Users Guide

E12 Gateway (22.04)

351 Electronics Blvd. SW // Huntsville, AL 35824 // (877) 982-7888 // synapsewireless.com

CONTENTS

The **SNAPconnect E12** is an embedded computer, powered by TI's AM335x processor, and ships with Ubuntu Linux. The **SNAPconnect E12** also includes a Synapse **RF220SU** RF module which allows the gateway to communicate with your entire **SNAP**-based network. It provides remote connectivity via an Ethernet interface, additional memory via micro-SD card slot, and additional communications and storage via USB connection.

Topics

CHAPTER

ONE

GETTING STARTED

These directions provide the steps for terminal access to your **SNAPconnect E12** from Windows, Linux or macOS.

1.1 Requirements

- DC power; the **SNAPconnect E12** requires 12 to 24 volts nominal, 7 to 36 volts absolute, via the terminal blocks on the side of the unit.
- A standard USB to Micro-USB cable, such as might be used for charging a cell phone.
- Terminal emulation software such as Tera Term, PuTTY, cu, screen, etc.

1.2 Connect USB to Host PC

You will need to connect the USB cable to the **SNAPconnect E12** and determine which serial port was assigned. This process is different based on the OS.

> **Note**
>
> If you find that your host PC cannot connect to the **SNAPconnect E12** over the USB connection, you may need to install the FTDI virtual COM port driver.

1.2.1 Windows

Check under “Ports” in the Device Manager and look for “USB Serial Port (COMxx)”. The “COMxx” will indicate the serial port assigned (e.g., COM3, COM88).

1.2.2 Linux

The USB connection will be in the `/dev` directory. Check the directory before and after connecting the USB cable to identify the new `ttyUSBx` where `x` indicates the USB connection assigned (e.g., ttyUSB0). For example:

```
$ ls /dev/ttyUSB*

  /dev/ttyUSB0
  /dev/ttyUSB1   # <- new device
```

1.2.3 macOS

The USB connection will be in the `/dev` directory. Check the directory before and after connecting the USB cable to identify the new device. It is typically `/dev/tty.usbserial-xxxxxxxx`.

```
$ ls /dev/tty.*

  /dev/tty.Bluetooth
  /dev/tty.usbserial-xxxxxxx # <- new device
```

1.3 Terminal Access

Use a terminal emulator to communicate to the **SNAPconnect E12**'s serial port using the following serial port settings:

- 115200 baud
- 8 bits
- No parity
- 1 stop bit
- No flow control

For example, using cu in **Linux**:

```
$ sudo cu -l /dev/ttyUSB0 -s 115200
```

Another example, using screen in **macOS**:

```
$ sudo screen /dev/tty.usbserial-xxxxxxx 115200
```

1.4 Login

Login to your **SNAPconnect E12** gateway the first time using the default credentials:

- Username: snap
- Password: synapse

> **Note**
>
> You must change your password the first time you log in.

1.5 Connect to the Internet

The easiest way to do this is to make a wired connection to an Internet-connected router that supports DHCP. If you wish to configure your device for a static IP address or configure your Wi-Fi at this point, you may do so via the serial connection before making your Internet connection.

1.6 Update Python Development Libraries

Before starting work with the **SNAPconnect E12**, you'll want to make sure you have the latest versions of the software installed on the unit. The gateway 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 update
sudo apt-get upgrade
```

Depending on your use case, you may also want to install development dependencies for Python 2:

```
sudo apt-get install python2-dev
```

1.7 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. You can run it with the command:

```
sudo dpkg-reconfigure tzdata
```

Next, and only if the gateway'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, 2017, at 12:30:59 p.m.`

```
sudo date --set "2017-04-30 12:30:59"
Sat Apr 30 12:30:59 CDT 2017
```

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

```
sudo hwclock -wu
```

1.8 Update SNAPconnect

The SNAPconnect software enables the connection from your **SNAPconnect E12** device to the rest of your SNAP-powered network.

To update SNAPconnect, type the command:

```
sudo -H python2 -m pip install --extra-index-url https://update.synapse-wireless.com/
↪pypi/ --upgrade snapconnect
```

1.9 Update PyCrypto

The PyCrypto project is required for using AES128 encryption on your radio network.

To update PyCrypto type the command:

```
sudo -H python2 -m pip install --extra-index-url https://update.synapse-wireless.com/
↪pypi/ --upgrade pycrypto
```

That's it! Your **SNAPconnect 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 gateway's SNAP module directly, and the rest of your nodes through that. You can also have full Internet access (either through a wired connection or a Wi-Fi connection).

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 a gateway-hosted web server. Download the code there, or fork it for your own projects. Better yet, contribute to the code base for other users.

CHAPTER

TWO

MIGRATION GUIDE

This page highlights significant changes from E12s running Ubuntu 14.04 LTS to E12s running Ubuntu 22.04 LTS.

2.1 Init System: systemd

Ubuntu 14.04 LTS used Upstart as its init system. Later versions use systemd. Though a full systemd manual is beyond the scope of this document, Ubuntu does provide some useful information:

systemd for Upstart Users

2.1.1 Example UserMain.py service

Below is an example of a systemd service which runs the `/home/snap/UserMain.py` script as the `snap` user.

```
[Unit]
Description=UserMain

[Service]
Type=simple
User=snap
Group=snap
ExecStart=/usr/bin/python2 /home/snap/UserMain.py
Restart=always

[Install]
WantedBy=default.target
```

This systemd unit might live in `/etc/systemd/system/usermain.service`. After creating this file, you would need to reload the config, enable the service (so it would run on subsequent boots), and start the service:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable usermain
$ sudo systemctl start usermain
```

By default, output from your service will be captured in the systemd journal. To see the journal for this service:

```
$ sudo journalctl -u usermain
```

2.2 Python versions:

Ubuntu 14.04 LTS provided Python 2.7.6, 3.4, and 3.6.

Ubuntu 22.04 LTS provides Python 2.7.18 and 3.10.4.

See What's new in Python for details of changes between Python versions.

CHAPTER

THREE

E12 SOFTWARE SPECIFICS

The **SNAPconnect E12** uses Canonical's Ubuntu 22.04 operating system. 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.

3.1 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 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 can perform all device administrative tasks. You can create user accounts on the device and grant them sudo access as well. For added security, you can create sudo-enabled user accounts and then delete the snap user account to further reduce a potential hacker's knowledge of how to access the gateway.

If you would rather work with the root account anyway, you can enable it by assigning 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.

3.2 E12 Software Packages

The **SNAPconnect 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 and 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 upgrade
```

> **Note**
>
> You can upgrade SNAPconnect, SNAPtoolbelt, and the encryption libraries necessary for AES128 communications using these commands:
>
> ```
> sudo -H python2 -m pip install --extra-index-url https://update.synapse-wireless.com/
> ↪pypi --upgrade snapconnect snaptoolbelt pycrypto
> ```

CHAPTER

FOUR

LEDS AND BUTTON

4.1 LEDs

The four processor-controlled LEDs are single color - green - and by default are activity indicators. The LEDs are numbered 1 - 4 proceeding from left to right. You can set the LED state using the following commands:

```
led-1 [on|off]
led-2 [on|off]
led-3 [on|off]
led-4 [on|off]
```

LED 1 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.

```
led-a [red|green|amber|off]
```

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.

4.1.1 SNAP Module-Controlled LED

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

These two IO lines from the SNAP module 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()
```

(continues on next page)

(continued from previous page)

```
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 B is the only LED controllable directly from the SNAP module.

4.2 Button

The button on the side of the E12 next to the microSD card slot is fully user-accessible. 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.

CHAPTER

FIVE

WORKING WITH THE SNAP MODULE

) The **SNAPconnect E12** contains a Synapse Wireless RF200SU surface-mount SNAP module, which it can access via serial ports `/dev/snap0` and `/dev/snap1` connected to `UART0` and `UART1` on the module, respectively. By default, SNAP-powered modules communicate serially over `UART1`, so when making your **SNAPstack** or **SNAPtoolbelt** connection to the module, you should use `/dev/snap1` unless you have modified your modules's default UART settings.

For detailed instructions on **SNAPstack**, please consult the SNAPstack Users Guide.

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

E12 Pin	RF220SU Pin	Purpose
GPIO 48	RESET	You can use this pin to reboot the SM220.

By default, the **SNAPconnect E12** ships with **SNAPtoolbelt** installed. **SNAPtoolbelt** can be very handy when needing to perform simple operations or maintenance on the SNAP module.

5.1 Waking the SNAP Module

At times it may be helpful to have the SNAP module in your E12 sleep, and then be woken by the E12's processor. You can easily do this by defining `GPIO_12` on the SNAP module as a wake pin, like this SNAPpy script:

```
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 SNAP module is in a timed sleep or an untimed sleep, having the code on your E12 invoke this command will wake the SNAP module:

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

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 SNAP module.

5.2 Resetting the SNAP Module

Just as you can wake a sleeping RF220SU, there is a pin you can use to reset your module should you need to.

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
```

5.3 Recovering the SNAP Module

Several things can make a module unresponsive, including: setting an encryption key that you then forget, or a script that sends the node to sleep with an invalid wake pin defined, or even a script that sends the node into an infinite loop.

SNAPtoolbelt provides help here, with the `snap recover` options.

If you find that your SNAP module is unresponsive or unreachable over the air or serially, the first suspect is typically the user script on the module. So, the first thing to try in module recovery is forcibly removing the SNAPpy script from your SNAP module:

```
snap recover erase-script RF200SU
```

This leaves your SNAP module'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:

```
snap recover default-nvparam RF220SU
```

This clears the encryption settings (no key, no encryption), sets UART connections to their default settings (UART1, 38400 baud, 8N1), and clears other settings to their default levels. (Refer to the SNAP Reference Guide for other defaults.)

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 reset (away from default values) parameters that you just reset (to default values).

Note

Resetting your SNAP module to 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 SNAP module. You can then use that radio connection to move the gateway's SNAP module to your preferred network settings. Or, you could change those settings serially from your **SNAPconnect E12** — if your gateway is set to communicate serially the way that your SNAP module 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.

CHAPTER

SIX

USING USB ACCESSORIES

The **SNAPconnect 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 devices fall outside the scope of this document, there are some common considerations that may prove useful.

6.1 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.

6.2 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 DB9 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.

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#`. 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')
```

6.3 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.

CHAPTER

SEVEN

ACCESSING THE MICROSD SLOT

The **SNAPconnect E12** includes a 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 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.)**
   - For cards formatted with the ext4 file system:

     ```
     sudo mount -t ext4 /dev/mmcblk0p1 /mnt/sdcard
     ```

   - For cards formatted with the FAT32 file system:

     ```
     sudo mount -t vfat /dev/mmcblk0p1 /mnt/sdcard
     ```

   - 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 use the ls command to list the available partitions:

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

CHAPTER

EIGHT

FACTORY RESTORE

From time to time you may need to restore a **SNAPconnect E12** to the factory default state.

8.1 Restoring from a microSD Card

8.1.1 Download image

Download the newest microSD card **SNAPconnect E12** installer image.

8.1.2 Write new image to a microSD card

Windows

Using an application such as Win32DiskImager, program the microSD card with the downloaded image.

Linux/macOS

Use `dd` from the command line, where `if` is the sdcard image file and `of` is the device location of the microSD card device. For example:

```
$ dd if=e20-VERSION-sdcard.img of=/dev/sdX bs=1M

$ sync
$ eject /dev/sdX
```

Warning

Ensure that the `of` variable points to the correct microSD card device location before executing this command.

8.1.3 Remove the E12 from the Case

Remove power and remove the E12 board from its metal casing. Use proper ESD strategies while touching the bare E12 board.

8.1.4 Insert microSD

Insert the microSD card into the microSD card slot.

8.1.5 Use microSD to restore E12

- Hold the button on the back of the PCB (not the button near the slot)
- Apply power
- Release the button

After a few moments, the LED should turn from amber to red, indicating the restore process is in progress. After about five minutes (your time may vary depending on factors such as the speed of your microSD card), the LED will turn green, indicating that the process is complete. Finally, disconnect power from the device, remove the microSD card, reassemble the E12.

Your **SNAPconnect E12** is now restored and ready to use.

CHAPTER

NINE

TECHNICAL SPECIFICATIONS

The Synapse Wireless E12 gateway is a free-standing ARM Cortex-A8 based Linux computer running Ubuntu 22.04, and incorporating a 2.4 GHz Synapse RF module that connects the device to SNAP-powered mesh networks. The gateway is available with serial connectivity through a microUSB connection, and Ethernet connectivity through the RJ-45 connector.

This arrangement provides for a wide range of possibilities for monitoring sensor networks, controlling remote devices, and driving the internet of things.

9.1 Hardware Specifications

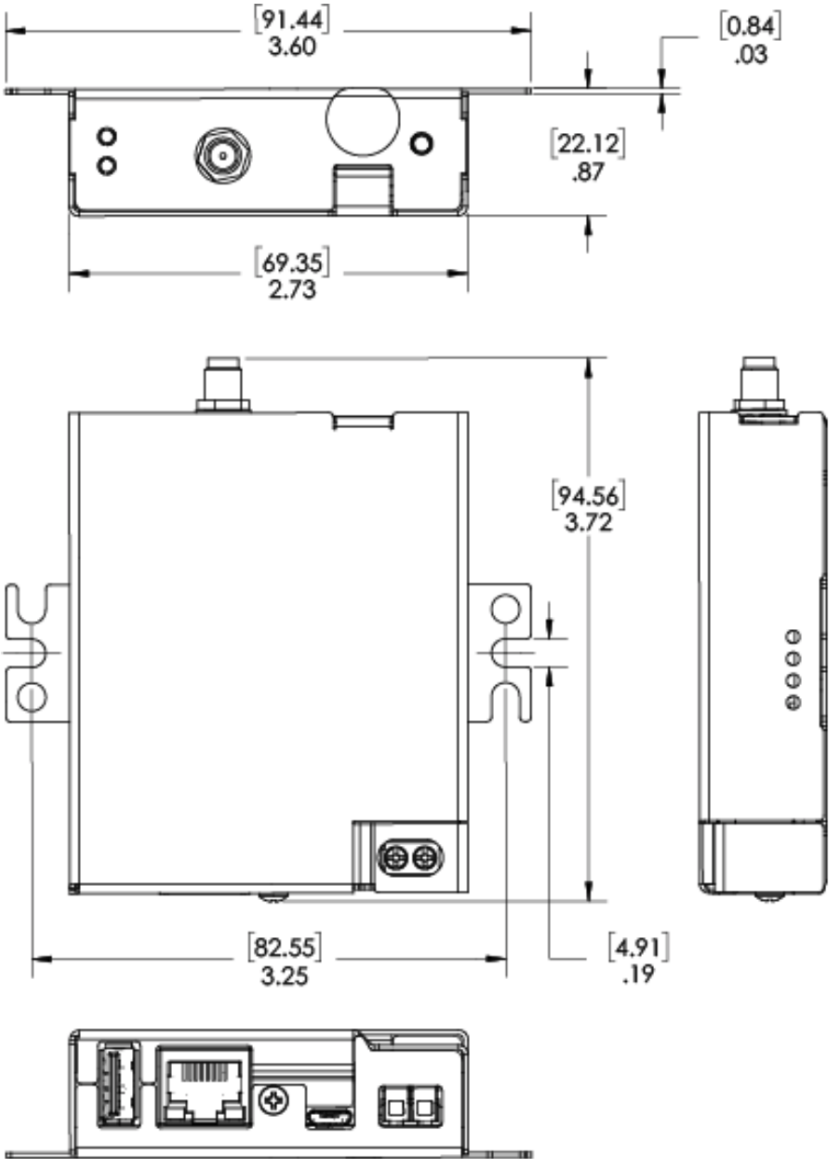
OS	Ubuntu 22.04 LTS
CPU	TI AM335x 1GHz
Flash	4GB NAND FLASH memory
RAM	512M DDR3
Network	10/100 Ethernet, WiFi, SNAP 2.4 GHz
USB host	USB 2.0 type A
USB client	Micro-USB (serial)
Min Operating Temperature	-40°C
Max Operating Temperature	70°C
Board Size	15.5cm x 9cm x 2cm
Unit weight	582 grams (without antennas)
Input Voltage	Nominal: 12-24V DC Absolute: 7-36V DC
Storage Expansion	µSD
LEDs / Buttons	4 Green LEDs 2 Red/Green LEDs 1 Button

Note

This equipment is certified by Underwriters Laboratories for operation in a maximum ambient temperature of 65°C. The product safety standard to which this unit is evaluated and certified, IEC 60950-1, and UL 60950-1 for the US and Canada, specifies a maximum temperature limit of 70°C on metal surfaces that may be touched. During operation, there is a slight temperature rise on the surfaces of the E20. Therefore, when installed into an ambient environment at 70°C, the surface temperatures on the E20 may exceed the limit of 70°C.

9.2 E12 Dimensions

E12 Dimensions



CHAPTER

TEN

TROUBLESHOOTING

10.1 The Ethernet port does not work or eth0 does not appear in ifconfig

Most likely, your MAC address has not been set. If you run `ip address` and see output similar to:

```
eth_badmac Link encap:Ethernet HWaddr 00:1c:2c:ff:ff:ff
```

Your MAC address has reverted to the default and needs to be set. See SNAPconnect is not working below.

10.2 SNAPconnect is not working

Check for one of the following possible issues:

- SNAPconnect may be out of date.
  Prior versions of SNAPconnect were incompatible with some version of the Python Tornado package or may have failed to obtain the MAC address correctly. Upgrade to the latest version of SNAPconnect by using the instructions in E12-Specific Software Packages.
- Your MAC address is not set.
  From the linux command line, run:

  ```
  sudo fw_setenv ethaddr "00:1c:2c:xx:xx:xx"
  ```

  (Replace the XX entries with your last three octets of your Ethernet MAC address, which should be found on the label on your E12's case. Be sure to use the Ethernet MAC address (labeled ETH MAC) and not the SNAP MAC address or Wi-Fi MAC address.)
  Then reboot.

10.3 Cannot SSH into my E12

You cannot SSH into the E12 as root, or any user account which does not have the password set. Be sure to have set a password for the account you want to use to connect.

CHAPTER

ELEVEN

REGULATORY INFORMATION AND CERTIFICATIONS

11.1 RF Exposure Statement

This equipment complies with FCC radiation exposure limits set forth for an uncontrolled environment. This equipment should be installed and operated with minimum distance of 20cm between the radiator and your body. This transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.

11.2 FCC Certifications and Regulatory Information (USA Only)

11.2.1 FCC Part 15 Class B

These devices comply with part 15 of the FCC rules. Operation is subject to the following two conditions: (1) These devices may not cause harmful interference, and (2) These devices must accept any interference received, including interference that may cause harmful operation.

11.2.2 Radio Frequency Interference (RFI) (FCC 15.105)

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However,there is no guarantee that the interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures: Reorient or relocate the receiving antenna. Increase the separation between the equipment and receiver. Connect the equipment into an outlet on a circuit different from that of the receiver. Consult the dealer or an experienced radio/TV technician for help. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

11.2.3 Labeling Requirements (FCC 15.19)

This device complies with Part 15 of FCC rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation. If the FCC ID for the module inside this product enclosure is not visible when installed inside another device, then the outside of the device into which this product is installed must also display a label referring to the enclosed module FCC ID.

11.2.4 Modifications (FCC 15.21)

Changes or modifications to this equipment not expressly approved by Synapse Wireless, Inc. may void the user's authority to operate this equipment.

11.2.5 Declaration of Conformity

(In accordance with FCC 96-208 and 95-19)

Manufacturer's Name: Synapse Wireless, Inc.

Headquarters: 6723 Odyssey Drive, Huntsville, AL 35806

Synapse Wireless, Inc. declares that the product:

Product Name: E12-0

to which this declaration relates, meet the requirements specified by the
Federal Communications Commission as detailed in the following specifications:

- Part 15, Subpart B, for Class B equipment
- FCC 96-208 as it applies to Class B personal computers and peripherals

The products listed above have been tested at an External Test Laboratory certified per FCC rules and has been
found to meet the FCC, Part 15, Emission Limits. Documentation is on file and available from Synapse Wireless, Inc

11.3 Industry Canada (IC) Statement

This Class B digital apparatus complies with Canadian ICES-003. Cet appareil numérique de la classe B est conforme à la norme NMB-003 du Canada.

11.4 CE Transmit Power

By default, the **SNAPconnect E12** ships with transmit levels that are not within the power levels allowed by CE. It is up to the customer to ensure that their system is CE compliant.

In order to set the SNAP radio's transmit power to CE levels, the following command should be issued from the **SNAPconnect E12**:

```
snap node bridge nvparam 11 0x0200
```