AGL/Phase2 - Devkit

How to bake a low level service

Version 1.1

March 2016

Abstract

System DevKit allows developers to rebuild a complete bootable image from source code. It uses Yocto/Pocky version 2.x with latest version of Renesas BSP and enables low level development as driver, system services.

System DevKit is composed of:

- A Docker container with AGL distribution preconfigured on Yocto 2.x.
- A documentation guide on how to build from scratch an image for Porter
- A documentation guide on how to bake a new low level system service

This document focuses on the procedure to add a new low level system service to the target firmware image. More specially, it highlights the integration of the system daemon named "hostapd" in the Yocto image built within the Devkit container.

Document revisions

Date	Version	Designation	Author
15 Feb. 2016	0.1	Initial release	Y. Gicquel [Iot.bzh]
24 Feb. 2016	0.2	Rework after internal review	Y. Gicquel [Iot.bzh]
25 Feb 2016	0.3	Review, add packagegroup section	Y. Gicquel [Iot.bzh] J. Bolo [Iot.bzh]
29 Feb 2015	1.0	Final review	S. Desneux [iot.bzh]
29 Mar 2016	1.1	Public version	S. Desneux [iot.bzh]

Table of contents

1. Introduction

1.1. Document scope

This document details the required steps to add a new low-level service to an AGL image. More specially, it highlights the proposed workflow by integrating "hostapd" daemon to a Porter AGL Yocto image.

This system service provides a pluggable Wifi network interface to the AGL Access Point. For the illustration, this example will use a USB to Wifi dongle adapter plugged into a Renesas Porter board, and will use the Devkit container to integrate it into our firmware.

The intent of this document is to propose a workflow on how system developer and integrator can handle this kind of activities using the Devkit container.

1.2. Prerequisites

The procedures proposed in this document assume that the user already has a Devkit container installed and configured on his computer.

More particularly, we assume that the user is able to start the container, to connect to it, to master the Yocto environment allowing the build of a new image, to deploy a SD-Card image on a Porter board.

All these steps are covered with more details in the document named: "AGL_Phase2-Devkit-Image_for_porter". Please refer to it if needed.

In the next sections, it is assumed that commands are issued in the openembedded environment of the container. For reminder, the following sequence of command should be done. From your host SSH client, log into the container shell and set the Yocto build environment:

```
$ ssh -p 2222 devel@localhost
devel@localhost's password: devel

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Feb 12 11:28:26 2016 from pollux.ygl.iot
devel@agl-porter-bsp:~$ prepare_meta -o /xdt -t porter -f iotbzh -l /home/de-
vel/mirror/ -p /home/devel/mirror/proprietary-renesas-r-car/
[…]
=== setup build for porter
```

```
Using proprietary Renesas drivers for target porter
=== conf: build.conf
=== conf: download caches
=== conf: sstate caches
=== conf: local.conf
=== conf: bblayers.conf.inc -> bblayers.conf
=== conf: porter_bblayers.conf.inc -> bblayers.conf
=== conf: bblayers_proprietary.conf.inc is empty
=== conf: porter_bblayers_proprietary.conf.inc is empty
=== conf: local.conf.inc -> local.conf
=== conf: porter_local.conf.inc is empty
=== conf: local_proprietary.conf.inc is empty
=== conf: porter_local_proprietary.conf.inc -> local.conf
==========================================================================

Build environment is ready. To use it, run:

# source /xdt/meta/poky/oe-init-build-env /xdt/build

then

# bitbake agl-demo-platform


devel@agl-porter-bsp:~$ source /xdt/meta/poky/oe-init-build-env /xdt/build
### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common target are:
    agl-demo-platform
```

1.3. A dedicated Yocto layer for your product

In this document we describe a workflow which can illustrate a use case where a vendor is preparing a future product based on AGL distribution.

Yocto brings out a powerful concept of “layers” for the description of complex systems. This concept has a direct impact on the file-system organization – especially on meta-data – and on the package recipes themselves. It allows the integrators to easily inherit subsets of generic meta-data and to make them consistent regarding a particular project scope.

Using a layer for a product will simplify its maintenance along the time, and isolates all its specificities in a reasonably traceable subset of the whole distribution.

2. Functional integration of *hostapd* service

In this section, we focus on the functional integration of the service, from the upstream project retrieval to the first run on the Porter board.

2.1. Prepare a new target image

2.1.1. Locate *hostapd* in existing layers

Before packaging a service, advised users should first have a look in common recipes repositories to check if the service is already available and ready to be built. When the relevant service is already packaged, the time required to include it on a particular project is significantly reduced.

In this scope, we will check for the presence of *hostapd*'s recipe inside AGL meta-data. In the latest snapshot as of Feb. 2016, AGL is using the following layers (all referenced in the "`bblayers.conf`" configuration file).

```
devel@agl-porter-bsp:/xdt/build$ bitbake-layers show-layers
layer                 path                                      priority
==========================================================================
meta                  /xdt/meta/poky/meta                       5
meta-yocto            /xdt/meta/poky/meta-yocto                 5
meta-yocto-bsp        /xdt/meta/poky/meta-yocto-bsp             5
meta-ivi-common       /xdt/meta/poky/../meta-agl/meta-ivi-common  7
meta-agl              /xdt/meta/poky/../meta-agl/meta-agl       7
meta-oe               /xdt/meta/poky/../meta-openembedded/meta-oe  6
meta-multimedia       /xdt/meta/poky/../meta-openembedded/meta-multimedia  6
meta-ruby             /xdt/meta/poky/../meta-openembedded/meta-ruby  7
meta-efl              /xdt/meta/poky/../meta-openembedded/meta-efl  7
meta-renesas          /xdt/meta/poky/../meta-renesas            5
meta-rcar-gen2        /xdt/meta/poky/../meta-renesas/meta-rcar-gen2  6
meta-agl-demo         /xdt/meta/poky/../meta-agl-demo           7
meta-qt5              /xdt/meta/poky/../meta-qt5                7
meta-security-smack   /xdt/meta/meta-intel-iot-security/meta-security-smack  8
meta-security-framework  /xdt/meta/meta-intel-iot-security/meta-security-framework  6
meta-app-framework    /xdt/meta/meta-iot-agl/meta-app-framework  20
meta-agl-security     /xdt/meta/meta-agl/meta-agl-security      9
```

We can also search for a particular recipe using the "`bitbake-layers`" tools. In our example, looking for all "hostapd" related packages can be done using:

```
devel@agl-porter-bsp:/xdt/build$ bitbake-layers show-recipes | grep -A1 hostapd
Parsing recipes..done.
hostapd:
  meta-oe              2.4
```

Here the tool shows the *hostapd* has already a recipe and the version 2.4 is ready to be used for our project. Thanks to these previous informations, we can locate its recipe using "`find`" in the layer meta-oe:

```
devel@agl-porter-bsp:/xdt/build$ find /xdt/meta/poky/../meta-openembedded/meta-oe \
-name "hostapd*bb*"
/xdt/meta/poky/../meta-openembedded/meta-oe/recipes-connectivity/hostapd/hostapd_2.4.bb
```

2.1.2. Locate recipes on Internet search engines

Alternatively to the local search, recipes can be grabbed using a web browser and looking on specialized repositories. Thus, the OpenEmbedded project references available layers and recipes through a search engine directly on their website:

http://layers.openembedded.org/layerindex/branch/master/layers/

In the same way, you can check the Yocto Project repositories at this location:

http://git.yoctoproject.org/

And more "meta" from open-source community are also available on popular development website such as GitHub for example.

2.1.3. Add *hostapd* to the target image

As we have identified the *hostapd* recipe, we can now add it to the Yocto build. Basically, a package addition requires to register the new recipe in the list of packages for generated target image.

This can be done thanks to the following directive:

```
IMAGE_INSTALL_append = " hostapd"
```

Warning: you should pay attention to the space character inserted at the beginning of the string. It allows a consistent append operation to an already existing IMAGE_INSTALL variable if any. Bitbake also provide the += operator that append and insert space but be aware that it is not supported in all files. At least for `local.conf` files and *IMAGE_INSTALL* variable, the syntax _append must be used.[1]

1 Few links for new bitbake users: http://www.yoctoproject.org/docs/2.0/bitbake-user-manual/bitbake-user-manual.html and http://elinux.org/Bitbake_Cheat_Sheet

A first try out of the new service integration can be easily de-risked by adding the *IMAGE_INSTALL_append* directive in your `local.conf` file in the build directory:

```
devel@agl-porter-bsp:/xdt/build$ vi /xdt/build/conf/local.conf
```

We are using here the provided editor: `vi`. For people that begin with this editor this link http://www.atmos.albany.edu/daes/atmclasses/atm350/vi_cheat_sheet.pdf could be of help. The distributed image also contains a tutorial that is run with the command `vimtutor`.

And for debugging purpose, we will also temporarily add packages providing useful tools:

```
IMAGE_INSTALL_append = " hostapd wireless-tools"
```

Save the file, quit the editor and finally launch a rebuild of the target image:

```
devel@agl-porter-bsp:/xdt/build$ bitbake agl-demo-platform
Parsing recipes: 100% |
##################################################################################| Time:
00:00:28
Parsing of 1862 .bb files complete (0 cached, 1862 parsed). 2399 targets, 368 skipped, 9 masked, 0
errors.
NOTE: Resolving any missing task queue dependencies
NOTE: preferred version 3.10% of nativesdk-linux-libc-headers not available (for item nativesdk-
linux-libc-headers)
NOTE: versions of nativesdk-linux-libc-headers available: 4.1
NOTE: multiple providers are available for runtime media-ctl (media-ctl, v4l-utils)
NOTE: consider defining a PREFERRED_PROVIDER entry to match media-ctl
NOTE: preferred version 3.10% of nativesdk-linux-libc-headers not available (for item nativesdk-
linux-libc-headers-dev)
NOTE: versions of nativesdk-linux-libc-headers available: 4.1
NOTE: multiple providers are available for virtual/mesa (mesa, mesa-gl)
NOTE: consider defining a PREFERRED_PROVIDER entry to match virtual/mesa

Build Configuration:
BB_VERSION        = "1.28.0"
BUILD_SYS         = "x86_64-linux"
NATIVELSBSTRING   = "Debian-8.3"
TARGET_SYS        = "arm-poky-linux-gnueabi"
MACHINE           = "porter"
DISTRO            = "poky-agl"
DISTRO_VERSION    = "1.0+snapshot-20160219"
TUNE_FEATURES     = "arm armv7a vfp neon callconvention-hard cortexa15"
TARGET_FPU        = "vfp-neon"meta
meta-yocto
meta-yocto-bsp    = "iotbzh:5b12268f6e17574999f91628a60e21711cf62ee4"
meta-ivi-common
meta-agl          = "iotbzh:2a50a1ee20d39db50b700e53a550985d69a984a5"
meta-oe
meta-multimedia
meta-ruby
meta-efl          = "iotbzh:7bc138a365e20653ddfb9229561e3e9e50b89ee8"
meta-renesas
meta-rcar-gen2    = "iotbzh:d2cf62fcf5dfb535d7527fbc646b6b115e8d8121"
meta-agl-demo     = "iotbzh:69f78be85976e8dac1c0ca70178a18910e4a04bf"
meta-qt5          = "iotbzh:d5536e34ec985c82b621448ab4325e5cbba38560"
meta-security-smack
meta-security-framework = "iotbzh:6a38543694b0e397d3ffbec5f7cc769b363ef4b4"
meta-app-framework = "iotbzh:8e938554beb1c23234273de6ae962672c56a0561"
meta-agl-security = "iotbzh:2a50a1ee20d39db50b700e53a550985d69a984a5"

NOTE: Preparing RunQueue
```

```
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
Currently 3 running tasks (4868 of 5160):
0: linux-renesas-3.10+gitb8ca8c397343f4233f9f68fc3a5bf8e1c9b88251-r0 do_configure (pid 8609)
1: linux-firmware-1_0.0+gitAUTOINC+75cc3ef8ba-r0 do_fetch (pid 8863)
2: hostapd-2.4-r0 do_compile (pid 9086)

NOTE: Tasks Summary: Attempted 5160 tasks of which 4798 didn't need to be rerun and all succeeded.
```

For this particular package, the build should succeed. Anyway, on some more complex service integrations, this step should highlight errors or missing dependencies.

2.1.4. Handle functional dependencies

Yocto build system handles build dependencies of packages and some runtime dependencies. But, users should keep in mind that runtime dependencies[2] are not all listed in recipes. In this particular case, the Linux kernel must be configured to support Wifi hardware and network related framework in order to gain a well supported wireless network interface for the user-space.

In our example, we established the link with a USB dongle Wifi adapter which uses a RaLink RT2501 chipset: the TP-Link TL-WN321G (official support page)

This particular peripheral requires some specific kernel modules and firmware blob that should be added to the kernel configuration.

As this document focuses on the service integration, we will not go in details in the way to integrate this particular chipset into the BSP layer. By the way, the integration of such a feature to the BSP layer should be quite similar with the service integration we are describing here.

For reference, here are the kernel defconfig required:

```
# Add wifi support
CONFIG_CFG80211 y
CONFIG_CFG80211_WEXT y
CONFIG_MAC80211 y

# Add USB RaLink drivers
CONFIG_USB_USBNET y
CONFIG_RT2X00 y
CONFIG_RT73USB y
```

For the current state of meta-renesas, the configuration of this kernel setting can be done by editing the file `meta-renesas/meta-rcar-gen2/recipes-kernel/linux/linux.inc`.

And here is the binary blob dependency, inserted for convenient in the file `local.conf`:

```
IMAGE_INSTALL_append = " linux-firmware"
```

2: Yocto at some point is designed to propagate some of these dependencies. For more informations, you could refer to IMAGE_FEATURE and DISTRO_FEATURE variables descriptions.

Once again, it is time to launch a new rebuild of the target image and this time we will be quite ready for a real test of the system image on the board:

```
devel@agl-porter-bsp:/xdt/build$ bitbake agl-demo-platform
[snip]
NOTE: Tasks Summary: Attempted 5160 tasks of which 4798 didn't need to be rerun and all
succeeded.
devel@agl-porter-bsp:/xdt/build$ mksdcard /xdt/build/tmp/deploy/images/porter/agl-demo-
platform-porter.tar.bz2 $XDT_WORKSPACE/agl-demo-platform-porter-sdcard.img
Creating the image /xdt/workspace/agl-demo-platform-porter-sdcard.img ...
0+0 records in
0+0 records out
0 bytes (0 B) copied, 6.5007e-05 s, 0.0 kB/s
[sudo] password for devel: devel
mke2fs 1.42.12 (29-Aug-2014)
Discarding device blocks: done
Creating filesystem with 524287 4k blocks and 131072 inodes
Filesystem UUID: 1093b3cf-88dc-4e64-8ac3-7e185024a053
Superblock backups stored on blocks:
      32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

Extracting image tarball...
done
[snip]
devel@agl-porter-bsp:/xdt/build$
```

At this step, the image with the new service is ready to be tested.

2.2. Bring-up the new service on target

Our freshly built service and kernel are now embedded in our custom fresh built AGL image. After “flashing” the SD-Card with this image, let's boot the Porter board and figure out if our previous steps are correctly in place.

2.2.1. Start the new image

With the SD-Card and the USB Wifi dongle inserted, we can power on the board and finally reach the login prompt:

```
Automotive Grade Linux 1.0+snapshot-20160210 porter ttySC6

porter login: root
Last login: Wed Feb 10 19:24:23 UTC 2016 on tty2
root@porter:~#
```

Checking at the kernel logs, we can check whether the Wifi adapter is correctly recognised:

```
root@porter:~# dmesg | grep 802
[   13.113162] cfg80211: Calling CRDA to update world regulatory domain
[   14.923097] ieee80211 phy0: rt2x00_set_chip: Info - Chipset detected - rt:
2573, rf: 0002, rev: 000a
[   15.580106] ieee80211 phy0: Selected rate control algorithm 'minstrel_ht'
```

And so does the wireless interface:

```
root@porter:~# iwconfig
wlP4p1s2u1u2  IEEE 802.11bg  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=0 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:on

sit0      no wireless extensions.

lo        no wireless extensions.

eth0      no wireless extensions.

can0      no wireless extensions.

root@porter:~#
```

2.2.2. Configure the service on target

As the drivers and network stack are available, we can now configure the *hostapd* daemon. In addition to the wireless interface, *hostapd* requires to setup a network configuration to forward IP packets between the Access Point and Internet.

First, we edit the new wireless network file:

```
root@porter:~# vi /etc/systemd/network/wireless.network
[Match]
Name=wl*

[Network]
Address=192.168.1.1/24
DHCPServer=yes
```

Note: we should use an IP address out of the DHCP range from the wired interface.

Secondly, we configure *hostapd* daemon to create an access point with SSID "test" secured with WPA2:

```
root@porter:~# vi /etc/hostapd.conf
ssid=test
wpa_passphrase=Somepassphrase
interface=wlP4p1s2u1u2
auth_algs=3
channel=7
driver=nl80211
hw_mode=g
logger_stdout=-1
logger_stdout_level=2
max_num_sta=5
rsn_pairwise=CCMP
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP CCMP
```

Then, we enable the systemd service to start *hostapd* automatically at boot:

```
root@porter:~# systemctl enable hostapd.service
Created symlink from /etc/systemd/system/multi-user.target.wants/hostapd.ser-
vice to /lib/systemd/system/hostapd.service.
```

2.2.3. Verify new service operation

We can verify that the whole configuration is working by rebooting the system, or step by step by:

1. Restarting the service:

```
root@porter:~# systemctl restart systemd-networkd
ieee80211 phy0: rt2x00lib_request_firmware: Info - Loading firmware file 'rt73.bin'
ieee80211 phy0: rt2x00lib_request_firmware: Info - Firmware detected - version: 1.7
IPv6: ADDRCONF(NETDEV_UP): wlP4p1s2u1u2: link is not ready
IPv6: ADDRCONF(NETDEV_CHANGE): wlP4p1s2u1u2: link becomes ready
root@porter:~# systemctl restart hostapd
```

2. Checking wireless link:

```
root@porter:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: can0: <NOARP,ECHO> mtu 16 qdisc noop state DOWN group default qlen 10
    link/can
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group de-
fault qlen 1000
    link/ether 2e:09:0a:00:87:e5 brd ff:ff:ff:ff:ff:ff
    inet 10.20.105.41/24 brd 10.20.105.255 scope global dynamic eth0
       valid_lft 86337sec preferred_lft 86337sec
```

```
    inet6 fe80::2c09:aff:fe00:87e5/64 scope link
       valid_lft forever preferred_lft forever
4: sit0: <NOARP> mtu 1480 qdisc noop state DOWN group default
    link/sit 0.0.0.0 brd 0.0.0.0
5: wlP4p1s2u1u2: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group
default qlen 1000
    link/ether 00:27:19:be:eb:b3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global wlP4p1s2u1u2
       valid_lft forever preferred_lft forever
root@porter:~# iwconfig
wlP4p1s2u1u2  IEEE 802.11bg  Mode:Master  Tx-Power=20 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Power Management:on
```

At this time, the access point should be up and running. After a client connects, you can check its service logs:

```
root@porter:~# systemctl -l status hostapd
● hostapd.service - Hostapd IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authentic-
ator
   Loaded: loaded (/lib/systemd/system/hostapd.service; disabled; vendor preset: en-
abled)
   Active: active (running) since Fri 2016-02-19 12:34:23 UTC; 8min ago
  Process: 652 ExecStart=/usr/sbin/hostapd /etc/hostapd.conf -P /run/hostapd.pid -B
(code=exited, status=0/SUCCESS)
 Main PID: 654 (hostapd)
   CGroup: /system.slice/hostapd.service
           └─654 /usr/sbin/hostapd /etc/hostapd.conf -P /run/hostapd.pid -B

Feb 19 12:34:23 porter hostapd[652]: wlP4p1s2u1u2: AP-ENABLED
Feb 19 12:34:23 porter systemd[1]: Started Hostapd IEEE 802.11 AP, IEEE
802.1X/WPA/WPA2/EAP/RADIUS Authenticator.
Feb 19 12:34:53 porter hostapd[654]: wlP4p1s2u1u2: STA b8:76:3f:43:ec:b7 IEEE 802.11:
authenticated
Feb 19 12:34:53 porter hostapd[654]: wlP4p1s2u1u2: STA b8:76:3f:43:ec:b7 IEEE 802.11:
associated (aid 1)
Feb 19 12:34:54 porter hostapd[654]: wlP4p1s2u1u2: STA b8:76:3f:43:ec:b7 RADIUS: start-
ing accounting session 56C70BCF-00000000
Feb 19 12:34:54 porter hostapd[654]: wlP4p1s2u1u2: STA b8:76:3f:43:ec:b7 WPA: pairwise
key handshake completed (RSN)
root@porter:~#
```

2.3. Requirements summary

In summary, below are the topics we covered in the previous sections:

- How to build a new service within the Devkit container,
- We identified the inherent BSP dependencies,
- We delimited the required user-space configurations files on the target,
- We confirmed briefly the functional state of the service.

These steps achieved, in the following section we will details the workflow proposal and prepare a consistent integration in our product layer.

3. Integrate the service in a product layer

3.1. Layer creation

As introduced, we will uses the layer mechanism of Yocto build system to integrate the service.

Let's illustrate this by preparing the system service integration in a sample product layer named "agl-product".

We create the Yocto layer which is named "meta-agl-product" using the following commands:

```
devel@agl-porter-bsp:/xdt/build$ pushd .
devel@agl-porter-bsp:/xdt/build$ cd ../meta
devel@agl-porter-bsp:/xdt/meta$ yocto-layer create agl-product
Please enter the layer priority you'd like to use for the layer: [default: 6]
Would you like to have an example recipe created? (y/n) [default: n]
Would you like to have an example bbappend file created? (y/n) [default: n]

New layer created in meta-agl-product.

Don't forget to add it to your BBLAYERS (for details see meta-agl-
product\README).
devel@agl-porter-bsp:/xdt/meta$ tree meta-agl-product/
meta-agl-product/
|-- COPYING.MIT
|-- README
`-- conf
    `-- layer.conf

1 directory, 3 files
devel@agl-porter-bsp:/xdt/meta$ popd
devel@agl-porter-bsp:/xdt/build$
```

Note: Prefixing directories with "meta-" is a common naming convention when referring to layers in Yocto, and the helper script "`yocto-layer`" automatically prepend this to the given identifier.

The next step is to register this new layer in the current build paths. To do so, we have to add an entry to the variable BBLAYERS in the file "`bblayers.conf`" of the `conf` directory:

```
devel@agl-porter-bsp:/xdt/build$ vi /xdt/build/conf/bblayers.conf
```

The "BBLAYERS" variable lists the registered layer of the current build environment. We can add the new layer location by adding the line in bold:

```
BBLAYERS ?= " \
  /xdt/meta/poky/meta \
[snip]
  /xdt/meta/poky/../meta-qt5 \
  /xdt/meta/poky/../meta-agl-product \
  "
```

Now, the new layer is ready to be populated either with new recipes for building new specific source code, or new extensions to existing recipes using overlaying technics. In section 2.2.2, we identified the configuration files required to start our service:

- A **new** wireless link, related to *systemd* recipe,
- A **modified** *hostapd* daemon configuration file, related to *hostapd* recipe.

3.2. Wireless service

The "wireless.network" file does not exist in the current AGL snapshot. This file would extend the systemd general configuration to include the new wireless interface. Thus, we have to deliver that file as an extension of systemd recipe. This mechanism is handled by Yocto build system using "bbappend" file.

From DevKit container:

```
$ cd /xdt/meta/meta-agl-product
$ mkdir -p recipes-core/systemd
$ vi recipes-core/systemd/systemd_%.bbappend
```

The bbappend file, below, extends the "install" task by deploying the new configuration:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"

SRC_URI += "file://wireless.network"

do_install_append() {
    install -m 644 ${WORKDIR}/wireless.network ${D}${sysconfdir}/systemd/network
}
```

And now, we create the configuration file "wireless.network" in the subdirectory commonly used to put extra files and patches related to recipes:

```
$ mkdir -p recipes-core/systemd/systemd
$ vi meta-agl-product/recipes-core/systemd/systemd/wireless.network
```

The file "wireless.network" below is the same that the one used when prototyping, as seen in chapter 2.2.2:

```
[Match]
Name=wl*

[Network]
Address=192.168.1.1/24
DHCPServer=yes
```

3.3. *hostapd* configuration file

The "hostapd.conf" file is already deployed when the original *hostapd* recipe is installed. To provide our own custom version, we can override using the same bbappend technic:

```
$ cd /xdt/meta/meta-agl-product
$ mkdir -p recipes-connectivity/hostapd
$ vi recipes-connectivity/hostapd/hostapd_%.bbappend
```

Here its content:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"

SRC_URI += "file://hostapd.conf"

do_install_append() {
    install -m 644 ${WORKDIR}/hostapd.conf ${D}${sysconfdir}
}
```

And now, we create the configuration file "hostapd.conf" in the subdirectory commonly used to put extra files and patches related to recipes:

```
$ mkdir -p recipes-connectivity/hostapd/hostapd
$ vi recipes-connectivity/hostapd/hostapd/hostapd.conf
```

The file "hostapd.conf" below is the same that the one used when prototyping, as seen in chapter 2.2.2:

```
ssid=test
wpa_passphrase=Somepassphrase
interface=wlP4p1s2u1u2
auth_algs=3
channel=7
driver=nl80211
hw_mode=g
logger_stdout=-1
logger_stdout_level=2
max_num_sta=5
rsn_pairwise=CCMP
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP CCMP
```

3.4. Package group integration

The service recipes are now adapted in the product layer. The next step of this integration is to overlay the initial AGL distribution by registering them in a packagegroup, instead of the temporary changes we made in "local.conf" file.

First, we remove the *IMAGE_INSTALL_append* directive introduced in 2.1.3 of the "local.conf" file.

Then, we locate the packagegroup recipe to be extended:

```
$ cd /xdt/meta/meta-agl/meta-agl
$ find . -name "packagegroup*bb*"
./recipes-core/packagegroups/packagegroup-agl-core-automotive.bb
./recipes-core/packagegroups/packagegroup-agl-core-connectivity.bb
./recipes-core/packagegroups/packagegroup-agl-core-graphics.bb
./recipes-core/packagegroups/packagegroup-agl-core-kernel.bb
./recipes-core/packagegroups/packagegroup-agl-core-multimedia.bb
./recipes-core/packagegroups/packagegroup-agl-core-navi-lbs.bb
./recipes-core/packagegroups/packagegroup-agl-core-os-commonlibs.bb
./recipes-core/packagegroups/packagegroup-agl-core-security.bb
./recipes-core/packagegroups/packagegroup-agl-core-speech-services.bb
./recipes-core/packagegroups/packagegroup-agl-core.bb
./recipes-core/packagegroups/packagegroup-core-boot-agl.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi-automotive.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi-connectivity.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi-graphics.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi-kernel.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi-multimedia.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi-navi-lbs.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi-os-commonlibs.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi-security.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi-speech-services.bb
./recipes-ivi/packagegroups/packagegroup-agl-ivi.bb
$
```

Any AGL distribution release should include "`meta-agl/meta-agl/recipes-core`" as a common basis for the distribution. The *hostapd* service is related to connectivity domain, thus we should extend the "`packagegroup-agl-core-connectivity.bb`":

```
$ cd /xdt/meta/meta-agl-product
$ mkdir -p recipes-core/packagegroups
$ vi recipes-core/packagegroups/packagegroup-agl-core-connectivity.bbappend
```

The packagegroup recipe is extended with the line below:

```
RDEPENDS_${PN} += "hostapd"
```