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

SNAPconnect E12

SNAP Enabled Gateway

Doc# 116-081614-030-B000

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

Disclaimers

Information contained in this manual is provided in connection with Synapse Wireless 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 terms and conditions 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 Wireless. 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

Redistribution of code and use in source and binary forms, with or without modification, are permitted provided that it retains the copyright notice, operates only on **SNAP**® networks, and the paragraphs below in the documentation and/or other materials are provided with the distribution:

Neither the name of Synapse nor the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided “AS IS,” without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SYNAPSE AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SYNAPSE OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SYNAPSE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Table of Contents

Overview

IoT solutions demand reliable and flexible wireless connectivity at scale. SNAPconnect Gateway devices are the interface between your SNAP mesh networks and the rest of the world. Whether performing data aggregation to quickly make real-time decisions, merging multiple IoT data streams, or connecting to on-site servers or the cloud, SNAPconnect Gateways provide the connectivity your network needs

Powered by Ubuntu - Easy to Customize

The SNAPconnect E12 Gateway is an embedded Ubuntu Linux computer with the flexibility to add custom or 3rd party software as needed for local databases, IoT platform clients, or other elements. It bridges SNAP edge networks over Ethernet or optional USB connectivity devices such as cell modems, Bluetooth, or Wi-Fi adapters.

Start Quickly with Included Tools

The E12 is ready-made to host SNAP Thing Services – a suite of software tools that unite IoT devices and applications via REST and MQTT for tasks such as data collection, device updates, and network management.

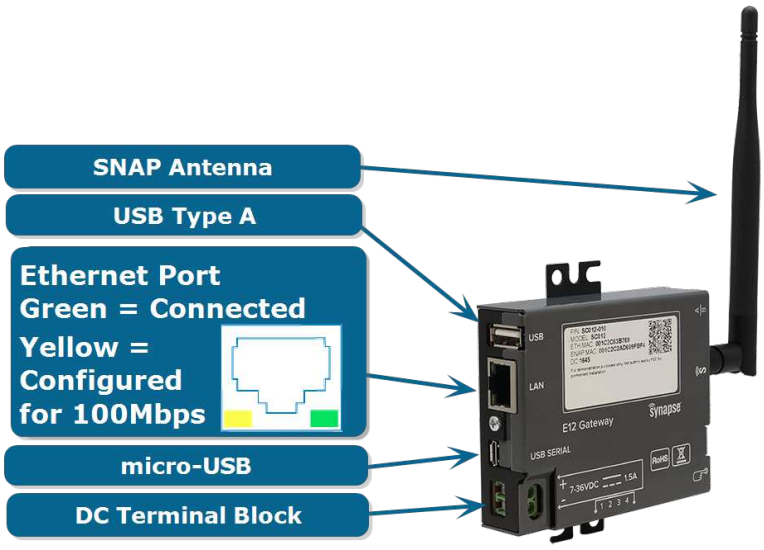
Custom programs can be created with the help of SNAPconnect software libraries making communications with IoT devices in edge networks easy, even in challenging environments.

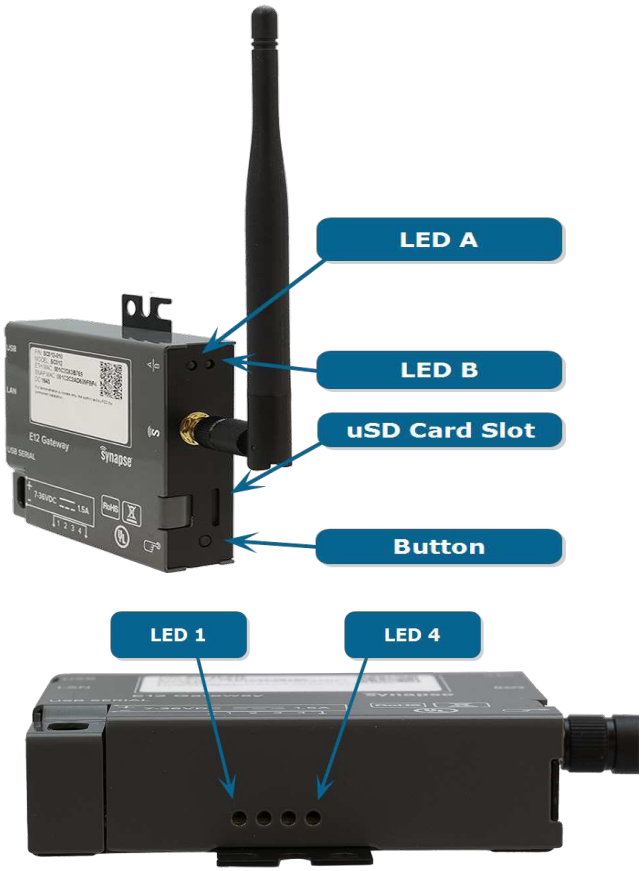
Protect your Deployment

SNAPconnect Gateways emphasize security using AES-128 to encrypt edge network traffic, and standard Linux security tools to secure the backhaul network. SNAP Thing Services utilizes tools like TLS and HTTPS to secure outbound interfaces.

The Device

The images below show an E12 gateway with all features marked:





All E12s include a SNAP-powered module, and thus will come with an antenna for the connection labeled with the Synapse S on the E12.

Getting Started

Adding an E12 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 E12 from either a Windows PC or a Linux PC, which we will refer to as your host PC. These instructions 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.

To get started with the E12, you'll first need to apply power.

Powering the E12

The E12 is powered through the screw terminals with a voltage between 7 VDC and 36 VDC. All transient voltage spikes must stay below 36 VDC. Any wire between 12 AWG and 26 AWG can be used to connect power to the E12. If you are using the 12 volt Phihong power supply, the polarity is such that the wire with the white stripe is the positive lead. If you accidentally switch the leads when connecting power to the E12, don't worry because the input is reverse polarity protected.

NOTE: The E12 cannot be powered over the USB SERIAL port.

Establishing a Serial Connection

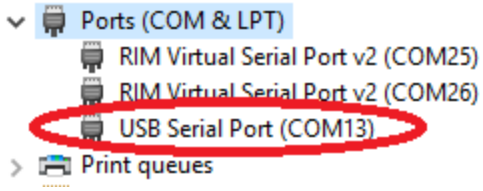
Next you'll need to establish a connection. For the purposes of this document, this will be accomplished through the use of a USB serial connection.

To establish a serial connection to the E12 and update the software:

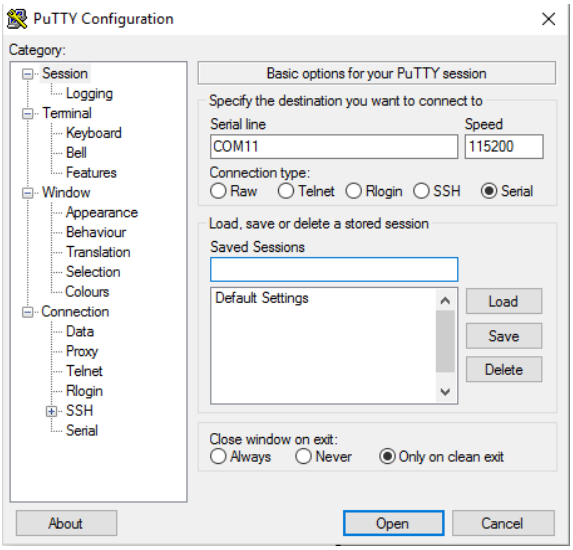
1. Ensure that you have terminal emulation software installed on your host PC. Popular software for this purpose includes Tera Term, PuTTY, minicom, screen, or any of many others.
2. Connect the micro-USB port on your E12 to an open USB port on your host PC using a standard USB to micro-USB cable.
3. Apply power to the E12.

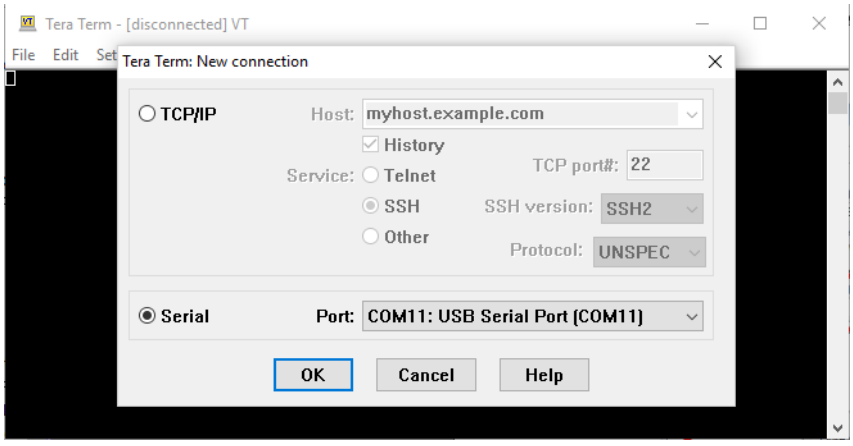
NOTE: During the upcoming step 4, if you find that your host PC cannot connect to the E12 over the USB connection, you may need to install the FTDI USB to UART VCP drivers, available from http://www.ftdichip.com

4. Find the serial port that your host PC has assigned to the E12 (over that USB connection)
    a. Windows:
    i. Check under “Ports” in the Device Manager.



    ii. Look for USB Serial Port (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 E12's USB cable, check for ttyUSB connections in the /dev directory.

ii. Plug in the E12 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).

5. Using your terminal emulator, connect to the E12 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 E12 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 E12 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. Connect your E12 to the internet.
   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, you may do so via the serial connection before making your internet connection.

8. Update the Python Development Libraries
   Before starting work with the E12, you'll want to make sure you have the latest versions of the software installed on the unit. The E12 makes use of Python Development Libraries that are sometimes updated to add new functionality and correct bugs. You can update the version installed on your unit by running the command sudo apt-get install python-dev.

   NOTE: Remember that Ubuntu Linux does not, by default, enable root as a user. The sudo command temporarily escalates your privileges to su (super user), so the E12 will prompt you for your password.

   NOTE: If you get a "Requirement already satisfied" message from one of the updates, don't be alarmed. It just means you already have the latest version.

9. Set the Clock
   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 only if the E12'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
   ```
10. Update SNAPconnect
    The SNAPconnect software enables the connection from your E12 device to the rest of your SNAP-powered network. To update SNAPconnect, type the command:
    ```
    sudo -H pip install --upgrade snapconnect -i https://update.
    synapse-wireless.com/pypi/
    ```
11. Update PyCrypto
    The PyCrypto project is required for using AES128 encryption on your radio network. To update PyCrypto type:
    ```
    sudo -H pip install --upgrade pycrypto
    ```

That's it! Your E12 is now ready to work with your SNAP-powered network. Your Python program, using the SNAPconnect library or the SNAPtoolbelt utilities, interfaces with the RF220SU module directly, and the rest of your nodes through that. You can also have full internet access via the wired Ethernet 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 E12-hosted web server. Download the code there, or fork it for your own projects. Better yet, contribute to the code base for other users.

E12 Software Specifics

The E12 uses Canonical's Ubuntu 14.04, running a custom Linux kernel based on the Beaglebone Black kernal. 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 E12 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.

E12-Specific Software Packages

The E12 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 E12 with the package servers so you will obtain the newest version. This action may take a few minutes, depending on your internet connection speed.

You can pull the latest updates for your gateway by calling:

```
sudo apt-get update
```

```
sudo apt-get install gateway-updates-latest
```

Optionally, you can install updates individually.

As mentioned in the Getting Started section, the SNAPconnect libraries that allow the E12 to connect to the rest of your SNAP-based network are not delivered on your E12 so that you will be sure to have the latest version available to you as you configure your gateway.

First, you can upgrade SNAPconnect and the encryption libraries necessary for AES128 communications using these commands:

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

```
sudo -H pip install --extra-index-url https://update.synapse-wireless.com/pypi snaptoolbelt
```

```
sudo -H pip install pycrypto
```

Finally, check for any updates to other E12-specific packages[1]:

```
sudo apt-get install E12-leds E12-buttons E12-gpio-scripts E12-network-help E12-snap-utils
```

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

E12-leds, E12-buttons packages - a simple LED and button control scripts package	led-1, led-2, led-3, led-4, led-a	Controls lighting for leds
	button	Reads button states
E12-gpio-scripts package - Initializes GPIO lines (/etc/rc2.d)	S30gpios	Startup script to initialize GPIO lines package
E12-snap-utils package - maintenance and support scripts for RF220SU.	reset-snap-node	resets the RF220SU
	wake-snap-node	wakes the RF220SU (if it was sleeping)
	flash-bridge	performs maintenance on the RF220SU

[1] 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.

E12 Physical Interfaces

The E12 includes a tri-color LED that you can control from your programs, plus a button you can monitor. Control scripts in the E12-led and E12-button packages assist with controlling the LED and monitoring the button.

E12 LEDs

The four processor-controlled LEDs are single color - green - and by default are activity indicators. You can set the LED state using the following commands:

- led-1 on / led-1 off
- led-2 on / led-2 off
- led-3 on / led-3 off
- led-4 on / led-4 off
- led-a [red | green | amber | off]

led1 is closest to the terminal blocks and is labeled "1".

An additional LED is on the side of the E12 next to the antenna. This LED, known as LED A, is controlled via the Linux instance running on the E12. The other LED, LED B, is controlled via the SNAPpy script running on the internal SM220 module.

Finally, there are two LEDs on the E12 ethernet port. A green LED that indicates the E12 is connected to the network, and a yellow LED that illuminates when the E12 is configured for 100Mbps communications.

The RF220SU-Controlled LED

The RF220SU controls the tri-color LED on the antenna side of the case (LED A) via GPIO_1 (green) and GPIO_0 (red). (For amber, use both green and red.) This LED is only accessible via the RF220SU. It cannot be controlled by the E12's AM335x processor, except through calls to the RF220SU.

These two IO lines from the RF220SU will light their respective colors when written high. This sample code demonstrates its use:

```
from synapse.platforms import *

GREEN=GPIO_1

RED=GPIO_0

@setHook(HOOK_STARTUP)

def onStartup():

    setPinDir(RED, True)

    setPinDir(GREEN, True)

    LED_off()
```

```
def LED_off():
    writePin(RED, False)
    writePin(GREEN, False)
def LED_green():
    writePin(GREEN, True)
    writePin(RED, False)
def LED_red():
    writePin(GREEN, False)
    writePin(RED, True)
def LED_amber():
    writePin(RED, True)
    writePin(GREEN, True)
```

LED A is the only LED controllable directly from the RF220SU. The other three LEDs are controlled from the E12's AM335x processor.

The E12 Button

The button on the bottom of the E12 is fully user-accessible, too. You can monitor the button state at GPIO 112. The E12-buttons package provides a Bash script that prints the button status to STDIO and returns the button status (as 1 for up or 0 for pressed).

You can monitor the AM335x processor GPIO directly rather than using the Bash script if you find that to be easier. Unlike the Bash script that set states on the E12, this script does not require sudo access to run.

Working With the RF220SU

The E12 contains a Synapse Wireless RF220SU 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 SNAPconnect or SNAPtoolbelt connection to the RF220SU, you should use `/dev/snap1` unless you have modified your RF220SU's default UART settings.

For detailed instructions on SNAPconnect, please consult the SNAPconnect Python Package Manual, available from **http://developer.synapse-wireless.com/software/snapconnect**.

In addition to the serial connections, there is one GPIO pin from the E12 that is tied to the RF220SU for controlling and signaling.

- GPIO 48: Tied to the Reset pin on the RF220SU, you can use this pin to reboot the module.

Waking the RF220SU

At times it may be helpful to have the RF220SU in your E12 sleep, and then be woken by the E12's processor. If you have installed the recommended E12-snap-utils package, you can easily do this by defining GPIO_F1 on the RF220SU as a wake pin, like this:

```
from synapse.pinWakeup import *
from synapse.platforms import *
@setHook(HOOK_STARTUP)
def onStartup():
    setPinDir(GPIO_12, False)
    setPinPullup(GPIO_12, True)
    wakeupOn(GPIO_12, True, True)
```

Now, whether your RF220SU is in a timed sleep or an untimed sleep, having the code on your E12 invoke this command will wake the RF220SU:

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

This command invokes a Bash script to pull the E12s 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 RF220SU.

Resetting the RF220SU

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 E12-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 RF220SU daily, you may have code issues you need to address.)

Restoring Functionality to an Unresponsive RF220SU

The risk of having a module that provides several configuration options is it expands the possibility of a misconfiguration causing you lose contact with the module. Several things can make a module 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.

The Portal software from Synapse Wireless provides mechanisms for node recovery, but since you cannot make a serial connection from the RF220SU in your E12 to Portal, that functionality needs to exist on the E12 as well.

If you find that your RF220SU 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 RF220SU:

```
sudo flash-bridge -e -p RF220
```

This leaves your RF220SU'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 -p RF220
```

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).

NOTE: Resetting your RF220SU to its default settings does not automatically 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 RF220SU. You can then use that radio connection to move the E12's RF220SU to your preferred network settings. Or, you could change those settings serially from your E12 — if your E12 is set to communicate serially the way that your RF220SU 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 RF220SU 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 RF220SU, you can do it over the air or you can do it serially from the E12. You'll need to download the RF220SU firmware by typing:

```
sudo -H pip install snap_firmware_2.8.1 -i https://update.synapse-
wireless.com/pypi/
```

The flash-bridge command we've been using for clearing scripts and resetting parameters saves the day again:

```
sudo flash-bridge –i <imageName> -p RF220
```

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).

For example:

First, download SNAP firmware version 2.8.1 by typing:

```
sudo -H pip install snap_firmware_2.8.1 -i https://update.synapse-
wireless.com/pypi/
```

This will download the firmware images and put them in the folder:
`/usr/local/lib/python2.7/dist-packages/snap_firmware_2_8_1/`

Then you can update the E12 by typing:

```
sudo flash-bridge -i /usr/local/lib/python2.7/dist-packages/snap_firmware_
2_8_1/RF220SU_AES128_SnapV2.8.1.sfi
```

Accessing the MicroSD Slot

The E12 includes an on the board[2] 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. You will need to run the following command for 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 at the end of the E12:
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. (For these commands, replace *p1* with the partition number you want to mount.)
   a. For cards formatted with the ext4 file system:
      ```
      sudo mount –t ext4 /dev/mmcblk0p1 /mnt/sdcard
      ```
   b. For cards formatted with the FAT32 file system:
      ```
      sudo mount –t vfat /dev/mmcblk0p1 /mnt/sdcard
      ```
   c. For cards formatted with the exFAT file system:
      ```
      sudo mount –t exfat /dev/mmcblk0p1 /mnt/sdcard
      ```

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)
```

You can us the ls command to list the available partitions **:**

```
ls /dev/mmcblk0p*
```

[2] Early versions of the hardware may not include this feature. See the **Factory Restore / Re-Flashing Your E12 on page 22.** section later in this document to determine whether your hardware has it.

Common Linux Operations

The E12 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 E12, 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 E12 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.[3]

For people who prefer a little more help on screen, the popular nano text editor is included in the basic E12 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 E12 is to create the files on another system completely and them move them into place on the E12. 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.

[3] Emacs is not included in the base Linux distribution on the E12. You can get it using sudo apt-get install emacs

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 E12 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.

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 SNAPconnect application run as a service, create a file named `/etc/init/MyOwnApp.conf` and put the following text in it:

```
# SNAPconnect - start a SNAPconnect application as a service
#
description     "Start SNAPconnect"
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).

If you want your application to automatically restart (such as if it crashes) you can add the respawn option:

```
# SNAPconnect - start a SNAPconnect application as a service
#
description "Start SNAPconnect"
start on runlevel [2]
stop on runlevel [!2]
respawn
exec python /home/snap/my_snapconnect_example.py
```

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 E12's Clock

The E12 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 E12 using the hardware clock's time when internet connectivity isn't available.

NOTE: These steps are typically performed during setup. The information is repeated here for ease of reference.

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 only if the E12'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 forgotten all the passwords for every account, the only way to regain control of your gateway is to reflash your gateway.

As a preventative measure, you can use the built-in recovery utility to create a flash drive key which will unlock a single user account on your gateway if the flash drive is present on bootup. The key will be unique and for one time use, meaning after you reset a password with it, that key won't work again (the flash drive will still be a normal flash drive.)

To make a recovery flash drive, first ensure you have the newest version of the utility:

```
sudo apt-get install 50-password-reset
```

Then, insert the flash drive you wish to store your recovery key, and run:

```
sudo generate-usb-password-reset USERNAME
```

Where username is the account you wish to be unlocked if this flash drive is present on boot. The key will then be written to the flash drive and stored locally on your gateway. On subsequent boots, if the flash drive containing the key is present, the account specified will have it's password deleted, and you will have to supply a new password upon logging in. The key will NOT work again after use.

If you need to create recovery keys for multiple gateways, you can use the same flash drive multiple times. A unique key will be generated per gateway, but they can all be stored on the same flash drive and the only one which will be removed is the one which was used.

Mounting an External Drive

The USB connection on the E12 is available for mounting external storage, whether that be a flash drive for “sneaker-netting” files, or a larger drive for data aggregation. You can mount and unmount the external drive using these commands, changing the number from 1 to the number appropriate for the partition on your drive:

```
sudo mount /dev/sda1 /mnt
sudo umount /dev/sda1
```

These will mount (and then unmount) the external drive to the /mnt directory in your E12’s file system. You can specify the mount point of your choice, but the mount point must exist as a directory before the drive can be mounted to it.

Extending the E12 with USB Accessories

The E12 has drivers to support many USB devices, such as a second SNAP-powered bridge (using an SN220 or SN132 carrier board), Wi-Fi devices, cell modems, or external storage. While the complete details of configuration options available on these types of devices fall outside the scope of this document, there are some common considerations that may prove useful.

USB Power

The USB 2.0 connection on the E12 is not rated as a "battery-charging" connection, and may not provide sufficient current for high-drain devices, such as some external hard drives.

If you find you are having problems with your USB devices (e.g., external hard drives failing to mount, or cellular connections losing their connections), we recommend you try connecting the device to the E12 through a powered USB hub.

Connecting to an Additional SNAP Device

The E12 can support a second SNAP-powered node through its USB port. You can connect an SN132 or SN220 SNAPstick, or you can use an FTDI USB-serial cable to connect to an SN171 ProtoBoard or some other hardware that uses a DE9 connector to make an RS232 serial connection.

This can allow your E12 to act as a bridge between two radio subnets, where radios are on some combination of different frequencies, different network IDs, and/or different channels.

The E12 provides the drivers that support the FTDI USB-serial cable and the SN132 inherently. To use the SNAPstick SN220, you need to modify it to operate as a serial device rather than a USB device. For instructions on doing this, refer to the Synapse Wireless technical brief "Configuring an SN220 SNAPstick as a COM Port," available from the Synapse support forum.

Whichever device you use, plug the device into the E12's host USB port and (among other messages) you should see something similar to:

```
usb 1-1: FTDI USB Serial Device converter now attached to ttyUSB0
```

or:

```
usb 1-1: cp210x converter now attached to ttyUSB0
```

The key here is the line that says "converter now attached to ttyUSB#" (ttyUSB0, in the example shown). You will use this device handle, "ttyUSB#", to communicate with the SNAP device. In your SNAPconnect application, you would open a connection to the device like this:

```
com.open_serial(type=SERIAL_TYPE_RS232, '/dev/ttyUSB0')
```

Using usb_modeswitch

Many USB Wi-Fi and cell modems now come with a small amount of onboard storage, typically used to automatically install drivers when connected to a Windows host. When the device first connects, it appears as a small flash drive or virtual CD-ROM. After installing the necessary drivers, the Windows host sends a signal to the device instructing it to "mode switch" – to unmount the storage and expose itself as a Wi-Fi (or cellular) device.

Ubuntu Linux also automatically handles many of these devices. But there may be some out there that Ubuntu does not recognize by default. If you find that the E12 is not recognizing your device, consider installing `usb_modeswitch`, which contains a library of parameters for converting devices like these.

```
sudo apt-get install usb-modeswitch
```

Then, plug your device in again and you are likely to find that it works as expected.