



Permissions

1 Contents

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

36 This document extends the higher-level [Applications](#)¹ and [Security](#)² design doc-
37 uments to go into more detail about the Flatpak-based permissions model.

38 Applications can perform many functions on a variety of user data. They may
39 access interfaces that read data (such as contacts, network state, or the users
40 location), write data, or perform actions that can cost the user money (like send-
41 ing SMS). As an example, the [Android](#) operating system has a comprehensive

¹<https://www.apertis.org/concepts/archive/application/applications/>

²https://www.apertis.org/concepts/archive/application_security/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 discuss the permissions model currently defined by Flat-
63 pak and thus in use in Apertis. This document aims to define a general approach
64 to permissions, so that future work on a particular feature that requires permis-
65 sions only requires the designer of that feature to define a permission or a series
66 of permissions, and does not require the designer of the feature to design the
67 entire permissions framework.

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

71 This document does not aim to define permissions for features that are not
72 already present in Apertis or supported by Flatpak, or that are not already
73 well-understood. For example, defining detailed permissions for [egress filtering](#)⁶
74 is out of scope.

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

⁴https://www.apertis.org/concepts/archive/application_security/security/#integrity-confidentiality-and-availability

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

⁶https://www.apertis.org/concepts/archive/application_framework/egress_filtering/

75 Flatpak

76 Apertis currently uses [Flatpak](#)⁷ for its application distribution, and thus Flatpak
77 serves as the implementation of the permissions detailed in this document.

78 Permissions model

79 Flatpak permissions are categorized according to the resource being controlled.
80 Available permissions include:

- 81 • Hardware-accelerated graphics rendering via [Direct Rendering Manager](#)⁸
82 devices
- 83 • Hardware-accelerated virtualization via [Kernel-based Virtual Machine](#)⁹
84 devices
- 85 • Full access to the host’s device nodes
- 86 • Sharing specific filesystem areas on a read-only or read/write basis
- 87 • Sharing the host’s X11 socket (not used in production on Apertis)
- 88 • Sharing the host’s Wayland socket (always available to graphical programs
89 on Apertis)
- 90 • Full access to the host’s D-Bus session bus
- 91 • Full access to the host’s D-Bus system bus
- 92 • Full access to the host’s PulseAudio socket
- 93 • Sharing the host system’s network namespace (Internet and LAN access)
- 94 • Sharing the host system’s IPC namespace (this does not control D-Bus or
95 `AF_UNIX` sockets, but would allow the app-bundle to be treated as uncon-
96 fined for the purposes of services that use [Unix System V IPC](#)¹⁰ or [POSIX](#)
97 [message queues](#)¹¹)
- 98 • Sending and receiving messages to communicate with a specific D-Bus
99 well-known name (`talk` access)
- 100 • Permission to own (provide) specific D-Bus well-known names (`own` access)

101 Portals

102 Flatpak’s [XDG portals](#)¹² are similar to Android `intents`. These components
103 expose a subset of desktop functionality as D-Bus services that can be used
104 by contained applications: they are part of the security boundary between a
105 contained app and the rest of the desktop session. The aim is for portals to
106 get the user’s permission to carry out actions, while keeping it as implicit as
107 possible, avoiding an “are you sure?” step where feasible. For example, if an
108 application asks to open a file, the user’s permission is implicitly given by them
109 selecting the file in the file-chooser dialog and pressing OK: if they do not want
110 this application to open a file at all, they can deny permission by cancelling.

⁷<https://flatpak.org/>

⁸https://en.wikipedia.org/wiki/Direct_Rendering_Manager

⁹https://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine

¹⁰[https://manpages.debian.org/svipc\(7\)](https://manpages.debian.org/svipc(7))

¹¹[https://manpages.debian.org/mq_overview\(7\)](https://manpages.debian.org/mq_overview(7))

¹²<https://flatpak.github.io/xdg-desktop-portal/portal-docs.html>

111 Similarly, if an application asks to stream webcam data, the expected UX is
112 for GNOME’s Cheese app or a similar non-GNOME app to appear, open the
113 webcam to provide a preview window so they can see what they are about to
114 send, but not actually start sending the stream to the requesting app until
115 the user has pressed a “Start” button. When defining the API “contracts” to
116 be provided by applications in that situation, portal designers need to be clear
117 about whether the provider is expected to obtain confirmation like this: in most
118 cases we anticipate that it will be expected to do this.

119 If this sort of implicit permission is not feasible for a particular portal, it is
120 possible for the portal implementation to fall back to a model similar to iOS, by
121 asking the user for explicit consent to access particular data. Flatpak provides a
122 portal-facing API (the *permissions store*) with which a portal can check whether
123 the user already gave permission for particular operations, or store the fact that
124 the user has now given permission. Each portal can define its own permissions,
125 but app-bundles cannot normally do so.

126 Different Flatpak portals use different mechanisms to send the result of a request
127 to the portal back to the requesting app-bundle. For example, many portals send
128 and receive small requests and results over D-Bus, but the file chooser makes
129 the selected file available in a FUSE filesystem that is visible inside the Flatpak
130 sandbox. This avoids having to stream the whole file over D-Bus, which could
131 be very slow and inefficient, particularly the file is very large and the app will
132 carry out random access within it (such as seeking within a video).

133 More information on Flatpak portals can be found in the article [The flatpak
134 security model, part 3](#)¹³.

135 Use cases

136 Internet access

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

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

143 A simple game might not require any network access at all, or might only require
144 the indirect network access (launching URIs) that can be obtained by sending
145 messages to the XDG portals¹⁴.

146 Many intermediate levels of Internet access are possible, but for the purposes of
147 this document we do not consider them. See the [Egress filtering design notes](#)

¹³<https://blogs.gnome.org/alex1/2017/01/24/the-flatpak-security-model-part-3-the-long-game/>

¹⁴<https://flatpak.github.io/xdg-desktop-portal/portal-docs.html>

148 on the [Apertis website](#)¹⁵ for initial work on finer-grained control.

149 **Security implications** An application with Internet access might be compro-
150 mised by malicious inputs from the Internet (integrity failure). If an application
151 cannot contact the Internet, we can be confident that it cannot be subject to this,
152 although it could still be compromised by malicious content that is downloaded
153 locally and passed to it by another entity via the [XDG portals](#)¹⁶.

154 If an application with Internet access is compromised either remotely or by
155 opening malicious local content, it could be induced to send private data to an
156 attacker-controlled server (a confidentiality failure). This attack applies equally
157 to applications with access to a download manager like Newport, because the
158 private data could be encoded in the URI to be downloaded. If the download
159 manager offers control over request headers such as cookies or the `HTTP_REFERER`,
160 the private data could also be encoded in those. Applications that register their
161 own mime type handlers service could also be susceptible to this attack, but
162 only if the service that invokes the handler (such as the [XDG portals](#)¹⁷) will
163 pass the files without user interaction, *and* the handler for those URIs will fetch
164 them without user interaction.

165 If an application with Internet access is compromised, but does not already
166 contain malicious code to carry out actions of the attacker's choice (the *payload*),
167 a common technique is to download a payload from a server controlled by the
168 attacker. In particular, this allows an attacker to alter the payload over time
169 according to their current requirements, for example to form a botnet that
170 can be used for multiple purposes. An application with access to a download
171 manager like Newport is equally susceptible to this, even if it cannot access the
172 Internet itself, because it can ask Newport to download the new payload from
173 the attacker's server. However, an application that can only request opening
174 a URI, e.g. via [XDG portals](#)¹⁸, is not susceptible to this attack, because such
175 applications are not allowed to see the result of the HTTP request.

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

¹⁵https://www.apertis.org/concepts/archive/application_framework/egress_filtering/

¹⁶<https://flatpak.github.io/xdg-desktop-portal/portal-docs.html>

¹⁷<https://flatpak.github.io/xdg-desktop-portal/portal-docs.html>

¹⁸<https://flatpak.github.io/xdg-desktop-portal/portal-docs.html>

185 **In other systems** In **Android**, the web browser and podcast player described
186 in the use-cases above would have the `INTERNET` permission, but the game would
187 not. Using the Android `DownloadManager` service (equivalent to `Newport`) also
188 requires `INTERNET` permission, because it enables most of the same attacks as
189 direct Internet access.

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

191 **Geolocation**

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

196 **Security implications** The user's geographical location is sensitive informa-
197 tion, especially if it is precise, and is valuable to criminals. In some cases
198 disclosing it would be a threat to personal safety.

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

203 **In other systems** In **Android** the navigation app-bundle would have `AC-`
204 `CESS_FINE_LOCATION`, the restaurant guide would have `ACCESS_COARSE_LOCATION` and
205 the e-book reader would have neither.

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

208 **Initiating a phone call**

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

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

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

219 **Security implications** If an app-bundle can initiate calls without user con-
220 sent, this will result in the user’s microphone being connected to the call recipi-
221 ent, which is a confidentiality (privacy) failure. Making undesired calls can also
222 cost the user money, and in particular a malicious app author might place calls
223 to a premium rate number that pays them.

224 **In Flatpak** The contact management app-bundle would have `talk` access to
225 a D-Bus service offering immediate phone dialling, for example the Telepathy
226 Account Manager, or to a group of services, for example Telepathy. The restau-
227 rant guide and the e-book reader would not, but would be able to launch a `tel:`
228 URI, resulting in the system’s phone dialer app being shown, giving the user
229 the opportunity to confirm or cancel.

230 **In other systems** In **Android**, the contact management app-bundle would
231 have the `CALL_PHONE` permission. The restaurant guide and the e-book reader
232 would not, but would still be able to launch an intent, which would be handled
233 in much the same way as the `tel:` URI.

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

237 **Shared file storage**

238 A media player with a gallery-style user experience might require the ability to
239 read media files stored on external storage (a USB thumb drive or externally-
240 accessible SD card), or in a designated [shared area](#)¹⁹.

241 Similarly, a media player might require access to media indexing and browsing
242 as described in the [Media Management concept design](#)²⁰.

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

245 **Security implications** App-bundles with write access to this shared storage
246 can modify or delete media files; if this is done inappropriately, that would be
247 an availability or integrity failure. App-bundles with read access can observe
248 the media that the user consumes, which could be considered privacy-sensitive;
249 uncontrolled access would be a confidentiality failure. Malicious app-bundles
250 with write access could also write malformed media files that were crafted to
251 exploit security flaws in other app-bundles, in the platform, or in other devices
252 that will read the same external storage device, leading to an integrity failure.

¹⁹https://www.apertis.org/concepts/archive/application_framework/application-layout/#shared-data

²⁰https://www.apertis.org/concepts/archive/application_media/media-management/

253 **In Flatpak** Any directory of interest can be mapped into the filesystem names-
254 pace of sandboxed processes, either read-only or read/write, via the `filesystems`
255 metadata field. Values like `xdg-music` and `xdg-download/Podcasts` make common
256 use cases relatively straightforward, and provide considerably finer-grained con-
257 trol than some other systems listed below (most particularly Android).

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

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

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

267 **Launcher**

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

269 A vendor-specific application launcher, such as the [Maynard] in the Apertis
270 reference user interface, needs to list all the application entry points on the
271 system together with their metadata. It also needs to launch those entry points
272 on-demand.

273 **Security implications** Holding this permission negates the Apertis platform’s
274 usual concept of [application list privacy](#)²¹: an app-bundle with this permission
275 can enumerate the entry points, which is valuable if an attacker wishes to identify
276 particular user (fingerprinting). If unintended app-bundles gain this access, it
277 is a confidentiality failure.

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

280 **In other systems** **Android** does not appear to restrict the visibility of other
281 app-bundles.

282 **iOS** restricts the visibility of other app-bundles, although [fingerprinting](#)²² can
283 be carried out by abusing inter-app communication. Because iOS is a single-
284 vendor system, the security mechanisms used by platform components and by
285 the equivalent of our built-in app bundles do not have public documentation.

²¹https://www.apertis.org/concepts/archive/application_framework/application-entry-points/#security-and-privacy-considerations

²²<https://arxiv.org/abs/1605.08664>

286 **Settings**

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

288 Suppose a vendor has a [system preferences application](#)²³ that provides an
289 overview of all [system settings](#)²⁴, [user settings](#)²⁵ and [app settings](#)²⁶. That
290 application needs to list the app settings belonging to all store and built-in
291 app-bundles, and needs the ability to change them, without prompting the
292 user.

293 **Security implications** Holding this permission negates the Apertis platform’s
294 usual concept of [application list privacy](#)²⁷, similar to the [Launcher](#) use case.

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

298 **In Flatpak** When used with any platform relying on dconf-backed GSettings
299 (such as GNOME), granting write access to `dconf` (by making its files readable in
300 the sandbox, and granting `talk` access to the dconf service) gives unconstrained
301 access to all settings.

302 **In other systems** In [Android](#), the `CHANGE_CONFIGURATION` permission grants
303 the ability to change system configuration in some limited ways, and the
304 `WRITE_SETTINGS` permission grants the ability to carry out more settings changes.

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

308 **Restricted subsets of settings**

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

²³https://www.apertis.org/concepts/archive/application_customization/preferences-and-persistence/#user-interface

²⁴https://www.apertis.org/concepts/archive/application_customization/preferences-and-persistence/#system-settings

²⁵https://www.apertis.org/concepts/archive/application_customization/preferences-and-persistence/#user-settings

²⁶https://www.apertis.org/concepts/archive/application_customization/preferences-and-persistence/#app-settings

²⁷https://www.apertis.org/concepts/archive/application_framework/application-entry-points/#security-and-privacy-considerations

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

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

323 **In other systems** In **Android**, the photo viewer might have the `SET_WALLPAPER`
324 and `SET_WALLPAPER_HINTS` permissions, the travel-related app-bundle might have
325 `SET_TIME_ZONE`, and the media player might have `MODIFY_AUDIO_SETTINGS`.

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

327 **Granting permission on first use**

328 The author of a hotel booking app-bundle includes a feature to locate nearby
329 hotels by using the Apertis geolocation API. Because [users are more likely to grant permission to carry out privacy-sensitive actions if they can understand why it is needed](#)²⁸, the app author does not want the Apertis system to prompt
331 for access to the geolocation feature until the user actively uses that particular
332 feature.
333

334 **Not granting permission on first use** Conversely, an automotive vendor
335 wishes to minimize driver distraction in order to maximize safety. When the
336 same hotel booking app-bundle attempts to use geolocation while the vehicle is
337 in motion, the platform vendor might want the Apertis system to **not** prompt
338 for access to the geolocation feature, contrary to the wishes of the app author.
339 Instead, the user should be given the opportunity to enable geolocation at a
340 time when it is safe to do so, either during app-bundle installation or as a
341 configuration/maintenance operation while the vehicle is stationary at a later
342 time.

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

345 **In Flatpak** Flatpak prompts for permission to carry out each privileged op-
346 eration at the time of first use, with some pragmatic exceptions: lower-level
347 permissions, such as access to direct rendering devices for 3D games or direct
348 access to the host filesystem, are implemented in a way that precludes that
349 model. These are set up at installation time, and can be overridden by user

²⁸<https://savvyapps.com/blog/how-to-create-better-user-permission-requests-in-ios-apps>

350 configuration. When a Flatpak app is launched, it is given the level of access
351 that was appropriate at launch time.

352 **In other systems** iOS prompts at the time of first use, similarly to Flatpak,
353 but the mentioned exceptions do not apply here.

354 **Android** 6.0 and later has the same behaviour as iOS. Older Android versions
355 configured all permissions at installation time, with a simple UX: the user must
356 either accept all required permissions, or abort installation of the app. Some
357 permissions, notably access to shared storage (the real or emulated SD card),
358 were implemented in a way that precluded runtime changes: app processes
359 with access to shared storage ran with one or more additional Unix group IDs,
360 granting them DAC permission to the appropriate areas of the filesystem.

361 **Tightening control**

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

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

372 **In Flatpak** App-bundles can specify a minimum Flatpak version. There is
373 is currently no mechanism to specify a target API level, although one could be
374 inferred from the runtime branch that the app-bundle has chosen to use, such
375 as `org.aperties.headless.Platform//v2022` or `org.apertis.hmi.Sdk//v2023`.

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

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

386 **Loosening control**

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

393 **Security implications** This use-case is only applicable if the Apertis devel-
394 opers have decided that the security implications of the permission in question
395 (in this example, querying the paint colour) are not significant.

396 **In Flatpak and other systems** We are not aware of any permissions that
397 have been relaxed like this in Flatpak, Android, or iOS, but it would be straight-
398 forward for any of these frameworks to do so: they would merely have to stop
399 presenting a user interface for that permission, and make requests for it always
400 succeed.

401 **Changing access**

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

405 Later, that user becomes more concerned about their privacy. They want to
406 continue to use the Facebook app-bundle, but prevent it from accessing their
407 new locations. They use a user interface provided by the system vendor, perhaps
408 a [system preferences application](#)²⁹, to reconfigure the permissions granted to the
409 Facebook app-bundle so that it cannot access their location.

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

413 **Security implications** This use-case is applicable if the user's perception of
414 the most appropriate trade-off between privacy and functionality changes over
415 time.

416 **In Flatpak** Flatpak provides access to its permission store³⁰ via D-Bus APIs,
417 which would allow e.g. a settings application to modify the runtime granted
418 permissions of other applications.

²⁹https://www.apertis.org/concepts/archive/application_customization/preferences-and-persistence/#user-interface

³⁰<https://docs.flatpak.org/en/latest/libflatpak-api-reference.html#gdbus-org.freedesktop.impl.portal.PermissionStore>

419 **In other systems** Android 6.0 and later versions have a [user interface](#)³¹ to
420 revoke and reinstate broad categories of permissions. Older [Android](#) versions
421 had a hidden control panel named [App ops](#)³² controlling the same things at a
422 finer-grained level (individual permissions), but it was not officially supported.
423 [iOS](#) allows permissions to be revoked or reinstated at any time via the [Pri-
424 vacy page in its Settings app](#)³³, which is the equivalent of the [Apertis system
425 preferences application](#)³⁴.

426 **Continuing to run in the background**

427 Background services do not show any graphical windows, so to be useful they
428 must always run in the background.

429 **Security implications** Background programs consume resources, impacting
430 availability (denial of service). A background program that has other permis-
431 sions might make use of them without the user's knowledge: for example, if a
432 restaurant guide can track the user's location, this can be mitigated by only
433 allowing it to run, or only allowing it to make use of its permissions, while it
434 is (or was recently) visible, so that the user can only be tracked by the guide's
435 author at times when they are aware that this is a possibility.

436 Users might wish to be aware of which graphical programs have this property,
437 and user interfaces for managing permissions might display it in the same context
438 as other permissions, but it is not a permission in the sense that it is used to
439 generate security policies. Accordingly, it should potentially be handled outside
440 the scope of this document.

441 **In Flatpak** [XDG portals](#)³⁵ provide an API to allow an application to dy-
442 namically request permission to run in the background. There is currently no
443 support for foreground-only permissions.

444 **In other systems** Android does not have permissions that influence its be-
445 haviour for background programs.

446 [iOS](#) manages background programs via the `[UIBackgroundModes]` and `UIAppli-
447 cationExitsOnSuspend`³⁶ metadata fields. `NSSupportsAutomaticTermination`³⁷

³¹<https://www.howtogeek.com/230683/how-to-manage-app-permissions-on-android-6.0/>

³²<https://www.theguardian.com/technology/2015/jun/09/google-privacy-apple-android-lockheimer-security-app-ops>

³³<https://www.howtogeek.com/177711/ios-has-app-permissions-too-and-theyre-arguably-better-than-androids/>

³⁴https://www.apertis.org/concepts/archive/application_customization/preferences-and-persistence/#user-interface

³⁵<https://flatpak.github.io/xdg-desktop-portal/portal-docs.html>

³⁶[https://developer.apple.com/library/content/documentation/General/Reference/InfoPl
istKeyReference/Articles/iPhoneOSKeys.html#//apple_ref/doc/uid/TP40009252-SW23](https://developer.apple.com/library/content/documentation/General/Reference/Info.plistKeyReference/Articles/iPhoneOSKeys.html#//apple_ref/doc/uid/TP40009252-SW23)

³⁷[https://developer.apple.com/library/content/documentation/General/Reference/InfoPl
istKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW13](https://developer.apple.com/library/content/documentation/General/Reference/InfoPl
istKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW13)

448 is analogous, but is for desktop macOS.

449 **Running on device startup**

450 A background service might be run on device startup.

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

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

454 The security implications are essentially the same as **continuing to run in the**
455 **background**.

456 Users might wish to be aware of which graphical programs have this property,
457 and user interfaces for managing permissions might display it in the same context
458 as other permissions, but it is not a permission in the sense that it is used to
459 generate security policies. Accordingly, it is treated as outside the scope of this
460 document.

461 **In Flatpak** The exact same API in the **XDG portals**³⁸ that manages back-
462 ground services can also be used to request permission to run at startup.

463 **In other systems** In **Android**, a graphical program or service that runs in the
464 background would have the `RECEIVE_BOOT_COMPLETED` permission, which is specifi-
465 cally described as covering performance and not security.

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

468 **Potential future use-cases**

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

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

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

³⁸<https://flatpak.github.io/xdg-desktop-portal/portal-docs.html>

480 **Audio playback**

481 A music player requires the ability to play back audio while in the background.
482 A video player might require the ability to play ongoing audio, but only while its
483 window is in the foreground. An e-book reader might only require the ability to
484 play short notification sounds while in the foreground, or might not require any
485 ability to play sounds at all. A voice-over-IP calling client requires the ability
486 to play audio with an elevated priority while a call is in progress, pre-empting
487 other audio players.

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

490 **Security implications** Uncontrolled audio playback seems likely to cause
491 driver distraction. Additionally, if all applications can play back audio with a
492 priority of their choice, a malicious app-bundle could output silence at a high
493 priority as a denial of service attack (a failure of availability).

494 **In Flatpak** In **Flatpak**, audio playback currently requires making the
495 PulseAudio socket available to the sandboxed app, which also enables audio
496 recording and control. Finer-grained control over audio is planned for the
497 future.

498 **In other systems** In **Android** and **iOS**, audio playback does not require spe-
499 cial permissions.

500 **Audio recording**

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

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

506 **Security implications** An app-bundle that can record audio could record
507 private conversations in the vehicle (a failure of confidentiality).

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

510 **In other systems** In **Android**, audio recording requires the `RECORD_AUDIO` per-
511 mission.

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

513 **Bluetooth configuration**

514 A [system preferences application](#)³⁹, or a separate Bluetooth control panel built-
515 in app-bundle, might require the ability to reconfigure Bluetooth in detail and
516 communicate with arbitrary devices.

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

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

521 **Security implications** For the control panel use-case, communicating with
522 arbitrary devices might be an integrity failure if the app-bundle can reconfigure
523 the device or edit data stored on it, or a confidentiality failure if the app-bundle
524 can read sensitive data such as a phone's address book. The ability for untrusted
525 app-bundles to view MAC addresses and other unique identifiers would also be
526 a privacy problem.

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

529 **Flatpak** Full access could be achieved by configuring Flatpak's D-Bus filter
530 to allow `talk` access to BlueZ. There is currently no implementation of partial
531 access; this would likely require a Bluetooth [portal](#) service.

532 **In other systems** In [Android](#), the `BLUETOOTH` permission allows an app-bundle
533 to communicate with any Bluetooth device that is already paired. This is
534 stronger than is needed for a device-specific app-bundle. The `BLUETOOTH_ADMIN`
535 permission additionally allows the app-bundle to pair new Bluetooth devices.

536 In [iOS](#), the `NSBluetoothPeripheralUsageDescription`⁴⁰ metadata field controls
537 access to Bluetooth, which appears to be all-or-nothing. User consent is re-
538 quested the first time this permission is used, with the metadata field's content
539 included in the prompt.

540 **Calendar**

541 A general-purpose calendar/agenda user interface similar to [GNOME Calen-](#)
542 [dar](#)⁴¹ or the [AOSP Calendar](#)⁴² requires full read/write access to the user's cal-
543 endar.

³⁹https://www.apertis.org/concepts/archive/application_customization/preferences-and-persistence/#user-interface

⁴⁰https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW20

⁴¹<https://wiki.gnome.org/Apps/Calendar>

⁴²<https://fossdroid.com/a/standalone-calendar.html>

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

549 A non-calendaring application like an airline booking app-bundle might wish to
550 insert events into the calendar without further user interaction, or it might wish
551 to insert events into the calendar in a way that presents them for user approval,
552 for example by submitting a vCalendar file to the [XDG portals](#)⁴³.

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

554 **Security implications** The general-purpose user interface described above
555 would have the ability to send calendar events to a third party (a confidentiality
556 failure) or to edit or delete them (an integrity failure).

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

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

564 If the airline booking app-bundle operates via [intents](#), [portals](#) or a similar mech-
565 anism that will result in user interaction, a malicious app-bundle cannot insert
566 misleading events without user action, avoiding that integrity failure (at the
567 cost of a more prescriptive UX).

568 **In Flatpak** A general-purpose calendar might be given `talk` access to the
569 `evolution-data-server` service. There is currently no calendar [portal](#), but when
570 one is added it will presumably be analogous to Android intents.

571 **In other systems** In [Android](#), the `READ_CALENDAR` and `WRITE_CALENDAR` permis-
572 sions⁴⁴ are suitable for the general-purpose calendar use case. [Sync adapters](#)⁴⁵
573 receive different access; it is not clear from the Android documentation whether
574 their restriction to a specific subset of the calendar is enforced, or whether sync
575 adapters are trusted and assumed to not attack one another. Applications that
576 do not have these permissions, such as the hotel booking use-case above, can
577 use [calendar intents](#)⁴⁶ to send or receive calendar events, with access mediated

⁴³<https://flatpak.github.io/xdg-desktop-portal/portal-docs.html>

⁴⁴<https://developer.android.com/guide/topics/providers/calendar-provider.html#manifest>

⁴⁵<https://developer.android.com/guide/topics/providers/calendar-provider.html#sync-adapter>

⁴⁶<https://developer.android.com/guide/topics/providers/calendar-provider.html#intents>

578 through a general-purpose calendar user interface that is trusted to behave ac-
579 cording to the user’s intention. There is no way to prevent an app-bundle from
580 using those intents at all.

581 In **iOS**, the [NSCalendarsUsageDescription] metadata field controls access to
582 calendars. User consent is requested the first time this permission is used, with
583 the metadata field’s content included in the prompt.

584 **Contacts**

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

587 **In Flatpak** As with calendar access, a general-purpose contacts app-bundle
588 might be given `talk` access to the `evolution-data-server` service. There is cur-
589 rently no contacts **portal**, but when one is added it will presumably be analogous
590 to Android intents.

591 **In other systems** [Android contact management](#)⁴⁷ is analogous to calendar-
592 ing, using the `READ_CONTACTS` and `WRITE_CONTACTS` permissions or contact-specific
593 intents.

594 [iOS contact management][NSContactsUsageDescription] is analogous to iOS
595 calendaring.

596 **Inter-app communication interfaces**

597 Inter-app communication has not been designed in detail, but the draft design
598 on the Apertis website suggests that it might be modelled in terms of **inter-**
599 **face discovery**⁴⁸, with app-bundles able to implement “public interfaces”that
600 are made visible to other app-bundles. The draft design has some discussion of
601 how **restricting interface providers**⁴⁹ might be carried out by app-store curators.

602 Additionally, if app-bundles export public interfaces, this might influence
603 whether other applications are allowed to communicate with them: if a
604 particular public interface implies that other app-bundles will communicate
605 directly with the implementor, then the implementor’s AppArmor profile and
606 other security policies must allow that. A **sharing**⁵⁰ feature similar to the one
607 in Android is one possible use-case for this.

608 We recommend that this topic is considered as one or more separate concept
609 designs, with its security implications considered at the same time. This is

⁴⁷<https://developer.android.com/guide/topics/providers/contacts-provider.html>

⁴⁸https://www.apertis.org/concepts/archive/application_framework/interface_discovery/

⁴⁹https://www.apertis.org/concepts/archive/application_framework/interface_discovery/#Restricting_who_can_advertise_a_given_interface_2

⁵⁰https://www.apertis.org/concepts/archive/application_security/sharing/

610 likely to be more successful if a small number of specific use-cases are considered,
611 rather than attempting to define a completely abstract and general framework.

612 **In Flatpak** App-bundles that will communicate via D-Bus can be given `talk`
613 access to each other. If this is done, it is up to the app-bundles to ensure that
614 they do not carry out unintended actions in response to D-Bus method calls.

615 **In other systems** In **Android**, any app-bundle can define its own intents.
616 If it does, those intents can be invoked by any other app-bundle that holds
617 appropriate permissions, and it is up to the implementor to ensure that that is
618 a safe thing to do.

619 In **iOS**, any app-bundle can define non-standard URI schemes that it will handle,
620 and these non-standard URI schemes are the basis for inter-app communication.
621 There is no particular correlation between the URI scheme and the app's identity
622 (the iOS equivalent of our bundle IDs), and there have been successful attacks
623 against this, including the **URL masquerade attack**⁵¹ identified by FireEye.

624 **Non-use-cases**

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

626 **App's own data**

627 Each app-bundle should be allowed to read and write its own data, including
628 its own **app settings**⁵². However, this should not need any special permissions,
629 because it should be granted to every app-bundle automatically: accordingly, it
630 is outside the scope of this document. App settings are part of the scope of the
631 **Preferences and Persistence**⁵³ concept design, and other per-app private data
632 are in the scope of the **Applications**⁵⁴ concept design.

633 Similarly, programs from each app-bundle should be allowed to communicate
634 with other programs from the same app-bundle (using any suitable mechanism,
635 including D-Bus) without any special permissions, with the typical use-case
636 being a user interface communicating with an associated background service.
637 Because it does not require special permissions, that is outside the scope of this
638 document.

639 **Platform services**

640 This permissions framework is not intended for use by platform services, re-
641 gardless of whether they are upstream projects (such as `systemd`, `dbus-daemon`

⁵¹https://www.fireeye.com/blog/threat-research/2015/04/url_masques_on_apps.html

⁵²https://www.apertis.org/concepts/archive/application_customization/preferences-and-persistence/#app-settings

⁵³https://www.apertis.org/concepts/archive/application_customization/preferences-and-persistence/

⁵⁴<https://www.apertis.org/concepts/archive/application/applications/>

642 and Tracker), developed specifically for Apertis (such as the Newport download
643 manager), or developed for a particular vendor. Platform services should con-
644 tinue to contain their own AppArmor profiles, polkit rules and other security
645 metadata.

646 **Infotainment cameras**

647 **Android** and **iOS** mobile phones and tablets typically have one or more cameras
648 directed at the user or their surroundings, intended for photography, videocon-
649 ferencing, augmented reality and entertainment. The use cases and security
650 implications are very similar to audio recording, so we believe there is no need
651 to describe them in detail in this document.

652 **App-specific permissions**

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

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

659 If an app-bundle by a different author that requests one of these app-specific
660 permissions is installed, a description provided by the app-bundle that declared
661 the permission is shown to the user when they are choosing whether to allow
662 the requesting app-bundle to be installed. If the requesting app-bundle is in-
663 stalled before the declaring app-bundle, then its request to use that permission
664 is silently denied.

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

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

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

669 **General notes on other systems**

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

673 **Android**

674 Android includes permissions in its XML manifest file.

- 675 • **Introduction**⁵⁵

⁵⁵<https://developer.android.com/guide/topics/manifest/manifest-intro.html>

- 676 • [Permission API reference](#)⁵⁶
- 677 • [Permission group API reference](#)⁵⁷
- 678 • [Declaring that a permission is needed](#)⁵⁸

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

- 680 • [Permission element](#)⁵⁹
- 681 • [Permission group element](#)⁶⁰
- 682 • [Permission tree element](#)⁶¹

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

685 **Permissions not described in this document** The following access per-
686 missions, available as of API level 25, do not match any use-case described in
687 this document. Deprecated and unsupported permissions have been ignored
688 when compiling this document.

689 Normal permissions:

- 690 • ACCESS_LOCATION_EXTRA_COMMANDS
- 691 • ACCESS_NETWORK_STATE
- 692 • ACCESS_NOTIFICATION_POLICY
- 693 • ACCESS_WIFI_STATE
- 694 • ADD_VOICEMAIL
- 695 • BATTERY_STATS
- 696 • BODY_SENSORS
- 697 • BROADCAST_STICKY
- 698 • CAMERA
- 699 • CHANGE_NETWORK_STATE
- 700 • CHANGE_WIFI_MULTICAST_STATE
- 701 • CHANGE_WIFI_STATE
- 702 • DISABLE_KEYGUARD
- 703 • EXPAND_STATUS_BAR
- 704 • GET_ACCOUNTS
- 705 • GET_ACCOUNTS_PRIVILEGED
- 706 • GET_PACKAGE_SIZE
- 707 • INSTALL_SHORTCUT
- 708 • KILL_BACKGROUND_PROCESSES
- 709 • NFC
- 710 • PROCESS_OUTGOING_CALLS
- 711 • READ_CALL_LOG

⁵⁶<https://developer.android.com/reference/android/Manifest.permission.html>

⁵⁷<https://developer.android.com/reference/android/Manifest.permission.html>

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

⁵⁹<https://developer.android.com/guide/topics/manifest/permission-element.html>

⁶⁰<https://developer.android.com/guide/topics/manifest/permission-group-element.html>

⁶¹<https://developer.android.com/guide/topics/manifest/permission-tree-element.html>

- 712 • READ_EXTERNAL_STORAGE
- 713 • READ_PHONE_STATE
- 714 • READ_SMS
- 715 • READ_SYNC_SETTINGS
- 716 • READ_SYNC_STATS
- 717 • RECEIVE_MMS
- 718 • RECEIVE_SMS
- 719 • RECEIVE_WAP_PUSH
- 720 • REORDER_TASKS
- 721 • REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
- 722 • REQUEST_INSTALL_PACKAGES
- 723 • SEND_SMS
- 724 • SET_ALARM
- 725 • SET_TIME_ZONE
- 726 • SET_WALLPAPER
- 727 • SET_WALLPAPER_HINTS
- 728 • TRANSMIT_IR
- 729 • USE_FINGERPRINT
- 730 • USE_SIP
- 731 • VIBRATE
- 732 • WAKE_LOCK
- 733 • WRITE_CALL_LOG
- 734 • WRITE_EXTERNAL_STORAGE
- 735 • WRITE_SETTINGS
- 736 • WRITE_SYNC_SETTINGS

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

- 738 • ACCOUNT_MANAGER
- 739 • Several permissions starting with `BIND_` that represent the ability to bind to the identity of a platform service, analogous to the ability to own platform services' D-Bus names in Apertis
- 740
- 741
- 742 • BLUETOOTH_PRIVILEGED
- 743 • Several permissions starting with `BROADCAST_` that represent the ability to broadcast messages, analogous to the ability to own a platform service's D-Bus name and send signals in Apertis
- 744
- 745
- 746 • CALL_PRIVILEGED
- 747 • CAPTURE_AUDIO_OUTPUT
- 748 • CAPTURE_SECURE_VIDEO_OUTPUT
- 749 • CAPTURE_VIDEO_OUTPUT
- 750 • CHANGE_COMPONENT_ENABLED_STATE
- 751 • CLEAR_APP_CACHE
- 752 • CONTROL_LOCATION_UPDATES
- 753 • DELETE_CACHE_FILES
- 754 • DELETE_PACKAGES
- 755 • DIAGNOSTIC
- 756 • DUMP

- 757 • FACTORY_TEST
- 758 • GLOBAL_SEARCH, held by the global search framework to give it permission
- 759 to contact every global search provider
- 760 • INSTALL_LOCATION_PROVIDER
- 761 • INSTALL_PACKAGES
- 762 • LOCATION_HARDWARE
- 763 • MANAGE_DOCUMENTS
- 764 • MASTER_CLEAR
- 765 • MEDIA_CONTENT_CONTROL
- 766 • MODIFY_PHONE_STATE
- 767 • MOUNT_FORMAT_FILESYSTEMS
- 768 • MOUNT_UNMOUNT_FILESYSTEMS
- 769 • PACKAGE_USAGE_STATS
- 770 • READ_FRAME_BUFFER
- 771 • READ_LOGS
- 772 • READ_VOICEMAIL
- 773 • REBOOT
- 774 • SEND_RESPOND_VIA_MESSAGE
- 775 • SET_ALWAYS_FINISH
- 776 • SET_ANIMATION_SCALE
- 777 • SET_DEBUG_APP
- 778 • SET_PROCESS_LIMIT
- 779 • SET_TIME
- 780 • SIGNAL_PERSISTENT_PROCESSES
- 781 • STATUS_BAR
- 782 • SYSTEM_ALERT_WINDOW
- 783 • UPDATE_DEVICE_STATS
- 784 • WRITE_APN_SETTINGS
- 785 • WRITE_GSERVICES
- 786 • WRITE_SECURE_SETTINGS
- 787 • WRITE_VOICEMAIL

788 **Intents** Holding a permission is not required to use an *intent* that implicitly
789 asks the user for permission, such as taking a photo by sending a request to
790 the system camera application, which will pop up a viewfinder provided by the
791 system camera application, allowing the user to either take a photo when they
792 are ready, or cancel by pressing the Back button; if the user takes a photo, it
793 is sent back to the requesting application as the result of the intent. This is
794 conceptually similar to Flatpak **portals**.

795 iOS

796 The iOS 10 model for permissions is a hybrid of the **intents/portals** approaches,
797 and the approach of pre-declaring Android permissions. Apps that need access
798 to sensitive APIs (analogous to portals) must provide a description of why that
799 access is required. This gives the app-store curator an opportunity to check that

800 these permissions make sense, as with Android permissions. However, unlike
801 Android, user consent is requested at the time the app tries to exercise that
802 access, not during installation. The given description is included in the prompt,
803 and can be used to justify why access is needed.

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

806 **Permissions not described in this document** The usage descriptions that
807 are the closest equivalent of permissions in iOS appear to be a subset of the
808 [Cocoa Info.plist keys](#)⁶², where `Info.plist` is the iOS equivalent of our [applica-
809 tion bundle metadata](#)⁶³. They exist in the same namespace as non-permission-
810 related keys such as human-readable copyright notices.

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

- 812 • `NSCameraUsageDescription`
- 813 • `NSHealthShareUsageDescription`
- 814 • `NSHealthUpdateUsageDescription`
- 815 • `NSHomeKitUsageDescription`
- 816 • `NSMotionUsageDescription` (accelerometer)
- 817 • `NSRemindersUsageDescription`
- 818 • `NSSiriUsageDescription`
- 819 • `NSSpeechRecognitionUsageDescription`
- 820 • `NSVideoSubscriberAccountUsageDescription`

⁶²<https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html>

⁶³https://www.apertis.org/concepts/archive/application_framework/application-bundle-metadata/