



Audio management

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

46	GStreamer support . . . . .	9
47	Approach . . . . .	9
48	Streams metadata in applications . . . . .	10
49	Requesting permission to use audio in applications . . . . .	10
50	Audio routing principles . . . . .	10
51	Identification of applications . . . . .	11
52	Implementation of priority within streams . . . . .	11
53	Corking streams . . . . .	12
54	GStreamer support . . . . .	12
55	Remembering the previously playing stream . . . . .	12
56	Using different sinks . . . . .	12
57	Routing data structure example . . . . .	13
58	Testability . . . . .	13
59	Requirements . . . . .	14
60	Open questions . . . . .	14
61	Roles . . . . .	14
62	Policies . . . . .	15
63	Summary of recommendations . . . . .	15

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

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

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

## 78 Terminology and concepts

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

### 80 Standalone setup

81 A standalone setup is an installation of Apertis which has full control of the  
82 audio driver. Apertis running in a virtual machine is an example of a standalone  
83 setup.

---

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

<sup>2</sup><https://pipewire.github.io/pipewire/>

<sup>3</sup><https://jwd.pages.apertis.org/apertis-website/glossary/>

## 84 Hybrid setup

85 A hybrid setup is an installation of Apertis in which the audio driver is not fully  
86 controlled by Apertis. An example of this is when Apertis is running under an  
87 hypervisor or using an external audio router component such as [GENIVI audio  
88 manager](#)<sup>4</sup>. In this case, the Apertis system can be referred to as Consumer  
89 Electronics domain (CE), and the other domain can be referred to as Automotive  
90 Domain (AD).

## 91 Different audio sources for each domain

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

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

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

- 101 • FM Radio
- 102 • CD Player
- 103 • Driver assistance systems

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

## 106 Mixing, corking, ducking

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

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

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

## 111 Use cases

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

## 115 Application developer

116 An application developer uses the SDK in a virtual machine to develop an  
117 application. He needs to play sounds. He may also need to record sounds or

---

<sup>4</sup><http://docs.projects.genivi.org/AudioManager/>

118 test their application on a reference platform. This is a typical standalone setup.

### 119 **Car audio system**

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

### 124 **Different types of sources**

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

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

### 128 **Example 1**

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

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

### 141 **Example 2**

142 In another system of interest, sounds are classified as *main sources*, *interrupt*  
143 *sources* and *chimes*. Unlike the first example, in this system, a source is con-  
144 sidered a main source if it is an infinite or loopable source, which can only be  
145 interrupted by another main source such as FM radio or CD player. Interrupt  
146 sources are informational sources such as navigation instructions, and chimes  
147 are unsolicited events of short duration. Each of these sound sources is not  
148 necessarily generated by an application. It could come from a system service  
149 instead.

### 150 **Navigation instruction**

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

154 **Traffic bulletin**

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

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

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

164 While some music from a FM or DAB radio is playing, a new traffic bulletin  
165 is issued. Because the music cannot be paused, it should be silenced and the  
166 traffic bulletin should be heard. When it is finished, the music can be heard  
167 again.

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

170 **USB drive**

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

174 **Rear sensor sound**

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

177 **Blind spot sensor**

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

180 **Seat belt**

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

183 **Phone call**

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

189 **Resume music**

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

192 **VoIP**

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

195 **Emergency call priority**

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

200 **Mute**

201 The user can press a [mute hard-key](#)<sup>5</sup>. In this case, and according to OEM-  
202 specific rules, all sources of a specific category could be muted. For example, all  
203 *main* sources could be muted. The OEM might require that some sources are  
204 never muted even if the user pressed such a hard-key.

205 **Audio recording**

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

208 **Microphone mute**

209 If the user presses a “mute microphone” button (sometimes referred to as a  
210 “secrecy” button) during a phone call, the sound coming from the microphone  
211 should be muted. If the user presses this button in an application during a video  
212 conference call, the sound coming from the microphone should be muted.

213 **Application crash**

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

218 **Web applications**

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

---

<sup>5</sup><https://jwd.pages.apertis.org/apertis-website/concepts/hardkeys/>

## 220 **Control malicious application**

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

## 224 **Multiple roles**

225 Some applications can receive both a standard media stream and traffic infor-  
226 mation.

## 227 **External audio router**

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

## 232 **Non-use-cases**

### 233 **Automatic actions on streams**

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

### 237 **Streams' priorities**

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

### 241 **Multiple independent systems**

242 Some luxury cars may have multiple IVI touchscreens and/or sound systems,  
243 sometimes referred to as [multi-seat](#)<sup>6</sup> (please note that this jargon term comes  
244 from desktop computing, and one of these “seats” does not necessarily corre-  
245 spond to a space where a passenger could sit). We will assume that each of  
246 these “seats” is a separate container, virtual machine or physical device, run-  
247 ning a distinct instance of the Apertis CE domain.

## 248 **Requirements**

### 249 **Standalone operation**

250 The audio manager must support standalone operation, in which it accesses  
251 audio hardware directly ( [Application developer](#)).

<sup>6</sup><https://jwd.pages.apertis.org/apertis-website/concepts/multiuser/#multi-seat-logind-seats>



## 252 **Integrated operation**

253 The audio manager must support integrated operation, in which it cannot access  
254 the audio hardware directly, but must instead send requests and audio streams  
255 to the hybrid system. ( [Different types of sources](#), [External audio router](#)).

## 256 **Priority rules**

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

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

- 261 • silence the lower-priority stream, with a notification to the application so  
262 that it can pause or otherwise minimise its resource use (corking)
- 263 • leave the lower-priority stream playing, possibly with reduced volume  
264 (ducking)
- 265 • terminate the lower-priority stream altogether

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

269 When an application attempts to play audio and the audio manager is unable  
270 to allocate a necessary audio resource (for example because a higher-priority  
271 stream is already playing), the audio manager must inform the application using  
272 an appropriate error message. ( [Emergency call priority](#) )

## 273 **Multiple sound outputs**

274 The audio manager should be able to route sounds to several sound outputs. ( [Different types of sources](#) ).  
275

## 276 **Remember preempted source**

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

## 281 **Audio recording**

282 App-bundles must be able to record audio if given appropriate permission. ( [Audio recording](#) )  
283

## 284 **Latency**

285 The telephony latency must be as low as possible. The user must be able to  
286 hold a conversation on the phone or in a VoIP application without noticing any

287 form of latency. ( [VoIP](#) )

## 288 **Security**

289 The audio manager should not trust applications for managing audio. If some  
290 faulty or malicious application tries to play or record an audio stream for which  
291 permission wasn't granted, the proposed audio management design should not  
292 allow it. ( [Application crash](#), [Control malicious application](#) )

## 293 **Muting output streams**

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

## 298 **Muting input streams**

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

## 303 **Control source activity**

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

## 306 **Per stream priority**

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

## 311 **GStreamer support**

312 GStreamer should be supported.

## 313 **Approach**

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

319 [WirePlumber](#)<sup>7</sup> is a service that provides the policy logic for PipeWire. It's where  
320 the default policy like the one above is implemented, but unlike PulseAudio is  
321 explicitly designed to let people customize it. PipeWire and WirePlumber is  
322 what AGL has used to replace their previous audio manager in their latest  
323 Happy Halibut 8.0.0 release.

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

### 327 Streams metadata in applications

328 PipeWire provides the ability to attach metadata to a stream. The function  
329 `pw_fill_stream_properties()`<sup>8</sup> is used to attach metadata to a stream during cre-  
330 ation. The current convention in usage is to use a metadata named `media.role`,  
331 which can be set to values describing the nature of the stream, such as `Movie`,  
332 `Music`, `Camera`, `Notification`, and other defined by PipeWire.

333 See also [GStreamer support].

### 334 Requesting permission to use audio in applications

335 Each audio role is associated with a permission. Before an application can start  
336 playback a stream, the audio manager will check whether it has the permission  
337 to do so. See [Identification of applications]. [Application bundle metadata](#)<sup>9</sup>  
338 describes how to manage the permissions requested by an application. The  
339 application can also use bundle metadata to store the default role used by all  
340 streams in the application if this is not specified at the stream level.

### 341 Audio routing principles

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

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

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

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

---

<sup>7</sup><https://gitlab.freedesktop.org/pipewire/wireplumber>

<sup>8</sup>[https://pipewire.github.io/pipewire/classpw\\_\\_\\_pipewire.html#a841dbb7608dc9cdda4a320d33fbbd39a](https://pipewire.github.io/pipewire/classpw___pipewire.html#a841dbb7608dc9cdda4a320d33fbbd39a)

<sup>9</sup><https://jwd.pages.apertis.org/apertis-website/concepts/application-bundle-metadata/>

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

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

#### 364 **Standalone routing module maps streams metadata to priority**

365 An internal priority module can be written. This module would associate a  
366 priority to all different streams' metadata. It is loaded statically from the config  
367 file. See [Routing data structure example] for an example of data structure.

#### 368 **Hybrid routing module maps stream metadata to external audio 369 router calls**

370 In the hybrid setup, the audio routing functions could be implemented in a  
371 separate module that maps audio events to automotive domain calls. However  
372 this module does not perform the priority checks. Those are executed in the  
373 automotive domain because they can involve a different feature set.

#### 374 **Identification of applications**

375 Flatpak applications are wrapped in containers and are identified by an unique  
376 app-id which can be used by the policy manager. Such app-id is encoded in the  
377 name of the [transient systemd scope wrapping each application instance](#)<sup>10</sup> and  
378 can be retrieved easily.

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

#### 381 **Web application support**

382 Web applications are just like any other application. However, the web engine  
383 JavaScript API does not provide a way to select the media role. All streams  
384 emanating from the same web application bundle would thus have the same  
385 role. Since each web application is running in its own process, AppArmor can  
386 be used to differentiate them. Web application support for corking depends on  
387 the underlying engine. WebKitGTK+ has the necessary support. See [changeset  
388 145811](#)<sup>11</sup>.

---

<sup>10</sup><https://github.com/flatpak/flatpak/wiki/Sandbox#the-current-flatpak-sandbox>

<sup>11</sup><https://trac.webkit.org/changeset/145811>

389 **Implementation of priority within streams**

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

393 **Corking streams**

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

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

400 If an application is not able to cork itself, the audio manager should enforce  
401 corking by muting the stream as soon as possible. However, this has the side  
402 effect that the stream between the corking request and the effective corking  
403 in the application will be lost. A threshold delay can be implemented to give  
404 an application enough time to cork itself. The policy of the external audio  
405 manager must also be considered: if this audio manager has already closed the  
406 audio route when notifying the user, then the data will already be discarded. If  
407 the audio manager synchronously requests pause, then the application can take  
408 appropriate time to shutdown.

409 **Ensuring a process does not overrides its priorities**

410 Additionally to request a stream to cork, a stream could be muted so any data  
411 still being received would be silenced.

412 **GStreamer support**

413 GStreamer support is straightforward. `pipewiresink` support the `stream-`  
414 `properties` parameter. This parameter can be used to specify the `media.role`.  
415 The GStreamer pipeline states already changes from `GST_STATE_PLAYING` to  
416 `GST_STATE_PAUSED` when corking is requested.

417 **Remembering the previously playing stream**

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

423 **Using different sinks**

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

429 **Routing data structure example**

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

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

436 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

437 In a hybrid setup, the priority would be expressed in a data understandable  
438 by the automotive domain. The action meaning would be only internal to CE  
439 domain. Since the CE domain do not know what is happening in the automotive  
440 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

## 441 **Testability**

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

## 446 **Application developer testing**

447 The application developer is requested to implement corking and error path.  
448 Testing those features will depend on the policy in use.

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

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

## 454 **Testing the complete design**

455 Testability of the complete design must be exercised from application level. It  
456 consist of emulating several applications each creating independent connections  
457 with different priorities, and verifying that the interactions are reliable. The  
458 policy module could be provisioned with a dedicated test policy for which the  
459 results are already known.

## 460 **Requirements**

461 This design fullfill the following requirements:

- 462 • [Standalone operation] and [Integrated operation] are provided using sep-  
463 arate sets of configuration files.
- 464 • [Priority rules] are provided by the policy manager.
- 465 • the audio manager library interface is aware of [Multiple sound outputs].
- 466 • [Remember preempted source] can be implemented in the policy manager.
- 467 • [Audio recording] will use the same mechanisms.
- 468 • [Latency] is implemented by not adding additional audio processing layer.
- 469 • [Security] is implemented by relying on the Flatpak containerization,  
470 which could be further hardened by adding AppArmor support.
- 471 • [Muting output streams] and [Control source activity] uses PipeWire cork-  
472 ing infrastructure.
- 473 • [Per stream priority] uses the PipeWire API.
- 474 • [GStreamer support] is provided indirectly thanks to existing plugins.

## 475 Open questions

### 476 Roles

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

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

- 482 • Should we define roles for input?

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

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

490 No, it is explicitly recommended that every audio role requires permis-  
491 sion. An app-store curator from the OEM could still give permission to  
492 every application requesting a role.

### 493 Policies

- 494 • How can we ensure matching between the policy and application defined  
495 roles?

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

- 498 • Should applications start stream corked?

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

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



## 510 Summary of recommendations

- 511 • PipeWire is adopted as audio router and WirePlumber as policy manager.
- 512 • Applications keep using the PulseAudio API or higher level APIs like
- 513 GStreamer to be compatible with the legacy system.
- 514 • The default WirePlumber policy is extended to address the use-cases de-
- 515 scribed here.
- 516 • Static sets of configuration files can implement different policies depending
- 517 on hybrid setup or standalone setup.
- 518 • Each OEM must derive from those policies to implement their business
- 519 rules.
- 520 • WirePlumber must be modified to check that the application have the
- 521 permission to use the requested role and, if the `media.role` is not provided
- 522 in the stream, it must check if a default value is provided in the application
- 523 bundle metadata.
- 524 • If AppArmor support is made available in Flatpak, WirePlumber must be
- 525 modified to check for AppArmor identity of client applications.
- 526 • The application bundle metadata contains a default audio role for all
- 527 streams within an application.
- 528 • The application bundle metadata must contain a permission request for
- 529 each audio role in use in an application.
- 530 • For each stream, an application can choose an audio role and communicate
- 531 it to PipeWire at stream creation.
- 532 • The policy manager monitors creation and state changes of streams.
- 533 • Depending on business rules, the policy manager can request an applica-
- 534 tion to cork or mute.
- 535 • GStreamer's `pipewiresink` support a `stream.properties` parameter.
- 536 • A tool for corking a stream could be implemented.