Production Firmware Customization

Overview

You may wish to change the default configuration or look and feel of the Mesh Rider Radio for you end application before the radio is shipped to then end customer. We recommend either

1. running commands remotely over the Mesh Rider Radio's API. Have a look at our Remote Management Guide for details.
2. having the radio set itself up on factory reset through scripts saved in the radio.
3. using Image Builder to build a custom image

In general, you need to become familiar with the CLI Guide. Our customers will often lock general access to the radio (e.g. SSH or HTTPs) from your end user and only allow radio access through their own software. Their software would have access to the radio over it's API.

Running scripts in the radio

If you are using scripts running inside of the radio, one issue is that the scripts will be wiped if the radio is factory reset. Therefore, we recommend saving your scripts in the `/opt/` directory which is not wiped on a factory reset.

After a factory reset, the radio will run the script `/opt/factoryreset.sh`. Therefore, if you want your changes to survive a factory reset, you can modify the `/opt/factoryreset.sh` file and add your script calls before the `exit 0` line. Your scripts will also need to be saved in the `/opt/` directory.

You can issue a factory reset from the command prompt using the command `firstboot -y && reboot`.

Example - Changing the default SSID and Password

As an example, suppose you want to change the default SSID and Password on the radio so that it holds over a factory reset or firmware upgrade. First modify the file `/opt/factoryreset.sh` so that it looks like this

```
#!/bin/sh

/opt/default-settings.sh &
```

```
exit 0
```

Next create the file `/opt/default-settings.sh` with the following content.

```
#!/bin/sh

# This line is added so that the wireless interface is fully up before any
changes are made
while [ -z "$(ifconfig bat0 2> /dev/null)" ]; do
        sleep 5
done

# These are the configuration changes
uci set wireless.wifi0.mesh_id='MyPreferredSSID'
uci set wireless.wifi0.key='MyPreferredPassword'
uci commit

# This restarts the wireless interface
wifi
```

We added some code which makes sure that the wireless interface is up before any changes are made. It's a good idea to add this to any script. As an additional failsafe, you could temporarily add some code so that this script is only run on the first factory reset and not on subsequent factory resets. This is useful for development in case your factory reset script has problems and bricks the radio. For example, we could modify the `/opt/factoryreset.sh` file like this

```
#!/bin/sh

# check if the file /opt/RunOnlyOnce exists, and if it does, then exit
[ -f "/opt/RunOnlyOnce" ] && exit 0

touch /opt/RunOnlyOnce

/opt/default-settings.sh &

exit 0
```

The first time the radio is factory reset, the file `/opt/RunOnlyOnce` will be created, and the `/opt/default-settings.sh` script will be run. The second time it is factory reset, the file `/opt/default-settings.sh` will not be run, and a true factory reset will happen.

Example - Copying a full configuration

It is possible to copy the configuration of one radio to another radio using our `configclone.sh` utility, which is described here.

First modify the `/opt/factoryreset.sh` file like this

```
#!/bin/sh

/opt/default-settings.sh &

exit 0`
```

Next create the `/opt/default-settings.sh` file with the following content

```
#!/bin/sh

/opt/default-settings.sh &

exit 0
root@smartradio-301a402973:/opt# ^C
root@smartradio-301a402973:/opt# cat default-settings.sh
#!/bin/sh

while [ -z "$(ifconfig bat0 2> /dev/null)" ]; do
        sleep 5
done

cp /opt/backup.tar.gz /tmp/
configclone.sh -r -k
```

Next create a configuration backup using the `configclone.sh` utility. Copy the `backup.tar.gz` file to your local PC in case you need it later, and copy the `backup.tar.gz` file to `/opt/backup.tar.gz`. At this point, you can try and factory reset the radio. It should revert to the same settings. You can copy the files

```
/opt/factoryreset.sh
/opt/default-settings.sh
/opt/backup.tar.gz
```

to any radio of the same model, and it should work. In some cases, you may need to resolve some conflicts during the config clone process.

Building custom software and Images

In order to build customer software or firmware images for your radio, you will need to gain access to our SDK and Image Builder. This can be obtained with an NDA through one of our Sales representatives.

- The SDK can be used to build customer **user-space** software. It cannot be used to build kernel modules or full images.
- Image Builder is used to compile a firmware image. Custom software packages and default settings can be added.

Using the SDK

The Doodle Labs SDK allows you to build additional OpenWrt packages suitable for a subsequent installation to Doodle Labs device via opkg. These can be packages in feeds or your own packages. To use the Doodle Labs SDK, first download and unpack the `doodle-labs-sdk-*-Linux-x86_64.tar.xz` file on a 64-bit x86 Linux PC. Adjust `feeds.conf` and/or `feeds.conf.default` as desired. More information is available here [1, 2, 3].

For external packages and external kernel module founds in feeds, follow below steps.

1. Install packages/libraries required for compilation

   ```
   $ sudo apt update; sudo apt install -y build-essential libncurses5-dev
   libncursesw5-dev zlib1g-dev gawk git gettext libssl-dev xsltproc rsync wget
   unzip python
   ```

2. Update feeds

   Edit `feeds.conf.default` and insert this into the file

   ```
   src-git base https://github.com/openwrt/openwrt;v19.07.7
   ```

   Save file and execute the command

   ```
   $ ./scripts/feeds update -a; ./scripts/feeds install -a
   ```

3. You can list all available packages in feeds via

   ```
   $ ./scripts/feeds list
   ```

4. Select Packages

   ```
   $ make menuconfig
   ```

   In the main menu, enter Global Build Settings and in the submenu, deselect/exclude the following options by pressing space bar

   ```
   Select all target specific packages by default
   Select all kernel module packages by default
   Select all userspace packages by default
   ```

   Still in the menu, find the package you want to build and select it by pressing `m`, this will also select all the dependencies, and you will see that they are all tagged with `<M>` in the menu. You can select multiple packages too.

5. Save the configuration and exit the menu.

6. You can then install and build a package as follows (Suppose ethtool is chosen in step 3, for example)

```
$ make package/ethtool/compile
```

7. The final IPK package can then be found in

```
./bin/packages/mips_24kc/base/ethtool_5.2-1_mips_24kc.ipk
```

8. The .ipk can then be copied to the Doodle lab device and install via opkg.

```
$ scp ethtool_5.2-1_mips_24kc.ipk root@smartradio:/tmp/
root@smartradio:/# opkg install /tmp/your-package.ipk
```

Building Firmware Images

The Doodle Labs Image Builder helps in creating a ready-to-flash firmware image with a custom set of packages for Doodle Labs Device. It is based off of OpenWrt's Image Builder.

To use the Doodle Labs ImageBuilder, first download and unpack the `doodle-labs-imagebuilder-*-Linux-x86_64.tar.xz` file on a 64-bit x86 Linux PC. These instructions were executed on a PC running Ubuntu 20.04.3 LTS.

1. Install necessary packages/libraries required for compilation

```
$ sudo apt update; sudo apt install -y build-essential libncurses5-dev
libncursesw5-dev zlib1g-dev gawk git gettext libssl-dev xsltproc rsync wget
unzip python
```

2. Copy any custom OpenWrt IPK files to the `./packages` directory. You can also see which packages have been precompiled by listing the contents of this directory

```
$ ls packages
```

3. You may also add any custom files which should be added to the image in a directory such as `files`. For example,

```
$ mkdir -p files/root
$ echo "Hello" > files/root/README
```

Tip: A common requirement is to have the radio start with a custom configuration. This can be done by adding custom scripts to `/etc/uci-defaults`. More information is available here. Make sure your script is executed last by giving it a name starting with `99-`. For example, create the file

```
./files/etc/uci-defaults/99-myStartupScript
```

inside of your working directory.

4. Get a list of installed packages from a running Mesh Rider radio. SSH into the radio and run

```
root@smartradio:/# echo $(opkg list_installed | awk '{ print $1 }') >
/tmp/default-packages
```

Now copy this file to your Image Builder working directory

```
$ scp root@<IP address of Mesh Rider radio>:/tmp/default-packages ./
```

5. You can now build a firmware image using

```
$ make image PROFILE=smartradio FILES=files PACKAGES="$(cat default-
packages) -wpad-basic"
```

Note that it is necessary to instruct Image Builder not to package `wpad-basic`. If you have additional packages you wish to add, use

```
$ make image PROFILE=smartradio FILES=files PACKAGES="$(cat default-
packages) -wpad-basic <your package name>"
```

Replace `<your package name>` with the name of your package (it must be in the `./packages` folder).

6. The new firmware will be at `bin/targets/ar71xx/generic/doodle-labs-19.07.7-ar71xx-generic-smartradio-squashfs-sysupgrade.bin`. You can now flash this firmware to your router using the instructions here.

Appendix A - Creating a Hello World program

1. Suppose the SDK directory is `/home/test/sdk-xxx`

```
$ cd /home/test/sdk-xxx
$ mkdir helloworld
$ cd helloworld
$ touch helloworld.c
```

2. Edit helloworld.c as

```
#include <stdio.h>
int main(void)
{
    printf("\nHello, world!\nFrom Doodle Labs.\n\n");
```

```
    return 0;
}
```

3. Build and test the program

```
$ gcc -c -o helloworld.o helloworld.c -Wall
$ gcc -o helloworld helloworld.o
$ ./helloworld
```

4. Prepare package feeds

```
$ cd /home/test/sdk-xxx
$ mkdir -p mypackages/examples/helloworld
$ touch ./mypackages/examples/helloworld/Makefile
```

5. Edit `./mypackages/examples/helloworld/Makefile` using the listing below as a reference (take note that the big indent below is tab (not 8 spaces), also change `SOURCE_DIR` to your actual `helloworld.c` source code directory)

```
include $(TOPDIR)/rules.mk
# Name, version and release number
# The name and version of your package are used to define the variable to
point to the build directory of your package: $(PKG_BUILD_DIR)
PKG_NAME:=helloworld
PKG_VERSION:=1.0
PKG_RELEASE:=1

# Source settings (i.e. where to find the source codes)
# This is a custom variable, used below
SOURCE_DIR:= /home/test/sdk-xxx/helloworld

include $(INCLUDE_DIR)/package.mk

# Package definition; instructs on how and where our package will appear in
the overall configuration menu ('make menuconfig')
define Package/helloworld
SECTION:=examples
CATEGORY:=Examples
TITLE:=Hello, World!
endef

# Package description; a more verbose description on what our package does
define Package/helloworld/description
A simple "Hello, world!" -application.
endef

# Package preparation instructions; create the build directory and copy the
source code.
# The last command is necessary to ensure our preparation instructions
remain compatible with the patching system.
define Build/Prepare
    mkdir -p $(PKG_BUILD_DIR)
    cp $(SOURCE_DIR)/* $(PKG_BUILD_DIR)
    $(Build/Patch)
```

```
endef

# Package build instructions; invoke the target-specific compiler to first
compile the source file, and then to link the file into the final
executable
define Build/Compile
    $(TARGET_CC) $(TARGET_CFLAGS) -o $(PKG_BUILD_DIR)/helloworld.o -c
$(PKG_BUILD_DIR)/helloworld.c
    $(TARGET_CC) $(TARGET_LDFLAGS) -o $(PKG_BUILD_DIR)/$1
$(PKG_BUILD_DIR)/helloworld.o
endef

# Package install instructions; create a directory inside the package to
hold our executable, and then copy the executable we built previously into
the folder
define Package/helloworld/install
    $(INSTALL_DIR) $(1)/usr/bin
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/helloworld $(1)/usr/bin
endef

# This command is always the last, it uses the definitions and variables we
give above in order to get the job done
$(eval $(call BuildPackage,helloworld))
```

6. Update and install feed for the package. Add the following at the end of the `/home/test/sdk-xxx/feeds.conf.default`. Change `/home/test/sdk-xxx` to your actual SDK directory.

```
# Own packages
src-link mypackages /home/test/sdk-xxx/mypackages
```

7. Update feeds and install your package

```
$ cd /home/test/sdk-xxx
$ ./scripts/feeds update -a
$ ./scripts/feeds install -a -p mypackages
```

8. Select `helloworld` packages

```
$ make menuconfig
```

   In the main menu, enter Global Build Settings and in the submenu, deselect/exclude the following options by pressing space bar

```
Select all target specific packages by default
Select all kernel module packages by default
Select all userspace packages by default
```

   Still in the menu, choose the `helloworld` package under Examples menu by pressing `m`. Save the configuration and exit the menu.

9. Compile your package

```
$ make package/helloworld/compile
```

10. You can find the `helloworld_1.0-1_<arch>.ipk` in the `./bin/packages/<arch>/mypackages` folder
11. Copy the `.ipk` to the Smart Radio device and install

```
root@smartradio:/# opkg install helloworld_1.0-1_<arch>.ipk
```

12. Test it

```
root@smartradio:/# helloworld

Hello, world!
From Doodle Labs.
```