A contact name for device issues.
- The location (ZIP Code/City/ State) where the gateway will reside
- A unique device name for each E20 being activated.

  An example would be E20-071EC5. This uses the unique SNAP address on the unit label (shown highlighted in red). Using the last 6 hex numbers will ensure each unit is unique and visually traceable.

Modem Activation

Contact a Verizon agent at 1-800-837-4966 and set up a contract, or contact your corporate Verizon representative if an account already exists.

Note: The agent will ask specific questions about the type of plan that will be used. This will depend on your application and related complexity, so be sure to have all information. The agent will assign a phone number, inform you when activation will be complete, and finalize integration between the Gateway and your system.

Email confirmations will be required by the designated account owner. If installation is performed away from the designated account owner, consider arrangements for email confirmation and completion of the activation process.

Final Steps – Verifying the Modem was Successfully Added to the Network

After you complete the activation with your Verizon agent, your E20 should automatically perform a special download (called the Over The Air Service Provision, or OTASP) from Verizon to enable it on the network. This will happen automatically when powered on and does not require any input from you, but it cannot occur until it has been activated on a plan.

You can verify the OTASP process has completed by checking for the presence of two files on your E20:

- `/etc/DE910_programmed_datetime`
- `/etc/DE910_MEID`

`/etc/DE910_programmed_datetime` contains the date and time the modem successfully completed the OTASP procedure. If this file does not exist, the modem has not yet completed the OTASP. If it does, you can use the cat command to view the file's contents:

```
cat /etc/DE910_programmed_datetime
Thu May 10 17:20:52 CDT 2016
```

The file /etc/DE910_MEID contains the MEID of the modem. This is written after the OTASP process completes.

```
cat /etc/DE910_MEID
A1000042F15A7B
```

In most cases, the modem will be able to connect to the network and obtain an IP address within minutes after the OTASP completes. However, it rare cases it may take up to four hours.

Initiating a Data Connection on the Verizon Network

To initiate a PPPD session on Verizon Wireless:

```
/usr/local/bin/callvz &
```

To terminate the connection:

```
poff telit-verizon
```

These scripts are not guaranteed to work with your network or data plan, and are provided for illustrative purposes only.

Troubleshooting Cellular Connectivity

If you are deploying the E20 in a situation that will be dependent on the cellular connection for connectivity, you will most likely want to take some precautions to ensure that the connection re-establishes itself in the event of failure (signal and handshaking issues, etc.) There are several ways to potentially address this, and the best way is largely depended on your needs and setup.

- The addition of a 'persist' string to your PPP configuration file will make PPPD attempt to reconnect if it detects that the connection to the tower has been dropped. The PPPD session initiated by `callvz` uses `/etc/ppp/peers/telit-verizon` as the PPP configuration file.
- You can create a background shell script, monit utility, or cron job that monitors if PPPD is running, and re-launches it if it detects it is not.

- Use of the `reset-cell-modem` script, which pulls the reset line on the cell modem, hardware resetting it if needed.

8. Common Linux Operations

The E20 uses Ubuntu 14.04 as its operating system. You will need to have some Linux knowledge to be able to use the gateway device. The Internet provides ample documentation for all operations within the capability of the E20, and "Linux Manual" is beyond the scope of this document.

However there are a few operations that are likely to be popular, based on the nature of using a gateway device. The following information may save you some time (and frustration) on searching the Internet.

Editing Linux Files

Many of the configuration suggestions below instruct you to create new files on your E20 or edit existing files. There are several ways to go about this, depending on your choice of methods and tools.

The method that old-school Linux gurus might mock you for not using is the classic Vim (Vi IMproved) text editor. If you are already comfortable in vi or Vim, kindly skip to the next section.[4]

For people who prefer a little more help on screen, the popular nano text editor is included in the basic E20 distribution. You can edit a file directly by typing `nano /path/to/filename`, or open nano and then open the file directly from within the editor. Remember that if you are attempting to edit a file that your user does not own (e.g., files in `/etc` that are owned by `root`) you should preface your `nano` command with the `sudo` command in order to open the editor with escalated privileges.

The third option for creating or editing files for the E20 is to create the files on another system completely and them move them into place on the E20. You can move them over an SSH connection or by "sneaker net" using a USB drive. This is the most cumbersome of the options for edits to existing files or for small changes to files. But for more elaborate software suites, it may be appropriate to install your package this way. Remember, if you do, that Windows and Linux use different line endings. You may need to update your file's line endings to the Linux standard using a command like this:

```
sed -i -e 's/\r//' file
```

Making Your Software Run at Startup

There are two main types of things you might want to invoke at startup:

- Scripts that run to completion, such as configuration or logging scripts.
- Applications that you want to start as a service that can be started, stopped, and restarted.

As with many things in the Linux world, both of these are easy once you know how.

Running a Script to Completion

Like many Linux distributions, Ubuntu does not follow all the standards. One such place is that by default, Ubuntu boots to runlevel 2, which allows for multiuser connectivity (per the standards) and networking (which the standards provide at runlevel 3). This is important because it affects where you should add your run-once scripts to have the execute.

There is a Bash script located at `/etc/rc.local` that executes every time the runlevel changes to a new multiuser level. In normal operation, the E20 boots to runlevel 2 and stays there. If you are not actively initializing a new runlevel, this script will only run on boot. You can add commands to this Bash script, which by default does nothing.

[4] Emacs is not included in the base Linux distribution on the E20. You can get it using `sudo apt-get install emacs`

As an alternative, you can add a Bash script in the `/etc/rc2.d/` directory to have the script execute each time runlevel 2 is initialized. (This script would run before the `rc.local` script executes.) This `/etc/rc2.d/` directory contains a `README` file that provides some instructions for naming and configuring your script to run on boot.

Starting a Service

For applications you want to have started as a service, which can be started, stopped, and restarted, you can create an upstart service at `/etc/init/`.

As an example of making a SNAP Connect application run as a service, create a file named `/etc/init/MyOwnApp.conf` and put the following text in it:

```
# SNAP Connect - start a SNAP Connect application as a service
#

description     "Start SNAP Connect"

start on runlevel [2]
stop on runlevel [!2]

exec python /home/snap/my_snapconnect_example.py
```

The command `python /home/snap/my_snapconnect_example.py` would then be executed on boot and stopped on shutdown (or on transition to any other runlevel). You could also administer the application with these commands:

```
sudo service MyOwnApp start
sudo service MyOwnApp restart
sudo service MyOwnApp stop
```

This basic example gives you a starting point for starting your own services. Examine the other `*.conf` files in the `/etc/init/` directory for further examples of how to configure your services.

Setting Your E20's Clock

The E20 has an NTP client that connects to time servers on the Internet to keep its clock set appropriately (to UTC). However the system clock and the hardware clock can get out of sync over time, resulting in the E20 using the hardware clock's time when Internet connectivity isn't available.

First, you should specify the timezone in which your device will reside. An easy way to do this is to use tzdata, which allows you to select the general region, and then select the specific zone for your location.

```
sudo dpkg-reconfigure tzdata
```

Next, and <u>only if</u> the E20's date is not set (i.e. it is not connected to a network so it does not set the date from an NTP server, and the hardware clock has never been set), set the date manually. The following example sets the date to April 20, 2016, at 12:30:59 p.m.

```
sudo date --set "2016-04-30 12:30:59"
Sat Apr 30 12:30:59 CDT 2016
```

You can set the hardware clock from the system clock using the `hwclock` command.

```
sudo hwclock -wu
```

Resetting a Lost User Password

If there's one thing you can count on, it's that at some point a user will forget his or her password. If you have another administrative (sudo) user defined on the device, that user can reset the lost password. However if you have only one administrative user defined and lose that password, you can still recover your E20 — as long as you have physical access and can make a serial connection over the microUSB connection.

- Make your serial connection (as described earlier in this document) and reboot the E20.
- During the boot process, Das U-Boot prints text to STDOUT. When you see the message "`Hit any key to stop autoboot`" on the screen, press a key. This will drop you into a U-Boot prompt.
- From the U-Boot command prompt, enter the following command:
  ```
  U-Boot-E20> setenv mmcargs 'setenv bootargs console=${console},${baudrate} --no-log fec.macaddr=${macaddr} root=${mmcroot} rootdelay=2 rw single'
  ```
- Then, execute this command:
  ```
  U-Boot-E20> boot
  ```

This will boot the E20 gateway into a mode where the user is `root`, with no password specified. From there you can administratively set your user's password using the `passwd` command. (Try `passwd --help` for guidance.)

Typical Steps for Configuring Wi-Fi

By default, the Wi-Fi interface on the E20 is not active on startup. And, as with any system that offers a lot of options, figuring out how to set up your connections can be daunting. These pointers should make your task a lot easier.

Enabling Wi-Fi

Edit the interfaces file at /etc/network/interfaces file using the editor of your choice.

Remove the `#` from the beginning of the following line:

```
#auto wlan0
```

On the next reboot, the Wi-Fi connection will automatically activate, though additional configuration is necessary for it to connect.

Connecting to an Access Point

Connecting to a access point using WPA encryption is fairly easy. You need to provide the `/etc/wpa_supplicant.conf` file with your desired network SSID and a passphrase key.

The easiest way to do this is to use the `wpa_passphrase` application:

```
sudo wpa_passphrase 'myssid' 'mypassword' >> /etc/wpa_supplicant.conf
```

This command generates a passphrase key from your password and then appends the appropriate text to the `/etc/wpa_supplicant.conf` file.

After you do this, you should edit the /etc/wpa_supplicant.conf file to remove the line that includes the clear text of your passkey, and to make sure there are not issues with conflicting network entries. You may also need additional options, depending on your network setup. (Such configuration is beyond the scope of this document.)

You can now reboot (or use `ifup wlan0`) to bring up the interface and connect to the network.

Setting Up Access-Point (AP) Mode

You can establish your E20 gateway to work as an access point for other Wi-Fi devices. This can be useful if, for example, you want to be able to connect directly to your gateway with a laptop or phone to administer your application.

Update udhcpd

Begin by making sure your `udhcpd` application is up-to-date.

```
sudo apt-get install udhcpd
```

Set Your SSID and Passphrase

Generate your passphrase using the wpa_passphrase application. You can direct this output to a file for recall later, or track it in the method of your choice.

```
$ wpa_passphrase 'myssid' 'mypassword'
network={
      ssid="myssid"
      #psk="mypassword"
      psk=2f0568b3492812bd56b946dbaf3fd7dd669b9a4602a09aa6462ff057949b025c
}
```

Set up /etc/udhcpd.conf

The `interface` definition in the `/etc/udhcpd.conf` file defaults to `eth0`. Find the definition in the file (typically within the first dozen lines, but it may vary depending on your current configuration) and change it to `wlan0` instead.

```
# The start and end of the IP lease block
start          192.168.0.20    #default: 192.168.0.20
end            192.168.0.254   #default: 192.168.0.254
# The interface that udhcpd will use
interface      wlan0           #default:eth0
```

Set udhcpd to Run by Default

Edit the `/etc/default/udhcpd` file to comment the line that sets the `DHCPD_ENABLED` parameter to `"no"`.

```
#DHCPD_ENABLED="no"
```

Note that if you are feeling contrary, you can instead set this parameter to `"yes"`.

Assign a Static IP Address

Edit the /etc/network/interfaces file to assign a static IP address so the gateway can act as an access point. By default, the file contains this configuration text:

```
iface wlan0 inet dhcp
    wpa-conf /etc/wpa_supplicant.conf
    wpa-driver wext
```

You can either replace that text with the new configuration text, or comment those lines by inserting a # character at the beginning of each line, and then add your new configuration text.

```
iface wlan0 inet static
    address 192.168.0.1
    netmask 255.255.255.0
```

Specify the IP address you wish to use for your access point, 192.168.0.1 in the example above. You should choose an appropriate private network address, suitable for your needs.