
# AGL/Phase 2 - Pulse Audio Routing Module

Developer Documentation

Version 1.0

September 2016

# Abstract

This document is the "**Developer Documentation**", as a part of the "AGL/Phase2-AppFw-Audio: Audio Routing" SoW deliverable.

# Document revisions

| Date | Version | Designation | Author |
|---|---|---|---|
| 14 Sept 2016 | 0.1 | First version | Y. Gicquel [ IoT.bzh ] |
| 20 Sept 2016 | 1.0 | Review & document approval | S. Desneux [ IoT.bzh ] |
| | | | |

# Table of contents

# 1. Introduction

The PulseAudio routing module for AGL presented here is targeted to fill the gap between open-source audio stacks and the current Automotive Grade Linux audio specifications.

The current routing module is a fork of the “pulseaudio-murphy-ivi-plugin” developed for Tizen [1], and its refit was done in the scope of a proof of concept which had been performed end of July this year.

Whereas the POC had confirmed it's ability to fulfill some typical routing scenarios which will be described in the next sections of this document, it also highlights the need to some further changes.

*This document presents the audio related solution on which the plugin interacts with: Alsa, PulseAudio, and their respective scope. In a second part, it presents the requirements for IVI systems and the current state and design of the routing plugins.*

*The last section presents the necessary steps to activate and test this plugin on a Renesas Porter board on AGL-2.0.*

# 2. Linux Audio frameworks

Before describing the module itself, this section presents some of the functional scopes of common Linux audio solutions which are involved in the routing feature.

## 2.1. Advanced Linux Sound Architecture

Advanced Linux Sound Architecture (ALSA) is a software framework which takes place in the Linux Kernel and in a user-space library: *libalsa*. It brings a set of API to enable the management of sound devices & data streams from user-space applications.

ALSA abstracts the hardware specificity through a common Kernel API, to:

- Reference *cards*, *devices*, and *sub-devices*, to get access to sound *streams*,
- Configure session *streams* at a specific sampling rate, a specific sample format/resolution, a specific number of channels, whereas they are input/output or both,
- Manage the sounds volumes setup and the hardware mixing feature if supported.

Accesses to ALSA devices are exclusive, meaning when a user-space application acquires the device resources, all others requesting applications are not able to perform a playback nor an acquisition. Accordingly, when a stream is captured or played, its characteristics such as sampling rate or resolution are fixed.

## 2.2. PulseAudio

PulseAudio is a proxy system which can run on top of ALSA API and which brings new functionalities to provide a more powerful & extendable audio solution. It is composed of a process (which can be executed as a daemon) and an associated library: *libpulse*.

PulseAudio most relevant features are listed below:

- It supports *multiple application streams mixing* and thus enables the sharing of the same hardware resources. This feature may trigger the use of an internal re-sampler when the different streams have different characteristics,

- It can be employed in the *users sessions*, and thus enforce the isolation of allocated resources & the security level of the whole sound system. This feature is enabled by the launch of a dedicated daemon for the system and the different users,
- It is *expandable with plugins* thanks to its module API. This is the way the routing feature is currently interfaced with PulseAudio,
- It can be *used over network* interfaces to share the local sound resources to remote clients,
- It can apply some treatments on the data streams (i.e. sound effects),

Whereas PulseAudio main target is the desktop environment, some of its features can be reused to fulfill the requirements of an IVI system.

## 2.3. IVI systems requirements

Targeted IVI systems are composed of multiples sounds peripherals, dispatched in different places inside the vehicle. Thus, some are part of the main IVI system, others may require to be remotely accessed through a dedicated link or the network. The system also requires to manage some transient peripherals such as Bluetooth headsets.

On the client side, as an embedded IVI system aims to allow end-users to install some 3rd party applications, some mechanisms should be available to handle those multiples applications to coexists with the pre-installed applications from OEMs. On the security aspects, this means handling application priority to keep the safety related sound available whatever the applications are.

PulseAudio in its current implementation (6.0 is integrated in AGL-2.0) has some native features and/or allows plugins to enable such requirements, but it also lacks some of them, including a consistent way to handle dynamic routing and events: these features are covered by the routing plugin itself.

# 3. Audio routing module design

The previous section briefly presents functional scopes of the most common open-source Audio frameworks. In this section, the document presents the routing module itself.

## 3.1. Functional scope

The functional scope for audio routing in Automotive systems introduces new concepts described below. They are inherited from its predecessor (pulseaudio-murphy-ivi-plugin) and are subject to further modifications.

Definitions will be detailed in following sections, but basically the aim of the routing plugin is to dynamically associate *sound stream producers* to *sound stream consumers*.

The features in the current release of the routing module are:

- Application's audio *streams classification* by rules which can be explicitly defined from a configuration file, or implicitly applied by following a default behavior,
- Configuration of *stream priorities* by a set of rules,
- Audio stream routing between classified streams and dedicated sound cards,
- Automatic volume ramp-down, ramp-up or mute, in the respect of applications defined priorities and applications classes,

The following features are planned to be covered, or are subject to further research:

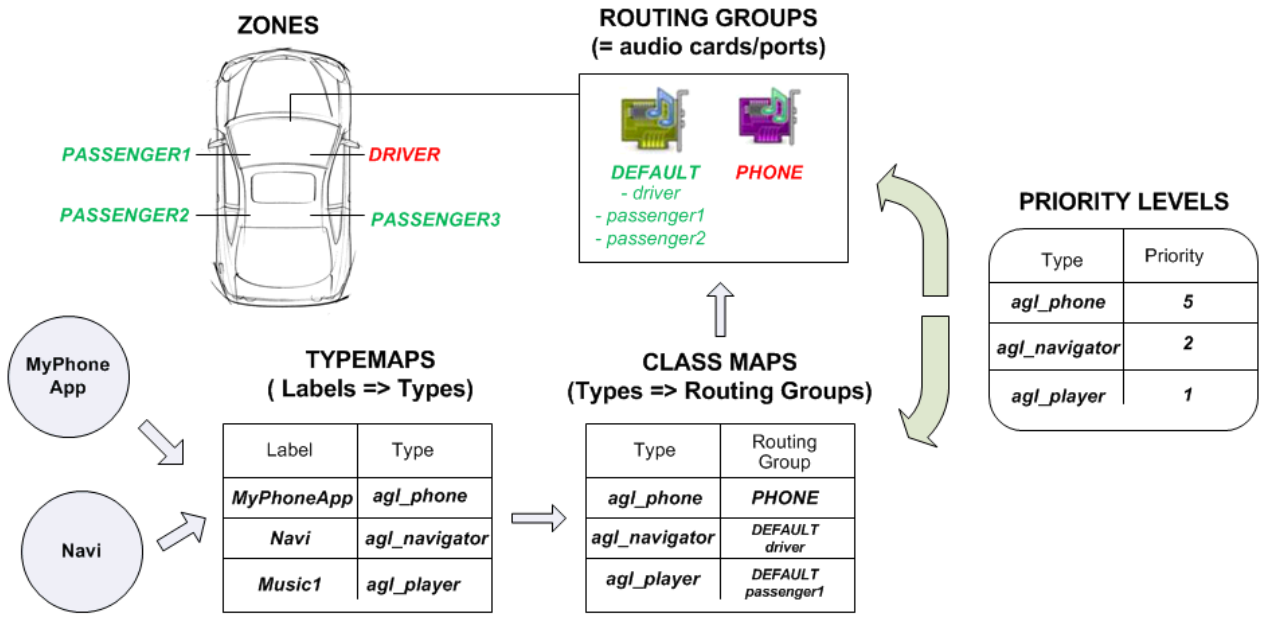
- Established stream routing on a hot-pluggable devices: this feature can be managed by the plugin, but it is currently under development by the pulseaudio upstream team. Some matching should be done to avoid some feature “overlap” between main stream & the plugin,

## 3.2. Plugin definitions

Some IVI specifics definitions are introduced by the routing plugin:

- **zone**: it refers to a set of physicals speakers inside the vehicle. (This setting is currently not managed).
- **type map**: this is a n-to-1 associative table: one or more "application identifier" (the pulseaudio media.role property) can be configured to belong to one specific "application class".
- **application class**: a set of applications grouped regarding their role in the system.
- **priority map**: this is a 1-to-1 table which statically defines the "application class" statics priorities.
- **routing groups**: this is a set of possible routing target (i.e. the sounds cards),
- **class map**: this is were the routing rules are defined. This is a list which defines how streams within an application class should be associated to one ore multiple routing groups.

Here below is a figure representing how those associations interacts in the routing process:



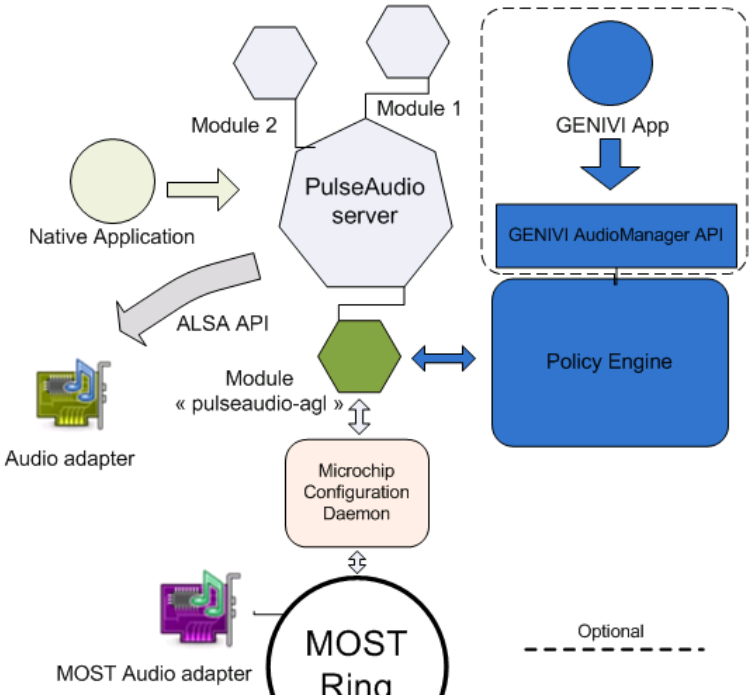
## 3.3. Architecture overview

As from its predecessor, the plugin is a shared object which implements the module API as defined by PulseAudio [2].

While Tizen policy engine had a large scope with a tight dependency to the murphy policy manager daemon, the current routing policy architecture for AGL is much lighter and focuses on audio domain only.

Thus, the dependency with murphy service has been removed and the policy engine is currently based on the rules defined in the plugin configuration file.

Here below is a figure summarizing the different parts of the architecture:

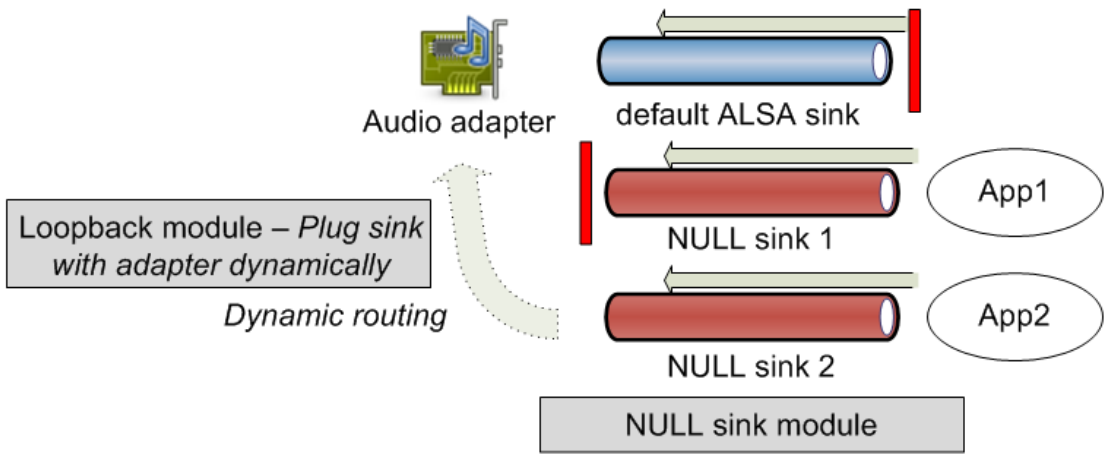


## 3.4. Dynamic routing

While PulseAudio supports dynamic addition and removal of audio sources and sinks, it does not support to dynamically change the setup of an established stream between those one. As soon as a stream is established between a source and a sink, it is not possible to change it's configuration (or route) without stopping the stream, and thus without impacting the application themselves.

To bypass this constraint, a particular design is proposed by the plugin:

1. all streams clients (i.e. the applications) are connected to a dedicated "null" sink, which is nothing more than a /dev/null equivalent in PulseAudio framework,
2. by default, application streams routes are thus established to an inaudible end-point.
3. when a new route is required and granted on such an established stream, the plugin instantiates a "loopback module" to dynamically connect the application null sink to a real sound card sink.




## 3.5. Source code repositories

At the time of writing this documentation, its Yocto recipe is already integrated in AGL distribution [3] and it builds the source code accessible through a public repository [4].

The plugin has few dependencies and requires 'json-c' & 'pulseaudio-module-devel' packages for building.

## 3.6. Known limitations

- Some jitter/timing issue can be audible on some stream playback. A workaround is possible by tuning the latency parameter passed to the loopback module, but it needs some further investigation to avoid any transient,
- Json configuration is not validated and can produce silent error and apply some default configuration,
- Effects functions are hardcoded,

# 4. Audio routing on Porter board

In this section, we present a procedure to activate the plugin and to start a stream prioritization process by launching two different streams with different IVI roles.

## 4.1. Prerequisites

To start the bench, you need:

- a Porter board with AGL-2.0 image installed,
- a 3.5" Jack headset, or loudspeaker plugged to the Audio Out jack connector,
- a set of sound files to be played (some samples are provided below),

## 4.2. Files deployed on target

The routing PulseAudio module is implemented by those following files:

| File path | Description |
|---|---|
| /etc/pulse/pulseaudio-agl.cfg | Routing module configuration file |
| /usr/lib/pulse-6.0/modules/agl-audio-plugin.so | Routing module shared object |

We also provides two sound sample file to perform further tests. Once logged on the Porter board, you can get them:

```
cd /home/root
wget http://iot.bzh/download/public/2016/medias/celtic_irish.mp3
wget http://iot.bzh/download/public/2016/medias/Phone_Ringing.wav
```

## 4.3. Launch PulseAudio server in debug mode

In AGL-2.0, the routing module is not loaded by default, and the PulseAudio configuration is kept with default configuration files. For the next steps, we propose to prepare the environment with a single system instance of the PulseAudio server with its logs visible in the console.

Also, PulseAudio server can be launched on-demand when clients require a sound session: this is a feature implemented in libpulse, and transparently available to applications. To clarify the following steps, we will disable the autospawn feature.

First, we can disable all PulseAudio servers:

```
rm -Rf /home/root/.config/systemd/user/*
echo "autospawn = no" >> /etc/pulse/client.conf
pulseaudio --kill
```

We can confirm that no more daemon is running by using:

```
# ps x | grep pulseaudio
 2121 pts/2    S+     0:00 grep pulse
#
```

Or by trying to play sound:

```
# aplay Phone_Ringing.wav
ALSA lib /ssd/agl2016-sept/build/tmp/work/cortexa15hf-vfp-neon-poky-linux-
gnueabi/alsa-plugins/1.0.29-r0/alsa-plugins-1.0.29/pulse/pulse.c:243:
(pulse_connect) PulseAudio: Unable to connect: Connection refused

aplay: main:730: audio open error: Connection refused
#
```

Now, we can launch a single instance using:

```
/usr/bin/pulseaudio --daemonize=no -v --log-target=stderr \
--exit-idle-time=3600 --log-time
(   0.000|   0.000) W: [pulseaudio] main.c: This program is not intended to be
run as root (unless --system is specified).
(   0.000|   0.000) I: [pulseaudio] core-util.c: Successfully gained nice level
-11.
(   0.000|   0.000) I: [pulseaudio] main.c: This is PulseAudio 6.0
(   0.000|   0.000) I: [pulseaudio] main.c: Page size is 4096 bytes
(   0.000|   0.000) I: [pulseaudio] main.c: Machine ID is
d28ff6cd7f154f35a75cd72acdf5d1d0.
[snip]
(   0.134|   0.000) I: [pulseaudio] main.c: Daemon startup complete.
```

If the last line "Daemon startup complete" appears, it means everything is now ready for the next steps. We can confirm functional state, by playing a sound. Using a new terminal window, let's connect to the Porter board and launch a playback client:

```
# aplay Phone_Ringing.wav
Playing WAVE 'Phone_Ringing.wav' : Signed 16 bit Little Endian, Rate 44100 Hz,
Stereo
#
```

## 4.4. Load the routing module

Enter the following command to load the routing module:

```
# pactl load-module agl-audio-plugin
17
#
```

If we look at the PulseAudio terminal, we can see the logs resulting of the plugin load:

```
[snip]
( 500.163| 250.686) I: [pulseaudio] client.c: Created 3 "Native client (UNIX
socket client)"
( 500.164|   0.000) I: [pulseaudio] protocol-native.c: Got credentials: uid=0
gid=0 success=1
( 500.166|   0.002) E: [pulseaudio] module.c: Initializing "pulseaudio-agl"
module
( 500.166|   0.000) E: [pulseaudio] module.c: cfgdir : /etc/pulse
( 500.166|   0.000) E: [pulseaudio] module.c: cfgfile : pulseaudio-agl.cfg
( 500.167|   0.000) I: [pulseaudio] config.c: parsing configuration file
'/etc/pulse/pulseaudio-agl.cfg'
Key : alsa_input.platform-sound.6.(null)@analog-input
Key : alsa_output.platform-sound.6.analog-stereo@analog-output
Key : alsa_output.platform-sound.6.analog-stereo@analog-output
Key : alsa_input.platform-sound.6.analog-stereo@analog-input
( 500.168|   0.001) I: [pulseaudio] module.c: Loaded "agl-audio-plugin" (index:
#17; argument: "").
( 500.168|   0.000) I: [pulseaudio] client.c: Freed 3 "pactl"
( 500.169|   0.000) I: [pulseaudio] protocol-native.c: Connection died.
[snip]
```

Now, we can play a sound which will be handled by the routing module. For example, we can start a stream with an “application type” set to “navi” class by using:

```
PULSE_PROP='media.role=navi' \
gst-launch-1.0 playbin uri=file:///home/root/celtic_irish.mp3
```

## 4.5. Launch multiple stream with different priorities

To illustrate the priority handling mechanism, we can launch two different streams with different priorities. The table below summarize default priority map as defined in the configuration file:

| Stream class | Priority |
|---|---|
| event | 5 |
| phone | 4 |
| navi | 2 |
| radio | 1 |
| music | 1 |

**Test 1**: A stream is played and a new one with *higher* priority stream is launched:

For this test, we will launch two instances of gstreamer, with the first stream associated to the "navi" class (priority 2) and the second stream associated to the "event" class (priority 5).

Step 1: Start stream priority 2

```
PULSE_PROP='media.role=navi' \
gst-launch-1.0 playbin uri=file:///home/root/celtic_irish.mp3 &
```

Step 2: Start stream priority 5

```
PULSE_PROP='media.role=event' \
gst-launch-1.0 playbin uri=file:///home/root/Phone_Ringing.wav
```

Results: New high priority stream mutes the lower one.

**Test 2**: A stream is played and a new one with *lower* priority stream is launched:

For this test, we will launch two instances of gstreamer, with the first stream associated to the "navi" class (priority 2) and the second stream associated to the "radio" class (priority 1).

Step 1: Start stream priority 2

```
PULSE_PROP='media.role=navi' \
gst-launch-1.0 playbin uri=file:///home/root/celtic_irish.mp3 &
```

Step 2: Start stream priority 1

```
PULSE_PROP='media.role=radio' \
gst-launch-1.0 playbin uri=file:///home/root/Phone_Ringing.wav
```

Results: New low priority stream mutes the lower one.

# 5. Sample clients

In the routing module, the stream classification is based on the "media.role" PulseAudio property. This property is not related to a particular client, but can be passed to audio application which use the *libpulse* library. The library will look for "PULSE_PROP" environment variable, and will append its value to the streams properties.

## 5.1. AlsaPlayer client

Here is an example of the property setup for GStreamer client:

```
PULSE_PROP='media.role=radio' \
gst-launch-1.0 playbin uri=file:///home/root/Phone_Ringing.wav
```

## 5.2. GStreamer client

Here is an example of the property setup for GStreamer pulsesink client:

```
gst-launch-1.0 playbin uri=file:///home/root/celtic_irish.mp3 \
audio-sink="pulsesink stream-properties=props,media.role=navi"
```

# 6. Externals references

[1] ***Murphy Pulseaudio module for IVI***
https://github.com/otcshare/pulseaudio-module-murphy-ivi

[2] ***PulseAudio module API***
https://freedesktop.org/wiki/Software/PulseAudio/Documentation/Developer/ModuleAPI/

[3] ***PulseAudio routing module Yocto recipe in AGL***
meta-agl/recipes-multimedia/pulseaudio/agl-audio-plugin_0.1.bb

[4] ***PulseAudio routing module for AGL source code***
https://github.com/iotbzh/agl-audio-plugin.git

[5]: ***Murphy policy engine documentation***
https://01.org/sites/default/files/documentation/audio-policy-configuration.pdf