



Text To Speech

1 Contents

2	Text To Speech	2
3	Introduction	2
4	Terminology and concepts	2
5	Text to speech (TTS)	2
6	Voice	2
7	Use cases	2
8	News application	2
9	Back in a news application	3
10	New e-mail notification	3
11	New e-mail notification then going back	3
12	New meeting notification then cancelled	3
13	Incoming phone call	4
14	Voice installed with the SDK	4
15	Installable voice bundle	4
16	Voice backend in the automotive domain	4
17	Installable languages	4
18	Voice configuration	4
19	Per-request emphasis	4
20	Non-phonetic place names	4
21	Driving abroad	5
22	Multiple concurrent TTS requests	5
23	Permissions to use TTS API	5
24	Multiple output speakers	5
25	Custom TTS implementation in an application	5
26	Non-use-cases	6
27	Accessibility for users with reduced vision	6
28	Requirements	6
29	Basic TTS API	6
30	Progress signalling API	6
31	Output policy decided by audio manager	6
32	Output streams are mixable	7
33	Runtime-swappable voice backends	7
34	Installable voice backends	7
35	Default SDK voice backend	7
36	Voice backends are not latency sensitive	8
37	System-wide voice configuration	8
38	Pronunciation data for non-phonetic words	8
39	Per-request language support	8
40	Support for concurrent requests	8
41	Prioritisation for concurrent requests	9
42	Output routing policy	9
43	Permission for using TTS system	9
44	Existing text to speech systems	9
45	Android	10

46	iOS	10
47	Previous eCore TTS API	10
48	speech-dispatcher	11
49	TTS voices	11
50	Approach	12
51	Overall architecture	13
52	Alternative centralised design	13
53	Use of speech-dispatcher	13
54	TTS library	14
55	Installable and swappable backends	14
56	SDK default backend	15
57	Global configuration	15
58	Per-request configuration	16
59	Sound icons	16
60	Request prioritisation	16
61	PulseAudio output	17
62	Testability	18
63	Security	19
64	Suggested roadmap	20
65	Requirements	20
66	Summary of recommendations	21
67	Appendix A: Suggested TTS API	21

68 **Text To Speech**

69 **Introduction**

70 This documents possible approaches to designing an API for text to speech
71 (TTS) services for an Apertis system in a vehicle.

72 This document proposes an API for the text to speech service in **Appendix: A**
73 **suggested TTS API**. This API is not finalised, and is merely a suggestion. It
74 may be refined in future versions of this document, or when implementation is
75 started.

76 The major considerations with a TTS API are:

- 77 • Simple API for applications to use
- 78 • Swappable voices through the application bundling system and application
79 store
- 80 • Output priorities controlled by the same set of audio manager policies
81 which control other application audio output

82 **Terminology and concepts**

83 **Text to speech (TTS)**

84 *Text to speech* (TTS) is the process of converting a string of text into spoken
85 words in the user's language, to be outputted as an audio stream.

86 **Voice**

87 In the context of TTS, a *voice* is an engine for producing spoken words. As with
88 the conventional meaning of the word, the voice may have certain characteristics,
89 such as gender, regionality or manners of speech. The most important quality
90 of a voice is its understandability and correctness of pronunciation.

91 **Use cases**

92 A variety of use cases for application usage of TTS services are given below.
93 Particularly important discussion points are highlighted at the bottom of each
94 use case.

95 **News application**

96 The user has installed a news application, and wants it to read the headlines
97 and articles aloud as they drive. If they are waiting in a traffic queue, they want
98 to be able to quickly find the current paragraph in the article on-screen so they
99 can read it themselves to speed things up.

100 **Back in a news application**

101 The user has a news reader application open on a specific article, which is being
102 read aloud. The user presses the back button to close the article and return
103 to the list of headlines. TTS output needs to stop for that article. If an audio
104 source was playing before the user started reading the article (for example, some
105 music), its playback may be resumed where it was paused.

106 **New e-mail notification**

107 The user's e-mail client is reading an e-mail aloud to the user, scrolling the
108 e-mail as reading progresses. A new e-mail arrives, which causes a 'new e-mail'
109 notification to be sent to the TTS system.

110 The OEM wants control over the policy of how the two TTS requests are played:

- 111 • The system could pause reading the original e-mail, read the notification,
112 then resume reading the original e-mail; or
- 113 • it could pause reading the original e-mail, read the notification, then *not*
114 resume reading the original e-mail; or
- 115 • it could continue reading the original e-mail at a lower volume, and read
116 the notification louder mixed over the top.

117 The OEM wants these policies to not be overridable by any application-specific
118 policy such as the ones described in [New e-mail notification then going back](#),
119 [New meeting notification then cancelled](#), [Incoming phone call](#).

120 **New e-mail notification then going back**

121 The user's e-mail client is reading an e-mail aloud to the user, scrolling the
122 e-mail as reading progresses. A new e-mail arrives, which causes a 'new e-mail'
123 notification to be sent to the TTS system. This pauses reading the original
124 e-mail and starts reading the notification, as notifications have a higher priority
125 than reading e-mails.

126 While the notification is being read, the user presses the 'back' button to go back
127 to their list of e-mails. This should cancel reading out the old e-mail (which is
128 currently paused), but should not cancel the 'new e-mail' notification, which is
129 still being played.

130 **New meeting notification then cancelled**

131 The user's e-mail client is reading them an invitation to a meeting. While read-
132 ing the invitation, the meeting is cancelled by the organiser, and a notification
133 is displayed informing the user of this. This notification is read by the TTS
134 system, interrupting it reading the original meeting invitation. Once the notifi-
135 cation has finished being read, the e-mail client should not resume reading the
136 original invitation.

137 **Incoming phone call**

138 The user's e-mail client is reading an e-mail aloud to the user. Part-way through
139 reading, a phone call is received. TTS output for the e-mail needs to be auto-
140 matically paused while the phone ringtone is played and the call takes place.
141 Once the call has finished, the e-mail application may want to continue reading
142 the user's e-mail aloud, or may cancel its output.

143 **Voice installed with the SDK**

144 A developer wants to develop an application using the SDK with TTS function-
145 ality, and needs to test it using a voice available in the SDK.

146 **Installable voice bundle**

147 A user does not like how the default TTS voice for their vehicle sounds, and
148 wishes to change it to another voice which they can download from the Apertis
149 application store. They wish this new voice to be used by default in future.

150 **Voice backend in the automotive domain**

151 An OEM may wish to provide a proprietary TTS voice as part of the software
152 in their automotive domain. They want this voice to be used as the default for
153 TTS requests from the CE domain as well.

154 **Installable languages**

155 A vehicle has already been released in various countries, but the OEM wishes to
156 expand into other countries. They need to add support for additional languages
157 to the TTS system.

158 **Voice configuration**

159 The user finds that the TTS system reads text too slowly for them, and they
160 wish to speed it up. They edit their system preferences to increase the speed,
161 and want this to take effect across all applications which use TTS.

162 **Per-request emphasis**

163 A news reader application needs to differentiate between TTS output for article
164 headings and bodies. It wishes to read headings slightly louder and more slowly
165 than it reads bodies. However, the application must not be allowed to make
166 TTS requests so loud that they distract the driver.

167 **Non-phonetic place names**

168 The navigation application is reading turn-by-turn route guidance aloud, includ-
169 ing place names. Various place names are not pronounced phonetically, and the
170 navigation system needs to make sure the TTS system pronounces them cor-
171 rectly.

172 **Driving abroad**

173 When driving abroad, the navigation application needs to read the instructions
174 “Turn left at the next junction, signposted ‘Paris nord’”, a sentence which con-
175 tains both English and French. The speech in each language should be pro-
176 nounced using the correct pronunciation rules for that language.

177 **Multiple concurrent TTS requests**

178 The user is listening to their e-reader read a book aloud using TTS, while they
179 are driving and using the audio turn-by-turn instructions from the navigation
180 application. Whenever the navigation application needs to read an instruction,
181 the e-reader output should be temporarily paused or its volume reduced, and
182 resumed after the navigation instruction has been read, so that the user doesn’t
183 get confused.

184 *It is understood that the current quality of TTS implementations is not sufficient*
185 *to read an e-book to the user without causing them significant discomfort. This*
186 *use case is intended to demonstrate the need for the system to handle multiple*
187 *pending TTS requests. E-reader output may become possible in the future.*

188 **Permissions to use TTS API**

189 The user has installed a game application for their passenger to play, and wants
190 to be sure that it will not start reading instructions aloud using the TTS service
191 while they are driving. They want to disallow the application permission to use
192 the TTS API — either entirely, or just while driving.

193 **Multiple output speakers**

194 A vehicle has a single main field speaker, plus two sets of headphones. Each
195 set of headphones is associated with a different head unit. TTS audio which
196 pertains to the entire system should be output through all three speakers; TTS
197 audio which pertains to an application only on one of the head units should
198 only be output through that head unit’s headphones.

199 **Custom TTS implementation in an application**

200 An application developer wants to port an existing application from another
201 platform to Apertis. The application is a large one, and has its own tightly
202 integrated TTS system which would output directly to the audio manager. This
203 must be possible.

204 **Non-use-cases**

205 The following use cases are not in scope to be handled by this design — but
206 they may be in scope to be handled by other components of Apertis. Further
207 details are given in each subsection below.

208 **Accessibility for users with reduced vision**

209 While TTS is often used in software to provide accessibility for users with re-
210 duced vision, who otherwise cannot see the graphical UI clearly, that is not a
211 goal of the TTS system in Apertis. It is intended to reduce driver distraction by
212 reducing the need for the driver to look at the graphical UI, rather than making
213 the UI more accessible.

214 **Requirements**

215 **Basic TTS API**

216 Implement a basic TTS API with support for speaking text; and pausing, re-
217 suming and cancelling specific requests.

218 See [News application](#), [Back in a news application](#), [New e-mail notification then](#)
219 [going back](#).

220 **Progress signalling API**

221 The TTS system must be able to signal an application as output progresses
222 through the current request. Signals must be supported for output start and
223 end, and may be supported for per-word progress through the text. Signals
224 must also be supported for pausing and resuming output.

225 These signals are intended to be used to update the client application's UI to
226 correspond to the output progress. For example, if a notification is being read
227 aloud, the notification window should be hidden when, and only when, output
228 is finished.

229 See [News application](#), [New e-mail notification then going back](#)

230 **Output policy decided by audio manager**

231 The policy deciding which TTS requests are played, which are paused, when
232 they are resumed, and which are cancelled altogether, must be determined by
233 the system's audio manager.

234 An application may be able to implement its own policy (for example, to always
235 cancel a TTS request if it is paused), but it must not be able to override the
236 audio manager's policy, for example by preventing a request from being paused,
237 or by increasing the priority of a request so it is played in preference to another.

238 If the audio manager corks a TTS output stream (for example, if all audio output
239 needs to be stopped in order to handle a phone call), the TTS daemon must
240 pause the corresponding client application request, and notify the application.

241 Once the output stream is uncorked, the client application request must be
242 resumed, and the application notified, unless the application has cancelled that
243 request in the meantime. By cancelling the request in the signal handler, a client
244 application can ensure that TTS output is not resumed after the stream would
245 have been uncorked, allowing for various resumption policies to be implemented.

246 See [New e-mail notification then going back](#), [New meeting notification then](#)
247 [cancelled](#), [Incoming phone call](#).

248 **Output streams are mixable**

249 Multiple TTS audio streams from within a single application, and from multiple
250 applications, must be mixable by the audio manager, to allow implementing the
251 policy of lowering the volume of one stream while playing a more important
252 stream over the top.

253 See [New e-mail notification](#).

254 **Runtime-swappable voice backends**

255 The TTS system must support different voice backends. Only one backend
256 has to be active at once, but backends must be swappable at runtime if, for
257 example, the user installs a new voice from the store, or if the OEM installs a
258 voice backend supporting more languages (requirement 5.6).

259 TTS requests queued or being output at the time a new voice backend is selected
260 should continue using the old voice. New TTS requests should use the new voice.

261 See [Voice installed with the SDK](#), [Voice configuration](#).

262 **Installable voice backends**

263 The user must be able to install additional voices from the Apertis application
264 store; and an OEM must be able to install additional voices before sale of
265 a vehicle to support additional languages. These voices must be available to
266 choose as the default for all TTS output.

267 See [Installable voice bundle](#), [Installable languages](#).

268 **Default SDK voice backend**

269 A voice backend must be shipped with the SDK by default, to allow application
270 development against the TTS system.

271 See [Voice installed with the SDK](#).

272 **Voice backends are not latency sensitive**

273 Some vehicles may have a TTS voice backend implemented in the automotive
274 domain, which means all TTS requests would be carried over the inter-domain
275 communications link, incurring some latency. The TTS system must not be
276 sensitive to this latency.

277 See [Voice backend in the automotive domain](#).

278 **System-wide voice configuration**

279 The system must have a single default voice, which is used for all TTS out-
280 put. The configuration settings for this voice must be settable in the system
281 preferences, but not settable by individual applications.

282 Specific preferences, such as volume or speech rate, may be settable on a per-
283 application basis to modify the system-wide defaults if needed. These mod-
284 ifications must have limited ability to distract the driver. For example, an
285 application may apply a modifier to the volume of between 0.8 and 1.2 times
286 the current system-wide output volume.

287 See [Voice configuration](#).

288 **Pronunciation data for non-phonetic words**

289 There must be a way for applications to provide pronunciations for non-phonetic
290 words. This may be implemented as a static list of overrides for certain words,
291 or may be implemented as a runtime API. Pronunciations must be associated
292 with a specific language, so that the correct pronunciation is used for the user's
293 current system language. If no more suitable pronunciation is available for a
294 word, the system must use the current voice's default pronunciation.

295 See [Non-phonetic place names](#), [Driving abroad](#).

296 **Per-request language support**

297 The TTS system must support specifying the language of each request (or even
298 parts of a request), so that requests which contain text in multiple languages
299 (for example 'Turn left onto Rue de Rivoli') are pronounced correctly.

300 The system language should be used by default if the application doesn't specify
301 a language, or if the specified language is not supported by the current voice.

302 See [Driving abroad](#).

303 **Support for concurrent requests**

304 The TTS system must support accepting TTS output requests from multiple
305 applications concurrently, and queueing them for output sequentially.

306 See [Multiple concurrent TTS requests](#).

307 **Prioritisation for concurrent requests**

308 The TTS system must support prioritising TTS requests from certain appli-
309 cations over requests from other applications, according to the urgency of the
310 output (for example, turn-by-turn navigation instructions are more urgent than
311 news reading). Similarly, it must support prioritising requests from within a
312 single application.

313 Prioritisation must be performed on a per-request basis, as one application
314 may make requests which are high and low priority. Note that this does not
315 necessarily mean that the priority policy is implemented in the TTS system; it
316 may be implemented in the audio manager. This requirement simply means that
317 the TTS API must expose support for prioritising requests, and must forward
318 that prioritisation information as 'hints' to whichever component implements
319 the priority policy.

320 See [Multiple concurrent TTS requests](#).

321 **Output routing policy**

322 On high-end vehicles, there may be multiple output speakers, attached to differ-
323 ent head units. The audio manager must be able to associate each TTS request
324 with an application so that it can determine which speaker or speakers to play
325 the audio on.

326 See [Multiple output speakers](#).

327 **Permission for using TTS system**

328 Applications must only be allowed to use the TTS system if they are allowed to
329 output audio. This is subject to the application's permissions from its manifest,
330 and may additionally be subject to the user's preferences for audio output. The
331 user may be able to temporarily disable audio output for a specific application.

332 If any TTS-specific permissions are implemented in the system, it must be
333 understood that an application may circumvent them by embedding its own
334 TTS system (or by playing pre-recorded audio files, for example).

335 See [Permissions to use TTS API, Custom TTS implementation in an applica-](#)
336 [tion](#).

337 **Existing text to speech systems**

338 This chapter describes the approaches taken by various existing systems for
339 allowing applications to use TTS services, because it might be useful input for
340 Apertis' decision making. Where available, it also provides some details of the
341 implementations of features that seem particularly interesting or relevant.

342 **Android**

343 [Android provides a text to speech API](#)¹ for converting text to audio to output,
344 or to audio in a file.

345 It provides an API for matching pieces of text with custom pre-recorded sounds
346 (which it calls 'earcons'), for the purpose of embedding custom noises (such as
347 ticking noises) into TTS output, or for providing custom pronunciations for the
348 text.

349 It supports voices which support different languages, and provides the union of
350 those languages to the developer, who may specify which language the provided
351 text is in.

352 The user controls the preferences for the voice, apart from pitch and speech
353 rate, which applications may set individually.

354 For determining the progress of the TTS engine through an utterance, the API
355 provides a callback function which is called on starting and ending audio output.

¹<http://developer.android.com/reference/android/speech/tts/package-summary.html>

356 iOS

357 iOS provides TTS support through its [speech synthesiser API](#)². In this API, text
358 to be spoken is passed to a new utterance object, which allows its voice, volume,
359 speech rate and pitch to be modified. The utterance is then passed to the service,
360 which queues it up to be spoken, or starts speaking it if nothing else is queued.
361 Methods on the service allow output to be paused, cancelled or resumed. When
362 pausing speech, the API provides the option to pause immediately, or after
363 finishing speaking the current word.

364 Progress through speaking an utterance can be tracked using a delegate, which
365 receives calls when speech starts, stops, is paused, resumes, and for each word
366 in the text as it is spoken (intended for the purposes of highlighting words
367 on-screen).

368 It is worth noting that iOS is recognised as highly competent in the field of
369 accessibility for the blind or partially sighted, partly due to its well designed
370 TTS system.

371 Previous eCore TTS API

372 The TTS API previously exposed by eCore gave a method to speak a given
373 string of text, a method to stop speaking, and one to check whether speech was
374 currently being output. It gave the choice of two voices, but no other options for
375 configuring them. It provided two signals for notifying of audio output starting
376 and ending.

377 speech-dispatcher

378 [speech-dispatcher](#)³ is an abstraction layer over multiple TTS voices. It uses a
379 client-server architecture, where multiple clients can connect and send text to
380 the server to be outputted as audio. The protocol used between clients and the
381 server is the [Speech Synthesis Interface Protocol](#)⁴, a custom text-based protocol
382 operated over a Unix domain socket.

383 Prioritisation between text from different clients is supported, but clients are
384 not strictly separated by the server: one client can control the settings and
385 output for another client.

386 The client library has C and Python APIs. The C API is pure C, and is not GLib-
387 based. The backend supports a few different voices (see [TTS voices](#)): Festival,
388 espeak, pico, and a few proprietary systems. Writing a new voice backend, to
389 connect an existing external voice engine to speech-dispatcher, is not a major
390 task.

²[https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/
AVSpeechSynthesizer_Ref/index.html](https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVSpeechSynthesizer_Ref/index.html)

³<http://devel.freebsoft.org/speechd>

⁴<http://devel.freebsoft.org/doc/speechd/ssip.html>

391 The system supports ‘sound icons’ which associate a sound file with a given text
392 string, and allow that sound to be played when that string is found in input.

393 The settings allow control over the output language, whether to speak punctu-
394 ation characters, the speech rate, pitch, and volume.

395 Speech output can be paused, resumed and cancelled once started. The API
396 supports notifying when output is started, stopped, and when pre-defined ‘index
397 marks’ are reached in the input string.

398 Backends for speech dispatcher are run as separate processes, communicating
399 with the daemon via stdin and stdout. They have individual configuration files.

400 **TTS voices**

401 Here is a brief comparative evaluation of various TTS engines and voices which
402 are available already.

403 **espeak**

- 404 • Supports many languages (importantly, non-Latin languages)
- 405 • Sounds robotic
- 406 • Can be used with mbrola voices to make it more natural; not supported
407 very well by speech-dispatcher (<http://espeak.sourceforge.net/mbrola.html>)
- 408
- 409 • Already packaged for Ubuntu (as are mbrola voices)
- 410 • <http://espeak.sourceforge.net/>

411 **Festival**

- 412 • Sounds less robotic than espeak, but still quite robotic (example [here](#)⁵)
- 413 • A bit slower
- 414 • Already packaged for Ubuntu
- 415 • Supports 3 languages (English, Spanish and Welsh)
- 416 • <http://www.cstr.ed.ac.uk/projects/festival/>

417 **pico**

- 418 • License: Apache License v2
- 419 • By SVOX; used in Android
- 420 • Written in Java; C API available in picoapi.h
- 421 • Supports 37 languages (importantly, non-Latin languages)
- 422 • Sounds very good (example here: <https://svoxmobilevoices.wordpress.com/demos/>)
- 423
- 424 • Not as well tested through speech-dispatcher
- 425 • <https://en.wikipedia.org/wiki/SVOX>

⁵<https://www.cstr.ed.ac.uk/projects/festival/onlinedemo.html>

- 426 • Publicly available source; <https://android.googlesource.com/platform/external/svox/>
- 427
- 428 • Already packaged for Debian and Ubuntu
- 429 • As this is a component of Android, we are not sure about the openness
- 430 of the development practices, and whether it's possible to get involved in
- 431 them.
- 432 • It's certainly possible to file bugs about the packaging with the [Debian](#)
- 433 [bug tracker](#)⁶, but that won't necessarily help for bugs in the source itself.

434 **acapela**

- 435 • Non-FOSS
- 436 • Best quality
- 437 • <http://www.acapela-group.com/>

438 **Nuance**

- 439 • Non-FOSS
- 440 • Has been used previously in eCore
- 441 • [Demo](#)⁷

442 **Approach**

443 Based on the above research ([Existing text-to-speech systems](#)) and [Requirements](#), we recommend the following approach as an initial sketch of a text to
444 speech system. A suggested API for the TTS service is given in [Appendix: A](#)
445 [suggested TTS API](#).

447 **Overall architecture**

448 As TTS output from an application is essentially another audio stream, and
449 no privilege separation is required for turning a string of text into an audio
450 stream, the design follows a 'decentralised' pattern similar to how GStreamer is
451 implemented.

452 In order to produce TTS output, an application can link to a TTS library,
453 which provides functionality for turning a text string into an audio stream. It
454 then outputs this audio stream as it would any other, sending it to the audio
455 manager, along with some metadata including an unforgeable identifier for the
456 application, and potentially other metadata hints for debugging purposes. The
457 audio manager applies the same priority policy which it applies to all audio
458 streams, and determines whether to allow the stream to be played, pause it
459 while another stream is played then resume it, or cancel it entirely. This is done
460 using standard audio manager mechanisms using PulseAudio.

⁶<https://bugs.debian.org/cgi-bin/pkgreport.cgi?pkg=libttspic0;dist=unstable>

⁷<https://www.nuance.com/omni-channel-customer-engagement/voice-and-ivr/text-to-speech.html#!>

461 The TTS library receives feedback about the state of the audio channel, and
462 passes this back to the application in the form of signals, which the application
463 may use to update its UI, or implement its own policy for enqueueing or cancelling
464 requests (or it may ignore the signals).

465 **Alternative centralised design**

466 The other major option is for a centralised design, where all TTS requests are
467 sent to a TTS service (running as a separate process), which decides on relative
468 priorities for them, converts them from text to audio, and forwards them to the
469 audio manager.

470 There is no need for this design: there is no need for the additional privilege
471 separation, and it complicates the application of audio policy, since it now has
472 to be applied in the TTS service *and* the audio manager.

473 **Use of speech-dispatcher**

474 **Speech dispatcher** is an existing FOSS system which is the standard choice
475 for systems like this. However, it is based around a centralised design which
476 does not fit with our suggested architecture — a large part of speech-dispatcher
477 is concerned with implementing a central daemon which handles connections
478 and requests from multiple clients, prioritises them, then outputs them to the
479 audio manager. As described in **Overall architecture** and **Alternative centralised
480 design**, this is functionality which our recommended design does not need.

481 Additionally, speech-dispatcher has the disadvantages that it:

- 482 • does not enforce separation between clients, meaning they may control
483 each others' output; and
- 484 • provides a C API which is not GLib-based, so would be hard to introspect
485 and expose in other languages (such as JavaScript).

486 For these reasons, and due to its centralised architecture, we recommend *not*
487 using speech-dispatcher. However, it may be possible and useful to extract
488 relevant parts of its code and turn them into shared libraries to be used in the
489 Apertis TTS library. The rest of this document will cover the design with no
490 reference to speech-dispatcher, in the knowledge that it might substitute for
491 some of the implementation work where possible.

492 **TTS library**

493 The TTS library would be a new shared library which can be linked into appli-
494 cations to essentially provide the functionality of turning a text string into an
495 audio stream. It would provide the following major APIs:

- 496 • Say a text string.
- 497 • Stop, pause and resume speech.

- 498 • Signal on starting, pausing, resuming and ending audio output, plus on
499 significant progress through output.
- 500 • Set the language for a request.
- 501 • ‘Sound icon’ API for associating audio files with specific strings.

502 The stop, pause and resume APIs would operate on specific requests, rather
503 than all pending requests from the application. This allows for an application
504 to cancel one TTS output while continuing to say another; or to cancel one
505 output while another is paused. The API should be implemented as a queue-
506 based one, where the application enqueues a string to be read, and receives
507 a handle identifying it in the queue. The TTS library can prioritise requests
508 within the queue, and hence requests may not be processed for some time after
509 being enqueued. Signals convey this information to the application.

510 The progress signal should be emitted at the discretion of the TTS library, to
511 signal significant progress to the application in outputting the TTS request. For
512 example, it could be emitted once per sentence, or once per word, or not at all.
513 It returns an offset (in Unicode characters) from the start of the input text.

514 The library’s audio output would provided in a format suitable for passing
515 directly to PulseAudio, or into GStreamer for further processing.

516 The TTS library would implement loading of a TTS backend into the process,
517 and would load and apply the system settings for TTS output.

518 **Installable and swappable backends**

519 The TTS library would implement voice backends as dynamically loaded shared
520 libraries, all installed into a common directory. It must monitor this directory
521 at runtime to detect newly installed voice backends; for an application bundle
522 to install a new backend, it would have to install or symlink the library into this
523 directory.

524 The TTS library should not care how a voice backend is implemented internally,
525 as long as it implements a standard backend interface. It may be possible, for
526 example, to re-use a lot of the code from speech-dispatcher’s [backend modules](#)⁸.

527 Each voice backend must provide an interface for converting text to audio, and
528 returning that audio to the TTS library — it should *not* implement outputting
529 the audio to the audio manager itself. Backends must provide a way of enumer-
530 ating and configuring their voice options (such as volume, pitch, accent, etc.),
531 including a way of specifying that an option is read-only or unsupported. It is
532 not expected that all backends will support all functionality of the TTS library.

533 The backend interface must be tolerant of latency in the backends, in order
534 to support backends which are implemented in the automotive domain. This
535 means that all functions must be [asynchronous](#)⁹.

⁸<http://git.freebsoft.org/?p=speechd.git;a=tree;f=src/modules;hb=HEAD>

⁹<https://developer.gnome.org/gio/stable/GAsyncResult.html>

536 **SDK default backend**

537 We recommend [Pico](#)¹⁰ as the default backend to ship with the SDK. It is freely
538 licenced, and supports 37 languages including non-Latin languages. It is used
539 on Android, so is relatively stable and mature.

540 **Global configuration**

541 Configuration options for the voice backends should be stored in GSettings (See
542 [Preferences and Persistence](#)¹¹), and should be stored once (not per-backend).
543 The semantics of each configuration option must be rigorously defined, as each
544 backend must convert those options to fit its own configuration interface. If a
545 backend has more options in its configuration interface than are provided by the
546 global TTS library configuration, it must use sensible, unconfigurable, defaults
547 for the other options.

548 Configuration options may include:

- 549 • Voice to use
- 550 • Whether to vocalise punctuation
- 551 • Voice type (male or female)
- 552 • Speech rate
- 553 • Pitch
- 554 • Volume

555 By storing the options in GSettings, it becomes possible to apply AppArmor
556 policy to control access to them so that, for example, applications which use
557 the TTS library are only allowed to read the settings, and only the system
558 preferences application is allowed to modify them.

559 **Per-request configuration**

560 Configuration which is exposed to applications via the TTS API could be:

- 561 • Pitch
- 562 • Speech rate
- 563 • Volume

564 These options must be exposed purely as *modifiers* on the system-wide values.
565 These modifiers could be defined symbolically, for example as a set of three
566 volume modifiers:

- 567 • Emphasised (120% of system-wide volume)
- 568 • Normal (100% of system-wide volume)
- 569 • De-emphasised (80% of system-wide volume)

570 A non-symbolic numerical modifier might be introduced in future.

¹⁰<https://android.googlesource.com/platform/external/svox/>

¹¹<https://em.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/>

571 The audio manager is responsible for limiting the maximum volume of any audio
572 stream, to avoid a malicious or faulty application from setting the volume too
573 high as to distract the driver.

574 **Sound icons**

575 Sound icons are a feature provided by speech-dispatcher, which we could use as
576 the basis for our own implementation, as this would allow re-use of the relevant
577 features in voice backends.

578 Sound icons could be used for identifying punctuation, for example, or for clari-
579 fying the pronunciation of certain words. It's suggested that applications install
580 sound icons at install time, in a per-application directory which the application
581 points the TTS library at to look up when asked to play a sound icon. Each
582 sound icon should have an associated language (or explicitly no associated lan-
583 guage), so that the correct sound icon file can be loaded according to a TTS
584 request's language.

585 Sound icons should be playable via a TTS library API, similarly to how text
586 output is requested. They should be provided in WAV format, as this is what
587 the existing speech-dispatcher backends expect.

588 **Request prioritisation**

589 There are two dimensions to prioritisation of requests: within a single applica-
590 tion, and across multiple applications.

591 Requests from within a single application should be handled using a request
592 queue within the TTS library. This allows squashing similar requests, or bump-
593 ing other requests so they are played before other requests from the same appli-
594 cation.

595 It is suggested that the speech-dispatcher [priorities](#)¹² are used for within a single
596 application, including their semantics. For example, the TTS library request
597 queue would squash multiple progress requests so that only one is played at
598 once.

599 These priorities should be attached to audio output when it is sent to the audio
600 manager, as a hint to assist it in its policy decisions.

601 Requests from multiple applications are prioritised by the audio manager, which
602 uses the audio priority of each application (whether it is an entertainment or
603 interrupt source, and its numerical audio priority) from the application's man-
604 ifest to determine which requests to play, which to pause then resume, and
605 which to cancel entirely. The application's audio priority is under the control
606 of the OEM, rather than the application developer, so application developers
607 cannot use this to always output audio at an inflated priority and deny other
608 applications audio output.

¹²<http://devel.freebsoft.org/doc/speechd/ssip.html#Priority-Categories>

609 See the Audio Management design

610 There is one situation where an application with a low priority may need to
611 output a TTS request at a higher overall priority than an application with a
612 high priority: when emitting a pop-up notification via the notification service.
613 This should be handled by having notifications submitted as TTS requests by
614 the notification service itself, rather than by the application which produced
615 the notification. This allows the audio manager to use the notification service's
616 priority for policy decisions, rather than the original application's priority.

617 **PulseAudio output**

618 Output from the TTS library should be sent to PulseAudio in order to be mixed
619 with other TTS and non-TTS audio streams and sent to the hardware for output.
620 It is PulseAudio and the audio manager which implement the priority policies
621 described above.

622 In order to differentiate TTS output from different applications, appropriate
623 metadata should be attached to the audio stream to identify the application,
624 its internal priority for the TTS request, and the fact that the audio is a TTS
625 request (as opposed to other audio content). The application identifier must
626 be unforgeable (i.e. it must come from a trusted source, like the kernel or
627 D-Bus daemon), as it is used as the basis for policy decisions. The internal
628 priority and TTS request flag are entirely under the control of the application
629 (i.e. forgeable), and therefore must only be used as hints by the audio manager.
630 Additional unforgeable metadata may come from the application's manifest file,
631 which is not under the control of the application developer, and can be uniquely
632 looked up by the application's trusted identifier.

633 The audio manager most likely will *not* use forgeable metadata from the applica-
634 tion, but this data could be useful for identifying audio streams when debugging,
635 for example.

636 If an application wishes to submit multiple TTS requests simultaneously, and
637 have the audio manager mix them or decide which one to prioritise, it must
638 have multiple connections to PulseAudio.

639 If, as a result of applying the priority policy, the audio manager corks an ap-
640 plication's TTS output stream, the TTS library must pause the corresponding
641 TTS request and notify the application using a signal. Once the request is un-
642 corked, the TTS library must unpause the request and notify the application
643 again — unless the application has cancelled the request in the meantime, in
644 which case the request is already cancelled and removed.

645 The same is true if the audio manager *cancels* an application's TTS output
646 stream: the TTS library must cancel the corresponding TTS request and notify
647 the application using a signal.

648 Note that the audio manager's pausing and resuming of TTS requests is separate

649 from the pause and resume APIs available to the application. The application
650 cannot call its resume method to resume a TTS request which the audio manager
651 has paused. Similarly, the audio manager cannot call its resume method to
652 resume a TTS request which the application has paused. This can be thought
653 of as separately pausing or resuming both ends of the audio channel between
654 an application and the audio manager.

655 **Testability**

656 Testing the TTS system can be split into three major areas: checking that
657 the TTS library and its various voice backends work; checking that the audio
658 manager correctly applies its priority policies to incoming TTS audio streams
659 and normal audio streams; and integration testing of audio output from an
660 application calling a TTS API.

661 The former can be achieved using unit tests within the TTS library project,
662 which test various components of the library in isolation. For example, they
663 could compare TTS audio output streams against stored ‘golden’ expected out-
664 put sound files.

665 The audio manager testing should be implemented as part of the audio man-
666 ager’s test plan, ensuring that TTS audio channel metadata is included in a
667 variety of test situations.

668 This should be described in the Audio Management design.

669 Finally, the integration testing requires the audio output to be checked, so is
670 infeasible to implement as an automated test, and would have to be a manual
671 test where the human tester verifies that the output sounds as expected for a
672 given set of input situations (requests from a test client).

673 **Security**

674 The security properties being implemented by the system are:

- 675 • Applications should be independent, in that one application cannot change
676 the TTS settings for another application, or affect another application’s
677 TTS output other than through prioritisation of requests as controlled by
678 the audio manager.
- 679 • Applications must not be able to play a TTS request if the audio man-
680 ager has disallowed or paused it (availability of audio output to other
681 applications).
- 682 • Applications should not be able to set the system-wide TTS preferences.
- 683 • Applications should not be able to determine the content of other appli-
684 cations’ TTS requests (confidentiality of requests).
- 685 • Applications must only be allowed to use the TTS system if they have
686 permission to output audio.

687 These are implemented through the separation of audio priority policies from the
688 TTS library, by implementing them in the audio manager. The audio manager
689 has a non-forgeable identifier for the application which originated each TTS
690 audio stream, and the forgeable priority hints which come from the application
691 are not allowed to override the application's audio priority.

692 Audio output from an application is subject to that application having permis-
693 sion to output audio, which is enforced by the audio manager.

694 Independence and confidentiality of application audio channels is implemented
695 as for all audio channels, by having separate connections from each application
696 to the audio manager.

697 Integrity of system-wide TTS preferences is implemented by the AppArmor
698 policy controlling access to those preferences in GSettings.

699 **Loadable voice backends**

700 The TTS library, and hence each application which links to it, needs read-only
701 and execute access to the loadable voice backend libraries, plus any resources
702 needed by those voices. It also needs read-only access to the TTS system-wide
703 preferences in GSettings.

704 **Suggested roadmap**

705 There are few opportunities for splitting this system up into stages. The TTS
706 library needs to be written first, including its loadable voice backend interface
707 and the first voice backend. More complex features like sound icons could be
708 ignored in the first version of the library. With this working, applications could
709 start to use the TTS APIs. The unit tests and integration tests for the TTS
710 library should be written from the very beginning.

711 With TTS output working, a second stage could implement the priority policies
712 in the audio manager, and ensure those are working. The system preferences
713 could also be integrated at this stage.

714 A third stage could produce more voice backends (if needed), potentially includ-
715 ing a voice backend which is implemented in the automotive domain, to ensure
716 that asynchronous calls to the backends work.

717 It is worth highlighting that aside from initially ignoring features like sound
718 icons, there is little scope for simplifying the TTS API for its first implemen-
719 tation. Specifically, we feel it would be a mistake to implement a non-queue-
720 based API for scheduling TTS requests to begin with, and then 'expand' it into
721 a queue-based API later on. To do so would expose applications to a lot of
722 semantic changes in the API which they would then have to adapt to use. The
723 TTS library API should be implemented as a queue-based one from the start.

724 Requirements

- 725 • **Basic TTS API:** Implemented as a C API on the TTS library.
- 726 • **Progress signalling API:** Implemented using GObject signals emitted by
727 the TTS library.
- 728 • **Output policy decided by audio manager:** Implemented by passing priority
729 and application identifiers to the audio manager, and it corking, uncork-
730 ing, or cancelling audio streams according to its policy, using standard
731 PulseAudio functionality.
- 732 • **Output streams are mixable:** Audio manager may choose to *not* cork two
733 streams, and mix them instead.
- 734 • **Runtime-swappable voice backends:** TTS library loads backends from a
735 directory as dynamically loaded libraries, and monitors that directory for
736 changes.
- 737 • **Installable voice backends:** Installed or symlinked into the backend library
738 directory.
- 739 • **Default SDK voice backend:** Pico to be shipped as the default backend
740 for the SDK.
- 741 • **Voice backends are not latency sensitive:** Voice backend interface uses
742 asynchronous functions to avoid blocking the TTS library.
- 743 • **System-wide voice configuration:** Stored in GSettings and read by the
744 TTS library in each application which uses it. The system preferences
745 application can modify the settings in GSettings.
- 746 • **Pronunciation data for non-phonetic words:** Provided by an API in the
747 TTS library similar to the speech-dispatcher API for ‘sound icons’.
- 748 • **Per-request language support:** Provided as a per-request API to hint at
749 the language the source text is written in.
- 750 • **Support for concurrent requests:** Implemented by allowing multiple audio
751 channel connections to the audio manager, which prioritises between them.
- 752 • **Prioritisation for concurrent requests:** Implemented by allowing multiple
753 audio channel connections to the audio manager, which prioritises between
754 them. In-application priorities are handled by a per-application request
755 queue within the TTS library.
- 756 • **Permission for using TTS system:** Checked by the audio manager for each
757 application which attempts to play audio (including TTS output), using
758 permissions from the application’s manifest.

759 Summary of recommendations

760 As discussed in the above sections, we recommend:

- 761 • Implementing a new TTS library, using an API like the one suggested in
762 [Appendix: A suggested TTS API](#). Parts of speech-dispatcher may be used
763 to aid the implementation if appropriate.
- 764 • Implementing voice backends as dynamically loaded libraries, potentially
765 reusing much of the existing backends from speech-dispatcher.
- 766 • Modifying the audio manager to support applying a priority policy to TTS
767 requests, using the application's audio priority, and potentially logging
768 TTS-specific metadata for debugging purposes.
- 769 • Implementing unit and integration tests for the TTS library, audio man-
770 ager and TTS system as a whole.
- 771 • Packaging and using Pico as the default voice backend in the SDK.
- 772 • Modifying the Apertis software installer to generate AppArmor rules to
773 allow access to the TTS voice backends and their resources, plus the TTS
774 system settings, if an application is allowed to output audio.

775 **Appendix A: Suggested TTS API**

776 The code listing is given in pseudo-code.

```

777 /* TTS context to contain relevant state and loaded resources and
778  * settings. */
779 class TtsContext {
780     async TtsRequest send_request (const string text_to_say,
781                                   TtsPriority priority=TEXT,
782                                   const string language=null,
783                                   TtsVoiceRate voice_rate=TtsVoiceRate.NORMAL,
784                                   TtsVolume volume=TtsVolume.NORMAL,
785                                   TtsPitch pitch=TtsPitch.NORMAL);
786
787     async TtsRequest send_sound_icon_request (const string icon_name,
788                                               TtsPriority priority=TEXT,
789                                               const string language=null,
790                                               TtsVoiceRate voice_rate=TtsVoiceRate.NORMAL,
791                                               TtsVolume volume=TtsVolume.NORMAL,
792                                               TtsPitch pitch=TtsPitch.NORMAL);
793 }
794
795 /* This represents a single pending TTS request. The object may persist
796  * after the underlying request has been handled, until the application
797  * programmer unrefs the object. */
798 class TtsRequest {
799     async void pause ();
800     async void resume ();
801     async void cancel ();

```

```

802
803  /* The current state of the request. */
804  property TtsRequestState state;
805
806  /* The current progress of reading through the request, as an offset
807   * into the original text in Unicode characters. */
808  property unsigned int current_offset;
809
810  /* In a GLib API, these would be GObject::notify::state and
811   * GObject::notify::current_offset. */
812  signal notify_state (TtsRequestState state);
813  signal notify_current_offset (unsigned int current_offset);
814 }
815
816 enum TtsRequestState {
817     PREROLL,
818     PLAYING,
819     PAUSED,
820     FINISHED,
821     CANCELLED,
822 }
823
824 enum TtsPriority {
825     IMPORTANT,
826     MESSAGE,
827     TEXT,
828     NOTIFICATION,
829     PROGRESS,
830 }
831
832 enum TtsVoiceRate {
833     SLOW,
834     NORMAL,
835     FAST,
836 }
837
838 enum TtsVolume {
839     DEEMPHASIZED,
840     NORMAL,
841     EMPHASIZED,
842 }
843
844 enum TtsPitch
845 {
846     LOW,
847     NORMAL,

```


848 HIGH,
849 }