



Permissions

Contents

2	Terminology	2
3	Scope of this document	2
4	Use cases	3
5	Internet access	3
6	Geolocation	4
7	Initiating a phone call	5
8	Shared file storage	6
9	Launcher	7
10	Settings	7
11	Restricted subsets of settings	8
12	Granting permission on first use	9
13	Tightening control	10
14	Loosening control	11
15	Changing access	11
16	Potential future use-cases	12
17	Audio playback	12
18	Audio recording	13
19	Bluetooth configuration	13
20	Calendar	14
21	Contacts	16
22	Inter-app communication interfaces	16
23	Continuing to run in the background	17
24	Running on device startup	18
25	Non-use-cases	18
26	App's own data	18
27	Platform services	19
28	Built-in app-bundles with specialized requirements	19
29	Driving cameras	19
30	Infotainment cameras	19
31	App-specific permissions	20
32	General notes on other systems	20
33	Android	20
34	Flatpak	23
35	iOS	25

This document extends the higher-level [Applications](https://jwd.pages.apertis.org/apertis-website/concepts/applications/)¹ and [Security](https://jwd.pages.apertis.org/apertis-website/concepts/security/)² design documents to go into more detail about the permissions framework.

Applications can perform many functions on a variety of user data. They may access interfaces that read data (such as contacts, network state, or the users location), write data, or perform actions that can cost the user money (like sending SMS). As an example, the **Android** operating system has a comprehensive

¹<https://jwd.pages.apertis.org/apertis-website/concepts/applications/>

²<https://jwd.pages.apertis.org/apertis-website/concepts/security/>

42 [manifest](#)³ that govern access to a wide array of functionality.

43 Some users may wish to have fine grained control over which applications have
44 access to specific device capabilities, and even those that don't should likely be
45 informed when an application has access to their data and services.

46 Terminology

47 [Integrity, confidentiality and availability](#)⁴ are defined in the Security concept
48 design.

49 Discussions of the security implications of a use-case in this document often
50 mention the possibility of a *malicious* or *compromised* app-bundle. For brevity,
51 this should be understood to cover all situations where malicious code might
52 run with the privileges of a particular app-bundle, including:

- 53 • An app-bundle whose author or publisher deliberately included malicious
54 code in the released version (a Trojan horse)
- 55 • An app-bundle whose author or publisher accidentally included malicious
56 code in the released version, for example by using a maliciously altered
57 compiler like [XcodeGhost](#)⁵
- 58 • An app-bundle where there is no directly malicious code in the released
59 version, but there is a security vulnerability that an attacker can exploit
60 to run malicious code of their choice with the privileges of the app-bundle

61 Scope of this document

62 This document aims to define a general approach to permissions, so that future
63 work on a particular feature that requires permissions only requires the designer
64 of that feature to define a permission or a series of permissions, and does not
65 require the designer of the feature to design the entire permissions framework.

66 This document also aims to define permissions for basic features that are already
67 present in Apertis and already well-understood. For example, access to external
68 and shared storage is in-scope.

69 This document does not aim to define permissions for features that are not al-
70 ready present in Apertis, or that are not already well-understood. For example,
71 defining detailed permissions for [egress filtering](#)⁶ is out of scope.

72 The permissions framework is not intended to cover all the needs of built-in
73 application bundles. See [non use cases](#) below. However, some classes of built-in
74 application bundles are anticipated to be implemented by every Apertis ven-
75 dor as a means of product differentiation, and these could benefit from having

³<http://developer.android.com/reference/android/Manifest.permission.html>

⁴<https://jwd.pages.apertis.org/apertis-website/concepts/security/#integrity-confidentiality-and-availability>

⁵<https://en.wikipedia.org/wiki/XcodeGhost>

⁶https://jwd.pages.apertis.org/apertis-website/concepts/egress_filtering/

76 a shortcut way to request particular permissions; these use cases are marked
77 below.

78 Use cases

79 Internet access

80 A general-purpose Internet application like a web browser might require full,
81 unfiltered Internet access, including HTTP, HTTPS, DNS, WebSockets and
82 other protocols.

83 A podcast player might require the ability to download arbitrary files via HTTP
84 using a service like the Apertis Newport download manager, but might not
85 require any other Internet access.

86 A simple game might not require any network access at all, or might only re-
87 quire the indirect network access (launching URIs) that can be obtained by
88 communicating with the Didcot [content handover](#)⁷ service.

89 Many intermediate levels of Internet access are possible, but for the purposes of
90 this document we do not consider them. See the [Egress filtering design notes](#)
91 [on the Apertis wiki](#)⁸ for initial work on finer-grained control.

92 Security implications

93 An application with Internet access might be compromised by malicious inputs
94 from the Internet (integrity failure). If an application cannot contact the Inter-
95 net, we can be confident that it cannot be subject to this, although it could still
96 be compromised by malicious content that is downloaded locally and passed to
97 it via [content handover](#)⁹.

98 If an application with Internet access is compromised either remotely or by
99 opening malicious local content, it could be induced to send private data to an
100 attacker-controlled server (a confidentiality failure). This attack applies equally
101 to applications with access to a download manager like Newport, because the
102 private data could be encoded in the URI to be downloaded. If the download
103 manager offers control over request headers such as cookies or the HTTP `Referer`,
104 the private data could also be encoded in those. Applications that can request
105 URIs via a [content handover](#)¹⁰ service could also be susceptible to this attack,
106 but only if the content handover service will pass them to a handler without
107 user interaction, *and* the handler for those URIs will fetch the URI without user
108 interaction.

109 If an application with Internet access is compromised, but does not already
110 contain malicious code to carry out actions of the attacker's choice (the *payload*),

⁷https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/

⁸https://jwd.pages.apertis.org/apertis-website/concepts/egress_filtering/

⁹https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/

¹⁰https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/

111 a common technique is to download a payload from a server controlled by the
112 attacker. In particular, this allows an attacker to alter the payload over time
113 according to their current requirements, for example to form a botnet that
114 can be used for multiple purposes. An application with access to a download
115 manager like Newport is equally susceptible to this, even if it cannot access
116 the Internet itself, because it can ask Newport to download the new payload
117 from the attacker's server. However, an application that can only use a [content](#)
118 [handover](#)¹¹ service is not susceptible to this attack, because such applications
119 are not allowed to see the result of the HTTP request.

120 In other systems

121 In **Android**, the web browser and podcast player described in the use-cases
122 above would have the `INTERNET` permission, but the game would not. Using the
123 Android DownloadManager service (equivalent to Newport) also requires `INTER-`
124 `NET` permission, because it enables most of the same attacks as direct Internet
125 access.

126 In **Flatpak**, the web browser would have the `shared=network` permission and the
127 game would not. The game could still send requests to the URI-opening **portal**:
128 assuming that only one web browser is installed, the URI-opening portal would
129 normally pass on HTTP and HTTPS URIs to a web browser without user
130 consent (with the result that the browser makes `GET` requests to the appropriate
131 web server), but prompt the user before passing other URIs to the URI handler.
132 The podcast player could have `talk` access to a D-Bus service equivalent to
133 Newport, but as noted above, that would be essentially equivalent to arbitrary
134 HTTP access in any case.

135 In **iOS**, all app-bundles have Internet access.

136 Geolocation

137 A navigation app-bundle needs to know the precise location of the vehicle, but an
138 app-bundle to suggest nearby restaurants might only need to know the location
139 within a few miles, and an e-book reader does not need to know the location at
140 all.

141 Security implications

142 The user's geographical location is sensitive information, especially if it is pre-
143 cise, and is valuable to criminals. In some cases disclosing it would be a threat
144 to personal safety.

145 In other systems

¹¹https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/

146 In **Android** the navigation app-bundle would have `ACCESS_FINE_LOCATION`, the
147 restaurant guide would have `ACCESS_COARSE_LOCATION` and the e-book reader
148 would have neither.

149 In **Flatpak**, the user is asked for permission to use geolocation the first time it
150 is used, with the option to remember that permission for all future requests.
151 The app-bundle is not required to declare in advance whether it might use
152 geolocation.

153 In **iOS**, two forms of geolocation can be requested, by using `NSLocationAlways-`
154 `UsageDescription` OR `NSLocationWhenInUseUsageDescription`.

155 **Initiating a phone call**

156 A contact management application might wish to initiate a phone call without
157 further user consent, for example when the user taps a phone icon next to a
158 contact.

159 An application that is only tangentially related to phone calls, such as an app-
160 bundle to suggest nearby restaurants, might not wish to request permission to
161 do that. Instead, it could initiate a phone call by launching an appropriate
162 `tel:` URI, which would normally result in a built-in application or a platform
163 service popping up a call dialog with buttons to initiate the call or cancel the
164 transaction, the same as would happen on selecting a `tel:` link in a web browser.

165 An e-book reader does not need to initiate phone calls at all.

166 **Security implications**

167 If an app-bundle can initiate calls without user consent, this will result in the
168 user's microphone being connected to the call recipient, which is a confidentiality
169 (privacy) failure. Making undesired calls can also cost the user money, and in
170 particular a malicious app author might place calls to a premium rate number
171 that pays them.

172 **In other systems**

173 In **Android**, the contact management app-bundle would have the `CALL_PHONE`
174 permission. The restaurant guide and the e-book reader would not, but would
175 still be able to launch an intent that results in the system phone dialler app
176 being shown, giving the user the opportunity to confirm or cancel.

177 In **Flatpak**, the contact management app-bundle would have `talk` access to a
178 D-Bus service offering immediate phone dialling, for example the Telepathy Ac-
179 count Manager, or to a group of services, for example Telepathy. The restaurant
180 guide and the e-book reader would not, but would be able to launch a `tel:` URI,
181 which would be handled in much the same way as the Android intent.

182 In **iOS**, user-installable app bundles would presumably launch `tel:` URIs. There
183 does not appear to be a way for a non-platform-level component to dial phone
184 numbers directly.

185 **Shared file storage**

186 A media player with a gallery-style user experience might require the ability to
187 read media files stored on external storage (a USB thumb drive or externally-
188 accessible SD card), or in a designated **shared area**¹².

189 Similarly, a media player might require access to media indexing and browsing
190 as described in the **Media Management concept design**¹³.

191 A podcast player might wish to store downloaded podcasts on external storage
192 devices or in the shared storage area so that media players can access them.

193 **Security implications**

194 App-bundles with write access to this shared storage can modify or delete media
195 files; if this is done inappropriately, that would be an availability or integrity
196 failure. App-bundles with read access can observe the media that the user con-
197 sumes, which could be considered privacy-sensitive; uncontrolled access would
198 be a confidentiality failure. Malicious app-bundles with write access could also
199 write malformed media files that were crafted to exploit security flaws in other
200 app-bundles, in the platform, or in other devices that will read the same external
201 storage device, leading to an integrity failure.

202 **In other systems**

203 In recent **Android**, the `READ_EXTERNAL_STORAGE` permission is required (the shared
204 area on Android devices was traditionally a removable SD card, leading to the
205 name of the relevant permission and APIs, even though in more recent devices it
206 is typically on non-removable flash storage). In older Android, that permission
207 did not exist or was not enforced.

208 Similarly, the `WRITE_EXTERNAL_STORAGE` permission governs writing; that permis-
209 sion was always enforced, but is very widely requested.

210 In **Flatpak**, any directory of interest can be mapped into the filesystem names-
211 pace of sandboxed processes, either read-only or read/write, via the `filesystems`
212 metadata field. Values like `xdg-music` and `xdg-download/Podcasts` make common
213 use cases relatively straightforward, and provide considerably finer-grained con-
214 trol than in Android.

215 In **iOS**, access to media libraries is mediated by the `NSAppleMusicUsageDescription`
216 and `NSPhotoLibraryUsageDescription` metadata fields.

¹²<https://jwd.pages.apertis.org/apertis-website/concepts/application-layout/#shared-data>

¹³<https://jwd.pages.apertis.org/apertis-website/concepts/media-management/>

217 **Launcher**

218 This use-case is only applicable to built-in app-bundles.

219 A vendor-specific application launcher, such as the [Mildenhall Launcher](#)¹⁴ in
220 the Apertis Mildenhall reference user interface, needs to list all the application
221 entry points on the system together with their metadata. It also needs to launch
222 those entry points on-demand.

223 **Security implications**

224 Holding this permission negates the Apertis platform's usual concept of [application list privacy](#)¹⁵: an app-bundle with this permission can enumerate the entry
225 points, which is valuable if an attacker wishes to identify particular user (fin-
226 gerprinting). If unintended app-bundles gain this access, it is a confidentiality
227 failure.
228

229 **In other systems**

230 **Android** does not appear to restrict the visibility of other app-bundles.

231 **Flatpak** app-bundles can only observe the existence of other app-bundles if their
232 D-Bus filtering is configured to be able to `see` their well-known names.

233 **iOS** restricts the visibility of other app-bundles, although [fingerprinting](#)¹⁶ can
234 be carried out by abusing inter-app communication. Because iOS is a single-
235 vendor system, the security mechanisms used by platform components and by
236 the equivalent of our built-in app bundles do not have public documentation.

237 **Settings**

238 This use-case is probably only applicable to built-in app-bundles.

239 Suppose a vendor has a [system preferences application](#)¹⁷ that provides an
240 overview of all [system settings](#)¹⁸, [user settings](#)¹⁹ and [app settings](#)²⁰, such as the
241 [Mildenhall Settings](#)²¹ application bundle in the Apertis Mildenhall reference
242 user interface. That application needs to list the app settings belonging to all

¹⁴<https://gitlab.apertis.org/hmi/mildenhall-launcher>

¹⁵[https://jwd.pages.apertis.org/apertis-website/concepts/application-entry-points/
#security-and-privacy-considerations](https://jwd.pages.apertis.org/apertis-website/concepts/application-entry-points/#security-and-privacy-considerations)

¹⁶<https://arxiv.org/abs/1605.08664>

¹⁷[https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/
#user-interface](https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/#user-interface)

¹⁸[https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/
#system-settings](https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/#system-settings)

¹⁹[https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/
#user-settings](https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/#user-settings)

²⁰[https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/
#app-settings](https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/#app-settings)

²¹<https://gitlab.apertis.org/hmi/mildenhall-settings>

243 store and built-in app-bundles, and needs the ability to change them, without
244 prompting the user.

245 **Security implications**

246 Holding this permission negates the Apertis platform’s usual concept of [appli-](#)
247 [cation list privacy](#)²², similar to the [Launcher](#) use case.

248 Unconstrained settings changes are also very likely to allow arbitrary code exe-
249 cution with the privileges of other components that trust those settings, which
250 would be a serious integrity failure if carried out by an attacker.

251 **In other systems**

252 In [Android](#), the `CHANGE_CONFIGURATION` permission grants the ability to change
253 system configuration in some limited ways, and the `WRITE_SETTINGS` permission
254 grants the ability to carry out more settings changes.

255 In [Flatpak](#) when used with GNOME, granting write access to `dconf` (by making
256 its files readable in the sandbox, and granting `talk` access to the `dconf` service)
257 gives unconstrained access to all settings.

258 Because [iOS](#) is a single-vendor system, the security mechanisms used by platform
259 components and by the equivalent of our built-in app bundles do not have public
260 documentation.

261 **Restricted subsets of settings**

262 A photo viewer might have an option to set a particular photo as “wallpaper”. A
263 travel-related app-bundle might have an option to set the time zone, and media
264 player might have options to change audio parameters. An e-book reader does
265 not require the ability to do any of those.

266 **Security implications**

267 In general, these subsets of settings are chosen so that an attacker changing them
268 would be an annoyance rather than a serious integrity failure, mitigating the
269 attacks that are possible in the use-case above. However, the effect of changing
270 a setting is not always immediately obvious: for example, setting untrusted
271 images as wallpaper could lead to a more serious integrity failure if there is an
272 exploitable flaw in an image decoder used by the platform component or built-in
273 app-bundle that displays the wallpaper.

274 **In other systems**

²²[https://jwd.pages.apertis.org/apertis-website/concepts/application-entry-points/
#security-and-privacy-considerations](https://jwd.pages.apertis.org/apertis-website/concepts/application-entry-points/#security-and-privacy-considerations)

275 In **Android**, the photo viewer might have the `SET_WALLPAPER` and `SET_WALLPAPER_HINTS`
276 permissions, the travel-related app-bundle might have `SET_TIME_ZONE`, and the
277 media player might have `MODIFY_AUDIO_SETTINGS`.

278 **Flatpak** does not currently have portals for these, but a Flatpak app-bundle
279 could be given `talk` access to a D-Bus service that would allow these actions.

280 **iOS** does not appear to provide this functionality to third-party app-bundles.

281 **Granting permission on first use**

282 The author of a hotel booking app-bundle includes a feature to locate nearby
283 hotels by using the Apertis geolocation API. Because **users are more likely to**
284 **grant permission to carry out privacy-sensitive actions if they can understand**
285 **why it is needed**²³, the app author does not want the Apertis system to prompt
286 for access to the geolocation feature until the user actively uses that particular
287 feature.

288 **Not granting permission on first use**

289 Conversely, an automotive vendor wishes to minimize driver distraction in order
290 to maximize safety. When the same hotel booking app-bundle attempts to use
291 geolocation while the vehicle is in motion, the platform vendor might want the
292 Apertis system to **not** prompt for access to the geolocation feature, contrary to
293 the wishes of the app author. Instead, the user should be given the opportunity
294 to enable geolocation at a time when it is safe to do so, either during app-bundle
295 installation or as a configuration/maintenance operation while the vehicle is
296 stationary at a later time.

297 Note that those two use cases have contradictory expectations: this is a user
298 experience trade-off for which there is no single correct answer.

299 **In other systems**

300 **iOS** prompts for permission to carry out each privileged operation at the time
301 of first use.

302 **Flatpak** mostly does the same, but with some pragmatic exceptions: lower-level
303 permissions, such as access to direct rendering devices for 3D games or direct
304 access to the host filesystem, are implemented in a way that precludes that
305 model. These are set up at installation time, and can be overridden by user
306 configuration. When a Flatpak app is launched, it is given the level of access
307 that was appropriate at launch time.

308 **Android** 6.0 and later has the same behaviour as iOS. Older Android versions
309 configured all permissions at installation time, with a simple UX: the user must
310 either accept all required permissions, or abort installation of the app. Some
311 permissions, notably access to shared storage (the real or emulated SD card),

²³<https://savvyapps.com/blog/how-to-create-better-user-permission-requests-in-ios-apps>

were implemented in a way that precluded runtime changes: app processes with access to shared storage ran with one or more additional Unix group IDs, granting them DAC permission to the appropriate areas of the filesystem.

Tightening control

Suppose that Apertis version 1 allows all app-bundles to query the vehicle model, but the Apertis developers later decide this is a privacy risk, and so Apertis version 2 restricts it with a permission. The app framework should be able to detect that an app-bundle was compiled for version 1, and behave as though that app-bundle had requested the necessary permission to query the vehicle model. It should not do that for an app-bundle compiled for version 2.

Security implications

App-bundles that were compiled for version 1 would still be able to carry out any attacks that were applicable before version 2 was released. This use-case is only applicable if those attacks are considered to be less serious than breaking backwards compatibility with older app-bundles.

In other systems

In **Android**, a simple integer “API level” is used to indicate the version of the Android API. Each app-bundle has a *minimum API level* and a *target API level*. The app framework enables various compatibility behaviours to make APIs resemble those that were present at the target API level; one of these compatibility behaviours is to behave as though app-bundles whose target API level is below a threshold had requested extra permissions. For example, Android behaves as though app-bundles with a target API level below 4 had requested `android.READ_PHONE_STATE`.

In **Flatpak**, app-bundles can specify a minimum Flatpak version. There is currently no mechanism to specify a target API level, although one could be inferred from the runtime branch that the app-bundle has chosen to use, such as `org.freedesktop.Platform/1.4` or `org.gnome.Platform/3.22`.

In **iOS**, keys like `[NSAppleMusicUsageDescription]` are documented as behaving like permissions, but only if the app was linked on or after iOS 10.0.

Loosening control

Suppose Apertis version 1 restricts querying the vehicle paint colour with a permission, but the Apertis developers later decide that this does not need to be restricted, and Apertis version 2 allows all app-bundles to do that. The app framework should never prompt the user for that permission. If an app-bundle designed for version 1 checks whether it has that permission, the app framework should tell it that it does.

349 Security implications

350 This use-case is only applicable if the Apertis developers have decided that the
351 security implications of the permission in question (in this example, querying
352 the paint colour) are not significant.

353 In other systems

354 We are not aware of any permissions that have been relaxed like this in Android,
355 Flatpak or iOS, but it would be straightforward for any of these frameworks to
356 do so: they would merely have to stop presenting a user interface for that
357 permission, and make requests for it always succeed.

358 Changing access

359 An Apertis user uses a Facebook app-bundle. The user wants their location at
360 various times to appear on their Facebook feed, so they give the app-bundle
361 permission to monitor his location, as in [geolocation](#) above.

362 Later, that user becomes more concerned about their privacy. They want to
363 continue to use the Facebook app-bundle, but prevent it from accessing their
364 new locations. They use a user interface provided by the system vendor, perhaps
365 a [system preferences application](#)²⁴, to reconfigure the permissions granted to the
366 Facebook app-bundle so that it cannot access their location.

367 Later still, that user wants to publish their location to their Facebook feed
368 while on a road trip. They reconfigure the permissions granted to the Facebook
369 app-bundle again, so that it can access their location again.

370 Security implications

371 This use-case is applicable if the user's perception of the most appropriate trade-
372 off between privacy and functionality changes over time.

373 In other systems

374 Android 6.0 and later versions have a [user interface](#)²⁵ to revoke and reinstate
375 broad categories of permissions. Older [Android](#) versions had a hidden control
376 panel named [App ops](#)²⁶ controlling the same things at a finer-grained level
377 (individual permissions), but it was not officially supported.

378 [iOS](#) allows permissions to be revoked or reinstated at any time via the [Pri-
379 vacy page in its Settings app](#)²⁷, which is the equivalent of the Apertis [system](#)

²⁴[https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/
#user-interface](https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/#user-interface)

²⁵<https://www.howtogeek.com/230683/how-to-manage-app-permissions-on-android-6.0/>

²⁶[https://www.theguardian.com/technology/2015/jun/09/google-privacy-apple-android-
lockheimer-security-app-ops](https://www.theguardian.com/technology/2015/jun/09/google-privacy-apple-android-lockheimer-security-app-ops)

²⁷[https://www.howtogeek.com/177711/ios-has-app-permissions-too-and-theyre-arguably-
better-than-androids/](https://www.howtogeek.com/177711/ios-has-app-permissions-too-and-theyre-arguably-better-than-androids/)

380 [preferences application](#)²⁸.

381 **Potential future use-cases**

382 Use cases described in this section are not intended to generate requirements in
383 the near future, and are not described in detail here. We recommend that these
384 use cases are expanded into something more detailed as part of design work on
385 the relevant feature: for example, Bluetooth permissions should be considered
386 as part of a more general Bluetooth feature design task.

387 However, as input to the design of the general feature of permissions, it might
388 be instructive to consider whether a proposed implementation could satisfy the
389 requirements that these use-cases are conjectured to have.

390 Because these use-cases have not been examined in detail, it is possible that
391 future work on them will result in the conclusion that they should be outside
392 the scope of the permissions framework described in this document.

393 **Audio playback**

394 A music player requires the ability to play back audio while in the background.
395 A video player might require the ability to play ongoing audio, but only while its
396 window is in the foreground. An e-book reader might only require the ability to
397 play short notification sounds while in the foreground, or might not require any
398 ability to play sounds at all. A voice-over-IP calling client requires the ability
399 to play audio with an elevated priority while a call is in progress, pre-empting
400 other audio players.

401 We recommend that these and related use cases are captured in detail as part
402 of the design of the Apertis audio manager.

403 **Security implications**

404 Uncontrolled audio playback seems likely to cause driver distraction. Addition-
405 ally, if all applications can play back audio with a priority of their choice, a
406 malicious app-bundle could output silence at a high priority as a denial of ser-
407 vice attack (a failure of availability).

408 **In other systems**

409 In **Android** and **iOS**, audio playback does not require special permissions.

410 In **Flatpak**, audio playback currently requires making the PulseAudio socket
411 available to the sandboxed app, which also enables audio recording and control.

412 Finer-grained control over audio is planned for the future.

²⁸[https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/
#user-interface](https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/#user-interface)

413 **Audio recording**

414 A memo recorder requires the ability to record audio. A voice-over-IP calling
415 client also requires the ability to record audio. Most applications, including
416 most of those that play back audio, do not.

417 We recommend that these and related use cases are captured in detail as part
418 of the design of the Apertis audio manager.

419 **Security implications**

420 An app-bundle that can record audio could record private conversations in the
421 vehicle (a failure of confidentiality).

422 **In other systems**

423 In **Android**, audio recording requires the `RECORD_AUDIO` permission.

424 In **Flatpak**, audio recording currently requires making the PulseAudio socket
425 available to the sandboxed app, which also enables audio playback and control.

426 In **iOS**, audio recording is mediated by `NSMicrophoneUsageDescription`.

427 **Bluetooth configuration**

428 A [system preferences application](#)²⁹, or a separate Bluetooth control panel built-
429 in app-bundle, might require the ability to reconfigure Bluetooth in detail and
430 communicate with arbitrary devices.

431 A less privileged app-bundle, for example one provided by the manufacturer of
432 peripheral devices like FitBit, might require the ability to pair and communicate
433 with those specific Bluetooth devices.

434 A podcast player has no need to communicate with Bluetooth devices at all.

435 **Security implications**

436 For the control panel use-case, communicating with arbitrary devices might be
437 an integrity failure if the app-bundle can reconfigure the device or edit data
438 stored on it, or a confidentiality failure if the app-bundle can read sensitive
439 data such as a phone's address book. The ability for untrusted app-bundles
440 to view MAC addresses and other unique identifiers would also be a privacy
441 problem.

442 The device-specific use case is a weaker form of the above, mitigating the confi-
443 dentiality and integrity impact.

²⁹[https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/
#user-interface](https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/#user-interface)

444 In other systems

445 In **Android**, the `BLUETOOTH` permission allows an app-bundle to communicate with
446 any Bluetooth device that is already paired. This is stronger than is needed for a
447 device-specific app-bundle. The `BLUETOOTH_ADMIN` permission additionally allows
448 the app-bundle to pair new Bluetooth devices.

449 In **Flatpak**, full access could be achieved by configuring Flatpak's D-Bus filter
450 to allow `talk` access to BlueZ. There is currently no implementation of partial
451 access; this would likely require a Bluetooth **portal** service.

452 In **iOS**, the `NSBluetoothPeripheralUsageDescription`³⁰ metadata field controls
453 access to Bluetooth, which appears to be all-or-nothing. User consent is re-
454 quested the first time this permission is used, with the metadata field's content
455 included in the prompt.

456 Calendar

457 A general-purpose calendar/agenda user interface similar to **GNOME Calen-**
458 **dar**³¹ or the **AOSP Calendar**³² requires full read/write access to the user's cal-
459 endar.

460 A calendar synchronization implementation, for example to synchronize with
461 calendar events stored in Google Calendar, Windows Live or OwnCloud, re-
462 quires full read/write access to its subset of the user's calendar. For example,
463 a Google Calendar synchronization app-bundle should have access to Google
464 calendars, but not to Windows Live calendars.

465 A non-calendar application like an airline booking app-bundle might wish to
466 insert events into the calendar without further user interaction, or it might wish
467 to insert events into the calendar in a way that presents them for user approval,
468 for example by submitting a vCalendar file for **content handover**³³.

469 A podcast player has no need to interact with the calendar at all.

470 Security implications

471 The general-purpose user interface described above would have the ability to
472 send calendar events to a third party (a confidentiality failure) or to edit or
473 delete them (an integrity failure).

474 The calendar synchronization example is a weaker form of the user interface use-
475 case: if malicious, it could cause the same confidentiality or integrity failures,
476 but only for a subset of the user's data.

³⁰https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW20

³¹<https://wiki.gnome.org/Apps/Calendar>

³²<https://fossdroid.com/a/standalone-calendar.html>

³³https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/

477 If the airline booking app-bundle described above has the ability to insert cal-
478 endar events without user interaction, a malicious app-bundle could insert mis-
479 leading events, an integrity failure; however, it would not necessarily be able to
480 break confidentiality.

481 If the airline booking app-bundle operates via content handover, **intents**, **por-**
482 **tals** or a similar mechanism that will result in user interaction, a malicious
483 app-bundle cannot insert misleading events without user action, avoiding that
484 integrity failure (at the cost of a more prescriptive UX).

485 In other systems

486 In **Android**, the `READ_CALENDAR` and `WRITE_CALENDAR` permissions³⁴ are suitable
487 for the general-purpose calendar use case. **Sync adapters**³⁵ receive different
488 access; it is not clear from the Android documentation whether their restriction
489 to a specific subset of the calendar is enforced, or whether sync adapters are
490 trusted and assumed to not attack one another. Applications that do not have
491 these permissions, such as the hotel booking use-case above, can use **calendar**
492 **intents**³⁶ to send or receive calendar events, with access mediated through a
493 general-purpose calendar user interface that is trusted to behave according to
494 the user's intention. There is no way to prevent an app-bundle from using those
495 intents at all.

496 In **Flatpak**, a general-purpose calendar might be given `talk` access to the
497 `evolution-data-server` service. There is currently no calendar **portal**, but when
498 one is added it will presumably be analogous to Android intents.

499 In **iOS**, the `[NSCalendarsUsageDescription]` metadata field controls access to
500 calendars. User consent is requested the first time this permission is used, with
501 the metadata field's content included in the prompt.

502 Contacts

503 The use cases and security implications for contacts are analogous to those for
504 the **calendar** and are not discussed in detail here.

505 In other systems

506 **Android contact management**³⁷ is analogous to calendaring, using the
507 `READ_CONTACTS` and `WRITE_CONTACTS` permissions or contact-specific intents.

508 In **Flatpak**, as with contacts, a general-purpose contacts app-bundle might be
509 given `talk` access to the `evolution-data-server` service. There is currently no

³⁴<https://developer.android.com/guide/topics/providers/calendar-provider.html#manifest>

³⁵<https://developer.android.com/guide/topics/providers/calendar-provider.html#sync-adapter>

³⁶<https://developer.android.com/guide/topics/providers/calendar-provider.html#intents>

³⁷<https://developer.android.com/guide/topics/providers/contacts-provider.html>

510 contacts [portal](#), but when one is added it will presumably be analogous to
511 Android intents.

512 [iOS contact management][NSContactsUsageDescription] is analogous to iOS
513 calendaring.

514 **Inter-app communication interfaces**

515 Inter-app communication has not been designed in detail, but the draft design
516 on the Apertis wiki suggests that it might be modelled in terms of [interface](#)
517 [discovery](#)³⁸, with app-bundles able to implement “public interfaces” that are
518 made visible to other app-bundles. The draft design has some discussion of how
519 [restricting interface providers](#)³⁹ might be carried out by app-store curators.

520 Additionally, if app-bundles export public interfaces, this might influence
521 whether other applications are allowed to communicate with them: if a
522 particular public interface implies that other app-bundles will communicate
523 directly with the implementor, then the implementor’s AppArmor profile and
524 other security policies must allow that. A [sharing](#)⁴⁰ feature similar to the one
525 in Android is one possible use-case for this.

526 We recommend that this topic is considered as one or more separate concept
527 designs, with its security implications considered at the same time. This is likely
528 to be more successful if a small number of specific use-cases are considered,
529 rather than attempting to define a completely abstract and general framework.

530 In [Android](#), any app-bundle can define its own intents. If it does, those intents
531 can be invoked by any other app-bundle that holds appropriate permissions,
532 and it is up to the implementor to ensure that that is a safe thing to do.

533 In [Flatpak](#), app-bundles that will communicate via D-Bus can be given `talk`
534 access to each other. If this is done, it is up to the app-bundles to ensure that
535 they do not carry out unintended actions in response to D-Bus method calls.

536 In [iOS](#), any app-bundle can define non-standard URI schemes that it will handle,
537 and these non-standard URI schemes are the basis for inter-app communication.
538 There is no particular correlation between the URI scheme and the app’s identity
539 (the iOS equivalent of our bundle IDs), and there have been successful attacks
540 against this, including the [URL masque attack](#)⁴¹ identified by FireEye.

541 **Continuing to run in the background**

542 [Agents](#)⁴² do not show any graphical windows, so to be useful they must always
543 run in the background.

³⁸https://jwd.pages.apertis.org/apertis-website/concepts/interface_discovery/

³⁹[https://jwd.pages.apertis.org/apertis-website/concepts/interface_discovery/
#Restricting_who_can_advertise_a_given_interface_2](https://jwd.pages.apertis.org/apertis-website/concepts/interface_discovery/#Restricting_who_can_advertise_a_given_interface_2)

⁴⁰<https://jwd.pages.apertis.org/apertis-website/concepts/sharing/>

⁴¹https://www.fireeye.com/blog/threat-research/2015/04/url_masques_on_apps.html

⁴²<https://jwd.pages.apertis.org/apertis-website/concepts/applications/#start>

544 Graphical programs that have windows open, but no windows visible to the user,
545 might be terminated by the application framework. The author of a graphical
546 program that needs to be available without delay might wish to request that it
547 is not terminated.

548 **Security implications**

549 Background programs consume resources, impacting availability (denial of ser-
550 vice). A background program that has other permissions might make use of
551 them without the user's knowledge: for example, if a restaurant guide can track
552 the user's location, this can be mitigated by only allowing it to run, or only
553 allowing it to make use of its permissions, while it is (or was recently) visible,
554 so that the user can only be tracked by the guide's author at times when they
555 are aware that this is a possibility.

556 Users might wish to be aware of which graphical programs have this property,
557 and user interfaces for managing permissions might display it in the same con-
558 text as other permissions, but it is not a permission in the sense that it is used to
559 generate security policies. Accordingly, it should potentially be handled outside
560 the scope of this document.

561 Future work on this topic is tracked in Apertis task [T3438](#)⁴³ and its future
562 subtasks.

563 **In other systems**

564 Android does not have permissions that influence its behaviour for background
565 programs.

566 Flatpak does not currently attempt to monitor background programs or force
567 them to exit.

568 iOS manages background programs via the [UIBackgroundModes] and UIAppli-
569 cationExitsOnSuspend⁴⁴ metadata fields. NSSupportsAutomaticTermination⁴⁵
570 is analogous, but is for desktop macOS.

571 **Running on device startup**

572 An [agent](#)⁴⁶ might be run on device startup.

573 A typical graphical program has no need to start running on device startup.

⁴³<https://phabricator.apertis.org/T3438>

⁴⁴https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/iPhoneOSKeys.html#//apple_ref/doc/uid/TP40009252-SW23

⁴⁵https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW13

⁴⁶<https://jwd.pages.apertis.org/apertis-website/concepts/applications/#start>

574 A graphical program that is expected to be frequently but intermittently used
575 might be pre-loaded (but left hidden) on device startup.

576 The security implications are essentially the same as **continuing to run in the**
577 **background**.

578 Users might wish to be aware of which graphical programs have this property,
579 and user interfaces for managing permissions might display it in the same con-
580 text as other permissions, but it is not a permission in the sense that it is used
581 to generate security policies. Accordingly, it is treated as outside the scope of
582 this document.

583 We suggest that this should be handled alongside **continuing to run in the back-**
584 **ground**.

585 In other systems

586 In **Android**, a graphical program or service that runs in the background would
587 have the `RECEIVE_BOOT_COMPLETED` permission, which is specifically described as
588 covering performance and not security.

589 **Flatpak** does not natively provide this functionality.

590 iOS manages autostarted background programs via certain values of the
591 `[UIBackgroundModes]` metadata field.

592 Non-use-cases

593 The following use cases are specifically excluded from the scope of this document.

594 App's own data

595 Each app-bundle should be allowed to read and write its own data, including
596 its own **app settings**⁴⁷. However, this should not need any special permissions,
597 because it should be granted to every app-bundle automatically: accordingly, it
598 is outside the scope of this document. App settings are part of the scope of the
599 **Preferences and Persistence**⁴⁸ concept design, and other per-app private data
600 are in the scope of the **Applications**⁴⁹ concept design.

601 Similarly, programs from each app-bundle should be allowed to communicate
602 with other programs from the same app-bundle (using any suitable mechanism,
603 including D-Bus) without any special permissions, with the typical use-case
604 being a user interface communicating with an associated **agent**⁵⁰. Because it
605 does not require special permissions, that is outside the scope of this document.

⁴⁷[https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/
#app-settings](https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/#app-settings)

⁴⁸<https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/>

⁴⁹<https://jwd.pages.apertis.org/apertis-website/concepts/applications/>

⁵⁰<https://jwd.pages.apertis.org/apertis-website/concepts/applications/#start>

606 **Platform services**

607 This permissions framework is not intended for use by platform services, re-
608 gardless of whether they are upstream projects (such as systemd, dbus-daemon
609 and Tracker), developed specifically for Apertis (such as the Canterbury app
610 manager, the Newport download manager and the Ribchester volume mounting
611 service), or developed for a particular vendor (such as the compositor that imple-
612 ments a vendor-specific user interface, for which the [Mildenhall Compositor](#)⁵¹ is
613 a reference implementation). Platform services should continue to contain their
614 own AppArmor profiles, polkit rules and other security metadata.

615 **Built-in app-bundles with specialized requirements**

616 This permissions framework is not intended for use by built-in application bun-
617 dles with specialized or highly-privileged requirements, such as a built-in ap-
618 plication that communicates directly with specialized hardware. These built-in
619 application bundles should have their own AppArmor profiles, polkit rules and
620 other security metadata.

621 **Driving cameras**

622 Some vehicles have external cameras for purposes such as facilitating reversing,
623 watching for hazards in the vehicle's blind spots, or improving night vision by
624 using thermal imaging.

625 Our understanding is that images from these cameras should only be made
626 available to platform components or to specialized built-in app-bundles, so they
627 are outside the scope of this document.

628 **Infotainment cameras**

629 **Android** and **iOS** mobile phones and tablets typically have one or more cameras
630 directed at the user or their surroundings, intended for photography, video-
631 conferencing, augmented reality and entertainment. Our understanding is that
632 this is not a normal use-case for an automotive operating system that should
633 minimize driver distraction.

634 If a vehicle does have such cameras, their use cases and security implications
635 are very similar to audio recording, so we believe there is no need to describe
636 them in detail in this document.

637 **App-specific permissions**

638 In **Android**, any app-bundle can declare its own unique permissions namespaced
639 by its author's reversed domain name, and any other app-bundle can request
640 those permissions. It is not clear how an app-store vendor can be expected to
641 make an informed decision about whether those requests are legitimate.

⁵¹<https://gitlab.apertis.org/hmi/mildenhall-compositor>

642 If an app-bundle signed by the same author requests one of these permissions,
643 it is automatically granted; Android documentation recommends this route.

644 If an app-bundle by a different author that requests one of these app-specific
645 permissions is installed, a description provided by the app-bundle that declared
646 the permission is shown to the user when they are choosing whether to allow
647 the requesting app-bundle to be installed. If the requesting app-bundle is in-
648 stalled before the declaring app-bundle, then its request to use that permission
649 is silently denied.

650 **Flatpak** does not directly have this functionality, although cooperating app-
651 bundles can be given `talk` access to each other's D-Bus well-known names.

652 **iOS** does not appear to have this functionality.

653 We recommend that this feature is not considered in the short term.

654 General notes on other systems

655 Specific permissions corresponding to those for which we see a need in Apertis
656 are covered in the individual use cases above. This section describes other
657 operating systems and app frameworks in more general terms.

658 Android

659 Android includes permissions in its XML manifest file.

- 660 • [Introduction](https://developer.android.com/guide/topics/manifest/manifest-intro.html)⁵²
- 661 • [Permission API reference](https://developer.android.com/reference/android/Manifest.permission.html)⁵³
- 662 • [Permission group API reference](https://developer.android.com/reference/android/Manifest.permission.html)⁵⁴
- 663 • [Declaring that a permission is needed](https://developer.android.com/guide/topics/manifest/uses-permission-element.html)⁵⁵

664 Android apps can declare new permissions in the XML manifest.

- 665 • [Permission element](https://developer.android.com/guide/topics/manifest/permission-element.html)⁵⁶
- 666 • [Permission group element](https://developer.android.com/guide/topics/manifest/permission-group-element.html)⁵⁷
- 667 • [Permission tree element](https://developer.android.com/guide/topics/manifest/permission-tree-element.html)⁵⁸

668 Since Android 6.0, it is possible to request additional permissions (not declared
669 in the manifest) at runtime.

670 Permissions not described in this document

⁵²<https://developer.android.com/guide/topics/manifest/manifest-intro.html>

⁵³<https://developer.android.com/reference/android/Manifest.permission.html>

⁵⁴<https://developer.android.com/reference/android/Manifest.permission.html>

⁵⁵<https://developer.android.com/guide/topics/manifest/uses-permission-element.html>

⁵⁶<https://developer.android.com/guide/topics/manifest/permission-element.html>

⁵⁷<https://developer.android.com/guide/topics/manifest/permission-group-element.html>

⁵⁸<https://developer.android.com/guide/topics/manifest/permission-tree-element.html>

671 The following access permissions, available as of API level 25, do not match any
672 use-case described in this document. Deprecated and unsupported permissions
673 have been ignored when compiling this document.

674 Normal permissions:

- 675 • ACCESS_LOCATION_EXTRA_COMMANDS
- 676 • ACCESS_NETWORK_STATE
- 677 • ACCESS_NOTIFICATION_POLICY
- 678 • ACCESS_WIFI_STATE
- 679 • ADD_VOICEMAIL
- 680 • BATTERY_STATS
- 681 • BODY_SENSORS
- 682 • BROADCAST_STICKY
- 683 • CAMERA
- 684 • CHANGE_NETWORK_STATE
- 685 • CHANGE_WIFI_MULTICAST_STATE
- 686 • CHANGE_WIFI_STATE
- 687 • DISABLE_KEYGUARD
- 688 • EXPAND_STATUS_BAR
- 689 • GET_ACCOUNTS
- 690 • GET_ACCOUNTS_PRIVILEGED
- 691 • GET_PACKAGE_SIZE
- 692 • INSTALL_SHORTCUT
- 693 • KILL_BACKGROUND_PROCESSES
- 694 • NFC
- 695 • PROCESS_OUTGOING_CALLS
- 696 • READ_CALL_LOG
- 697 • READ_EXTERNAL_STORAGE
- 698 • READ_PHONE_STATE
- 699 • READ_SMS
- 700 • READ_SYNC_SETTINGS
- 701 • READ_SYNC_STATS
- 702 • RECEIVE_MMS
- 703 • RECEIVE_SMS
- 704 • RECEIVE_WAP_PUSH
- 705 • REORDER_TASKS
- 706 • REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
- 707 • REQUEST_INSTALL_PACKAGES
- 708 • SEND_SMS
- 709 • SET_ALARM
- 710 • SET_TIME_ZONE
- 711 • SET_WALLPAPER
- 712 • SET_WALLPAPER_HINTS
- 713 • TRANSMIT_IR
- 714 • USE_FINGERPRINT
- 715 • USE_SIP

- 716 • VIBRATE
- 717 • WAKE_LOCK
- 718 • WRITE_CALL_LOG
- 719 • WRITE_EXTERNAL_STORAGE
- 720 • WRITE_SETTINGS
- 721 • WRITE_SYNC_SETTINGS

722 Permissions described as not for use by third-party applications:

- 723 • ACCOUNT_MANAGER
- 724 • Several permissions starting with `BIND_` that represent the ability to bind to
- 725 the identity of a platform service, analogous to the ability to own platform
- 726 services' D-Bus names in Apertis
- 727 • BLUETOOTH_PRIVILEGED
- 728 • Several permissions starting with `BROADCAST_` that represent the ability to
- 729 broadcast messages, analogous to the ability to own a platform service's
- 730 D-Bus name and send signals in Apertis
- 731 • CALL_PRIVILEGED
- 732 • CAPTURE_AUDIO_OUTPUT
- 733 • CAPTURE_SECURE_VIDEO_OUTPUT
- 734 • CAPTURE_VIDEO_OUTPUT
- 735 • CHANGE_COMPONENT_ENABLED_STATE
- 736 • CLEAR_APP_CACHE
- 737 • CONTROL_LOCATION_UPDATES
- 738 • DELETE_CACHE_FILES
- 739 • DELETE_PACKAGES
- 740 • DIAGNOSTIC
- 741 • DUMP
- 742 • FACTORY_TEST
- 743 • GLOBAL_SEARCH, held by the global search framework to give it permission
- 744 to contact every global search provider
- 745 • INSTALL_LOCATION_PROVIDER
- 746 • INSTALL_PACKAGES
- 747 • LOCATION_HARDWARE
- 748 • MANAGE_DOCUMENTS
- 749 • MASTER_CLEAR
- 750 • MEDIA_CONTENT_CONTROL
- 751 • MODIFY_PHONE_STATE
- 752 • MOUNT_FORMAT_FILESYSTEMS
- 753 • MOUNT_UNMOUNT_FILESYSTEMS
- 754 • PACKAGE_USAGE_STATS
- 755 • READ_FRAME_BUFFER
- 756 • READ_LOGS
- 757 • READ_VOICEMAIL
- 758 • REBOOT
- 759 • SEND_RESPOND_VIA_MESSAGE
- 760 • SET_ALWAYS_FINISH

- 761 • SET_ANIMATION_SCALE
- 762 • SET_DEBUG_APP
- 763 • SET_PROCESS_LIMIT
- 764 • SET_TIME
- 765 • SIGNAL_PERSISTENT_PROCESSES
- 766 • STATUS_BAR
- 767 • SYSTEM_ALERT_WINDOW
- 768 • UPDATE_DEVICE_STATS
- 769 • WRITE_APN_SETTINGS
- 770 • WRITE_GSERVICES
- 771 • WRITE_SECURE_SETTINGS
- 772 • WRITE_VOICEMAIL

773 Intents

774 Holding a permission is not required to use an *intent* that implicitly asks the
 775 user for permission, such as taking a photo by sending a request to the system
 776 camera application, which will pop up a viewfinder provided by the system
 777 camera application, allowing the user to either take a photo when they are
 778 ready, or cancel by pressing the Back button; if the user takes a photo, it is
 779 sent back to the requesting application as the result of the intent. This is
 780 conceptually similar to Flatpak [portals](#).

781 Flatpak

782 Flatpak does not have a single flat list of permissions. Instead, its permissions
 783 are categorized according to the resource being controlled. Available permissions
 784 include:

- 785 • Hardware-accelerated graphics rendering via [Direct Rendering Manager](#)⁵⁹
- 786 devices
- 787 • Hardware-accelerated virtualization via [Kernel-based Virtual Machine](#)⁶⁰
- 788 devices
- 789 • Full access to the host's device nodes
- 790 • Sharing specific filesystem areas on a read-only or read/write basis
- 791 • Sharing the host's X11 socket (not used in production on Apertis)
- 792 • Sharing the host's Wayland socket (always available to graphical programs
- 793 on Apertis)
- 794 • Full access to the host's D-Bus session bus
- 795 • Full access to the host's D-Bus system bus
- 796 • Full access to the host's PulseAudio socket
- 797 • Sharing the host system's network namespace (Internet and LAN access)
- 798 • Sharing the host system's IPC namespace (this does not control D-Bus or
- 799 AF_UNIX sockets, but would allow the app-bundle to be treated as uncon-

⁵⁹https://en.wikipedia.org/wiki/Direct_Rendering_Manager

⁶⁰https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine

800 fined for the purposes of services that use [Unix System V IPC](#)⁶¹ or [POSIX](#)
801 [message queues](#)⁶²
802 • Sending and receiving messages to communicate with a specific D-Bus
803 well-known name (`talk` access)
804 • Permission to own (provide) specific D-Bus well-known names (`own` access)

805 Portals

806 [Flatpak portals](#)⁶³ are similar to Android [intents](#). These components expose a
807 subset of desktop functionality as D-Bus services that can be used by contained
808 applications: they are part of the security boundary between a contained app
809 and the rest of the desktop session. The aim is for portals to get the user’s per-
810 mission to carry out actions, while keeping it as implicit as possible, avoiding an
811 “are you sure?” step where feasible. For example, if an application asks to open
812 a file, the user’s permission is implicitly given by them selecting the file in the
813 file-chooser dialog and pressing OK: if they do not want this application to open
814 a file at all, they can deny permission by cancelling. Similarly, if an application
815 asks to stream webcam data, the expected UX is for GNOME’s Cheese app or
816 a similar non-GNOME app to appear, open the webcam to provide a preview
817 window so they can see what they are about to send, but not actually start
818 sending the stream to the requesting app until the user has pressed a “Start”
819 button. When defining the API “contracts” to be provided by applications in
820 that situation, portal designers need to be clear about whether the provider is
821 expected to obtain confirmation like this: in most cases we anticipate that it
822 will be expected to do this.

823 If this sort of implicit permission is not feasible for a particular portal, it is
824 possible for the portal implementation to fall back to a model similar to [iOS](#), by
825 asking the user for explicit consent to access particular data. Flatpak provides a
826 portal-facing API (the *permissions store*) with which a portal can check whether
827 the user already gave permission for particular operations, or store the fact that
828 the user has now given permission. Each portal can define its own permissions,
829 but app-bundles cannot normally do so.

830 There is currently no user interface for the user to review previously-granted
831 permissions and revoke them if desired, but one could be added in future, again
832 similar to [iOS](#).

833 Unlike Android intents, different Flatpak portals use different mechanisms to
834 send the result of a request to the portal back to the requesting app-bundle. For
835 example, many portals send and receive small requests and results over D-Bus,
836 but the file chooser makes the selected file available in a FUSE filesystem that
837 is visible inside the Flatpak sandbox. This avoids having to stream the whole
838 file over D-Bus, which could be very slow and inefficient, particularly the file is

⁶¹[https://manpages.debian.org/svipc\(7\)](https://manpages.debian.org/svipc(7))

⁶²[https://manpages.debian.org/mq__overview\(7\)](https://manpages.debian.org/mq__overview(7))

⁶³<https://github.com/flatpak/flatpak/wiki/Portals>

839 very large and the app will carry out random access within it (such as seeking
840 within a video).

841 More information on Flatpak portals can be found in the article [The flatpak
842 security model, part 3](#)⁶⁴.

843 iOS

844 The iOS 10 model for permissions is a hybrid of the [intents/portals](#) approaches,
845 and the approach of pre-declaring Android permissions. Apps that need access
846 to sensitive APIs (analogous to portals) must provide a description of why that
847 access is required. This gives the app-store curator an opportunity to check
848 that these permissions make sense, as with Android permissions. However,
849 unlike Android, user consent is requested at the time the app tries to exercise
850 that access, not during installation. The given description is included in the
851 prompt, and can be used to justify why access is needed.

852 There is also a user interface for the user to review previously-granted permis-
853 sions, and revoke them if desired.

854 Permissions not described in this document

855 The usage descriptions that are the closest equivalent of permissions in iOS
856 appear to be a subset of the [Cocoa Info.plist keys](#)⁶⁵, where `Info.plist` is the
857 iOS equivalent of our [application bundle metadata](#)⁶⁶. They exist in the same
858 namespace as non-permission-related keys such as human-readable copyright
859 notices.

860 Usage descriptions not corresponding to a use-case in this document include:

- 861 • `NSCameraUsageDescription`
- 862 • `NSHealthShareUsageDescription`
- 863 • `NSHealthUpdateUsageDescription`
- 864 • `NSHomeKitUsageDescription`
- 865 • `NSMotionUsageDescription` (accelerometer)
- 866 • `NSRemindersUsageDescription`
- 867 • `NSSiriUsageDescription`
- 868 • `NSSpeechRecognitionUsageDescription`
- 869 • `NSVideoSubscriberAccountUsageDescription`

⁶⁴<https://blogs.gnome.org/alex1/2017/01/24/the-flatpak-security-model-part-3-the-long-game/>

⁶⁵<https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html>

⁶⁶<https://jwd.pages.apertis.org/apertis-website/concepts/application-bundle-metadata/>