



AGL Kickstart on Renesas Porter Board

Version 2.0

July 2016

Abstract

This document explains step by step how to bootstrap a Porter board with AGL distribution and latest Renesas BSP then run a sample AGL application.

The following topics are covered:

- instructions to build an embedded image of AGL 2.0 “Brilliant Blowfish” based on Yocto Project 2.0 and Renesas BSP (meta-renesas)
- instructions on how to install and boot the image on Porter board
- instructions on how to add and run a “Hello World” application in the runtime system

Table of contents

1. Hardware setup.....	4
2. Building target image with Yocto project.....	5
2.1. Setting up your operating system.....	5
2.2. Setting up the build environment.....	7
2.3. Download Renesas proprietary drivers.....	8
2.4. Preparing to build.....	9
2.5. Configure for Release or Development.....	11
2.6. Build the image.....	12
3. Run the AGL Image.....	13
3.1. Boot the board using a micro-SD card.....	13
3.1.1. Format the SD-card.....	13
3.1.2. Set up the board to boot on SD-card.....	14
3.1.3. Copying the built image to the SD-card.....	16
3.1.4. Booting the board.....	17
3.2. Writing a “hello world” application.....	17
3.3. Running CES 2016 Demos.....	18
4. More Documentation.....	20

Document revisions

Date	Version	Designation	Author
14 Sept. 2015	0.9	Initial release	J. Bollo [lot.bzh]
14 Sept. 2015	1.0	Review and approval	S. Desneux [lot.bzh]
23 Dec. 2015	1.1	Updated for AGL 1.0 release	S. Desneux [lot.bzh] Y. Gicquel [lot.bzh]
22 June 2016	1.2	Update for Rev C Porter boards	S. Desneux [lot.bzh]
28 July 2016	2.0	Update for Brilliant Blowfish release	Y. Gicquel [lot.bzh] S. Desneux [lot.bzh]

1. Hardware setup

Here is a non exhaustive list of hardware parts that could be used to setup the Porter board development environment:

- Porter board with its power supply
- mini USB-A cable for serial console
- USB 2.0 Hub
- USB keyboard
- USB mouse
- Ethernet cable
- HDMI type A (full size HDMI) cable and associated display
- micro-SD Card (at least 4GB)
- USB touch screen device like the [GeChic 1502i](#)

For more information and latest news, please check the following page:
<http://elinux.org/R-Car/Boards/Porter>

The following documents may also be helpful:

- Porter Hardware Manual
http://elinux.org/images/8/83/Porter_HardwareManual_02242015.pdf
- Porter (Rev B) Setup Manual
http://elinux.org/images/1/11/PORTER_B_SetupManual_rev0.01.pdf

2. Building target image with Yocto project

The standard Yocto process is made of the following steps:

- Setting up your operating system,
- Setting up the build environment for R-Car BSP,
- Downloading the proprietary drivers and installing them in the build environment,
- Build the image,
- Boot using SD-CARD:
 - Create an SD-CARD,
 - Configure to boot on SD-CARD,
 - Copy the image to the SD-CARD,
 - Boot the board on it.

For convenience, the resulting development images are made available at the following location:

<https://download.automotivelinux.org/AGL/snapshots/master/latest/>.

If you want to bypass the build phase and quick boot the Porter board, you can download the image tarball and the kernel then follow the procedure described in section 3.

2.1. Setting up your operating system

The very first step is to ensure that your system can run the build system of the Yocto Project.

First, it only runs on Linux: if your system is Windows[®] or iOS[®] you should use a virtualization solution (Virtualbox, VMWare ...) to run a Linux VM on your system.

For AGL 2.0, Yocto Project 2.0, known as **jethro**, has been selected for the BSP and build system. This will change in the future but as of today [jul-2016] it is currently strongly recommended to use 2.0.

Reference data for configuring your system can be found in the Yocto documentation:

<http://www.yoctoproject.org/docs/2.0/ref-manual/ref-manual.html#detailed-supported-distros>

Here after an extract of this documentation for most common Linux distributions:

The build system should be able to run on any modern distributions that has the following versions for Git, tar, and Python, GCC, ...

- Git 1.7.8 or greater,
- tar 1.24 or greater,
- Python 2.7.3 or greater excluding Python 3.x, which is not supported.

Also note that for this tutorial, the utility 'curl' has been added to the list of packages to install.

Ubuntu and Debian

The essential and graphical support packages you need for a supported Ubuntu or Debian distribution are shown in the following command:

```
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
  build-essential chrpath socat libsdl1.2-dev xterm cpio curl
```

Fedora

The essential and graphical packages you need for a supported Fedora distribution are shown in the following command:

```
sudo yum install gawk make wget tar bzip2 gzip python unzip perl patch \
  diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath \
  ccache perl-Data-Dumper perl-Text-ParseWords perl-Thread-Queue socat \
  SDL-devel xterm curl
```

OpenSUSE

The essential and graphical packages you need for a supported OpenSUSE distribution are shown in the following command:

```
sudo zypper install python gcc gcc-c++ git chrpath make wget python-xml \
  diffstat texinfo python-curses patch socat libSDL-devel xterm curl
```

CentOS

The essential and graphical packages you need for a supported CentOS distribution are shown in the following command:

```
sudo yum install gawk make wget tar bzip2 gzip python unzip perl patch \
  diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath \
  socat SDL-devel xterm curl
```

2.2. Setting up the build environment

Yocto layers required by AGL distribution can be fetched using the “repo” tool. Here are the detailed steps to prepare a build environment using this method.

In the following, your top level directory is noted as “AGL_TOP”.

For example, we will set AGL_TOP to point to a directory “\$HOME/r-car-agl”:

```
export AGL_TOP=$HOME/r-car-agl
mkdir -p $AGL_TOP
```

Then in this directory, the developer should clone, using git, the reference sources of Yocto Project, OpenEmbedded, AGL, etc. All yocto layers required by the AGL distribution can be fetched using the “repo” tool¹.

First, we need to install “repo”:

```
mkdir -p ~/bin
export PATH=~/bin:$PATH
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

Then, we can use the “repo” command to initialize from the official AGL repository:

```
cd $AGL_TOP
repo init -u https://gerrit.automotivelinux.org/gerrit/AGL/AGL-repo \
  -b blowfish \
  -m default_blowfish_2.0.1.xml
```

And finally, we can use the tool to sync all local repositories with the manifest file:

```
repo sync
```

¹ <https://gerrit.googlesource.com/git-repo>

2.3. Download Renesas proprietary drivers

Developer may want to install Renesas non open-source drivers, at least in their evaluation version.

The evaluation version of these drivers can be found here:

http://www.renesas.com/secret/r_car_download/rcar_demoboard.jsp

under the **Target hardware: R-Car H2, M2 and E2** section:

Target hardware: R-Car H2, M2 and E2



Product name: R-Car Series Evaluation Software Package for Linux

Product Name	Download File
R-Car Series Evaluation Software Package for Linux	To download Multimedia and Graphics library, please click this link. Go to Download page >>
	To download related Linux drivers, please click here .

Note that you have to register with a free account on MyRenesas and accept the license condition before downloading them. The operation is fast and simple but nevertheless mandatory to access evaluation of non open-source drivers for free.

Once you registered, you can download two zip files. Note you can left them in your Downloads directory (please check the \$XDG_DOWNLOAD_DIR environment variable value).

Here after is an example of their names:

```
$ chmod a+r $XDG_DOWNLOAD_DIR/*.zip
$ ls -l $XDG_DOWNLOAD_DIR
total 8220
-rw-r--r-- 1 1000 1000 6047383 Jul 11 11:03 R-
Car_Series_Evaluation_Software_Package_for_Linux-20151228.zip
-rw-r--r-- 1 1000 1000 2394750 Jul 11 11:03 R-
Car_Series_Evaluation_Software_Package_of_Linux_Drivers-20151228.zip
```


2.4. Preparing to build

The `aglsetup.sh` helper script was introduced in AGL 2.0 and enables a flexible way to select features in the distribution. You can get more details of supported features and machines by sourcing this script with the “-h” option:

```
cd $AGL_TOP
source meta-agl/scripts/aglsetup.sh -h
----- aglsetup.sh: Starting
Usage: . aglsetup.sh [options] [feature [feature [... ]]]

Version: 1.0.0
Compatibility: bash, zsh, ksh

Options:
  -m|--machine <machine>
    what machine to use
    default: 'qemux86-64'
  -b|--build <directory>
    build directory to use
    default: './build'
  -s|--script <filename>
    file where setup script is generated
    default: none (no script)
  -f|--force
    flag to force overwriting any existing configuration
    default: false
  -v|--verbose
    verbose mode
    default: false
  -d|--debug
    debug mode
    default: false
  -h|--help
    get some help

Available machines:
  [meta-agl]
    dra7xx-evm
    dragonboard-410c
    intel-corei7-64
    porter
    porter-nogfx
    qemux86
    * qemux86-64
    raspberrypi2
    raspberrypi3
    wandboard
  [meta-agl-extra]

Available features:
  [meta-agl]
    agl-devel
    agl-netboot
  [meta-agl-extra]
    agl-appfw-smack
    agl-demo
```

```
agl-localdev
agl-sota
```

For porter, we can set the build environment using:

```
cd $AGL_TOP
source meta-agl/scripts/aglsetup.sh \
    -m porter \
    -b build \
    agl-devel agl-demo agl-appfw-smack
----- aglsetup.sh: Starting
Generating configuration files:
  Build dir: /ssd/agl2016-for-kickstart-update/build
  Machine: porter
  Features: agl-devel agl-demo agl-appfw-smack
  Running /ssd/agl2016-for-kickstart-update/poky/oe-init-build-env
  Templates dir: /ssd/agl2016-for-kickstart-update/meta-agl/templates/base
  Config: /ssd/agl2016-for-kickstart-update/build/conf/bblayers.conf
  Config: /ssd/agl2016-for-kickstart-update/build/conf/local.conf
  Setup script: /ssd/agl2016-for-kickstart-update/build/conf/setup.sh
  Executing setup script ... --- beginning of setup script
--- fragment /ssd/agl2016-for-kickstart-update/meta-
agl/templates/base/01_setup_EULAFunc.sh
--- fragment /ssd/agl2016-for-kickstart-update/meta-
agl/templates/machine/porter/50_setup.sh
/ssd/agl2016-for-kickstart-update /ssd/agl2016-for-kickstart-update/build
Archive: /home/yannick/Téléchargements/R-
Car_Series_Evaluation_Software_Package_for_Linux-20151228.zip
  extracting: R-Car_Series_Evaluation_Software_Package_for_Linux-20151228.tar.gz
  inflating: readme.txt
Archive: /home/yannick/Téléchargements/R-
Car_Series_Evaluation_Software_Package_of_Linux_Drivers-20151228.zip
  inflating: R-Car_Series_Evaluation_Software_Package_of_Linux_Drivers-
20151228.tar.gz
  inflating: readme.txt
/ssd/agl2016-for-kickstart-update/build
--- fragment /ssd/agl2016-for-kickstart-update/meta-
agl/templates/base/99_setup_EULAconf.sh
--- end of setup script
OK
Generating setup file: /ssd/agl2016-for-kickstart-update/build/agl-init-build-
env ... OK
----- aglsetup.sh: Done

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
- meta-agl:          (core system)
  agl-image-ivi
  agl-image-ivi-crosssdk
  agl-image-ivi-qa
  agl-image-minimal
  agl-image-minimal-qa
  agl-image-weston
- meta-agl-demo:    (demo with UI)
```

```
agl-demo-platform (* default demo target)
agl-demo-platform-qa
agl-demo-platform-crosssdk
```

Notes: after this command, the working directory is changed to `$AGL_TOP/build`.

In case the graphical drivers were not found, you could notice an error message as follow:

```
[snip]
--- fragment /ssd/agl2016-for-kickstart-update/meta-
agl/templates/machine/porter/50_setup.sh
/ssd/agl2016-for-kickstart-update /ssd/agl2016-for-kickstart-update/build
The graphics and multimedia acceleration packages for the R-Car M2 Porter board
can be download from :
  <http://www.renesas.com/secret/r_car_download/rcar_demoboard.jsp>

These 2 files from there should be store in your '/home/users/Downloads'
directory.
  R-Car_Series_Evaluation_Software_Package_for_Linux-20151228.zip
  R-Car_Series_Evaluation_Software_Package_of_Linux_Drivers-20151228.zip
Copying gfx drivers and multimedia packages for 'porter' failed.
ERROR: Script /ssd/agl2016-for-kickstart-update/build/conf/setup.sh failed
OK
Generating setup file: /ssd/agl2016-for-kickstart-update/build/agl-init-build-
env ... OK
----- aglsetup.sh: Done
[snip]
```

If you encounters this issue, or any other unwanted behavior, you can fix the error mentioned and then clean up by removing the "`$AGL_TOP/build`" directory then launch the procedure again.

Users may want to check that the board is correctly selected in the environment:

```
$ grep -w -e "^MACHINE" $AGL_TOP/build/conf/local.conf
MACHINE ??= "qemux86"
MACHINE = "porter"
```

2.5. Configure for Release or Development

Development images require extra tools for developer convenience, in particular:

- a debugger (gdb)
- some tweaks, including a disabled root password

- a SFTP server
- the TCF Agent for easier application deployment and remote debugging
- ...

In section 2.4, after the `aglsetup.sh` environment script was launched, we explicitly activate these Debug facilities by specifying the “agl-devel” feature.

2.6. Build the image

The process to build an image is simple:

```
cd $AGL_TOP/build
bitbake agl-demo-platform
```

Once done, what may take up to few hours, you should get the end result in the directory `$AGL_TOP/build/tmp/deploy/images/porter`.

In case of failure of the build it is safe to first check that the Linux distribution chosen for your host has been validated for version 2.0 of Yocto.

3. Run the AGL Image

3.1. Boot the board using a micro-SD card

NOTE: Porter boards have 2 SD slots: one for SD cards and another one for micro-SD cards. At the time of writing, we didn't succeed to boot a board using the SD slot with the current kernel (3.10): only the micro-SD slot was usable.

To boot the board using a micro-SD card, there are two operations that should be done prior to first initial boot:

- create a SD-card with one ext3 partition,
- set up the board to boot on the SD-card.

Then for each build, the SD-card is merely rewritten and used to boot the configured board.

3.1.1. Format the SD-card

First, insert the SD-card in your computer. Type "dmesg | tail -15" to get its associated device. Example:

```
dmesg | tail -10
...
[608743.302571] mmc0: new high speed SDHC card at address 0007
[608743.302748] mmcblk0: mmc0:0007 SD16G 14.9 GiB
[608743.303818]  mmcblk0: p1
```

Here, the SD-card is attached to the device **mmcblk0**.

You can also use 'lsblk' to confirm the SD-card location:

```
lsblk
NAME                MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                  8:0    0 223.6G  0 disk
├─sda1                8:1    0    2G  0 part /
├─sda2                8:2    0   12G  0 part [SWAP]
└─sda3                8:3    0 209.6G  0 part
   └─vg0-usr          254:0    0   32G  0 lvm  /usr
   └─vg0-data         254:1    0   50G  0 lvm  /data
   └─vg0-home         254:2    0   80G  0 lvm  /home
   └─vg0-var          254:3    0    4G  0 lvm  /var
   └─vg0-docker       254:4    0   24G  0 lvm  /docker
mmcblk0             179:0    0  14.9G  0 disk
└─mmcblk0p1         179:1    0  14.9G  0 part /var/run/media/sdx/dbc75ae4-beac-45e5-949f-c185b92c917a
```

In the previous example, we see the first SATA drive as 'sda' and the SD-Card as 'mmcblk0'.

Use "fdisk" to partition the card and set the MBR. Example:

```
sudo fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.26.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): o
Created a new DOS disklabel with disk identifier 0xa35c1e5b.

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p):

Using default response p.
Partition number (1-4, default 1):
First sector (2048-31268863, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-31268863, default 31268863):

Created a new partition 1 of type 'Linux' and of size 14.9 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

Then initialize the ext3 partition using "mke2fs":

```
sudo mke2fs -t ext3 /dev/mmcblk0p1
```

3.1.2. Set up the board to boot on SD-card

Install a serial client on your computer. This can be "screen", "picocom", "minicom". The lighter of the 3 is "picocom" (it has less dependencies).

Plug a USB cable from your computer to the serial CP2102 USB port of the porter board (near the power switch and fan connector).

With "dmesg" you can check the device created for the serial link. To get it, you must switch the board on. For example:

```
dmesg | tail
[609575.767056] usb 2-1.6.4: new full-speed USB device number 21 using ehci-pci
[609575.854083] usb 2-1.6.4: New USB device found, idVendor=10c4,
idProduct=ea60
[609575.854089] usb 2-1.6.4: New USB device strings: Mfr=1, Product=2,
SerialNumber=3
[609575.854100] usb 2-1.6.4: Product: CP2102 USB to UART Bridge Controller
[609575.854102] usb 2-1.6.4: Manufacturer: Silicon Labs
[609575.854104] usb 2-1.6.4: SerialNumber: 0001
[609575.990209] usbcore: registered new interface driver usbserial
```

```
[609575.990221] usbcore: registered new interface driver usbserial_generic
[609575.990229] usbserial: USB Serial support registered for generic
[609575.995184] usbcore: registered new interface driver cp210x
[609575.995198] usbserial: USB Serial support registered for cp210x
[609575.995239] cp210x 2-1.6.4:1.0: cp210x converter detected
[609576.068184] usb 2-1.6.4: reset full-speed USB device number 21 using ehci-pci
[609576.154125] usb 2-1.6.4: cp210x converter now attached to ttyUSB0
```

The link is attached to the device `/dev/ttyUSB0`.

It is time to launch your serial client. Example:

```
picocom -b 38400 /dev/ttyUSB0
```

OR

```
minicom -b 38400 -D /dev/ttyUSB0
```

OR

```
screen /dev/ttyUSB0 38400
```

Now you should reset/reboot the board and inside your client terminal, type a character to abort the boot and enter the U-boot menu:

```
KOELSCH SPI_LOADER(DDR3L_1333) V0.16a 2014.10.03
DEVICE S25FL512

U-Boot 2013.01.01-gb653737-dirty (Mar 26 2015 - 14:37:46)

CPU: Renesas Electronics R8A7791 rev 2.0
Board: Porter Board

DRAM: 1 GiB
MMC: sh-sdhi: 0, sh-sdhi: 1
SF: Detected S25FL512S with page size 256 KiB, total 64 MiB
In: serial
Out: serial
Err: serial
Net: sh_eth
Hit any key to stop autoboot: 0
=>
```

Then enter the following commands at the `'=>'` prompt to redefine the boot parameters, in three steps to avoid copy/paste problems:

```
setenv bootargs_console 'console=ttySC6,38400 ignore_loglevel'
setenv bootargs_video 'vmalloc=384M video=HDMI-A-1:1920x1080-32@60'
setenv bootargs_root 'root=/dev/mmcblk0p1 rootdelay=3 rw rootfstype=ext3
rootwait'
```

NOTE: if no display shows up when booting, e.g. for a non-full HD screen, users may want to replace the "1920x1080" value in the `bootargs_video` variable with another lower one such as "1024x768"; there is unfortunately no universally supported setting yet

```
setenv bootmmc          '1:1'
setenv bootcmd_sd      'ext4load mmc ${bootmmc} 0x40007fc0 boot/uImage+dtb'
setenv bootcmd 'setenv bootargs ${bootargs_console} ${bootargs_video} $
${bootargs_root}; run bootcmd_sd; bootm 0x40007fc0'
```

NOTE: depending on your board (Porter rev B or rev C, Koelsch etc.), the SD card slots may differ. You could have to set 'bootmmc' to '0:1' or '2:1' depending on the slot and card format.

```
saveenv
Saving Environment to SPI Flash...
SF: Detected S25FL512S with page size 256 KiB, total 64 MiB
Erasing SPI flash...Writing to SPI flash...done
```

You have now finished the setup of your board.

3.1.3. Copying the built image to the SD-card

Insert the SD-card into your build host: your desktop system may probably offer a choice to mount the SD-card automatically in some directory. In the next sample code, we'll suppose that the SD-card mount directory is stored in the variable \$SDCARD.

To extract the root file-system with extended attributes, your host requires the use of "tar" version 1.28 minimum. You can check the current version using following:

```
tar --version
tar (GNU tar) 1.28
[snip]
```

If your distribution is up to date on this dependency, you can use the host tool directly. Let's define a variable for the following steps:

```
TAR=$(which tar)
```

Otherwise, a native up-to-date version of tar is also generated while building AGL distribution:

```
TAR=$AGL_TOP/build/tmp/sysroots/x86_64-linux/usr/bin/tar-native/tar
$TAR --version
tar (GNU tar) 1.28
[snip]
```

Next step is to extract the AGL image tarball and copy the kernel onto the SD-card:

IMPORTANT: replace literally 'SOME_SPECIFIC_MOUNT_DIRECTORY' by the appropriate path of the SD-card on your system, or bad things may happen! If any doubt, do nothing.

```
export SDCARD=SOME_SPECIFIC_MOUNT_DIRECTORY
cd $AGL_TOP/build/tmp/deploy/images/porter
sudo rm -rf ${SDCARD:-bad_dir}/*
```



```
sudo $STAR xpj --xattrs-include='*' -f agl-demo-platform-porter.tar.bz2 \  
-C $SDCARD  
sudo cp uImage+dtb $SDCARD/boot
```

3.1.4. Booting the board

Turn the board off using the power switch.

Insert the microSD-card into the appropriate slot.

Verify that you have plugged in, at least, the following:

- external monitor on HDMI port
- input device (keyboard, mouse, touchscreen...) on USB port.

Turn the board on using the power switch.

After a few seconds, you'll see the AGL splash screen on the display and you'll be able to log in on the console terminal (login is 'root', no password):

```
Automotive Grade Linux 2.0.0 porter ttySC6  
porter login:
```

3.2. Writing a "hello world" application

Yocto project provides a good reference on its complete solution for developers:

- ADT: The Application Development Toolkit is the complete solution;
- the cross-toolchain is a simple build environment.

Reading <http://www.yoctoproject.org/docs/2.0/adt-manual/adt-manual.html> is a good starting point.

A Docker image with prebuilt AGL SDK is also made available by IoT.bzh. Check the following document for more information:

<http://iot.bzh/download/public/2016/sdk/AGL-Application-SDK-Kickstart-on-Renesas-Porter-board.pdf>.

Here, for a quick demo we will build the cross-toolchain and write a sample application.

First, let's create the build toolchain:

```
cd $AGL_TOP  
source poky/oe-init-build-env  
bitbake meta-ide-support
```

The small following "hello world" example:

```
cat hello.c
#include <stdio.h>
int main() { printf("Hello world\n"); return 0; }
```

... can now be compiled and executed this way:

```
. $AGL_TOP/build/tmp/environment-setup-*
$CC -o hello hello.c
scp hello root@porterboard:/
ssh root@porterboard /hello
```

where 'porterboard' is replaced by the IP address or the hostname of your Porter board.

3.3. Running CES 2016 Demos

The CES demos are located in /opt/AGL/CES2016 (on the microSD-Card).

To run the demo, execute the following commands on the target (from a weston terminal or from the serial console)

```
cd /opt/AGL/CES2016
export LD_PRELOAD=/usr/lib/libEGL.so
```

For the main demo, run:

```
/usr/bin/qt5/qmlscene --fullscreen -I imports Main.qml
```

To start the demo using IVI Shell, run the appropriate scripts located in /opt/AGL/CES2016:

```
./switch_to_ivi-shell.sh
./start_CES2016_ivi-shell.sh
```

This will restart Weston with IVI Shell enabled and launch the demo.

With the above commands, the demo application has still some decorations. They can be dropped by adding '--fullscreen' in the script. Use the following command once to modify the script.

```
sed -i 's/Main.qml/--fullscreen Main.qml/' start_CES2016_ivi-shell.sh
```

Then restart the demo:

```
killall qmlscene
./start_CES2016_ivi-shell.sh
```

IMPORTANT: Please note that the current image uses Evaluation drivers: as a consequence, the graphics and multimedia acceleration provided by these drivers will stop **after 3 hours**. When this happens, simply reboot the board and restart the demo.

For more information, you can check the embedded README:

```
cat /opt/AGL/CES2016/README.md
Open source QML UI

To run on target:
$ cd /opt/AGL/CES2016
$ /usr/bin/qt5/qmlscene -I imports Main.qml

For development it can be nice to use Scaled.qml instead so it fits your
screen.

© 2015 Jaguar Land Rover. All Rights Reserved.
Licensed under Creative Commons Attribution 4.0 International
https://creativecommons.org/licenses/by/4.0/legalcode

(Optional) switch shell for weston to ivi-shell and start demo apps if you want
to start demo apps with ivi-shell.
$ cd /opt/AGL/CES2016
$ ./switch_to_ivi-shell
(Optional a) $ ./start_CES2016_ivi-shell.sh
(Optional b) $ ./start_CES2016_with_navi_ivi-shell.sh

Option a: start QML UI only.
Option b: start QML + CarNavigation:/home/navi. For the time being,
CarNavigation expects to be Wayland native application, which will be showed on
top of QML by using LayerManagerControl.
```

4. More Documentation

More documents are available to guide developers with AGL and Renesas boards:

- [AGL-Devkit-Image-and-SDK-for-porter.pdf](#)
Detailed guide on how to build AGL for Renesas boards and using AGL SDK inside a ready-to-use Docker container.
- [AGL-Devkit-Build-your-1st-AGL-Application.pdf](#)
Generic guide on how to build various application types (HTML5, native, Qt, QML, ...) for AGL.
- [AGL-Devkit-HowTo_bake_a_service.pdf](#)
Generic guide on how to add a new service in the BSP.