# Yocto and FX30 Application Note

SIERRA
WIRELESS®

# Copyright

© 2017 Sierra Wireless Inc. All rights reserved.

The information in this manual is subject to change without notice and does not represent a commitment on the part of Sierra Wireless. SIERRA WIRELESS AND ITS AFFILIATES SPECIFICALLY DISCLAIM LIABILITY FOR ANY AND ALL DIRECT, INDIRECT, SPECIAL, GENERAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES INCLUDING, BUT NOT LIMITED TO,
LOSS OF PROFITS OR REVENUE OR ANTICIPATED PROFITS OR REVENUE ARISING OUT OF THE USE OR INABILITY TO USE ANY SIERRA WIRELESS PRODUCT, EVEN IF SIERRA WIRELESS AND/OR ITS AFFILIATES HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR THEY ARE FORESEEABLE OR FOR CLAIMS BY ANY THIRD PARTY.

Notwithstanding the foregoing, in no event shall Sierra Wireless and/or its affiliates aggregate liability arising under or in connection with the Sierra Wireless product, regardless of the number of events, occurrences, or claims giving rise to liability, be in excess of the price paid by the purchaser for the Sierra Wireless product.

# Revision history

**March 2018 Revision 1.2 Author: nmp**
**Add multiple information**

**Sept 2018 Revision 1.3 Author: nmp**
**Add R14 links for sdk and source download**

**Oct 2018 Revision 1.4 Author: nmp**
**Revised section 7.2**

**Oct 2018 Revision 1.5 Author: nmp**
**Corrected and improved diverse parts. Added auto start section.**

# Table of content

# 1. Introduction

This document shows how to use the Yocto toolchain to customize the Legato Linux Distribution. The nodejs package will be added to the FX30 standard image (hereafter referenced as “minimal image”) and a webserver demo application will be built on top of the new custom kernel. Finally, the user will be explained how to integrate his custom applications, firewall rules or other specifics directly into the customized Legato Linux Distribution and how to deploy it using AirVantage Platform.

## 1.1. Synopsis

The purpose of this application note is twofold:

1. illustrate the development phase and how to customize the Linux distribution and create a webserver application.

2. illustrate the industrialization phase and how to create images based on the custom distribution and embedding custom application / configuration on top of it.

## 1.2. Prerequisites

You should have exposure to the FX30/FX30S sales and/or technical materials to:

- understand the FX30/FX30S capabilities and constraints
- understand the general Legato and Yocto architecture.

## 1.3. Requirements

- An FX30 or FX30S.
- A Linux test pc or laptop with access to the internet
- Beware this application note is specifically targeted for the 3G variant of the FX30 / FX30S (WP85 based). Meaning all the steps and procedures, or the way to build recipes, navigate through the source files or meta-layers, really matches a given association of Linux kernel (3.14), Yocto version (16.10) and meta-layers (meta-swi and meta-columbia-x) that are specific to the WP85 based FX30.

# 2. Yocto and custom Linux distributions

Some applications will need specific libraries or drivers or packages to interface with different equipment or to do specific processing or tasks which will require adding given feature set to the default Linux distribution loaded at factory on the FX30.

For security reasons and especially in the growing IoT world where always more devices are deployed and connected it is important that those are not open and accessible either remotely or locally. By default the FX30 is locked down on all its interfaces except locally on the UB, and it is up to the customer to open and secure the relevant interfaces for his application. For the same reason, the major part of the Legato Linux Distribution root file system is read only. Other reasons being it also preserves the flash lifetime and allows using efficient flash compression algorithms which are two critical factors in embedded Linux.

## 2.1. About Yocto and open Embedded

Yocto: is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture: https://www.yoctoproject.org/

OpenEmbedded (OE): Framework than can create embedded Linux distributions.
http://www.openembedded.org/wiki/Main_Page

## 2.2. Some vocabulary

Recipe:
Instructions on how to add a package to a Linux image. Recipe files end with *.bb (bitbake) or *.bbappend extension.

Layer:
Metadata inside one meta-* folder. It is a collection of recipe(s). 1 layer can: add new recipe(s) or replace other layers recipes. Do add a layer if you want to change the default Linux image or if you plan to import or create recipes which are not pre-loaded onto the default Legato Linux Distribution source package. The source package contains lot of recipes which are not compiled at build (not specified in the make file). You can activate them and rebuild the image. However, if you plan to drag additional / external recipes to the source package, never add them into existing meta-layers, always do create your own meta-layer. Also, if you plan to modify the default configuration of the Linux distribution (eg default iptable rules …) do add those additional configurations into your own meta-layer.

Machine:
Target device for which the Linux image is built. For FX30 swi-mdm9x15/40 platform, where MDM9x15/40 is the Qualcomm chipset embedded inside FX30 WP85xx radio module.

Embedded Linux Image types:

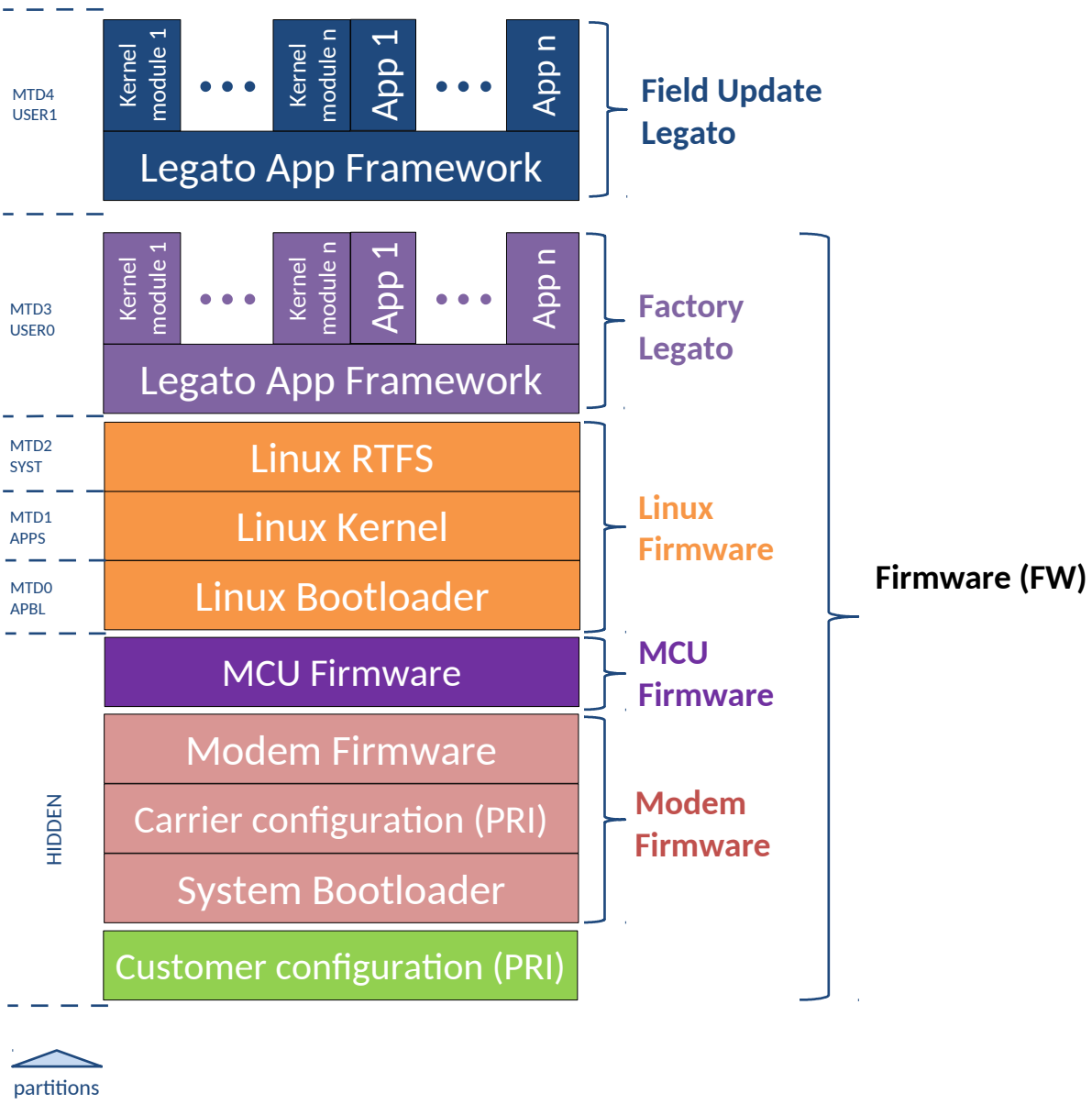
- Minimal image: loaded at factory onto the devices. Based on the Yocto project Poky Linux Distribution. Provides a given feature set. Linux system can have as many flavors as applications and it makes little sense to load them all for deployment on embedded systems. Contrary to desktop and laptops Linux distributions, embedded Linux distributions must meet minimum memory footprint and maximum efficiency with a given CPU core. Hence Sierra provides a minimal image and it is up to each customer, using Yocto, to customize this distribution according to his specifics.

- Development Image: for Sierra Wireless developers, contains additional debug tools. This image is accessible only when setting appropriate options at build (make command). User loads it on his device.

- Custom image: using Yocto, built and compiled by user to add specific packages to the minimal image.

- Color code: throughout this appnote the items in red do refer to directories of the build environment whereas the items in blue do refer to shell commands of the build environment or file edits of the build environment.

## 2.3. Image architecture



# 3. Build default distribution

## 3.1. Download the distribution source package

On your Linux machine, install the SDK through the SPM tool. Follow instructions at this page: https://source.sierrawireless.com/resources/airprime/software/legato_spm/

Then install the PDK (Package Development Kit). PDK is an advanced version of the SDK. Add following command in the Linux terminal (it will also install the FX30 source package):

```
legato-spm -s http://updatesite.sierrawireless.com/packages/Legato/fx30-wp85-R14.0.4.002 -i -m "FX30(WP85XX)" -k pdk.64
```

In case you run a 32 bits machine do replace in the last command pdk.64 by pdk.32.

Check within legato>packages that different legato packages have well been created. Typically legato.device, legato.framework, legato.recovery, legato.sdk, legato.source and legato.toolchain. For the fx30 distribution, the image, framework and sdk packages must as well contain “**fx30_wp85**” in the package name. Usually those files are created into the user directory: /home/user/Legato/packages for normal user or root directory /root/legato/packages in case of having run the command with root loggin.

If for some reasons you do not find the above-mentioned directories, it is most probably because you run an older version of the SDK which do not support FX30 packages.
In this case do run following command: "sudo apt-get upgrade" instead of "sudo apt-get install legato-spm" in the procedure on the Source webpage, and follow the rest of the procedure again.

Beware if you are trying to load the packages from the Studio you will need to activate, in the preferences menu, the displaying of the advanced packages in the packages tab. Note also that the studio is a mainstream oriented tool which is not tailored for advanced development like Yocto recompilation. Hence it is rather advised to switch to command lines, and this is what will be used in the present application note.

## 3.2. Unzip the FX30 distribution source

In the /legato/packages directory, go into the “legato.source.**SWI9X15Y**” directory.

Then go into resources/source. This will point you to the directory containing the FX30 tar source file.

Create a blank directory somewhere on your computer, this will be your working directory.

Caution no spaces in the directory name, otherwise compilation errors at build.

Copy the tar file into the directory you created. Untar the file. In the terminal cd into your created directory, then: tar xf <filename>. You can also just double-click on the tar file from the Linux GUI explorer.

Due to the size of the tar file (3.8GB), operation will take in the 10min time.

If this step is properly conducted there shall now be a « yocto » directory created.

Note: Starting R14 onwards the yocto directory is no more generated. Instead all the packages are available in the unzipped directory called after the tar ball name. Simply cd into this directory and work from here. For all subsequent references to the yocto directory in this app note, please simply understand the unzipped directory called after the tar ball name. Typically with R14 (FX30 3G): FX30_WP8545_full_R14.0.4.002. Also referred in this app note as main directory.

You can also just go the Sierra Wireless Source FX30 FW page, the R14 source code package will be there for direct download.

## 3.3. Prepare and configure PC for build

For legato 16.10 (the links provided in this app note section 3.1 are for legato 16.10):

- Follow instructions to add required tools on your Linux for building the image:

On the legato.io website, in section Build Apps>Concepts>Build Legato>Build Legato>PC host prerequisite packages: http://legato.io/legato-docs/16_10/basicBuildPrepDevHost.html

- Do execute the 3 steps:

1) Install Linux Host Packages

2) Uninstall ccache

3) Set Default System Shell to bash (do answer « no » to the prompt question). Dash must not be the default system shell.

- Be aware that depending on the Linux and GCC version you are using it is possible that you may need to fix problems at build. Typically on an Ubuntu 16.04 no additional configuration is required beside the 3 steps listed above.

For other versions of Legato, host pc configuration might change and you will need to check the exact instructions matching your target legato version.

For reference the instructions for host pc configuration for Legato 16.10 are appended at the end of this document.

Finally, be aware that the legato.io website is always living and that the link provided at the beginning of the paragraph may have moved or have been removed. The most secure path to retrieve the information pertaining to the host pc configuration prior to build is in the documentation downloaded alongside the source package of your target FX30 legato version.
After executing section 3.1 above, whole documentation of the legato 16.10 framework has been downloaded as well and is available under following directory:

legato/packages/legato.framework.16.10.1.fx30_wp85-01704031310/resources/legato/Documentation

Then inside this directory ctrl-f and search for the index.html page.

Open it in your browser, and within this page follow this path: build Apps>Concepts>Build Legato>Build Legato>PC host prerequisite packages.

## 3.4. Build the default image

Prior to generating a custom build, always start by generating a default build (it will generate the image loaded onto the FX30 at factory).

This will ensure the setup and configuration on your PC is correct.

Caution minimum disk space: 50GB needed. Unzipped source size is of 6GB. During compilation source will generate binaries, libraries and other log files, as well as download additional packages, generate images for the bootloader, kernel, rootfs, Legato application framework, custom applications … Depending on the build you want to generate, if you for instance on a WP or mangoH also add drivers for a eg WiFi chipset, this all can stack up to the 50GB (not the target fw which itself is only in the 30MB big, but the whole environment to build the target fw will be in the 50GB).

Beware to have internet access (build will access github or SourceForge to retrieve Linux packages and dependencies).

Note: FX30 drivers and radio stack are locked and cannot be accessed by user.

In the Linux terminal, in the « yocto » directory created after unzipping the source tar file, do « make ».

Depending on your PC, build can take from 45 minutes (8 cores, 3.2GHz each) to 2.5 hours (2 cores, 2.4GHz). After build is finished, go to following directory: yocto>build_bin>tmp>deploy>images>swi_mdm9x15

Look out for file boot-yocto_wp85.cwe.

This is the default bootloader, rootfs and kernel file created by the build.

Note that the binary generated does not contain the legato framework part. Only the boot loader, kernel and root file system. By default the legato framework part is not included in the build. You have to explicitly copy the legato directory inside the yocto directory (legato directory being available amongst the packages installed after execution of section 3.1, under legato/packages/legato.framework.xx.yy.z.fx30_wp85-yyyymmddhhmm/resources") before build so that the make detects it and generates as well the legato.cwe binary. Note also that such binary is available in the source package documentation in the sdk directory home/user/legato/packages/legato.sdk.16.10.1.fx30_wp85-pdk-x86_64-201704040842/resources/device.image (file legato-image.wp85.cwe) and is also already loaded at factory on the FX30. Main driver for wanting to generate one's own legato.cwe file is to generate a custom version of it. This is however not covered in this app note.

Note as well that you will find in the yocto>build_bin>tmp>deploy>images>swi_mdm9x15 directory a mdm9x15-image-minimal-swi-mdm9x15.manifest file listing all modules and packages integrated to the built distribution.

If you like you can jump to section « install image on target » before resuming to next section « build custom distribution ».

# 4. Build custom distribution

Now that we have a nominal setup, we can add custom packages and build our own custom Linux distribution.

Having compiled the default distribution presents advantage that new builds will go way faster to generate as the main image is already available and the build process will simply re-use it and only integrate and build the newly added package(s).

In this section we will add the nodejs package to the default distribution.

Note: sudo apt-get would have been a way to achieve the same on a PC or lab laptop, but is not the kind of tools we want to have on devices on the field. For both security reasons and industrial reasons, the suited approach is to generate one's own image and flash all at once in prod.

## 4.1. Find out matching recipe

At this stage one needs a recipe to add nodejs. 2 approaches:

- make your own recipe or retrieve it from an external source.

Means setting in config files corresponding patches and checksums. Recipes can be dragged from external sites and integrated. Yocto being an open source project one can find various communities and forums with various recipes for different cores. Yocto communities are very active and lot of recipes can be found there. However, addressing how to retrieve and adapt such recipes will need a separate application note.

- find a pre-integrated recipe into the framework.

We'll go for this option. Let's have a look into pre-integrated recipes. Let's search within /Yocto directory after the nodejs keyword. From the terminal on your Linux machine cd into /yocto directory. Issue following command: find ./ -name *nodejs*

You will see following hint:

./meta-openembedded/meta-oe/recipes-devtools/nodejs/nodejs_0.8.18.bb

Most of the recipes (*.bb files) are located in the /yocto/meta-openembedded directory.

If you edit the file using vi you will see the patch location (from nodejs.org in this case) as well as checksums.

## 4.2. Add recipe to make file

Having an existing pre-integrated recipe, one just needs to set the nodejs flag in mdm9x15-image.inc file, and issue make command again.

Browse to following directory: yocto>meta-swi>meta-swi-mdm9x15>recipes-core>images

Edit file « mdm9x15-image.inc », use vi in command line or double-click it.

Add the following lines after the last « # Add » paragraph:

```
# Add nodejs package
IMAGE_INSTALL += "nodejs"
```

Save and quit (« Esc ZZ » in vi).

Upon compilation toolchain will automatically look-up for the pre-loaded nodejs recipe.

Note: As an exercise, you can repeat steps 4.1 and 4.2 and lookout for keyword "openvpn" and add openvpn to the image if you need.

## 4.3. Build custom distro

cd back to the yocto directory. Do « make ».

Because compiler will notice only difference with previous build is the nodejs package, it will only compile this later one. Build shall only take in the 3 minutes this time.

Verify that the target packages have properly been integrated in the distribution. Browse to: yocto>build_bin>tmp>work>swi_mdm9x15-poky-linux-gnueabi>mdm9x15-imageminimal>1.0-r0.0>rootfs>usr>lib and check for the node directories.

Note: never create or modify anything inside the build_bin directory. It is an outptut directory only, it will be erased upon make clean.

Note: to modify the FX30 version at build time, edit the "version" file located in the "meta-columbia-x" folder. DO NOT remove the "R1x.x.x.xxx_modified" string. This string indicates the generated build is a modified build based on FW R1x.x.x.xxx. Simply add your own custom version info at next line. Removing the FW version from this file will lead to inability to know what is the base FW used in case of a troubleshooting.

# 5. Install image on target

Check current Firmware, send following commands from shell: "fwupdate query" and "legato version".

You might want to first install the FW matching your legato framework (in our case FX30 FW R13 and Legato framework 16.10). You can get the FX30 FW from the Source:

https://source.sierrawireless.com/resources/airlink/software_downloads/fx30-firmware/fx30-firmware/
(or navigate the source through Airlink>FX Series>FX30>FW)

- do a scp to copy it from your linux machine onto the FX30 root directory (or winscp if from a windows machine): scp /<path>/<packagename>.spk root@192.168.2.2:.
  Beware the "**:.**" at the end of the root@ command, this will copy file to root directory.

- on the target (FX30 legato shell) do "cd ~"" to set root directory as active directory.

- activate the fwupdate service if not already done: "app status" then if not running, "app start fwupdateService".

- Install file: fwupdate download <filename.spk/cwe>

Follow those same steps again to either install the default distribution compiled at paragraph 3 or the custom distribution compiled at paragraph 4.

In both cases navigate to following directory: yocto>build_bin>tmp>deploy>images>swi_mdm9x15 and install the "boot-yocto_wp85.cwe" file.

To be noted cwe files are image files you get upon one's own yocto compilation whereas .spk files are concatenated .cwe files (eg boot-yocto_wp85.cwe + legato.cwe + ...)

Verify nodejs is recognized in the command line: type « node --help » and check prompt displays the installed node version. Normally v0.8.18 as per Legato framework 16.10.

# 6. Create an application

Now that we have nodejs supported by our file system, let's develop a small webserver running on the FX30 ethernet port. This server will have a loggin page, user "user" and pwd "12345", and once logged in user will be displaid the present application note.

Log into the FX30. For demo purpose we will create our app into /mnt/flash. Cd in /mnt/flash. Create a webserver directory: "mkdir webserver". Cd into it.

1) Create a server.js file. Issue following command: "vi server.js"
Once in vi mode copy paste the script as per appendix section 8.2. Save and exit: "Esc ZZ".

2) Create an index.html file. Issue following command: "vi index.html".
Once in vi mode copy paste the script as per appendix section 8.3. Save and exit: "Esc ZZ".

3) Copy the jpg picture (Sierra Wireless FX30 logo) onto the FX30 inside /mnt/flash/webserver
From your Linux machine terminal send following command:
scp <path_to_the_image/image_name.jpg> root@192.168.2.2:/mnt/flash/webserver
The Sierra Wireless FX30 logo is copied into the appendix section.

4) Open port 80 onto the Ethernet interface.
From the FX30 shell issue following command:
iptables -I INPUT -i eth0 -p tcp -m tcp --dport 80 -j ACCEPT
Beware to use **–I** option before INPUT and not –A. By default the meta-columbia-x layer adds a drop rule for all type of traffic except DNS traffic. Using –I option will insert the rule before the drop rule execution. Using –A would have appended the rule after the drop rule execution and hence traffic would not have reached the Ethernet interface.

FX30 iptable rules after having added the tcp port 80 traffic:

```
root@fx30:~# iptables -nvL
Chain INPUT (policy ACCEPT 8 packets, 528 bytes)
 pkts bytes target     prot  opt   in                   out      source               destination
```

```
    0     0  ACCEPT     tcp  --  eth0       *       0.0.0.0/0            0.0.0.0/0           tcp dpt:80
    0     0  ACCEPT     icmp --  rmnet0     *       0.0.0.0/0            0.0.0.0/0           icmptype 0 state ESTABLISHED
    0     0  ACCEPT     tcp  --  rmnet0     *       0.0.0.0/0            0.0.0.0/0           tcp spt:53 state ESTABLISHED
    0     0  ACCEPT     udp  --  rmnet0     *       0.0.0.0/0            0.0.0.0/0           udp spt:53 state ESTABLISHED
    0     0  DROP       all  --  rmnet0     *       0.0.0.0/0            0.0.0.0/0
    4   336  ACCEPT     icmp --  eth0       *       0.0.0.0/0            0.0.0.0/0
    0     0  ACCEPT     tcp  --  eth0       *       0.0.0.0/0            0.0.0.0/0           tcp spt:53 state ESTABLISHED
    0     0  ACCEPT     udp  --  eth0       *       0.0.0.0/0            0.0.0.0/0           udp spt:53 state ESTABLISHED
   41  3552  DROP       all  --  eth0       *       0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 5 packets, 604 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

One can see only ping and DNS are allowed on cellular (rmnet0) and Ethernet (eth0) interfaces, and that tcp traffic port 80 has been added at top of the list.

5) Configure the FX30 to be DHCP server on ethernet port:

In /etc/dnsmasq.d create a dnsmasq.eth.conf file that will set dnsmasq to run a dhcp server on ethernet interface. On the FX30 shell:

```
$ cd /etc/dnsmasq.d
$ touch dnsmasq.eth.conf
$ vi dnsmasq.eth.conf
```

Add following to dnsmasq.eth.conf file

```
dhcp-range=interface:eth0,192.168.13.200,192.168.13.215,12h
dhcp-option=6,8.8.4.4
```

save and close:

```
ESC:wq!
```

You might want to restart the dnsmasq service:

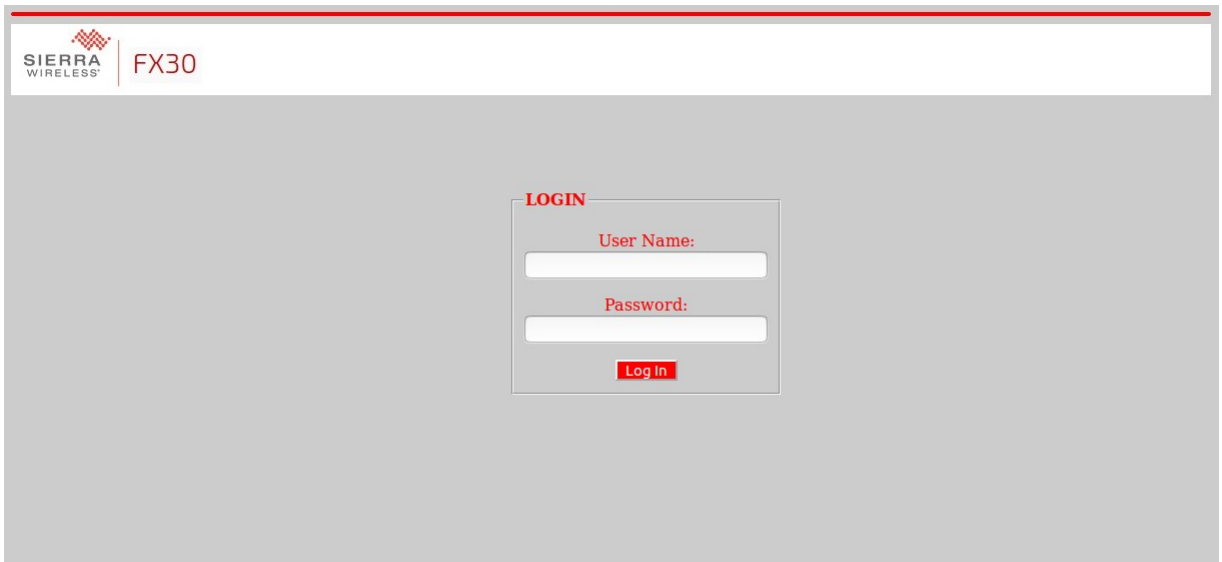
```
$ /etc/init.d/dnsmasq stop
$ /etc/init.d/dnsmasq start
```

6) Start the webserver:

from the FX30 shell, /mnt/flash/webserver directory, issue: “node server.js”

7) plug an Ethernet line between the FX30 and your machine.

From your browser go to http://192.168.13.31 and you will see following page displaid:




For demo purposes and not creating a too complicated demo, the java script after having entered the correct user and pwd simply displays a pdf file. For the script to be able to do so you will need to copy

the pdf file onto the FX30 (as per this example and as coded in the js script in appendix, the pdf file is the present app note). Issue following command:
scp <path_to_the_appnote/Yocto_nodejs_AppNote.pdf> root@192.168.2.2:/mnt/flash/webserver
Beware pdf file name in the script must match with the exact name of the file you copy. Do modify the script (section 8.2) line 48 if you load a different pdf file.

# 7. Industrialization

In the previous sections we saw how to add packages to the kernel and how to implement an application onto a given unit. This is good when in development, but for production one will need a single binary containing everything, including the custom application integrated in the rootfs. Not to be omitted, the different firewall rules matching the customer use case will have to be properly configured and eventually removing license constraining applications or services that can/could be dragged along installation of packages. We will see this all in the present section.

## 7.1. Configure the firewall rules

As mentioned earlier Legato Linux distribution is delivered from factory with most interfaces closed. Before generating a production image for deployment, we shall think of which interface shall be opened, for what type of traffic and over which port. For example, one might want for the Ethernet interface to open ssh access, set the unit to be dhcp server and as per our demo need, open tcp port 80. Let's add those 3 items in the appropriate files so that this configuration is added upon build of the root fs image.

Ideally one should add one's own firewall rules on top of the default ones (Columbia-x meta-layer), using one's own meta-layer and one's own recipe or patch. For simplicity'sake, and in this subsection only, we will add those directly in the existing Columbia-x layer.

Let's go to following directory: yocto/meta-columbia-x/rootfs/recipes/init-ifupdown/files and open the iptables.rules file. Before the DROP rule do add the 3 following rules:
-A INPUT -i eth0 -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT
-A INPUT -i eth0 -p udp --dport 67:68 -j ACCEPT
Save and close: ESC:wq!
The first rule allows http traffic, the second rule allows ssh traffic and the third one allows dhcp traffic. (note in this case where we edit the iptables.rules file and add rules before the drop rule, we can use the –A option. This is different from paragraph 6 where we dynamically added rules from the shell, in which case –I option *must* be used).

cd into the main directory and issue make command.
Note: the directory providing the exact image of the target root file system (root fs) created after compilation:
build_bin/tmp/work/swi_mdm9x15-poky-linux-gnueabi/mdm9x15-image-minimal/1.0-r0.0/rootfs/

In order for the FX30 image to have the dhcp server set on the ethernet interface we will need to create a recipe for this. Let's move to next section and we will see how to do this along with the other recipe that will allow us to load our demo app in the image.

## 7.2. Create a custom meta-layer to integrate a custom app

### 7.2.1. Create the layer

On your Linux machine cd to the yocto directory (the one created upon unzipping the tar source archive). Note the different meta-layers. Each of them adds specific configuration on top of the Linux distribution. For example, “meta-columbia-x” is the layer where is defined all the FX30 specific configuration. From the drivers to Ethernet interface configuration, LED behaviour and so on.

We must now create our own meta-layer. This will segregate what we add on top of the minimal image and allow better readability and tracking in the versioning as well as prevent mixing up custom configuration from the native configuration.

- cd into your main directory and type « make dev »

  This will load the special commands allowing creating a meta layer. Then cd to the parent folder where all other meta layers are available (typically meta-mangoh or meta-columbia-x), as the make dev command will set you into the build-bin directory. Do not issue those commands as root but as standard user, otherwise all the directories and files will be locked.

- Issue yocto-layer create <name of my custom layer>

  Do not add “meta layer” in the name of your layer. This will be added automatically. Do not call it explicitly after a given recipe, it must be more generic, as this layer will potentially contain many different recipes. For this example we will call the layer “custom”, so issue command “yocto-layer create custom”.
  Do set 10 as priority.
  Do answer no (leave default) when asked for creating an example recipe.
  Do answer no (leave default) when asked for creating an example bbappend file.
  See below screenshot.

- Issue exit to quit the special command mode.

$ cd <main *folder*>
$ make dev
$ cd ..
$ yocto-layer create custom → set a high priority if you are building and already existing service and you want yours to be executed in place of the standard one. Some other recipes, providing the same packages, could already exist (e.g. inside poky/or meta-openembedded/folder). Priority value must be the highest for the current recipe to be used (override policy). To be on the safe side do set 10 as priority. This will prevent that upon build your recipe is not considered (most other recipes have a priority of 6)
$ exit

make dev command will let you enter the yocto command line mode. You will enter a 2nd shell with access to special yocto commands like bitbake or create custom meta layers. Issue exit command to come back to initial terminal shell.

```
nmp@nmp-ThinkPad-T400:~/Documents/Yocto_FX30_nodejs/yocto/build_bin$ cd ..
nmp@nmp-ThinkPad-T400:~/Documents/Yocto_FX30_nodejs/yocto$ yocto-layer create custom
Please enter the layer priority you'd like to use for the layer: [default: 6] 10
Would you like to have an example recipe created? (y/n) [default: n]
Please enter the name you'd like to use for your example recipe: [default: example]
Would you like to have an example bbappend file created? (y/n) [default: n]

New layer created in meta-custom.

Don't forget to add it to your BBLAYERS (for details see meta-custom\README).
nmp@nmp-ThinkPad-T400:~/Documents/Yocto_FX30_nodejs/yocto$ exit
exit
nmp@nmp-ThinkPad-T400:~/Documents/Yocto_FX30_nodejs/yocto$
```

## 7.2.2. Modify the Linux image build scripts

Now that we have created our own layer, we must make the build scripts aware of it.

### 7.2.2.1. Modify the build.sh

Go into the yocto/meta-columbia-x directory. Beware to work on the build.sh from the meta-columbia-x meta layer and not on the one from other layers. Edit the build.sh file:
$ cd meta-columbia-x
$ vi build.sh

Add **those 4 modifications** to the build.sh:

**1°**
**in the Global section add:** (Beware always capital C)
-C <custom meta layer>

```
Usage:
$0 <options ...>

  Global:
    -p <poky_dir>
    -o <meta-openembedded dir>
    -l <SWI meta layer>
    -C <custom meta layer>
    -x <linux repo directory>
    -m <SWI machine type>
    -b <build_dir>
    -t <number of bitbake tasks>
    -j <number of make threads>
    -r (enable preempt-rt kernel <Test-only. Not supported>)
    -g (enable Legato setup and build Legato images)
    -a (pass extra options for recipes, key=value separated by ::)
```

**2°**
**in the while getopts line add:**
:C

```
while getopts ":p:o:b:l:f:C:x:m:t:j:w:v:a:F:McdrqsgkhzV" arg
```

**3°**
**In the while getopts body add:**

```
C)
    CUSTOM=$(readlink -f $OPTARG)
    echo "CUSTOM meta dir: $CUSTOM"
    ;;
```

```
    l)
        SWI=$(readlink -f $OPTARG)
        echo "SWI meta dir: $SWI"
        ;;

    C)
        CUSTOM=$(readlink -f $OPTARG)
        echo "CUSTOM meta dir: $CUSTOM"
        ;;

    x)
        LINUXDIR=$(readlink -f $OPTARG)
        echo "Linux repo dir: $LINUXDIR"
        ;;
```

**4°**
**In the enable meta layer section add:**

```
# Enable meta-custom layer
enable_layer "meta-custom" $CUSTOM
```

```
# Enable the meta-networking layer
enable_layer "meta-networking" "$OE/meta-networking"

# Enable the meta-python layer
enable_layer "meta-python" "$OE/meta-python"

# Enable the meta-custom layer
enable_layer "meta-custom" $CUSTOM

# Enable proprietary layers: from sources
if [ $ENABLE_PROPRIETARY_SRC = true ]; then
```

Beware it is always capital C!
Save and close: ESC ZZ.

### 7.2.2.2. Modify the external.mk.

Still into the yocto/meta-columbia-x directory. Edit the external.mk file:
$ vi external.mk

Add the following line in the common_args: -C meta-custom \

```
COMMON_ARGS := ${BUILD_SCRIPT} \
                               -p poky/ \
                               -o meta-openembedded/ \
                               -l meta-swi \
                               -C meta-custom \
                               -x "kernel/.git" \
                               -j $(NUM_THREADS) \
                               -t $(NUM_THREADS) \
```

Save and close: ESC ZZ.

### 7.2.2.3. Append recipe(s) to the image file

In previous steps we have just configured the meta-layer and made sure it is taken into account at build. Now we will have to specify which of the recipes from our meta-layer we want to be taken into account.
A Meta-layer can contain lot of recipes. Not all of them are supposed to end up in the final image. One user might need one recipe but not another user this same one. For the build to take a recipe into account it has to be specified explicitly.
For recipes of our meta-layer to be taken into account we will simply tell the compiler to add them to the minimal image file (see section 3, default build aka factory image).
Doing so we do not modify any process of the minimal image creation and only add on top of it through our own meta layer.
So we are going to create a *bbappend file referring to the minimal image bb file (factory image).
This has to follow precise naming for the directories in our own meta layer, namely we will have to follow the same directory naming as for the minimal image file.
In order to find out those names cd into the main directory (typically FX30_WP8545_full_R14.0.4.002 on the FX30 3G R14 sdk) issue following command from the shell:
$ find meta-swi* -name mdm*.bb
The output provides a variety of hints depending on the Qualcomm chipset. And inside a same chipset family there are multiple images, being development image, test image or minimal image. Identify the hind matching your Qualcomm chipset (mdm9x15 for FX30 3G WP85) and the minimal image:
…
meta-swi/ meta-swi-mdm9x15/recipes-core/images/mdm9x15-image-minimal.bb
…
This is the file based on which the factory image is created.
Beware to always follow this method to identify it as from one release to the other or one platform to the other this file might be located somewhere else.

Now replicate in your custom meta layer -only - the /recipes-core/images/ folders and in the /images folder create a bbappend file called after the minimal image bb file.
$ cd meta-custom
$ mkdir recipes-core
$ cd recipes-core
$ mkdir images
$ cd images
$ touch mdm9x15-image-minimal.bbappend

Edit the mdm9x15-image-minimal.bbappend file and into it add following:

# recipe for adding nodejs application files to the rootfs
IMAGE_INSTALL += "nodejsapp"
Save and close: ESC:wq!
(anticipating the next paragraph and our recipe being called “nodejsapp”)

Beware to not mess up at this point.
Forgetting the "+" and writing only "=" instead of "+=" will result in overwriting the minimal image bitbake file instead of appending to it, and lead to compilation error.

Note starting R17 FW onwards the new syntax will be:
IMAGE_INSTALL_APPEND = " nodejsapp"
(note the append and the SPACE after the double quote)

Still in the mdm9x15-image-minimal.bbappend file, add the following install line in order to have a DHCP server running on the ethernet interface:
# recipe to set ethernet interface to be dhcp server
IMAGE_INSTALL += "dnsmasq"

## 7.2.3. Create and bake recipe(s)

### 7.2.3.1. Create recipe(s)

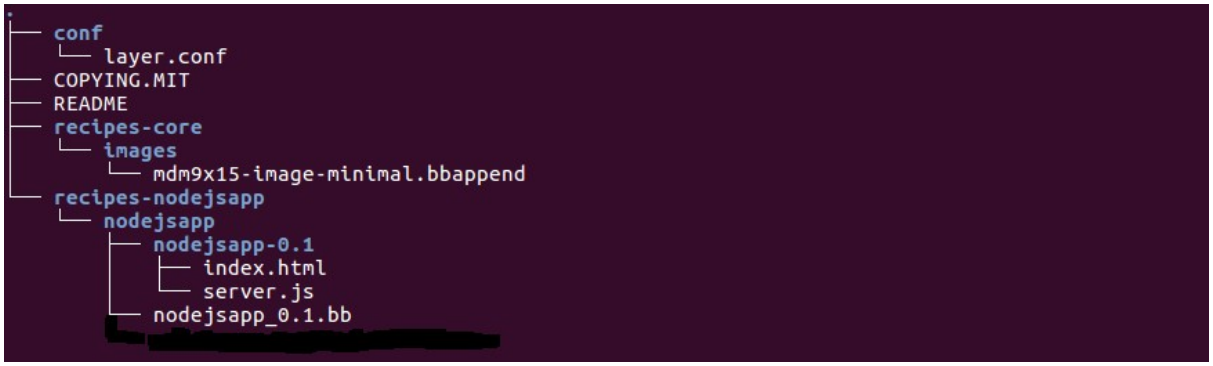
In this section we are going to create 2 recipes. One that will install some files in the minimal image rootfs (nodejsapp recipe). Another that will set the ethernet interface to be dhcp server (dnsmasq recipe).
If you want an example of a recipe compiling a file and installing the corresponding binary in the rootfs, just do section 7.2.1 again but this time mention “yes” when prompted for a recipe sample.

a) nodejsapp recipe

Let’s create the actual recipe directory. At the end of previous section we saw the recipe will be called “nodejsapp”. In the meta-custom layer directory issue:
$ mkdir recipes-nodejsapp
$ cd recipes-nodejsapp
$ mkdir nodejsapp
$ cd nodejsapp
$ touch nodejsapp_0.1.bb
$ mkdir nodejsapp-0.1

The nodejsapp-0.1 directory will contain data to be processed according to instructions in the example_0.1.bb file. Eg C file to compile, or script file (.sh or .js or …) to be copied into a given rootfs directory at build or …
In our case we will add the server.js file and index.html file of our application developed in section 6: do copy index.html and server.js files into nodejsapp-0.1 directory.

Beware of the names and syntax when creating your recipe.
The recipe name **must** remain the same throughout the tree (files and directories of our recipe):
$ cd meta-custom
$ tree




The version of the bitbake file (nodejsapp_0.1.bb) must be added in the bb file name using an underscore "_".
The version number in the directory containing recipe files (nodejsapp-0.1 containing index.html and server.js) must be added using a hyphen "-" and must be called after the exact recipe name.
Note it is not mandatory to add a version number to this directory. In this case its files will be called regardless of the bb file version.
If you call the bb file with a given version and the directory with another version, the build will not look into the directory when compiling the recipe and bitbake will fail.
If one works with a bbappend (not in the present example) and want your bbappend to be called for any version of its corresponding bb file, do set % as version number.
Finally do NOT put underscore "_" in the name of your recipe. This will make the build fail. Underscore is reserved for separation between recipe name and recipe version.
This is yocto ruling and must be respected for proper compilation.

Now we must define what the recipe actually does.

This will be done in the nodejsapp.bb file. See appendix 8.3 for the file content. Copy and paste it into the nodejsapp_0.1bb file. Some information on the content:

- You can add as many files as you like. Just do put them all in the nodejsapp-0.1 directory and you add as many SRC_URI += lines as needed as well as FILES_${PN} += lines as needed. Here again do not mess up and beware the +=. No just a = sign otherwise only just the one file will be used for build and this will generate errors. Also add as many install lines as necessary.
- If you like to create a directory just mention its path within the individual install lines and it will be create upon build. This is the case in our example for the webserver directory which is not a native directory but one created as part of our recipe.
- Beware the license information. In our case there is no proprietary nor licensed file or library dragged in our recipe. If it happens to be the case, modify this line accordingly. Depending on the license type the compiler can stop. Especially when dragging libraries or files referring to a commercial service. In this case after clearing the commercial terms you can add in the custom layer conf file (/meta-custom/conf/layer.conf) the name of the service to the whitelist.
  Eg for a mpeg decoder:
  #add mpg123 in whitelist:

LICENSE_FLAGS_WHITELIST = "commercial_mpg123"

b) dnsmasq recipe

similarly, as the dnsmasq service is an already existing service from the meta-swi layer, we will create (part of) the same directory structure at the root of our custom meta layer:

$ cd meta-custom
$ mkdir recipes-support
$ cd recipes-support
$ touch dnsmasq_%.bbappend
$ mkdir dnsmasq
$ cd dnsmasq
$ vi dnsmasq.eth.conf
dhcp-range=interface:eth0,192.168.13.200,192.168.13.215,12h

dhcp-option=6,8.8.4.4
save and close:
ESC:wq!
Note here we have create file dnsmasq.eth.conf in /meta-custom/recipes-support/dnsmasq/dnsmasq, the file contains the following DHCP server configuration for ethernet interface:
dhcp-range=interface:eth0,192.168.13.200,192.168.13.215,12h
dhcp-option=6,8.8.4.4

Now we must define what our dnsmasq_%.bbappend recipe does. Edit the dnsmasq_%.bbappend file and copy paste the content as from appendix 8.4. This will tell the compiler to add the dnsmasq.eth.conf file to the rootfs under /etc/dnsmasq.d

#### 7.2.3.2. Bake recipe(s)

Enter the dev environment again (make dev)

Make sure you are into build-bin directory and call bitbake nodejsapp

```
$ make dev
$ bitbake nodejsapp
```

The bitbake command works only from build-bin directory. This command will compile the newly created recipe (beware not to add any recipe version number in the recipe name of the bitbake command nor the .bb extension).

Note the recipe is compiled but nothing added to the image. To get the files created in the rootfs we will need to issue the make command from outside the dev environment (exit after make dev command).

The bitbake recipe command is useful to verify the recipe is properly set and running before launching a full make. Do use it with -e | grep option in order to get information on e.g. an environment variable or other.

You can make the same with the dnsmasq recipe as exercise:

```
$ bitbake dnsmasq
```

Once the recipe(s) compile fine, exit the dev environment and from the main directory issue make command again.

```
$ exit
$ make
```

Verify your newly created meta-layer is listed amongst the different meta-layers used for the build (in the shell log).

When make is finished, check for the “webserver” directory and server.js as well as index.html files under /usr/share in build_bin/tmp/work/swi_mdm9x15-poky-linux-gnueabi/mdm9x15-image-minimal/1.0-r0.0/rootfs/. This directory is the image of the built image rootfs.

Install the newly built boot-yocto_wp85.cwe on the FX30 (see section 5), then cd to /usr/share/webserver and call the server.js script from the shell: $ node server.js

Note:

- The directory specifying all packages and layers which are going to be used for the build: build_bin/tmp/work/armv7a-vfp-neon-poky-linux-gnueabi
  Your own custom layer shall be listed into it.
- The directory providing the exact image of the target root file system (root fs) created after compilation:
  build_bin/tmp/work/swi_mdm9x15-poky-linux-gnueabi/mdm9x15-image-minimal/1.0-r0.0/rootfs/
- in above example we modified the rootfs with our recipe. In case one uses a recipe accessing to the Linux kernel (bbappend recipe or new recipe), the new kernel image will be at:

yocto>build_bin>tmp>work>swi_mdm9x15-poky-linux-gnueabi>linux-yocto>3.14.29+gitAUTOINC**-r0.1>image. Inside the boot directory, file config-3.14.**abb will list all recipes integrated to the Linux kernel.

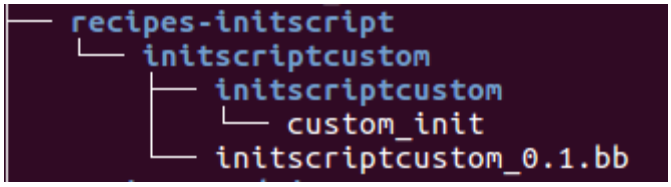
> Sidenote: as a method to know what recipes are integrated into the Kernel and/or what is the Kernel configuration on a given device, one can use following command on the target device: $ zcat /proc/config.gz or $ zcat /proc/config.gz | grep keyword if looking for a particular keyword. The config.gz file will only give information relating to the Linux Kernel. Not to the rootfs. See second bullet of the Note for the rootfs configuration.

### 7.2.4. Set the custom app to auto start

In order for the custom application to automatically start at device boot, we will write a recipe that will call a script, itself calling our application file (server.js)

We will do this in a similar fashion as in section 7.2.3.

Create under meta-custom layer a directory called recipes-initscript. Inside it create a directory called initscriptcustom. Inside it create an initscriptcustom_0.1.bb file and another initscriptcustom directory containing a file called custom_init:




See appendix for content of custom_init file and initscriptcustom.bb file.

In /recipes-core/images/ folder open file mdm9x15-image-minimal.bbappend.
Add this recipe to the list of recipes to bake upon image build:
#recipe to start the nodejs app at device boot
IMAGE_INSTALL += "initscriptcustom"

From the dev environment issue bitbake initscriptcustom.
Exit and from the yocto environment issue make.

## 7.3. Remove eventual license constraining modules

Look out into following manifest file and check for any license constraining package or application that could have been dragged along installation of your packages:

yocto>build_bin>tmp>deploy>licenses>mdm9x15-image-minimal-swi-mdm9x15-yyyymmddhhmmss>license.manifest

This file provides a list of all packages and corresponding licence versions.

The default distribution (minimal image loaded at factory and installed on units out of the box) will not have any. However, it is possible that adding a given package to customize the rootfs or kernel does import as well other license-constraining packages (eg bash). Do remove them if any, build again.

# 7.4. Remotely install the binary using AirVantage

One way to install the binary can be on production line. FX30 can also be mounted as second mount in the field on an already deployed application. In this case, or during the life cycle of the application, remote upgrade is the approach to prefer. Here is a link to a WP application note explaining how to create the AirVantage package based on the built binary (Do log on to be able to download the app note).

## 7.4.1. Create the AirVantage package

You can directly go to section 8.1.3 of the WP application note: “Bundle: Combined Package”.
In our case we will not create a bundle as such (we only have one single .cwe file) but steps are the same.

Create a blank folder, name it as you wish. Inside this folder create a blank file with .app extension. This will be the application model file.
Beware though to the type in your application model. FX30 has its own type, so do not set “WP8545” but do set “FX30” instead. If you leave the “WP8545” type or any other type, you will not be proposed your custom package when creating your install job on the platform.

- Do set your own name. Name will be the name displaid in the different menus of the AirVantage platform.
- Do set your own RFS revision. Beware the RFS revision is not something you have to populate randomly. The Root File System revision must be the RFS answer you get to the “ATI8” AT command. Upon build of your own custom Linux distribution, the custom RFS will be tagged with a unique identifier. Upon an install job, the platform will check what is supposed to be installed on the device as a RFS version (from what has been populated in the application model file) and what is the information remotely provided by the FX30 after the installation job. If those do not match, the install job will be marked as failed on the platform side (even if locally the installation on the FX30 succeeded).
- Do populate as well the exact binary file name of your .cwe file. Typically, it will be “yocto_wp85.cwe”.

In the present example, we only modify the RFS, so we do not use neither Little Kernel (bootloader) version nor OS version nor Legato version. In case you modify any of them make sure you provide in your application model file in the revision parameter, the corresponding revisions of each of those components as well. They will be tagged uniquely as for the RFS and the unique revision will be available from the ATI8 command. Syntax example for the revision parameter would be in this case: revision=”LK=<LKversionasperATI8answer>,RFS=<RFSversionasperATI8answer>,LE=<LEversionasperATI8answer>”.

Then save and paste your custom “yocto_wp85.cwe” file in the above created folder, next to the .app file. Do zip both and load the zip file on AirVantage (into the Develop menu go to the blue “Release” arrow and browse to the zip file. You will need once released to publish it using the publish button, unless you have selected the publish action in the release>advance menu).
See appendix for the FX30 application model example.

Note that prior to release and publish your custom package, you will need to create your test account on AirVantage. Follow this link to do so or contact your Sierra Wireless representative.

Note as well that the combination [name and revision] provided in the application model must be unique in the whole AirVantage platform. Name of the zip file is not a differentiator. Meaning giving a unique name to the zip file will not prevent you from the platform rejecting loading the zip file.
Also providing a generic name (eg test) and a generic revision (eg 1.0) has high chances that a custom package already exists with this name and revision and release operation will fail.

### 7.4.2. Deploy

Once the custom package is available on AirVantage (released and published) you can remotely deploy it. For the operation to succeed do follow precisely those 4 steps:

- Select “install bundle” action from the “more” button in the AirVantage GUI. Do not use install FW button.

- Do set time on the FX30. For AirVantage upgrades to work, time on the FX30 must be set through the modem service (and only through the modem service). Per AT command do use: AT!TIME=yyyy,mm,dd,hh,mm,ss. To access AT command from the FX30 shell, send microcom -E /dev/ttyAT, then ctrl-X to exit AT command mode. Beware there is no battery on the FX30 so in case of power loss (or if during tests you remove power) you will need to reset time. It is advised that you foresee a mean to synch time on your FX30 upon device boot, eg retrieve time per sntp and set it using above AT command with a script, a custom application, a Legato application (in this later case you can also use time API’s) …

- It is mandatory that the FW names are aligned between AirVantage (what is declared to be the running FW on your FX30) and your FX30 (the last line of the answer to “fwupdate query” command from the FX30 shell). And this for the current FW version, not for the target version you plan to install. A verification is made at install process start and if the current FW declared on AirVantage and the current FW installed on your FX30 are not exactly matching the update operation will fail. So, either synch on AirVantage side or in case of dev/tests manually install on your unit the FW as declared on AirVantage side, then only trigger your update operation.

- In case of development or test you can manually trigger a connection to the AirVantage platform by sending command AT+WDSS=1,1. It is advised that prior to this you do activate the +WDSI unsolicited indications. Those provide precious information about the different stages or states of the connection between the FX30 and the AirVantage platform.
  The WP AT command guide can be found here, you will find in section 10 “AirVantage commands” the +WDSI command syntax and unsolicited response values.
  In case you have omitted eg to set time on the unit and you have already started the manual connection to the platform and see nothing happens, you might want to stop the session by sending command AT+WDSS=1,0, send the AT!TIME command, and trigger again the connection by sending AT+WDSS=1,1.

# 8. Appendix

## 8.1. Node js webserver script

```
var http = require('http')
var url = require('url')
var querystring = require('querystring')
var fs = require('fs')
//var express = require ('express');
//var app = express ();
const port = 80

http.createServer(function(request, response) {
console.log(request.url)
var params = querystring.parse(url.parse(request.url).query);
if(request.url == "/")
{
  fs.readFile("index.html", function (err, data) {
     response.writeHead(200, {'Content-Type': 'text/html'});
     response.write(data);
     response.end();
  });
}
else if(request.url == "/logoFX30.jpg")
{
 var img = fs.readFileSync('/usr/share/webserver/logoFX30.jpg');
    response.writeHead(200, {'Content-Type': 'image/jpg' });
    response.end(img, 'binary');
}
else if ('user' in params && 'pwd' in params)
{
response.writeHead(200, {"Content-Type": "text/plain"});
response.write('user et pwd: ' + params['user'] + ' ' + params['pwd']);
}
else if (request.method == 'POST')
{
 var body = '';
       request.on('data', function (data) {
```

```
            body += data;
            // Too much POST data, kill the connection!
            // 1e6 === 1 * Math.pow(10, 6) === 1 * 1000000 ~~~ 1MB
            if (body.length > 1e6)
                request.connection.destroy();
        });
        request.on('end', function () {
                var post = querystring.parse(body);
                console.log(post.user)
                console.log(post.pwd)
                        if (post.pwd == '12345')
                                {
                                //console.log("correct pwd")
                                var pdf = fs.readFileSync('/usr/share/webserver/Yocto_nodejs_AppNote.pdf');
                                response.writeHead(200, {'Content-Type': 'application/pdf' });
                                response.end(pdf, 'binary');
                                }
        });
}
}).listen(port);
```

## 8.2. HTML page loaded by nodejs webserver

```
<!DOCTYPE html><html>
<head>
      <meta charset="utf-8" /><title>FX30 Admin page</title>
      <style>
            .button {background-color: #FF0000; color: white;}
            fieldset { margin-left: 500px; margin-right: 500px;}
             div {background-color: #FFFFFF;  width 2000px; height100px;}
     </style>
</head>
<body bgcolor="cccccc">
<hr style="border-color: #FF0000;" size="4" noshade>
<div>
<img src="/logoFX30.jpg"/>
<font color="cccccc">
</div>
```

```
<br/><br/><br/><br/><br/>
<form method="post" action="/login">
    <fieldset><legend><center><font color="#FF0000"><b>LOGIN</b></center></legend>
    <p><font face="verdana"><center><label>User Name</label><input type="text" name="user"
value="user"/></center></font
    <p><font face="verdana"><center><label>Password</label><input type="text"
name="pwd"/></center></font></p>
    <center><input type="submit" class="button" value="Log In"></center>
    </fieldset>
</form>
</body>
</html>
```

## 8.3. Recipe content for sample recipe “nodejsapp”

```
DESCRIPTION = "Adding application files into /usr/share/webserver directory"
LICENSE = "CLOSED"
PR = "r0"

SRC_URI += "file://server.js"
SRC_URI += "file://index.html"

do_install_append() {
        install -d ${D}/usr/share/webserver
        install -m 0755 ${WORKDIR}/server.js ${D}/usr/share/webserver/
        install -m 0755 ${WORKDIR}/index.html ${D}/usr/share/webserver/
}

FILES_${PN} += "/usr/share/webserver/server.js"
FILES_${PN} += "/usr/share/webserver/index.html"
```

## 8.4. Recipe content for sample recipe “nodejsapp”

```
DESCRIPTION = "Adding dnsmasq DHCP management for eth interface"
LICENSE = "CLOSED"
PR = "r0"

FILESEXTRAPATHS_prepend := "${THISDIR}/dnsmasq:"

SRC_URI += "file://dnsmasq.eth.conf"

do_install_append() {
        install -d ${D}/etc/dnsmasq.d
        install -m 0755 ${WORKDIR}/dnsmasq.eth.conf ${D}/etc/dnsmasq.d/
}

FILES_${PN} += "/etc/dnsmasq.d/dnsmasq.eth.conf"
```

## 8.5. Recipe content for application auto start

```
inherit update-rc.d

LICENSE = "Apache-2.0"
LIC_FILES_CHKSUM = "file://${COREBASE}/meta/files/common-licenses/Apache-2.0;md5=89aea4e17d99a7cacdbeed46a0096b10"
PR = "r0"

SRC_URI = "file://custom_init"
# one can add other script files by adding  \ and additionnal scripts at next line. Terminate with ".

# init scripts are called in alphabetical order, uper case first.
# depending on when you want / need your script to be executed you might pick a particular naming
# for instance script might need to start after eth interface is mounted or ...
# similarly scripts with same name priority (starting with same letter) are discriminated against their priority number
# see /etc/rcS.d for a list of init scripts
# in below example we choose to start our script with S 97 order of priority (almost at the end of all scripts)
INITSCRIPT_NAME = "custom_init"
INITSCRIPT_PARAMS = "start 97 S ."
```

```
# mandatory to let the do_compile section and let it emtpy
do_compile() {
}

do_install() {
    install -m 0755 ${WORKDIR}/custom_init -D ${D}${sysconfdir}/init.d/custom_init
    # install script in /etc/init.d directory in order for rc.d class to find it
}
```

## 8.6. custom_init file content

```
#!/bin/sh
DESC="starting custom meta layer application(s)"

#set -e

case "$1" in
    start)
        cd /usr/share/webserver
          /usr/bin/node server.js &
          # add & to run node server.js in the background
          export PID_NODE_SERVERJS=$!
          # retrieve and store pid of server.js in order to kill process when stop command is called (see
below)
        ;;
    stop)
          # to stop the node server.js script one must kill the corresponding process
          kill -9 $PID_NODE_SERVERJS
          ;;
esac

exit 0
```

## 8.7. Useful tools

- swi-cwe

This tool allows to concatenate different cwe files (eg custom Legato application and custom Legato Linux distribution. In this case one can skip integrating one's own legato app at Legato Linux distribution build and instead, simply concatenate the Legato Linux binary along with the Legato app binary)

- swiflash

This tool allows to flash an image (cwe or spk) onto the device.

- Command to flash an image:

cat <path to the yocto_wp85.cwe image> | ssh root@192.168.2.2 /legato/systems/current/bin/fwupdate download –

Do not forget the minus at the end of the command.

## 8.8. Image for the webserver demo page


## 8.9. FX30 AirVantage application model sample

```
<?xml version="1.0" encoding="UTF-8"?>
<app:application
xmlns:app="http://www.sierrawireless.com/airvantage/application/1.0"
type="FX30"
name="nmp_AV_customtestpackageFX30_R13.1.3"
revision="RFS=SWI9X15Y_07.11.21.00 2017-10-11_11:05:07">
<binaries>
<binary file="boot-yocto_wp85.cwe"/>
</binaries>
<application-manager use="LWM2M_AIRPRIME_BUNDLE" />
</app:application>
```