



Audio management

# 1 Contents

2	Terminology and concepts . . . . .	3
3	Standalone setup . . . . .	4
4	Hybrid setup . . . . .	4
5	Different audio sources for each domain . . . . .	4
6	Mixing, corking, ducking . . . . .	4
7	Playing, paused, stopped . . . . .	4
8	Use cases . . . . .	5
9	Application developer . . . . .	5
10	Car audio system . . . . .	5
11	Different types of sources . . . . .	5
12	Navigation instruction . . . . .	6
13	Traffic bulletin . . . . .	6
14	USB drive . . . . .	6
15	Rear sensor sound . . . . .	6
16	Blind spot sensor . . . . .	7
17	Seat belt . . . . .	7
18	Phone call . . . . .	7
19	Resume music . . . . .	7
20	VoIP . . . . .	7
21	Emergency call priority . . . . .	7
22	Mute . . . . .	7
23	Audio recording . . . . .	7
24	Microphone mute . . . . .	8
25	Application crash . . . . .	8
26	Web applications . . . . .	8
27	Control malicious application . . . . .	8
28	Multiple roles . . . . .	8
29	External audio router . . . . .	8
30	Non-use-cases . . . . .	8
31	Automatic actions on streams . . . . .	8
32	Streams' priorities . . . . .	8
33	Multiple independent systems . . . . .	9
34	Requirements . . . . .	9
35	Standalone operation . . . . .	9
36	Integrated operation . . . . .	9
37	Priority rules . . . . .	9
38	Multiple sound outputs . . . . .	10
39	Remember preempted source . . . . .	10
40	Audio recording . . . . .	10
41	Latency . . . . .	10
42	Security . . . . .	10
43	Muting output streams . . . . .	10
44	Muting input streams . . . . .	10
45	Control source activity . . . . .	10

46	Per stream priority . . . . .	11
47	GStreamer support . . . . .	11
48	Approach . . . . .	11
49	Stream metadata in applications . . . . .	11
50	Requesting permission to use audio in applications . . . . .	12
51	Audio routing principles . . . . .	12
52	Identification of applications . . . . .	13
53	Implementation of priority within streams . . . . .	13
54	Corking streams . . . . .	13
55	GStreamer support . . . . .	14
56	Remembering the previously playing stream . . . . .	14
57	Using different sinks . . . . .	14
58	Default media role . . . . .	14
59	Routing data structure example . . . . .	14
60	WirePlumber policy samples . . . . .	15
61	Testability . . . . .	18
62	Requirements . . . . .	18
63	Open questions . . . . .	19
64	Roles . . . . .	19
65	Policies . . . . .	19
66	Summary of recommendations . . . . .	20

67 Apertis audio management was previously built around PulseAudio but with  
68 the move to the Flatpak-based application framework [PipeWire](#)<sup>1</sup> offers a better  
69 match for the use-cases below. Compared to PulseAudio, PipeWire natively  
70 supports containerized applications and keeps policy management separate from  
71 the core routing system, making it much easier to tailor for specific products.

72 Applications can use PipeWire through [its native API](#)<sup>2</sup>, as the final layer to  
73 access sound features. This does not mean that applications have to deal directly  
74 with PipeWire: applications can still make use of their preferred sound APIs as  
75 intermediate layers for manipulating audio streams, with support being available  
76 for the PulseAudio API, for GStreamer or for the ALSA API.

77 In an analogous manner, applications can capture sound for various purposes.  
78 For instance, speech recognition or voice recorder applications may need to  
79 capture input from the microphone. The sound will be captured from PipeWire.  
80 ALSA users can use `pcm_pipewire`. GStreamer users can use `pipewiresrc`.

## 81 Terminology and concepts

82 See also the [Apertis glossary](#)<sup>3</sup> for background information on terminology.

---

<sup>1</sup><https://pipewire.org/>

<sup>2</sup>[https://docs.pipewire.org/page\\_api.html](https://docs.pipewire.org/page_api.html)

<sup>3</sup><https://www.apertis.org/glossary/>

83 **Standalone setup**

84 A standalone setup is an installation of Apertis which has full control of the  
85 audio driver. Apertis running in a virtual machine is an example of a standalone  
86 setup.

87 **Hybrid setup**

88 A hybrid setup is an installation of Apertis in which the audio driver is not fully  
89 controlled by Apertis. An example of this is when Apertis is running under  
90 an hypervisor or using an external audio router component such as [GENIVI  
91 audio manager]. In this case, the Apertis system can be referred to as Consumer  
92 Electronics domain (CE), and the other domain can be referred to as Automotive  
93 Domain (AD).

94 **Different audio sources for each domain**

95 Among others, a standalone Apertis system can generate the following sounds:

- 96 • Application sounds
- 97 • Bluetooth sounds, for example music streamed from a phone or voice call  
98 sent from a handsfree car kit
- 99 • Any kind of other event sounds, for example somebody using the SDK can  
100 generate event sounds using an appropriate command line

101 A hybrid Apertis system can generate the same sounds as a standalone sys-  
102 tem, plus some additional sounds not always visible to Apertis. For example,  
103 hardware sources further down the audio pipeline such as:

- 104 • FM Radio
- 105 • CD Player
- 106 • Driver assistance systems

107 In this case, some interfaces should be provided to interact with the additional  
108 sound sources.

109 **Mixing, corking, ducking**

110 *Mixing* is the action of playing simultaneously from several sound sources.

111 *Corking* is a request from the audio router to pause an application.

112 *Ducking* is the action of lowering the volume of a background source, while  
113 mixing it with a foreground source at normal volume.

114 **Playing, paused, stopped**

115 *Playing* describes the stream state when an audio stream is played.

116 *Paused* describes the state where an ongoing audio stream is suspended. When  
117 resuming, the stream shall restart from the point where it has been paused, if  
118 possible.

119 *Stopped* describes the state where no audio output is played. When resuming,  
120 the stream starts from scratch.

## 121 Use cases

122 The following section lists examples of usages requiring audio management. It  
123 is not an exhaustive list, unlimited combinations exists. Discussion points will  
124 be highlighted at the end of some use cases.

### 125 Application developer

126 An application developer uses the SDK in a virtual machine to develop an  
127 application. He needs to play sounds. He may also need to record sounds or  
128 test their application on a reference platform. This is a typical standalone setup.

### 129 Car audio system

130 In a car, Apertis is running in a hypervisor sharing the processor with a real  
131 time operating system controlling the car operations. Apertis is only used for  
132 applications and web browsing. A sophisticated Hi-Fi system is installed under  
133 a seat and accessible via a network interface. This is a hybrid setup.

### 134 Different types of sources

135 Some systems classify application sound sources in categories. It's important to  
136 note that no standard exists for those categories.

137 Both standalone and hybrid systems can generate different sound categories.

138 **Example 1** In one system of interest, sounds are classified as *main sources*,  
139 and *interrupt sources*. Main sources will generally represent long duration sound  
140 sources. The most common case are media players, but it could be sound sources  
141 emanating from web radio, or games. As a rule of thumb, the following can be  
142 used: when two main sources are playing at the same time, neither is intelligible.  
143 Those will often require an action from the user to start playing, should it be  
144 turn ignition on, press a play button on the steering wheel or the touchscreen.  
145 As a consequence, only policy mechanisms ensure that only one main source can  
146 be heard at a time.

147 Interrupt sources will generally represent short duration sound sources, they  
148 are emitted when an unsolicited event occurs. This could be when a message is  
149 received in any application or email service.

150 **Example 2** In another system of interest, sounds are classified as *main*  
151 *sources*, *interrupt sources* and *chimes*. Unlike the first example, in this system,  
152 a source is considered a main source if it is an infinite or loopable source, which  
153 can only be interrupted by another main source such as FM radio or CD player.  
154 Interrupt sources are informational sources such as navigation instructions, and  
155 chimes are unsolicited events of short duration. Each of these sound sources  
156 is not necessarily generated by an application. It could come from a system  
157 service instead.

#### 158 **Navigation instruction**

159 While some music from FM Radio is playing, a new navigation instruction has  
160 to be given to the driver: the navigation instructions should be mixed with the  
161 music.

#### 162 **Traffic bulletin**

163 Many audio sources can be paused. For example, a CD player can be paused,  
164 as can media files played from local storage (including USB mass storage), and  
165 some network media such as Spotify.

166 While some music from one of these sources is playing, a new traffic bulletin  
167 is issued: the music could be paused and the traffic bulletin should be heard.  
168 When it is finished, the music can continue from the point where the playback  
169 was paused.

170 By their nature, some sound sources cannot be paused. For example, FM or  
171 DAB radio cannot be paused.

172 While some music from a FM or DAB radio is playing, a new traffic bulletin  
173 is issued. Because the music cannot be paused, it should be silenced and the  
174 traffic bulletin should be heard. When it is finished, the music can be heard  
175 again.

176 Bluetooth can be used when playing a game or watching live TV. As with the  
177 radio use-case, Bluetooth cannot be paused.

#### 178 **USB drive**

179 While some music from the radio is playing, a new USB drive is inserted. If  
180 setting *automatic playback from USB drive* is enabled, the Radio sound stops  
181 and the USB playback begins.

#### 182 **Rear sensor sound**

183 While some music from the radio is playing, the driver selects rear gear, the rear  
184 sensor sound can be heard mixed with the music.

185 **Blind spot sensor**

186 While some music from Bluetooth is playing, a car passes through the driver's  
187 blind spot: a short notification sound can be mixed with the music.

188 **Seat belt**

189 While some music from the CD drive is playing, the passenger removes their  
190 seat belt: a short alarm sound can be heard mixed with the music.

191 **Phone call**

192 While some music from the CD drive is playing, a phone call is received: the  
193 music should be paused to hear the phone ringing and being able to answer the  
194 conversation. In this case, another possibility could be to notify the phone call  
195 using a ring sound, mixed in the music, and then pause the music only if the  
196 call is answered.

197 **Resume music**

198 If music playback has been interrupted by a phone call and the phone call has  
199 ended, music playback can be resumed.

200 **VoIP**

201 The driver wishes to use internet telephony/VoIP without noticing any difference  
202 due to being in a car.

203 **Emergency call priority**

204 While a phone call to emergency services is ongoing, an app-bundle process  
205 attempts to initiate lower-priority audio playback, for example playing music.  
206 The lower-priority audio must not be heard. The application receives the infor-  
207 mation that it cannot play.

208 **Mute**

209 The user can press a [mute hard-key](#)<sup>4</sup>. In this case, and according to OEM-  
210 specific rules, all sources of a specific category could be muted. For example, all  
211 *main* sources could be muted. The OEM might require that some sources are  
212 never muted even if the user pressed such a hard-key.

213 **Audio recording**

214 Some apps might want to initiate speech recognition. They need to capture  
215 input from a microphone.

---

<sup>4</sup><https://www.apertis.org/concepts/distribution/hardkeys/>

## 216 **Microphone mute**

217 If the user presses a “mute microphone” button (sometimes referred to as a “se-  
218 crecy” button) during a phone call, the sound coming from the microphone  
219 should be muted. If the user presses this button in an application during a  
220 video conference call, the sound coming from the microphone should be muted.

## 221 **Application crash**

222 The Internet Radio application is playing music. It encounters a problem and  
223 crashes. The audio manager should know that the application no longer exists.  
224 In an hybrid use case, the other audio routers could be informed that the audio  
225 route is now free. It is then possible to fall back to a default source.

## 226 **Web applications**

227 Web applications should be able to play a stream or record a stream.

## 228 **Control malicious application**

229 An application should not be able to use an audio role for which it does not  
230 have permission. For example, a malicious application could try to simulate a  
231 phone call and deliver advertising.

## 232 **Multiple roles**

233 Some applications can receive both a standard media stream and traffic infor-  
234 mation.

## 235 **External audio router**

236 In order to decide priorities, an external audio router can be involved. In this  
237 case, Apertis would only be providing a subset of the possible audio streams,  
238 and an external audio router could take policy decisions, to which Apertis could  
239 only conform.

## 240 **Non-use-cases**

### 241 **Automatic actions on streams**

242 It is not the purpose of this document to discuss the action taken on a media  
243 when it is preempted by another media. Deciding whether to cork or silence a  
244 stream is a user interface decision. As such it is OEM dependent.

### 245 **Streams' priorities**

246 The audio management framework defined by this document is intended to  
247 provide mechanism, not policy: it does not impose a particular policy, but  
248 instead provides a mechanism by which OEMs can impose their chosen policies.



## 249 **Multiple independent systems**

250 Some luxury cars may have multiple IVI touchscreens and/or sound systems,  
251 sometimes referred to as **multi-seat**<sup>5</sup> (please note that this jargon term comes  
252 from desktop computing, and one of these “seats” does not necessarily correspond  
253 to a space where a passenger could sit). We will assume that each of these “seats”  
254 is a separate container, virtual machine or physical device, running a distinct  
255 instance of the Apertis CE domain.

## 256 **Requirements**

### 257 **Standalone operation**

258 The audio manager must support standalone operation, in which it accesses  
259 audio hardware directly ( **Application developer**).

### 260 **Integrated operation**

261 The audio manager must support integrated operation, in which it cannot access  
262 the audio hardware directly, but must instead send requests and audio streams  
263 to the hybrid system. ( **Different types of sources**, **External audio router**).

### 264 **Priority rules**

265 It must be possible to implement OEM-specific priority rules, in which it is  
266 possible to consider one stream to be higher priority than another.

267 When a lower-priority stream is pre-empted by a higher-priority stream, it must  
268 be possible for the OEM-specific rules to choose between at least these actions:

- 269 • silence the lower-priority stream, with a notification to the application so  
270 that it can pause or otherwise minimise its resource use (corking)
- 271 • leave the lower-priority stream playing, possibly with reduced volume  
272 (ducking)
- 273 • terminate the lower-priority stream altogether

274 It must be possible for the audio manager to lose the ability to play audio  
275 (audio resource deallocation). In this situation, the audio manager must notify  
276 the application with a meaningful error.

277 When an application attempts to play audio and the audio manager is unable  
278 to allocate a necessary audio resource (for example because a higher-priority  
279 stream is already playing), the audio manager must inform the application using  
280 an appropriate error message. ( **Emergency call priority**)

---

<sup>5</sup>[https://www.apertis.org/concepts/archive/application\\_security/multiuser/#multi-seat-logind-seats](https://www.apertis.org/concepts/archive/application_security/multiuser/#multi-seat-logind-seats)

281 **Multiple sound outputs**

282 The audio manager should be able to route sounds to several sound outputs. (  
283 [Different types of sources](#)).

284 **Remember preempted source**

285 It should be possible for an audio source that was preempted to be remembered  
286 in order to resume it after interruption. This is not a necessity for all types  
287 of streams. Some OEM-specific code could select those streams based on their  
288 roles. ([Traffic bulletin](#), [Resume music](#))

289 **Audio recording**

290 App-bundles must be able to record audio if given appropriate permission. (  
291 [Audio recording](#))

292 **Latency**

293 The telephony latency must be as low as possible. The user must be able to  
294 hold a conversation on the phone or in a VoIP application without noticing any  
295 form of latency. ([VoIP](#))

296 **Security**

297 If some faulty or malicious application tries to play or record an audio stream  
298 for which permission wasn't granted, the proposed audio management design  
299 should not allow it. ([Application crash](#), [Control malicious application](#))

300 **Muting output streams**

301 During the time an audio source is preempted, the audio framework must notify  
302 the application that is providing it, so that the application can make an attempt  
303 to reduce its resource usage. For example, a DAB radio application might stop  
304 decoding the received DAB data. ([Mute](#), [Traffic bulletin](#))

305 **Muting input streams**

306 The audio framework should be able to mute capture streams. During that  
307 time, the audio framework must notify the application that are using it, so  
308 that the application can update user interface and reduce its resource usage. (  
309 [Microphone mute](#))

310 **Control source activity**

311 Audio management should be able to set each audio source to the playing,  
312 stopped or paused state based on priority. ([Resume music](#))

### 313 **Per stream priority**

314 We might want to mix and send multiple streams from one application to the  
315 automotive domain. An application might want to send different types of alert.  
316 For instance, a new message notification may have higher priority than ‘some  
317 contact published a new photo’. ( [Multiple roles](#))

### 318 **GStreamer support**

319 PipeWire includes 2 GStreamer elements called `pipewiresrc` and `pipewiresink`,  
320 which can be used in GStreamer’s pipelines.

321 PipeWire provides a device monitor as well so that `gst-device-monitor-1.0`  
322 shows the PipeWire devices and a camera application will automatically use  
323 the PipeWire video source when possible.

### 324 **Approach**

325 PulseAudio embeds a default audio policy so, for instance, if you plug an head-  
326 set on your laptop aux slot, it silences the laptop speakers. PipeWire has no  
327 embedded logic to do that, and relies on something else to control it, which  
328 suites the needs for Apertis better since it also targets special use-cases that  
329 don’t really match the desktop ones, and this separation brings more flexibility.

330 [WirePlumber](#)<sup>6</sup> is a service that provides the policy logic for PipeWire. It’s  
331 where the policies like the one above is implemented, but unlike PulseAudio is  
332 explicitly designed to let people define them using LUA scripts and they are  
333 also what AGL has used to replace their previous audio manager in their latest  
334 [Happy Halibut 8.0.0 release](#)<sup>7</sup>.

335 The overall approach is to adopt WirePlumber as the reference solution, but the  
336 separation between audio management and audio policy means that product  
337 teams can replace it with a completely different implementation with ease.

### 338 **Stream metadata in applications**

339 PipeWire provides the ability to attach metadata to a stream. The func-  
340 tion `pw_fill_stream_properties()`<sup>8</sup> is used to attach metadata to a stream  
341 during creation. The current convention in usage is to use a metadata  
342 named `media.role`, which can be set to values describing the nature of the  
343 stream, such as `Movie`, `Music`, `Camera`, `Notification`, …(defined in [PipeWire](#)’  
344 [s PW\\_KEY\\_MEDIA\\_ROLE](#)<sup>9</sup>), but not limited to them. This list of roles  
345 should be well defined between applications and WirePlumber.

---

<sup>6</sup><https://pipewire.pages.freedesktop.org/wireplumber/>

<sup>7</sup>[https://wiki.automotivelinux.org/agl-distro/release-notes#happy\\_halibut](https://wiki.automotivelinux.org/agl-distro/release-notes#happy_halibut)

<sup>8</sup><https://docs.pipewire.org/>

<sup>9</sup>[https://docs.pipewire.org/group\\_\\_\\_pw\\_\\_keys.html#ga7e7dcf769f9e253b0e3cde6534feed69](https://docs.pipewire.org/group___pw__keys.html#ga7e7dcf769f9e253b0e3cde6534feed69)

346 See also [GStreamer support](#).

### 347 **Requesting permission to use audio in applications**

348 Each audio role is associated with a permission. Before an application can start  
349 playback a stream, the audio manager will check whether it has the permission  
350 to do so. See [Identification of applications](#). [Application bundle metadata](#)<sup>10</sup>  
351 describes how to manage the permissions requested by an application. The  
352 application can also use bundle metadata to store the default role used by all  
353 streams in the application if this is not specified at the stream level.

### 354 **Audio routing principles**

355 The request to open an audio route is emitted in two cases:

- 356 • when a new stream is created
- 357 • before a stream changes state from Paused to Playing (uncork)

358 In both cases, before starting playback, the audio manager must check the  
359 priority against the business rules, or request the appropriate priority to the  
360 external audio router. If the authorization is not granted, the application should  
361 stop trying to request the stream and notify the user that an undesirable event  
362 occurred.

363 If an application stops playback, the audio manager will be informed. It will in  
364 turn notify the external audio router of the new situation, or handle it according  
365 to business rules.

366 An application that has playback can be requested to pause by the audio man-  
367 ager, for example if a higher priority sound must be heard.

368 Applications can use the PipeWire event API to subscribe to events. In partic-  
369 ular, applications can be notified about their mute status. If an event occurs,  
370 such as mute or unmute, the callback will be executed. For example, an applica-  
371 tion playing media from a source such as a CD or USB storage would typically  
372 respond to the mute event by pausing playback, so that it can later resume from  
373 the same place. An application playing a live source such as on-air FM radio  
374 cannot pause in a way that can later be resumed from the same place, but would  
375 typically respond to the mute event by ceasing to decode the source, so that it  
376 does not waste CPU cycles by decoding audio that the user will not hear.

377 **Standalone routing module maps streams metadata to priority** An  
378 internal priority module can be written. This module would associate a priority  
379 to all different streams' metadata. It is loaded statically from the config file.  
380 See [Routing data structure example](#) for an example of data structure.

---

<sup>10</sup>[https://www.apertis.org/concepts/archive/application\\_framework/application-bundle-metadata/](https://www.apertis.org/concepts/archive/application_framework/application-bundle-metadata/)

381 **Hybrid routing module maps stream metadata to external audio**  
382 **router calls** In the hybrid setup, the audio routing functions could be im-  
383 plemented in a separate module that maps audio events to automotive domain  
384 calls. However this module does not perform the priority checks. Those are  
385 executed in the automotive domain because they can involve a different feature  
386 set.

### 387 **Identification of applications**

388 Flatpak applications are wrapped in containers and are identified by an unique  
389 app-id which can be used by the policy manager. Such app-id is encoded in the  
390 name of the [transient systemd scope wrapping each application instance](#)<sup>11</sup> and  
391 can be retrieved easily.

392 If AppArmor support is added to Flatpak, AppArmor profiles could also be  
393 used to securely identify applications.

394 **Web application support** Web applications are just like any other applica-  
395 tion. However, the web engine JavaScript API does not provide a way to select  
396 the media role. All streams emanating from the same web application bundle  
397 would thus have the same role. Since each web application is running in its own  
398 process, AppArmor can be used to differentiate them. Web application support  
399 for corking depends on the underlying engine. WebKitGTK+ has the necessary  
400 support. See [changeset 145811](#)<sup>12</sup>.

### 401 **Implementation of priority within streams**

402 The policy manager should be able to cork streams: when a new stream with a  
403 certain role is started, all other streams within a user defined list of roles will  
404 get corked.

### 405 **Corking streams**

406 Depending on the audio routing policy, audio streams might be “corked”,  
407 “ducked” or simply silenced (moved to a null sink).

408 As long as the role is properly defined, the application developer does not have  
409 to worry about what happens to the stream except corking. Corking is part of  
410 PipeWire API and can happen at any time. Corking *should* be supported by  
411 applications. It is even possible that a stream is corked before being started.

412 If an application is not able to cork itself, the audio manager should enforce  
413 corking by muting the stream as soon as possible. However, this has the side  
414 effect that the stream between the corking request and the effective corking  
415 in the application will be lost. A threshold delay can be implemented to give  
416 an application enough time to cork itself. The policy of the external audio

---

<sup>11</sup><https://github.com/flatpak/flatpak/wiki/Sandbox#the-current-flatpak-sandbox>

<sup>12</sup><https://trac.webkit.org/changeset/145811>

417 manager must also be considered: if this audio manager has already closed the  
418 audio route when notifying the user, then the data will already be discarded. If  
419 the audio manager synchronously requests pause, then the application can take  
420 appropriate time to shutdown.

421 **Ensuring a process does not overrides its priorities** Additionally to  
422 request a stream to cork, a stream could be muted so any data still being  
423 received would be silenced.

#### 424 **GStreamer support**

425 GStreamer support is straightforward. `pipewiresink` support the `stream-`  
426 `properties` parameter. This parameter can be used to specify the `media.role`.  
427 The GStreamer pipeline states already changes from `GST_STATE_PLAYING` to  
428 `GST_STATE_PAUSED` when corking is requested.

#### 429 **Remembering the previously playing stream**

430 If a stream was playing and has been preempted, it may be desirable to switch  
431 back to this stream after the higher priority stream is terminated. To that effect,  
432 when a new stream start playing, a pointer to the stream that was currently  
433 playing (or an id) could be stored in a stack. The termination of a playing  
434 stream could restore playback of the previously suspended stream.

#### 435 **Using different sinks**

436 A specific `media.role` metadata value should be associated to a priority and a  
437 target sink. This allows to implement requirements of a sink per stream category.  
438 For example, one sink for main streams and another sink for interrupt streams.  
439 The default behavior is to mix together all streams sent to the same sink.

#### 440 **Default media role**

441 If an audio stream does not have the `media.role` property set, the policy will  
442 assign the `Default` media role name to it. In addition to this, if the `Default`  
443 endpoint can not be found, the policy will link the stream audio node with the  
444 **lowest** priority endpoint.

445 This allows users to assign a particular endpoint for streams that don't have the  
446 `media.role` property set.

#### 447 **Routing data structure example**

448 The following table document routing data for defining a A-IVI inspired stream  
449 routing. This is an example, and in an OEM variant of Apertis it would be  
450 replaced with the business rules that would fulfill the OEM's requirements

451 App-bundle metadata defines whether the application is allowed to use this  
 452 audio role, if not defined, the application is not allowed to use the role. From  
 453 the role, priorities between stream could be defined as follows:

454 In a standalone setup:

role	priority	sink	action
music	0 (lowest)	main_sink	cork
phone	7 (highest)	main_sink	cork
ringtone	7 (highest)	alert_sink	mix
customringtone	7 (highest)	main_sink	cork
new_email	1	alert_sink	mix
traffic_info	6	alert_sink	mix
gps	5	main_sink	duck

455 In a hybrid setup, the priority would be expressed in a data understandable  
 456 by the automotive domain. The action meaning would be only internal to CE  
 457 domain. Since the CE domain do not know what is happening in the automotive  
 458 domain.

role	priority	sink	action
music	MAIN_APP1	main_sink	cork
phone	MAIN_APP2	main_sink	cork
ringtone	MAIN_APP3	alert_sink	mix
customringtone	MAIN_APP3	main_sink	cork
new_email	ALERT1	alert_sink	mix
traffic_info	INFO1	alert_sink	mix
gps	INFO2	main_sink	mix

## 459 WirePlumber policy samples

460 All the policies in WirePlumber are completely scriptable and written in Lua.  
 461 The Lua API Documentation can be found [here](#)<sup>13</sup>.

462 The default roles, priorities and related actions are defined in `/usr/share/wireplumber/policy.lua.d/50-`  
 463 `endpoints-config.lua` and can be re-written to `/etc/wireplumber/policy.lua.d/50-`  
 464 `endpoints-config.lua` to support the standalone setup defined in [Routing data](#)  
 465 [structure example](#):

```
466 default_policy.policy.roles = {
467     -- main sink
468     ["Multimedia"] = { ["priority"] = 0, ["action.default"] = "cork", ["alias"] = { "Movie", "Music", "Game" }, },
469     ["GPS"] = { ["priority"] = 5, ["action.default"] = "duck", },
```

<sup>13</sup>[https://pipewire.pages.freedesktop.org/wireplumber/scripting/lua\\_api.html](https://pipewire.pages.freedesktop.org/wireplumber/scripting/lua_api.html)

```

470 ["Phone"]          = { ["priority"] = 7, ["action.default"] = "cork", ["alias"] = { "CustomRingtone" }, },
471
472 -- alert sink
473 ["New_email"]      = { ["priority"] = 1, ["action.default"] = "mix", },
474 ["Traffic_info"]   = { ["priority"] = 6, ["action.default"] = "mix", },
475 ["Ringtone"]       = { ["priority"] = 7, ["action.default"] = "mix", },
476 }
477
478 default_policy.endpoints = {
479 ["endpoint.multimedia"] = { ["media.class"] = "Audio/Sink", ["role"] = "Multimedia", },
480 ["endpoint.gps"]        = { ["media.class"] = "Audio/Sink", ["role"] = "GPS", },
481 ["endpoint.phone"]      = { ["media.class"] = "Audio/Sink", ["role"] = "Phone", },
482 ["endpoint.ringtone"]   = { ["media.class"] = "Audio/Sink", ["role"] = "Ringtone", },
483 ["endpoint.new_email"]  = { ["media.class"] = "Audio/Sink", ["role"] = "New_email", },
484 ["endpoint.traffic_info"] = { ["media.class"] = "Audio/Sink", ["role"] = "Traffic_info", },
485 }

```

486 And, for example, a policy to automatically switch Bluetooth from A2DP to  
487 HSP/HFP profile when a specific application starts, e.g. Zoom, could be defined  
488 like:

```

489 #!/usr/bin/wpexec
490 --
491 -- WirePlumber
492 --
493 -- Copyright © 2021 Collabora Ltd.
494 --   @author George Kiagiadakis <george.kiagiadakis@collabora.com>
495 --
496 -- SPDX-License-Identifier: MIT
497 --
498 -- This is an example of a standalone policy making script. It can be executed
499 -- either on top of another instance of wireplumber or pipewire-media-session,
500 -- as a standalone executable, or it can be placed in WirePlumber's scripts
501 -- directory and loaded together with other scripts.
502 --
503 -- The script basically watches for a client application called
504 -- "ZOOM VoiceEngine", and when it appears (i.e. Zoom starts), it switches
505 -- the profile of all connected bluetooth devices to the "headset-head-unit"
506 -- (a.k.a HSP Headset Audio) profile. When Zoom exits, it switches again the
507 -- profile of all bluetooth devices to A2DP Sink.
508 --
509 -- The script can be customized further to look for other clients and/or
510 -- change the profile of a specific device, by customizing the constraints.
511 -----
512 -
513
514 devices_om = ObjectManager {

```



```

515     Interest { type = "device",
516         Constraint { "device.api", "=", "bluez5" },
517     }
518 }
519
520 clients_om = ObjectManager {
521     Interest { type = "client",
522         Constraint { "application.name", "=", "ZOOM VoiceEngine" },
523     }
524 }
525
526 function set_profile(profile_name)
527     for device in devices_om:iterate() do
528         local index = nil
529         local desc = nil
530
531         for profile in device:iterate_params("EnumProfile") do
532             local p = profile:parse()
533             if p.properties.name == profile_name then
534                 index = p.properties.index
535                 desc = p.properties.description
536                 break
537             end
538         end
539
540         if index then
541             local pod = Pod.Object {
542                 "Spa:Pod:Object:Param:Profile", "Profile",
543                 index = index
544             }
545
546             print("Setting profile of '"
547                 .. device.properties["device.description"]
548                 .. "' to: " .. desc)
549             device:set_params("Profile", pod)
550         end
551     end
552 end
553
554 clients_om:connect("object-added", function (om, client)
555     print("Client '" .. client.properties["application.name"] .. "' connected")
556     set_profile("headset-head-unit")
557 end)
558
559 clients_om:connect("object-removed", function (om, client)
560     print("Client '" .. client.properties["application.name"] .. "' disconnected")

```

```
561     set_profile("a2dp-sink")
562 end)
563
564 devices_om:activate()
565 clients_om:activate()
```

## 566 **Testability**

567 The key point to keep in mind for testing is that several applications can execute  
568 in parallel and use PipeWire APIs (and the library API) concurrently. The  
569 testing should try to replicate this. However testing possibilities are limited  
570 because the testing result depends on the audio policy.

571 **Application developer testing** The application developer is requested to  
572 implement corking and error path. Testing those features will depend on the  
573 policy in use.

574 Having a way to identify the *lowest* and *highest* priority definition in the policy  
575 could be enough for the application developer. Starting a stream with the lowest  
576 priority would not succeed if a stream is already running. Starting a stream with  
577 the highest priority would cork all running streams.

578 The developer may benefit from the possibility to customize the running policy.

579 **Testing the complete design** Testability of the complete design must be  
580 exercised from application level. It consist of emulating several applications  
581 each creating independent connections with different priorities, and verifying  
582 that the interactions are reliable. The policy module could be provisionned  
583 with a dedicated test policy for which the results are already known.

## 584 **Requirements**

585 This design fulfill the following requirements:

- 586 • **Standalone operation** and **Integrated operation** are provided using separate sets of configuration files.
- 587
- 588 • **Priority rules** are provided by the policy manager.
- 589 • the audio manager library interface is aware of **Multiple sound outputs**.
- 590 • **Remember preempted source** can be implemented in the policy manager.
- 591 • **Audio recording** will use the same mechanisms.
- 592 • **Latency** is implemented by not adding additional audio processing layer.
- 593 • **Security** is implemented by relying on the Flatpak containerization, which could be further hardened by adding AppArmor support.
- 594
- 595 • **Muting output streams** and **Control source activity** uses PipeWire corking infrastructure.
- 596
- 597 • **Per stream priority** uses the PipeWire API.
- 598 • **GStreamer support** is provided indirectly thanks to existing plugins.

599 **Open questions**

600 **Roles**

- 601 • Do we need to define roles that the application developer can use?

602 It's not possible to guarantee that an OEM's policies will not nullify an  
603 audio role that is included in Apertis. However, if we do not provide  
604 some roles, there is no hope of ever having an application designed for one  
605 system work gracefully on another.

- 606 • Should we define roles for input?

607 Probably, yes, speech recognition input could have a higher priority than  
608 phone call input. (Imagine the use case where someone is taking a call,  
609 is not currently talking on the call, and wants to change their navigation  
610 destination: they press the speech recognition hard-key, tell the navigation  
611 system to change destination, then input switches back to the phone call.)

- 612 • Should we define one or several audio roles not requiring permission for  
613 use?

614 No, it is explicitly recommended that every audio role requires permission.  
615 An app-store curator from the OEM could still give permission to every  
616 application requesting a role.

617 **Policies**

- 618 • How can we ensure matching between the policy and application defined  
619 roles?

620 Each permission in the permission set should be matched with a media  
621 role. The number of different permissions should be kept to a minimum.

- 622 • Should applications start stream corked?

623 It must be done on both the application side and the audio manager side.  
624 Applications cannot be trusted. As soon as a stream opens, the PipeWire  
625 process must cork it - before the first sample comes out. Otherwise a ma-  
626 licious application could play undesirable sounds or noises while the audio  
627 manager is still thinking about what to do with that stream. The au-  
628 dio manager might be making this decision asynchronously, by asking for  
629 permission from the automotive domain. The audio manager can choose  
630 uncork, leave corked or kill, according to its policies. On the application  
631 side, it is only possible to *suggest* the best way for an application to behave  
632 in order to obtain the best user experience.

- 633 • Should we use `media.role` or define an apertis specific stream property?

## 634 Summary of recommendations

- 635 • PipeWire is adopted as audio router and WirePlumber as policy manager.
- 636 • Applications keep using the PulseAudio API or higher level APIs like
- 637 GStreamer to be compatible with the legacy system.
- 638 • The default WirePlumber policy is extended to address the use-cases de-
- 639 scribed here.
- 640 • Static sets of configuration files can implement different policies depending
- 641 on hybrid setup or standalone setup.
- 642 • Each OEM must derive from those policies to implement their business
- 643 rules.
- 644 • WirePlumber must be modified to check that the application have the
- 645 permission to use the requested role and, if the `media.role` is not provided
- 646 in the stream, it must check if a default value is provided in the application
- 647 bundle metadata.
- 648 • If AppArmor support is made available in Flatpak, WirePlumber must be
- 649 modified to check for AppArmor identity of client applications.
- 650 • The application bundle metadata contains a default audio role for all
- 651 streams within an application.
- 652 • The application bundle metadata must contain a permission request for
- 653 each audio role in use in an application.
- 654 • For each stream, an application can choose an audio role and communicate
- 655 it to PipeWire at stream creation.
- 656 • The policy manager monitors creation and state changes of streams.
- 657 • Depending on business rules, the policy manager can request an applica-
- 658 tion to cork or mute.
- 659 • GStreamer's `pipewiresink` support a `stream.properties` parameter.
- 660 • A tool for corking a stream could be implemented.