Connect Box - Preparing a release

Jim Adams - April 25, 2020 - This summary document created with input from material originally published by Edwin Steele, Nov 7, 2012 and my build experiences during January 2020.
Updated - May 8, 2020 - Fresh environment setup on 2015 MacBook Pro running Catalina - Note that this uses latest versions of Virtual Box (6.1.6) and Vagrant (2.2.8).
Updated - May 26, 2020 - addition to step 12; also added additional info on a couple of useful tools at the end of the document.
Updated - February 26, 2021 - Major rewrite… Added instructions to include how to make an RPi based build environment. Reformat document to split build instructions for MacBook and RPi based build environments. Include NEO and RPi image build instructions within the two build environments.

CONTENTS:

SECTION 1: Background

This document describes the process for creating a ConnectBox image, suitable for burning by end-users onto microSD. This process can generate images for a NanoPi NEO (NEO), and for a Raspberry Pi (RPi) (tested on Rpi3+ and RPi0w).

Overview
The general process to create an image is:

1. Prepare the **build environment** ("One off steps").
   This document will describe two versions of the build environment.
   a. **Mac based**: This environment is built on a Mac and consists of VirtualBox running Vagrant (in turn, running the Ubuntu operating system) and a terminal window to access the environment. (Note that while this environment is Mac based, the instructions are a good foundation for building a similar environment on a PC).

   b. **Raspberry Pi based**: This environment runs on a Raspberry Pi and does not require the installation of Virtual Box or Vagrant. Access to this environment is via a terminal window running on the RPi desktop.

2. Download and burn a **base image** (target machine specific, ie Raspberry Pi or NanoPi NEO) onto a uSD card.
3. Put the uSD card in the target machine, boot the machine, login and enable SSH password-less login as user "root".
4. Use the build environment to install the ConnectBox code onto the uSD card in the target machine.
5. Remove the uSD card from the target machine and insert it into the build environment computer to allow the build environment to read the uSD card, shrink its contents, and build the code image for distribution.

Typographical notes

Generally, we will be working in a Linux terminal window during the build. (This is true in both the Mac and RPi environments.) When we are typing terminal commands where we are interacting with the Linux environment**,** we will show this as:
~$ *some_command*

where the "~$" represents the prompt from the Linux environment, and the italicized *some_command* represents what the user needs to type.

When we have used the terminal to log into the target machine (ie, RPi or NEO) and are, in essence, typing in the target machine (ie, logged into the target machine and the typed commands are executed on the target machine) we will show those as:
~# *some_command*
(note the difference between the "~$" and "~#".)

Section 2: Creating a Mac Based Build Environment

In the following instructions for the Mac build environment, we will open a terminal window on the Mac and from within that window, launch *vagrant* to create a virtual Ubuntu/Linux machine. It is THIS terminal window which will be used in almost all of the steps in interacting with the target (ie, Raspberry or NEO) machine. When a terminal window on the Mac ***other than*** this window is needed, we will explicitly state that such is the case.

Step 2.1: First, create a home directory for your build activities. In the following, we will use <YHD> to represent the path to Your Home Directory. (I used Users/Jim/ConnectBox/ImageBuild/). Open a regular terminal window (we are not in vagrant yet) and type:

```
~$  cd <YHD>
~$  git clone https://github.com/ConnectBox/connectbox-tools
~$  chmod 777 <YHD>/connectbox-tools-master/deployment/make_cb.py
```

This will create directory below <YHD> named connectbox-tools-master and make the python tool, make_cb.py, executable.

Step 2.2: Set up VirtualBox and Vagrant:
- Install the latest version of the program Virtual Box (VB) (Currently at 6.1.6).
- If the VB fails to install, check Settings -> Security and Privacy to see if there is a line at the bottom re: Syetem Developer Oracle blocked from loading… if so, unlock the Security and Privacy page (lock icon at the bottom left) and click the “Allow” button to allow the program to load.
- Install the VirtualBox Extension pack which is required for mounting a USB device (needed later). (See https://www.nakivo.com/blog/how-to-install-virtualbox-extension-pack/).
- Install the latest version of Vagrant (currently at 2.2.8) from (https://www.vagrantup.com/downloads.html)

Step 2.3: Start VirtualBox and Vagrant:
- Start Virtual Box (if it isn’t already running)
- Open a terminal window and type

```
~$  cd <YHD>/connectbox-tools-master/deployment
~$  vagrant up
~$  vagrant ssh
```

NOTE: We are now in the terminal window that we will use for the remainder of our interactions in building our image. When we reference the "terminal window", it is this window to which we are referring.

(The first time this is run it will install the Ubuntu/Linux machine in Virtual Box… note that this can take a bit of time and may seem to hang the terminal window. Just be patient and WAIT. (Possibly 20 minutes to completion of the initial “vagrant up” process.)

Step 2.4: In this step we will make needed adjustments to the VirtualBox settings.

- Start the VirtualBox (VB) application (if it isn’t already up).
- In the left pane of the application window, check to see if the virtual machine is running. If it shows “Running”, right click on the blue box with the virtual machine name and select “Close -> Power Off”.
- In the next dialog, click “Power Off”.
- Click the click the “Settings” icon (looks like a gear) at the top.
- In the Settings window, click the “Ports” icon and then select the “USB” tab.
- Make sure that “Enable USB Controller” is checked and that “USB 3.0” is also selected.
- Click “OK” to save the settings.
- Click the “Start” icon (green right pointing arrow) at the top. (If you get a request to allow keystrokes click “Open Settings” to open the Settings -> Security and Privacy dialog. Checkmark “Virtual Box and select “Later”.)
- In the terminal window that opened from Virtual Box, note the small icons at the bottom of this screen. Click on the icon which is third from left (looks like a USB plug).
- The window that pops up lists the USB devices that the virtual machine knows about. Click anywhere on the desktop to close this window.
- Now insert a USB memory stick and click that same icon again. You should see your USB stick listed as "**Generic USB Mass Storage Device [0500]**". (Perhaps greyed out because the Apple system has mounted it and so it isn’t available to the VB. That is OK. If you go to Finder and Eject the stick but don’t physically unplug the USB stick, then go back and again click the small USB icon, you will see it listed among the devices and NOT greyed out.)
- Click on your stick in the listing which will mount your stick to VB. Verify that there is now a check mark beside it.
- Now go back and click the "Settings" icon again and select Ports -> USB.
- In the lower half of the dialog on the right side of the "Filters" area, find and click the icon with a "+". You will see your USB stick listed.
- Click the USB stick entry to add your stick to the filters list.
- Click OK.
- Finally, go back to the small VB window we used previously to mount your USB stick to VB and uncheck the stick (unmounted from the Virtual Box).
- Go to the Apple Finder to eject the stick there.
- Remove the stick.

Step 2.5: Note: make sure that we have the vagrant window open (see step 2.3), then type the following to set up the vagrant environment:

```
~$ python3 -m pip install click
~$ python3 -m pip install pygithub
```

Step 2.6: This step will set up a public / private SSH key pair which will be used in doing password-less logins to your target device (NEO or RPi). In the terminal window type:

```
  ~$ ssh-keygen -t rsa
```

and take defaults for storage location (/home/vagrant/.ssh/id_rsa) and empty passphrase. (Note that the referenced path is not accessible from the Mac desktop… it is only available in the Linux environment through our terminal window.)

Step 2.7: Install balenaEtcher (https://www.balena.io/etcher/). We will use this to burn the starting image on our uSD card (later)

This completes the setup of the Mac build environment.

Section 3:
Building a NEO or RPi image in a Mac environment

Burn a Base Image on the uSD Card

Notes:

- The space required for the base image and additional ConnectBox code is less than 8 GB so a card of 8GB or greater should be ok in the creation process.
- Where there are differences between building a NEO image and an RPi image, we will preface the step number with "N" (NEO) or "R" (RPi). Where there is no difference, the step number will be shown without a prefaced letter.

Step N3.1: Use an internet browser to go to https://github.com/ConnectBox/armbian-build (the armbian build repo). Near the top you will see (on the line just above "Clone or download") something like, "6220 commits 9 branches 0 packages 15 releases …"… click on "releases" to see a list of the armbian-build releases available. The most recent is "base-image-190417". Click on the base-image-190417 text to go to the page for that image. Then find the image itself about halfway down the page under "Assets". (At the time of this writing, the image is "Armbian_5.78_Nanopineo_Debian-stretch_next_4.19.34.img.xz".) Click on that to download; then save in an appropriate directory of your choice.

Step R3.1: Use an internet browser to go to https://www.raspberrypi.org/downloads/raspbian/ and download the latest image. Probably want to use the version that has a desktop but without other applications included. I used the “Raspbian Buster with Desktop” image.

Step 3.2: Use balenaEtcher to burn the base image (from Step 3.1) to a uSD card.

Step R3.3: This step only required if we are making an RPi image.
Open a new Mac terminal (not our vagrant terminal window) and type:

```
  ~$  cd /Volumes/boot
  ~$  touch ssh
  ~$  cd
```

Close this new Mac terminal window and then use the Mac desktop tools to eject the uSD card. (It will have the name "boot".)

Enable Password-less SSH login to Target Machine
This step will boot up the target machine (NEO or RPi) with the uSD inserted. We will then make edits to the target machine environment to enable login as user “root” via ssh without password (ie, Keys).

Step 3.4: Put microSD into the target machine (NEO or RPi), and use an IP cable to connect the target machine into your network then power up the target machine.

Step 3.5: Find the IP of the target machine… I used the iPhone app Scany… you are looking for a used IP that comes and goes as you power the target machine up and down… use whatever tool you are familiar with.
In the following instructions, substitute the IP you found for the target in place of 10.0.0.118.

(Note: In the following step, you may need to edit the file “/home/vagrant/.ssh/known_hosts” to remove the credentials from a device earlier assigned the IP now used by your target machine. If you get an error announcing a problem related to that, simply edit the file “known_hosts” (the error message will give you the path) to remove the line beginning with the IP address that the target machine is now using. If there is just one line, remove it.)

Step N3.6: Using the terminal window, connect to the NEO by typing:

```
  ~$ ssh root@10.0.0.118
```

(use password 1234 and follow prompts to create new pw “connectbox”). After entering the new password twice, type Ctrl-c which will close the dialog and exit the NEO.

Step R3.6): Using the terminal window, connect to the RPi by typing:

```
  ~$ ssh pi@10.0.0.118
```

(answer “yes” to the dialog re: logging into a new machine and then use password “raspberry”)

Once we are logged in as user “pi”, we need to enable the “root” account on the RPi machine. So type the following commands:

```
        ~# sudo passwd root
              (follow prompts to give root the password “connectbox")
        ~# sudo echo 'PermitRootLogin yes' >> /etc/ssh/sshd_config
        ~# sudo service ssh restart   // Don’t forget this!
        ~# exit
```

Step 3.7: Display and copy the public key set up in Step 2.6:

```
   ~$ cat .ssh/id_rsa.pub
```

COPY the contents of this file by highlighting the text and doing Cmd-C (this text is the public key we need to put in the authorized_keys file on the target machine).

Step 3.8: Now we will log into the target machine, create the authorized_keys file and paste the public key into that file:

```
   ~$ ssh root@10.0.0.118
          (password is still “connectbox”)
      ~# mkdir .ssh
      ~# cd .ssh
      ~# touch authorized_keys
      ~# chmod 600 authorized_keys
      ~# echo ‘<copied_text>’  >> authorized_keys
```

to paste your ssh public key into the file “authorized_keys”. Note: Paste the required text into the command line in place of *<copied_text>* for the above command. And DON’T forget the single quote characters just before and after the pasted text.

```
      ~#  exit              (exit out of the target machine)
```

Step 3.9: Make sure you can do a PW-less login to to the target machine as user “root”:

```
   ~$ ssh root@10.0.0.118
```

This login should happen WITHOUT the target machine asking for a password

```
      ~#  exit              (exit out of the target machine)
```

Step 3.10: Connect a wifi hub antenna to a USB connector on the target machine. (The build will fail if it doesn’t find a USB wifi dongle.) Lots of offerings on Amazon. Here is one possibility:
https://www.amazon.com/Adapter-600Mbps-802-11ac-Wireless-Compatible/dp/B086WDF6VF/ref=sr_1_15?dchild=1&keywords=usb+wifi+dongle&qid=1589402526&sr=8-15

Step 3.11: Determine if you want to do a "step by step" build (creating a local image that, if desired, can be published to GitHub at a later time) or a fully automated build which, once started, will require only minimal intervention and result in a published build on the GitHub repository. **Note that the fully automated build process requires write permissions to the GitHub ConnectBox repository.** If you do not have write privileges to that repository, or if you just want to build a ConnectBox image for local use, continue to Step 3.12. If you want to do a fully automated build with writes to the GitHub ConnectBox repository, go to Step 3.13.

Step 3.12: Starting the "step by step" build.
From the terminal window run:

```
~$  cd  /home/vagrant
~$  /vagrant/make_cb.py
```

Typical script queries and your responses:

```
      Update ansible [y/N]:  <CR>
      Tag for release [v20200508]: <CR>
      IP of build device: 10.0.0.118       (or whatever IP you found)
```

After about 50 minutes of activity the script prints something like:

```
     PLAY RECAP
      10.0.0.118 : ok=134 changed=106 unreachable=0 failed=0
```

If the build to the uSD has succeeded, the PLAY RECAP (above) will show "failed=0".

This completes the creation of an uSD card containing the ConnectBox software for your target machine. At this point you can now power down the target machine and remove the uSD card. If you want to create a software image (*.img file) which can be used to create other uSD cards with this same image, proceed to Section 6, "Shrinking The Image". Otherwise you can just put this uSD chip into your target machine and power up to use you ConnectBox.

Step 3.13: Fully automated build build process.
The automated build process for a published github image uses the github v3 API to tag repositories, so you need to create a Github API personal access token. (See: https://help.github.com/en/github/authenticating-to-github/creating-a-personal-access-token-for-the-command-line.) The token only needs the *public_repo* scope (and should only be given that scope, to reduce the scope for damage if it is lost). Save this token in a text file. We will use this token in the automated script build (next). NOTE: You will only need to do this the very first time you do a fully automated build. You can use this same key on all future fully automated builds.

Step 3.14: Starting the fully automated build.
From within the terminal window run:

```
~$ cd  /home/vagrant
~$ CONNECTBOX_GITHUB_TOKEN=<your-GitHub-rep-token> python3 /vagrant/
prepare_release.py
```

(Copy the text of the token you created in Step 3.13 and paste it in in place of <your-GitHub-rep-token> in the command line above.)

Typical script queries and your responses:

```
    GitHub token [..]: <CR>
    Tag for release [v20200508]: <CR>
    IP of build device: 10.0.0.118                      (or whatever IP you found)
              Press any key: <CR>                       (this takes a few minutes…)
    Proceed with new tag ‘v20200508’? [y/N]: y
```

After about 50 minutes of activity the script prints something like:

PLAY RECAP
10.0.0.118 : ok=134 changed=106 unreachable=0 failed=0
Insert SD card from device (you may need to attach it to this VM)
If the build to the uSD has succeeded, the PLAY RECAP (above) will show “failed=0”. Continue to Step 3.15.

Step 3.15: Power down the target machine and remove the microSD card. This uSD card now contains a ConnectBox image suitable for running in your target machine. In the next section of the automated build, the Mac will shrink the image which is on the uSD card and from that, create and store an image file (*.img) suitable for creating other uSD cards with that same image.
Put the uSD card into a **microSD to USB** adapter. (**IMPORTANT**: a microSD to SD card adaptor WILL NOT WORK. **You MUST use a USB type adaptor**.) Plug the USB adaptor into the computer.

Step 3.16: Once the USB adapter stick is mounted in the Mac and recognized by vagrant, the terminal will print the message, “*Additional partition(s) detected*” and ask for a key press. Press a key and the build of the shrink process will begin.
(Note: If the build succeed through Step 3.14 but has problems at this point or beyond, you can still use the "step by step" instructions to create the *.img file from you uSD card, and if desired, publish your image to GitHub. See Section 6, "Shrinking The Image".)

Step 3.17: After about 30 more minutes…The script in the terminal next will print the message, “*Compressed image complete and located at: …*”.

An upoad to GitHub follows (about 10 minutes) and then a final message, “*Now, update release notes, inserting changelog and base image name*”. You can now remove your uSD from the Mac. The automated image creation process is complete.

At this point your Mac will have a compressed image file of your uSD suitable for use by Etcher (or other image burning tools) for creating duplicates of your uSD card. This image file is available both locally and on GitHub. The local file will be in your <YHD>/connectbox-tools-master/deployment directory and will end in .img.xz. The ending messages in the terminal will tell you the exact name of the image.

Step 3.18: Update release notes in GitHub by opening https://github.com/ConnectBox/connectbox-pi, the click on the “releases” tab, click the “Edit” button next to the release you just made, add notes as to what was changed or added (see earlier releases for examples), and click “Save draft” at the bottom to save the release notes.

Step 3.19: When you have verified the img works as expected and want to make it available to the public, open GitHub and go again to the page for editing your release. At the bottom you will find a “Publish” button. Click that button to take your build from the “Draft” mode into “Pre-release” mode where it will be available to the public.

Note that the path to your release is available by opening your release page on GitHub and copying the address in the address box at the top of your browser. For my v20200121 build, that address is:
https://github.com/ConnectBox/connectbox-pi/releases/tag/v20200121

Section 4: Creating a Raspberry Pi Build Environment

The following steps detail setting up the Build Environment on a Raspberry Pi. Specifically, I used a RPi4B with 4 GB of ram, but this process should work with an RPi3B+.

Step 4.1: Download a suitable RPi image from the Rasberry Pi website https://www.raspberrypi.org/software/operating-systems/. (Raspberry Pi with Desktop is a good choice.) Burn the image onto a uSD card (minimum 16GB) and insert in your RPi. Attach a keyboard, HDMI monitor and mouse to your RPi machine. Power up the RPi and go through the setup instructions to choose timezone, keyboard, wifi network, etc.

Step 4.2: **Optional:** Change the name of the hostname from raspberrypi to RPi4 by accessing the dialog Preferrences → RaspberryPi Configuration. (Pulldown the raspberry icon at the top left of the screen.) Change the Hostname to RPi4 (or your preferrence) and then click OK. Then do a sudo edit of /etc/hosts to change "localhost" to the new name. Reboot for the changes to take effect.

Step 4.3: Use the Desktop File Manager to create the following directory structure:

```
/home/pi/
        |
        |— connectbox/
                |
                |— Images/
```

Step 4.4: Open up a Terminal window and type the following:

```
~$ pip3 install --user ansible
~$ cd /home/pi/.local/bin
~$ sudo chmod 775 ansible*
~$ cd /home/pi
~$ echo 'export  PATH=$PATH:/home/pi/.local/bin/' >> .bashrc
~$ cd /home/pi/connectbox
~$ ssh-keygen -t rsa     (creates public/private key pair... take defaults)
~$ python3 -m pip install pygithub
~$ git clone https://github.com/ConnectBox/connectbox-tools
~$ chmod 777 connectbox-tools/deployment/make_cb.py
- Reboot the Rpi and open the terminal again
~$ cd /home/pi/connectbox/
~$ ansible --version      (verifies ansible installed)
```

Step 4.5: **Optional**: You may want to install the program Etcher to your RPi build machine. If so, follow this link for instructions:
https://www.hackster.io/Eat_Apples/balena-etcher-on-raspberry-pi-bf4188

This completes the setup of the RPi build environment

Section 5:
Building a NEO or RPi image in an RPi environment

Burn a Base Image on the uSD Card

Notes:

- The space required for the base image and additional ConnectBox code is less than 4 GB so a card of 4GB or greater should be ok in the creation process. (I used 32GB.)
- Where there are differences between building a NEO image and an RPi image, we will preface the step number with "N" (NEO) or "R" (RPi). Where there is no difference, the step number will be shown without a prefaced letter.

Step N5.1: Use an internet browser to go to https://github.com/ConnectBox/armbian-build (the armbian build repo). Near the top you will see (on the line just above "Clone or download") something like, "6220 commits 9 branches 0 packages 15 releases …"… click on "releases" to see a list of the armbian-build releases available. The most recent is "base-image-190417". Click on the base-image-190417 text to go to the page for that image. Then find the image itself about halfway down the page under "Assets". (At the time of this writing, the image is "Armbian_5.78_Nanopineo_Debian-stretch_next_4.19.34.img.xz".) Click on that to download; then save in an appropriate directory of your choice.

Step R5.1: Use an internet browser to go to https://www.raspberrypi.org/downloads/raspbian/ and download the latest image. Probably want to use the version that has a desktop but without other applications included. The "Raspbian Buster with Desktop" image is a good choice.

Step 5.2: Use BalenaEtcher or other image burning software to burn the base image (from Step 5.1) to a uSD card.

Enable Password-less SSH login to Target Machine

This step will boot up the target machine (NEO or RPi) with the uSD inserted. We will then make edits to the target machine environment to enable login as user "root" via ssh without password (ie, Keys).

Step 5.3: Put microSD into the target machine (NEO or RPi), and use an IP cable to connect the target machine into your network then power up the target machine.

Step 5.4: Open a terminal window on the RPi and type:

```
  ~$  cd /home/pi/connectbox/
```

Step 5.5: Find the IP of the target machine… I used the iPhone app Scany… you are looking for a used IP that comes and goes as you power the target machine up and down… use whatever tool you are familiar with.
In the following instructions, substitute the IP you found for the target in place of 10.0.0.118.

Step N5.6: Using the terminal window, connect to the NEO by typing:

```
 ~$ ssh root@10.0.0.118
```

(use password 1234 and follow prompts to create new pw "connectbox"). After entering the new password twice, type Ctrl-c which will close the dialog and exit the NEO.

Step R5.6): Using the terminal window, connect to the RPi by typing:

```
 ~$ ssh pi@10.0.0.118
```

(answer "yes" to the dialog re: logging into a new machine and then use password "raspberry")

Once we are logged in as user "pi", we need to enable the "root" account on the RPi machine. So type the following commands:

```
        ~# sudo passwd root
                (follow prompts to give root the password "connectbox"   then)
        ~# sudo echo 'PermitRootLogin yes' >> /etc/ssh/sshd_config
        ~# sudo service ssh restart   // Don't forget this!
        ~# exit
```

Step 5.7: Display and copy the public key set up in Step 4.4:

```
  ~$ cat /home/pi/.ssh/id_rsa.pub
```

COPY the contents of this file by highlighting the text and doing Cmd-C (this text is the public key we need to put in the authorized_keys file on the target machine.

Step 5.8: Log into the target machine and paste the public key into the file authorized_keys:

```
  ~$ ssh root@10.0.0.118.   (password is still "connectbox")
     ~# mkdir .ssh
     ~# cd .ssh
     ~# touch authorized_keys
     ~# chmod 600 authorized_keys
     ~# echo '<copied_text>'  >> authorized_keys
```

to paste your ssh public key into the file "authorized_keys". Note: Paste the required text into the command line in place of <copied_text> for the above command.

And DON'T forget the single quote characters just before and after the pasted text.

```
    ~#  exit                (exit out of the target machine)
```

Step 5.9: Make sure you can do a PW-less login to to the target machine as user "root":

```
  ~$ ssh root@10.0.0.118
        This login should happen WITHOUT the target machine asking for a password
    ~#  exit
       exit out of the target machine
```

Step 5.10: Connect a wifi hub antenna to a USB connector on the target machine. (The build will fail if it doesn't find a USB wifi dongle.) Lots of offerings on Amazon. Here is one possibility:
https://www.amazon.com/Adapter-600Mbps-802-11ac-Wireless-Compatible/dp/B086WDF6VF/ref=sr_1_15?dchild=1&keywords=usb+wifi+dongle&qid=1589402526&sr=8-15

Step 5.11: Starting the "step by step" build.
From the terminal window run:

```
  ~$  cd  /home/pi/connectbox
  ~$ connectbox-tools/deployment/make_cb.py
```

Typical script queries and your responses:

```
      Update ansible [y/N]:    <CR>
      Tag for release [v20200508]: <CR>
      IP of build device: 10.0.0.118      (or whatever IP you found)
```

After about 50 minutes of activity the script prints something like:

```
    PLAY RECAP
     10.0.0.118 : ok=134 changed=106 unreachable=0 failed=0
```

If the build to the uSD has succeeded, the PLAY RECAP (above) will show "failed=0".

This completes the creation of an uSD card containing the ConnectBox software for your target machine. At this point you can now power down the target machine and remove the uSD card. If you want to create a software image (*.img file) which can be used to create other uSD cards with this same image, proceed to Section 6, "Shrinking The Image". Otherwise you can just put this uSD chip into your target machine and power up to use you ConnectBox.

Proceed to **Section 6: Shrinking the Image**

Section 6: Shrinking the uSD Image

The following are instructions for shrinking the image on a uSD card. The process is very similar for both Mac and RPi development machines. Where differences exist, we will preface the step number with "M" (Mac) or "R" (Rpi).

Step 6.1: After powering down the target machine and removing the microSD card, put the uSD card into a **microSD to USB** adapter. (IMPORTANT: a microSD to SD card adaptor WILL NOT WORK. **You MUST use a USB type adaptor**.) Plug the USB adaptor into the computer (ie, Mac or RPi build machine).

Step 6.2: In the terminal window type:

```
~$ lsblk
```

If the image is for a NEO target, you should see entries for "sdb" and "sdb1". If the image is for an RPi target you should see entries for "sdb", "sdb1" and "sdb2". This will indicate that the uSD has been seen by the system and can be accessed by the next command.

Step M6.3: In the terminal window type:

```
~$ sudo /vagrant/shrink-image.sh /dev/sdb /vagrant/<nameYourImage>.img
```

where "<nameYourImage>.img" is the name you want to use for the image about to be created.

Step R6.2: In the terminal window type:

```
~$ umount /dev/sda1
~$ umount /dev/sda2.     (only if you found an sda2
                          in response to the lsblk command)
~$ cd /home/pi/connectbox
~$ sudo connectbox-tools/deployment/shrink-image.sh /dev/sda Images/
<nameYourImage>.img
```

where "<nameYourImage>.img" is the name you want to use for the image about to be created.

Step M6.3: When this step completes (10-20 minutes) you will have an image of the uSD which will be available in your /vagrant directory (from within the terminal window). (Note that this directory is the same as your connectbox-tools-master/deployment directory of your Mac.)

Step R6.3: When this step completes (10-20 minutes) you will have an image of the uSD which will be available in your /home/pi/connectbox/Images/ directory.

Step 6.4: If you want to compress your .img file using the xz compression tool, you can now type:

```
        ~$ xz -k <path to your image file>
```

and after about an hour, you should have a compressed version of your image located in the same folder as your image file and having the same name as your .img file but ending in ".img.xz".

APPENDIX - Edwin’s original document

Edwin Steele is the author of much of the ConnectBox code and for most of the life of the ConnectBox was the only builder of images for the project. The following are his original instructions on how to create an image from the ConnectBox code residing on GitHub.

Once-off steps

1. Checkout the connectbox-tools repo. It contains the tooling required to create an image
2. The build process uses the github v3 API to tag repositories, so you need to create a Github API personal access token. The token only needs the *public_repo* scope (and should only be given that scope, to reduce the scope for damage if it is lost)
3. The build process tags several repos (see the `CONNECTBOX_REPOS` in https://github.com/ConnectBox/connectbox-tools/blob/master/deployment/prepare_release.py#L18 for the up-to-date list). The Github account that created the API access token must have write access to each of these repos. If you don't have access to the repositories, ask a person with the *owner* role in the Github ConnectBox organisation (https://github.com/orgs/ConnectBox/people)

Release Machine Setup (automated)

1. Install VMWare or VirtualBox and also install a current version of Vagrant (the Vagrantfile can supports both VirtualBox and VMWare)
2. cd into the `deployment` directory of the `connectbox-tools` repository and run `vagrant up`

Release Machine Setup (manual)

1. Find a machine running Ubuntu 16.04 and login to it.
2. Run the `apt-get` commands listed in the `deployment/Vagrantfile` of the `connectbox-tools` repository.
3. Checkout the connectbox-tools repo, cd into the `deployment` directory of the checkout and run `pip3 install -r requirements.txt`

Per-build

This assumes you're using the automated machine setup above. If you're using the manual method, you can probably replace `/vagrant` with the `deployment` directory of the `connectbox-tools` repo that you've checked out on the machine.

1. Update the CHANGELOG.md file in the connectbox-pi repository to list any changes since the last build
2. Prepare a your target device with a base operating system image. On a NEO, use the most recent release from the https://github.com/ConnectBox/armbian-

build repository. For an RPi, try the most recent release of Raspbian lite, but know that ymmv because their image changes.

3. Login to the virtual machine that you've setup in the *Release Machine Setup* section. You can do that by running `vagrant ssh` in the `connectbox-tools/deployment` directory to login to the virtual machine.
4. Make sure that you can perform a passwordless login as root on the target device (the `setup-armbian-host.sh` in the `deployment` directory makes it easy if you have keys already available, and automates the 4 steps below, assuming you're building a NEO image - it's different for an RPi)
    - `ssh root@<neoip>` (pwd: 1234)
    - Set root password to *connectbox* per the prompts
    - Ctrl^c and do not create any extra accounts
    - Put your ssh public key on the device so you don't need to run ansible with `--ask-pass`
5. Run `CONNECTBOX_GITHUB_TOKEN=<whatever-it-is> python3 /vagrant/prepare_release.py` on the release machine and follow the instructions.
6. If the process fails before you remove the microSD from the target device, you can continue the process by running `CONNECTBOX_GITHUB_TOKEN=<whatever-it-is> python3 /vagrant/prepare_release.py --use-existing-tag --tag=<whatever-tag-you-used-when-it-failed>`
7. Update the description of the github release to add release notes, consider whether this is a pre-release and mark it accordingly (historically, release have been marked as pre-release when they've only had limited testing).
8. Publish the release to take it out of draft stage.
9. Broadcast availability on slack.