



Applications

1 Contents

2	Traditional package managers are unfit for applications	2
3	Terminology	3
4	Graphical program	3
5	Bundle	3
6	Store account	3
7	Software Categories	3
8	Pre-installed Applications	5
9	Responsibilities of the Application Store	6
10	Identifying applications	7
11	Application Releasing Process	10
12	Application Installation Tracking	10
13	Digital Rights Management	11
14	Permissions	12
15	Data Storage	13
16	Extending Storage Capabilities	14
17	Application Management	15
18	Store Applications	15
19	System Extensions	18
20	License Agreements	20
21	Application Run Time Life-Cycle	21
22	Start	21
23	Background Operation	22
24	End	23
25	Frozen	25
26	Resource Usage	25
27	Applications not Written for <i>Apertis</i>	26
28	Responsibilities of the Application Manager	26
29	References	27

30 This document is intended to give a high-level overview of application handling
31 by Apertis. Topics handled include the storage of applications and related data
32 on the device, the format of the distributable application bundle files, how
33 they're integrated into the system, and how the system manages them at run-
34 time. Topics related to the development of applications are covered by several
35 other designs.

36 Unfortunately, the term “application” has seen a lot of misuse in recent times.
37 While many mobile devices have an “application store” that distributes “applica-
38 tion packages”, what is actually in one of those packages may not fit any sensible
39 definition of an application – as an example, on the Nokia N9 one can download
40 a package from the application store that adds MSN Messenger capabilities to
41 the existing chat application.

42 To avoid ambiguity, this document will avoid using “application” as a jargon
43 term. Instead, we use two distinct terms for separate concepts that could in-
44 formally be referred to as applications: *graphical programs*, and *application*

45 *bundles*. See [Terminology](#).

46 Apertis is a multiuser system; see the [multiuser](#)¹ design document for more on the
47 specifics of the multiuser experience and the division of responsibilities between
48 middleware and HMI elements.

49 Apertis has first shipped with a custom application framework to address the
50 needs described in this document, see [canterbury legacy application framework](#)².
51 The custom legacy framework has been replaced with an evolution based on
52 upstream components, see [application framework](#)³.

53 **Traditional package managers are unfit for applications**

54 Apertis relies heavily on a traditional packaging system to compose the base
55 OS. However, it does not rely on it to distribute the composed system as it is not
56 a good fit for the use-cases Apertis addresses, see [system updates and rollback](#)⁴
57 for more details. Similarly, a traditional packaging system is not a good fit for
58 applications in Apertis since:

- 59 • Apertis relies on a immutable base OS to implement a robust update
60 mechanism, see [system updates and rollback](#)⁵ for more details. This means
61 that a traditional package manager is not used to distribute updates on
62 the field and that the writable application storage should be kept separate
63 from the read-only base OS.
- 64 • Application bundles don't depend on each other – this makes creating
65 a new special purpose package management solution much easier, and
66 removes the main reason for customizing an existing solution to fit Apertis-
67 specific needs.
- 68 • Much of the complexity in application bundle handling (DRM, rollbacks,
69 communicating security “permissions” to the user) is not part of the ex-
70 isting package management tools, and is not interesting to the upstream
71 tool maintainers.
- 72 • Applications can have conflicting dependencies which can't be shipped as
73 part of the base OS and should be somehow bundled with the application
74 itself.

¹<https://em.pages.apertis.org/apertis-website/concepts/multiuser/>

²<https://em.pages.apertis.org/apertis-website/architecture/canterbury-legacy-application-framework/>

³<https://em.pages.apertis.org/apertis-website/concepts/application-framework/>

⁴<https://em.pages.apertis.org/apertis-website/concepts/system-updates-and-rollback/>

⁵<https://em.pages.apertis.org/apertis-website/concepts/system-updates-and-rollback/>

75 Terminology

76 Graphical program

77 A *graphical program* is a program with its own UI drawing surface, managed
78 by the system’s window manager. This matches the sense with which “appli-
79 cation” is traditionally used on desktop/laptop operating systems, for instance
80 referring to Notepad or to Microsoft Word.

81 Bundle

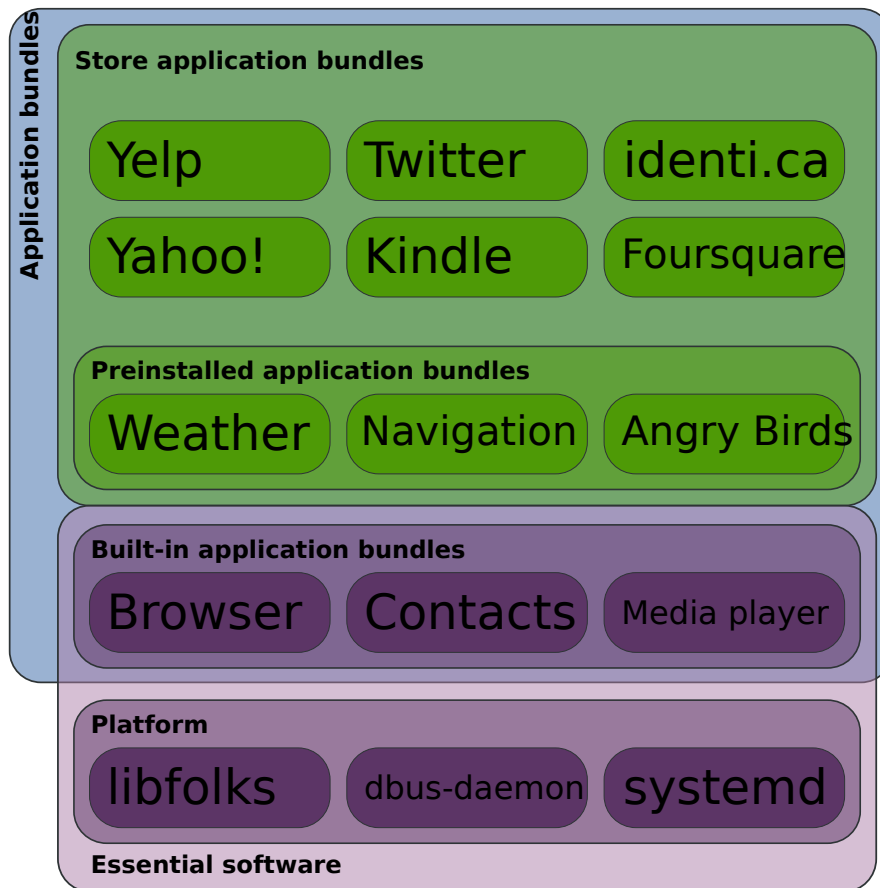
82 A *bundle* or *application bundle* is a group of functionally related components
83 (be they services, data, or programs), installed as a unit. This matches the sense
84 with which “app” is typically used on mobile platforms such as Android and iOS;
85 for example, we would say that an Android .apk file contains a bundle. Some
86 systems refer to this concept as a *package*, but that term is strongly associated
87 with dpkg/apt (.deb) packages in Debian-derived systems, so we have avoided
88 that term in this document.

89 Store account

90 The [Digital rights management](#) section discusses *store accounts*, anticipated
91 to have a role analogous to Google Play accounts on Android or Apple Store
92 accounts on iOS. If these accounts exist, we recommend against using the term
93 “user” for them, since that would be easily confused with the users found in the
94 Multiuser design document; it is not necessarily true that every user has access
95 to a store account, or that every store account corresponds to only one user.

96 Software Categories

97 The software in a Apertis device can be divided into three categories: *platform*,
98 *built-in application bundles* and *store application bundles*. Of these categories,
99 some store application bundles may be *preinstalled*.



100

101 The *platform* is comprised of all the facilities used to boot up the device and
 102 perform basic system checks and restorations. It also includes the infrastruc-
 103 tural services on which the applications rely, such as the session manager, win-
 104 dow manager, message bus and configuration storage service, and the software
 105 libraries shared between components.

106 *Built-in application bundles* are components that have a structure analogous
 107 to that of an application bundle from the application store, but can only be
 108 upgraded as part of an operating system upgrade, not separately. This should
 109 include all software laid on top of the platform that is on the critical path of
 110 user-facing basic functionality, and hence cannot be removed or upgraded except
 111 by installing a new operating system; this might include basic software such as
 112 the browser, email reader and various settings management applications.

113 The platform and built-in applications combine to make up *essential software*:
 114 the bare minimum Apertis will always have installed. Essential software has
 115 strict requirements both in terms of reliability and security.

116 **Store application bundles** are application bundles developed by third-parties
117 to be used as add-ons to the system: they are not part of the system image and
118 are made available for installation through the application store instead. While
119 they may be important to the user, their presence is not required to operate the
120 device properly.

121 It is important to note that store application bundles can be shipped pre-
122 installed on the device, which provides OEMs with a flexible way of providing
123 differentiation or a more complete user experience by default.

124 **Pre-installed Applications**

125 On most software platforms there are two kinds of applications that come pre-
126 installed on the device: what we call built-in application bundles and regular
127 store application bundles. The difference between built-in application bundles
128 and regular store application bundles that just happen to come pre-installed is
129 essentially that the former are considered part of the system's basic functionality,
130 are updated along with the system and cannot be removed.

131 Taking Apple's iPad as an example, we can see that approach being applied:
132 Safari, Weather, Mail, Camera and so on are built into the system.

133 See <http://www.apple.com/ipad/built-in-apps/> for a list

134 They cannot be removed and they are updated through system updates. Apple
135 doesn't seem to include any store applications pre-installed, though.

136 The Android approach is very similar: applications such as the browser are not
137 removable and are updated with the system, but it's much more common to
138 have store applications be pre-installed, including Google applications such as
139 Gmail, Google Maps, and so on.

140 The reason why browsers, mail readers, contacts applications are built-in soft-
141 ware that come with the system is they are considered integral parts of the core
142 user experience. If one of these applications were to be removed the user would
143 not be able to utilize the device at all or would have a lot of trouble doing so:
144 listening to music, browsing the web and reading email are basic expectations
145 for any mobile consumer device.

146 A second reason which is also important is that these applications often provide
147 basic services for other applications to call upon. The classic example here is
148 the contacts application that manages contacts used by text messaging, instant
149 Internet messaging, email, and several other use cases.

150 **Case Study: a navigation application, how would it work?**

151 The navigation application was singled out as a case that has requirements and
152 features that intersect those of built-in applications and those of store applica-
153 tions. On the one hand, the navigation application is core functionality, which

154 means it should be part of the system. On the other hand, it should be possible to make the application extensible or upgradable, enabling the selling of updated maps, for instance.

157 Collabora believes that the best way to solve this duality is to separate program and data, and to follow the lead of other platforms and their app stores in providing support for in-app purchases. This functionality is used often by games to provide additional characters, scenarios, weapons and such, but also used by applications to provide content for consumption through the application, such as magazine issues and also maps.

163 For such a feature to work, it needs to be provided as an API that applications can use to talk to the app store to place orders and to verify which data sets the user should be allowed to download. The actual data should be hosted at the app store for downloading post-validation. The disposition of the data, such as whether it should be made available as a single file or several, whether the file or files are compressed or not, should be left for the application author to decide on based on what makes more sense for the application.

170 **Responsibilities of the Application Store**

171 The application store will be responsible for packaging a developer's store application bundle into a bundle file along with a "store directory"(see [Store directory](#)) that contains some generated metadata and a signature. Special "SDK" system images will provide software development tools and allow the installation of unsigned packages, but the normal "target" system image will not allow the installation of packages that don't contain a valid store signature.

177 The owner of the store, via the signing authority of the application store, will have the ability to accept or reject any application to be run on Apertis. By disallowing any form of "self publication" by application developers, the store owner can ensure a consistent look and feel across all applications, screen applications for malicious behavior, and enforce rigorous quality standards.

182 However, pre-publication screening of applications will represent a significant time commitment, as even minor changes to applications must undergo thorough testing. High priority security fixes from developers may need to be given a higher priority for review and publication, and the priority of application updates may need to be considered individually. System updates will correspond to the busiest periods for both internal and external developers, and the application store will experience significant pressure at these times.

189 In order for the the application update system to work properly, each new release of an application needs to have a version number greater than the previous release. The store may need to make changes to application metadata between the developer's releases of that application. To allow the store to increment the application version without interfering with the developer's version numbers, the store will maintain a "store version" number to be appended to the developer's

195 version number. The store version will start at 1 and be reset to 1 any time the
196 developer increases the application version.

197 As an example, if a developer releases an application with a version of 2.5 for
198 publication, the store will release this under the version 2.5-1.

199 This approach closely resembles the versioning scheme used in dpkg
200 and rpm packages, which combine an “upstream version” with a
201 “packaging revision”

202 If the store ever needs to push an update to this application without waiting for
203 the developer to create a new version, then the store version can be incremented
204 from 1 to 2, and version 2.5-2 can be released without any intervention from the
205 developer. This is expected to be an uncommon occurrence, but may be done
206 to correct packaging problems, or even to disable an application if it’s found to
207 have critical security flaws and the developer isn’t responsive.

208 Identifying applications

209 During the design of other Apertis components, it has become clear that several
210 areas of the system design would benefit from a consistent way to identify and
211 label application bundles and programs. In particular, the ability to provide
212 a security boundary where inter-process communication is used relies on being
213 able to identify the peer, in a way that ensures it cannot be impersonated.

214 An application has several strings that might reasonably act as its machine-
215 readable name in the system:

- 216 • the name of the application bundle, being the [Flatpak app-id](#)⁶ or the name
217 discussed in [Application bundle metadata](#)⁷
- 218 • the D-Bus well-known name or names taken by the program(s) in the
219 bundle, for instance via GLib’s GApplication interface
- 220 • the name of the AppArmor profile attached to the program(s) in the bun-
221 dle, if they have them
- 222 • the name(s) of the freedesktop.org .desktop file(s) associated with the
223 program(s), if they have them
- 224 • the name of the systemd user service (.service file) associated with the
225 program(s), if they have them

226 We propose to align all of these as follows, matching the approach used by
227 [Flatpak for its application identifiers](#)⁸:

- 228 • The *bundle ID* is a case-sensitive string matching the syntactic rules for
229 a D-Bus interface name, i.e. two or more components separated by dots,

⁶<http://docs.flatpak.org/en/latest/using-flatpak.html#identifiers>

⁷<https://em.pages.apertis.org/apertis-website/concepts/application-bundle-metadata/>

⁸<http://docs.flatpak.org/en/latest/using-flatpak.html#identifiers>

230 with each component being a traditional C identifier (one or more ASCII
231 letters, digits, or underscores, starting with a non-digit).

232 This scheme makes every bundle ID a valid D-Bus well-known name,
233 but excludes certain D-Bus well-known names (those containing the hy-
234 phen/minus). This allows hyphen/minus to be used in filenames without
235 ambiguity, and facilitates the common convention in which a D-Bus ser-
236 vice's main interface has the same name as its well-known name.

- 237 • Application authors should be strongly encouraged to use a DNS name
238 that they control, with its components reversed (and adjusted to follow
239 the syntactic rules if necessary), as the initial components of the bundle
240 ID. For instance, the owners of collabora.com and 7-zip.org might choose
241 to publish `com.collabora.MyUtility` and `org._7_zip.Decompressor`, respec-
242 tively. This convention originated in the Java world and is also used for
243 Android application packages, Tizen applications, D-Bus names, GNOME
244 applications and so on.
- 245 • Application-specific filenames on disk should be based on the bun-
246 dle name. For instance, `com.collabora.MyUtility` might have its
247 program, libraries and data in appropriate subdirectories of `/Appli-
248 cations/com.collabora.MyUtility/`. Built-in applications should also
249 use the bundle ID; for instance, the Frampton executable might be
250 `/usr/Applications/org.apertis.Frampton/bin/frampton`.
- 251 • App-store curators should not allow the publication of a bundle whose
252 name is a prefix of a bundle by a different developer, or a bundle that is
253 in the essential software set. App-store curators do not necessarily need
254 to verify domain name ownership in advance, but if a dispute arises, the
255 app-store curator should resolve it in favour of the owner of the relevant
256 domain name.
- 257 • Well-known namespaces used by platform components (such as `aper-
258 tis.org`, `freedesktop.org`, `gnome.org`, `gtk.org`) should be restricted to app
259 bundles associated with the relevant projects. Example projects provided
260 in SDK documentation should use the names that are reserved for
261 examples (see [RFC2606](http://www.rfc-editor.org/info/rfc2606)⁹), such as `example.com`, but app-store curators
262 should not publish bundles that use such names.
- 263 • Programs in a bundle may use the D-Bus well-known name correspond-
264 ing to the bundle ID, or any D-Bus well-known name for which the
265 bundle ID is a prefix. For instance, the `org.apertis.Frampton` bundle
266 could include programs that take the bus names `org.apertis.Frampton`,
267 `org.apertis.Frampton.UI` and/or `org.apertis.Frampton.Agent`.
- 268 • Programs in a bundle all use the same AppArmor profile. As a result of the
269 convention that AppArmor profile names are equal to on-disk filenames,
270 its name must start with the installation location based on the bundle ID.

⁹<http://www.rfc-editor.org/info/rfc2606>

271 Further, to allow upgrade and rollback to be carried out without making
272 the system insecure, we currently require that every store app-bundle's
273 AppArmor profile is deterministically derived from the bundle ID, by being
274 exactly `/Applications/${bundle_id}/**` where `${bundle_id}` represents the
275 bundle ID. (See [AppArmor profiles](#) for rationale for this choice.)

276 For instance, all programs in the `org.apertis.Frampton` built-in
277 app-bundle would run under a profile whose name starts with
278 `/usr/Applications/org.apertis.Frampton/`, and all programs in the
279 `com.example.ShoppingList` store app-bundle would run under a profile
280 named `/Applications/com.example.ShoppingList/**`.

- 281 • If a program is a systemd user or system service, the service file should be
282 the program's D-Bus well-known name followed by `.service`, for example
283 `org.apertis.Frampton.Agent.service`. Similarly, if a program has a freedesk-
284 top.org `.desktop` file, its name should be the program's D-Bus well-known
285 name followed by `.desktop`, for example `org.apertis.Frampton.UI.desktop`.

286 In particular, using the bundle ID in the AppArmor profile name makes it trivial
287 for a D-Bus service to identify the application bundle to which a peer belongs:

- 288 • the service can learn the AppArmor profile name via the standard `GetCon-`
289 `nectionCredentials` D-Bus method call
- 290 • if the profile starts with `/Applications/`, followed by a syntactically valid
291 bundle ID, followed by either end-of-string or `/`, then the peer is a store
292 app-bundle with the bundle ID that appears after the second `/`
- 293 • if the profile starts with `/usr/Applications/`, followed by a syntactically
294 valid bundle ID, followed by either end-of-string or `/`, then the peer is a
295 built-in app-bundle with the bundle ID that appears after the third `/`
- 296 • if the profile starts with one of the well-known executable directories
297 for the platform (`/usr/`, `/bin/`, `/lib/` etc.) and does not start with
298 `/usr/Applications`, or the profile has the special value `unconfined` indicat-
299 ing the absence of AppArmor confinement, then the peer is a platform
300 component
- 301 • otherwise, the peer is in an unknown category and must not be given any
302 special privileges

303 The same approach can be used across any other IPC channel on which a process
304 can securely query the peer's LSM (Linux Security Module) context, such as
305 Unix sockets or `kdbus`.

306 A library available to platform services should provide a recommended imple-
307 mentation of this algorithm.

308 **Application Releasing Process**

309 Once application testing is complete and an application is ready to be dis-
310 tributed, the application releasing process should contain at least the following
311 steps:

- 312 • Verify that the application’s bundle ID does not collide with any bundle
313 by a different publisher (in the sense that neither is a prefix of the other).
- 314 • Generate an AppArmor profile for the application based on its [permis-
315 sions]
- 316 • Generate the application’s **Store directory**.
- 317 • Make the application available at the store.

318 **Application Installation Tracking**

319 The System Updates and Rollback design describes a method of migrating set-
320 tings and data from an existing Apertis system to another one. To work prop-
321 erly, the application store would need to have a list of applications installed on
322 a specific Apertis device.

323 If the application store keeps a database of vehicle IDs and the applications
324 purchased for them, this will help in order to facilitate software updates and to
325 simplify software re-installation after a system wipe.

326 The application store can only know which applications have been downloaded
327 for use in a specific vehicle – with no guarantee of a persistent Internet con-
328 nection, the store has no way to know whether the application has really been
329 installed or subsequently uninstalled. The store also can’t reliably track what
330 version of an application is installed.

331 If an application is downloaded on a computer with a web browser (presumably
332 for installation via external media), the store shouldn’t assume it was actually
333 installed anywhere. Only applications installed directly to the device should be
334 logged as installed. When the user logs in to the store (or the device logs into
335 the store with the users credentials to check for updates), the list of installed
336 packages can be synchronized.

337 If an application is installed from a USB storage device the application manager
338 could write a synchronization file back to the device that could subsequently be
339 uploaded back to the application store from a web browser. Care should be
340 taken to ensure these files can’t be used by malicious users to steal applications
341 – the store should check that the applications listed in the synchronization file
342 have been legitimately purchased by the user and the file’s contents should be
343 discarded if they have not.

344 To perform a migration for a device that hasn’t had a consistent Internet con-
345 nection, the device could be logged into the store to synchronize its application
346 list prior to beginning the migration process.

347 **Digital Rights Management**

348 Details of how DRM is to be used in Apertis are not finalized yet, but some
349 options are presented here.

350 The store is in a convenient position to enforce access control methods for appli-
351 cations. When an application is purchased, the application store can generate
352 the downloadable bundle with installation criteria built in.

353 The installation could be locked in the following ways:

- 354 • Locked to a specific vehicle ID – it will only install on a specific vehicle.
355 The Apertis unit will refuse to install the application if the vehicle ID does
356 not match the ID embedded in the downloaded application package.
- 357 • Locked to a specific device ID – it will only install on a specific Apertis
358 unit.
- 359 • Locked to a customer ID – It will only install for a specific person, as
360 represented by their store account - presumably a store account must be
361 present and logged in for this to work. The store account is assumed
362 to be analogous to an Apple Store or Google Play account: as noted in
363 [Terminology](#), we recommend avoiding the term “user” here, since a store
364 account does not necessarily correspond 1:1 to the “users” discussed in the
365 Multiuser design document.

366 Any “and” combination of these 3 locks could also be used. For example, an
367 application bundle may only be installable to a specific device in a specific
368 vehicle (in other words, locked to vehicle ID and device ID) – if the Apertis
369 unit is placed in another vehicle, or the vehicle’s Apertis unit is replaced, the
370 application bundle would not be installable.

371 Conversely, rights could also be combined with the “or” operator, such as al-
372 lowing an application bundle to be installed if either the correct Apertis unit
373 is used, or the correct vehicle. Collabora recommends these combinations not
374 be implemented. Most of the combinations provided by “or” aren’t obviously
375 useful.

376 It might also be useful to distribute some packages in an unlocked form – free
377 software, ad sponsored software, or demo software may not require any locking
378 at all. Ultimately, this is a policy decision, not a technical one, as they could
379 just as easily be locked to the downloader’s account.

380 Note that these are all install time checks, and if a device is moved to another
381 vehicle after successfully installing a bundle, it may result in running an app
382 somewhere that an application developer or OEM didn’t intend it to be run. In
383 order to prevent this from happening, it would be more reliable to do launch-
384 time testing of the applications.

385 The store would generate a file to be bundled with the application that listed the
386 launch criteria, and the application manager would check those criteria before

387 launching the application for use.

388 It should be considered that launch time testing would require a user to be
389 logged in to the store in some way if the applications are to be keyed to a store
390 account. This would make it impossible to launch certain applications when
391 Apertis is without network connectivity, and could be a source of frustration for
392 end users.

393 **Permissions**

394 Applications can perform many functions on a variety of user data. They may
395 access interfaces that read data (such as contacts, network state, or the users
396 location), write data, or perform actions that can cost the user money (like send-
397 ing SMS). As an example, the Android operating system has a comprehensive
398 [manifest](#)¹⁰ that govern access to a wide array of functionality.

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

402 See the [Permissions concept design](#)¹¹ for further details.

403 Application developers will declare the permissions their application depends
404 on in [application bundle metadata](#)¹², and Apertis will allow a user to approve
405 a subset of an application's required permissions.

406 There are some difficulties in allowing users to accept only some of the permis-
407 sions an application developer expected their software to have access to:

- 408 • Some of the permissions will be controlled by an AppArmor profile gen-
409 erated by the application store. The user is merely accepting the profile,
410 actually changing it would not be trivial.
- 411 • AppArmor profiles are per-application, not per user. AppArmor profiles
412 would need to be changed on user switch if different users required different
413 permission configurations for the same applications.
- 414 • A huge testing burden is placed on the application developer if they can't
415 rely on the requested permissions. They must test their applications in
416 all possible configurations.
- 417 • The permissions may be required for the application developer's business
418 model – be that network permissions for displaying advertising, or GPS
419 information for crowd sourcing traffic information. Allowing the user to
420 restrict permissions in these situations would be unfair to the developer.

421 To mitigate some of these problems, there must be two kinds of permissions:
422 required and optional. Required permissions are those that can't be removed

¹⁰<http://developer.android.com/reference/android/Manifest.permission.html>

¹¹<https://em.pages.apertis.org/apertis-website/concepts/permissions/>

¹²<https://em.pages.apertis.org/apertis-website/concepts/application-bundle-metadata/>

423 from an application – such as anything granted by the AppArmor profile. If a
424 user chooses to deny a required permission, an application can not run.

425 Optional permissions are handled by higher level APIs in the SDK and may be
426 influenced by system settings. Apertis-specific “wrapper” services that abstract
427 the functionality of lower level libraries can provide access controls. These wrap-
428 per services would act based on the individual user’s settings and preferences, so
429 each user would have control over the applications they use. Because these ser-
430 vices must act as a trust boundary between apps within the scope of a particular
431 user’s account, a privilege boundary must be imposed between the app bundle
432 and the wrapper service: to provide this boundary, they must be implemented
433 as a separate service process, rather than merely a library that is loaded by the
434 application program.

435 Some permissions may prove to be more of an annoyance than helpful to the
436 user. For example, if `Start` are employed by vast numbers of applications, users
437 may not wish to be informed every time a new application requires one. It may
438 be worth considering having some permission acceptance governed by system
439 settings, and only directly query the user if a permission is “important” (such
440 as sending SMS).

441 Data Storage

442 Applications will have access to several types of writable application storage.
443 Care should be taken to select the appropriate area as different areas are handled
444 differently if rollbacks (See `Roll-back`) occur. The storage types are:

- 445 • Application User – for settings and any other per-user files. In the event
446 of an application rollback, files in this area are rolled back with their
447 associated application.
- 448 • Application Everyone – for data that is rolled back with an application
449 but isn’t tied to a user account – such as voice samples or map data.
- 450 • Cache – for easily recreated data. If the system is low on storage space,
451 it may reclaim cache space for applications that aren’t currently running.
452 Caches will be cleared on update and rollback instead of being stored by
453 the rollback system.
- 454 • Shared – This area’s contents are not touched by the application manage-
455 ment framework for any reason. They are also not subject to any form of
456 rollback. This area is intended for storage of videos, music, photos and
457 other data in standard formats that aren’t tied to a single application,
458 analogous to `/sdcard` on Android devices. Since Apertis space may be lim-
459 ited, and since it is thought that users will usually want to share media
460 between accounts, the data in this storage area will be accessible by all
461 users. More details on this are available in the `multiuser`¹³ design.

¹³<https://em.pages.apertis.org/apertis-website/concepts/multiuser/>

462 The [XDG Base Directory Specification](#)¹⁴ environment variables will guide ap-
463 plications to find the appropriate locations for the different storage types.

464 **Extending Storage Capabilities**

465 It may be desirable for some Apertis devices to allow the user to install an SD
466 card to increase storage capacity. Since SD cards are removable – possibly even
467 at runtime – they present some problems that need to be addressed:

- 468 • Allowing applications to be run from SD cards makes it more difficult to
469 prevent software piracy.
- 470 • An SD card should be properly unmounted by the system before being
471 physically removed from the device.

472 It is recommended that SD card storage not be used for the installation of
473 applications or any manner of system software, as this could give users a way
474 to run untrusted code, or tamper with application settings or data in ways the
475 developers haven't anticipated. Media files are obvious candidates for placement
476 on this type of removable storage, as they don't provide key system functionality,
477 and are not trusted data.

478 If it is critical that applications (or other trusted data, such as navigation maps)
479 be run off of removable storage, allowing the system to “format” the device be-
480 fore use, deleting all data already on the card and replacing it with an encrypted
481 BTRFS filesystem would allow a secure method of placing application storage
482 on the device.

483 The dm-crypt [LUKS](#)¹⁵ system would be used to encrypt the storage device
484 using a random binary key file. These key files would be generated at the time
485 the external storage device is formatted and stored along with the device serial
486 number. One way to generate a key file would be to read an appropriate number
487 of bytes (such as 32) from `/dev/random`.

488 The key store will be in a directory in the var subvolume (but not in `/var/lib`)
489 as the var subvolume is not tracked by system rollbacks. If the key files were in
490 a volume subject to rollbacks, they would disappear and render external storage
491 unreadable after a system rollback that crossed their creation date.

492 It is imperative that the key store not be accessible to a user as it would allow
493 them to directly access their removable storage device on another computer and
494 potentially copy and distribute applications.

495 The device could be recognized by its label as reported by the `blkid` command,
496 and added to the startup application scan in [Boot time procedures](#).

497 If this is extended to multiple SD cards, difficulties arise in deciding which
498 storage device to install an application to. Either configuration options will

¹⁴<http://standards.freedesktop.org/basedir-spec/basedir-spec-latest.html>

¹⁵<https://gitlab.com/cryptsetup/cryptsetup>

499 need to be added to control this, or the device with the greatest free space at
500 the time of installation can be selected.

501 Many embedded devices require some manner of disassembly to remove an SD
502 card, preventing the user from removing it while the system is in operation (such
503 as a mobile phone that hides the SD card behind the battery). If an approach
504 such as this is used, there is no need for special “eject” procedures for the SD
505 storage. If this is not possible however, some manner of interface will need to
506 be provided so the user can safely unmount the SD card before removal.

507 If it’s physically possible for a user to remove the SD card while the system
508 is running, the operating system and applications may be exposed to difficult
509 to recover from situations and poorly tested code paths. These sorts of SD
510 card sockets should probably not be used for cards using the BTRFS filesystem.
511 Instead, the better tested FAT-32 filesystem should be used.

512 **Application Management**

513 Applications will be distributed by the application store as compressed “appli-
514 cation bundles” containing programs and services that can be launched in a
515 variety of ways – with the limitation that an application bundle can’t contain
516 more than one program launchable from the application launcher, or more than
517 one agent.

518 All communication with the application store will take place over a
519 secure HTTPS connection.

520 The metadata in this bundle provides information about the application such
521 as it’s user friendly name, services it needs from the system (such as querying
522 the GPS), permissions it needs from the user, and the versions of system APIs
523 it depends on.

524 An application bundle may provide back-ends to existing system functionality
525 and add new features to installed software without necessarily adding any new
526 applications to the application manager. These are called “system extensions”
527 and are detailed in [System extensions](#).

528 **Store Applications**

529 **Acquisition**

530 Applications will be made available through the application store as compressed
531 files.

532 Since Apertis may have limited or no Internet connectivity, it must be possible
533 to download an application elsewhere and install it from a USB storage device.
534 Even if Internet connectivity is available the download process must be reliable
535 – it must be possible to resume a partially completed application download if the
536 connection is broken or Apertis is shut down before the download completes.

537 A background download service will be provided by Apertis (See [Reliable down-](#)
538 [load service](#)). This service will continue downloads if they are interrupted or if
539 the system is restarted. When the download is completed, or if the download
540 is incapable of being completed, a callback will be made to the requester via
541 D-Bus.

542 The user interface components will be able to query status from the download
543 service in order to display status about the installation – including a completion
544 percentage or a position in the download queue.

545 **Installation**

546 If an application is being installed directly from the store, the application bundle
547 metadata will be downloaded and the user allowed to select a subset of [permis-
548 sions] to allow, or cancel the installation. If the installation is to proceed, an
549 icon to be displayed in the launcher while the download and installation takes
550 place will now be acquired from the application store.

551 If the application is being provided from a USB device, the application bundle
552 metadata and icon are extracted from the application bundle.

553 Displaying an accurate progress indicator while installing an application is non-
554 trivial. One simple option is to include the full decompressed size of the ap-
555 plication in its metadata and send an update to the user interface occasionally
556 based on the amount of bytes written.

557 This assumes that “number of bytes left to install” directly correlates to “amount
558 of time left to completion”, and suffers from a couple of common problems:

- 559 • Eventually storage caches are filled and begin writing out causing a dra-
560 matic slowdown in apparent installation speed for larger applications.
- 561 • Decompression speed may vary for different parts of the same archive.

562 However, users are unlikely to notice even moderate inaccuracies in an instal-
563 lation percentage indicator, so this may be adequate without requiring compli-
564 cated development that may not solve these problems anyway.

565 **Upgrades**

566 If configured with a suitable Internet connection, the system will periodically
567 check whether upgrades are available for any store applications that have been
568 installed. Apertis will provide its vehicle ID to the application store and the
569 application store will reply with a list of the most recent versions of the ap-
570 plications authorized for the vehicle. If Apertis has had software installed or
571 removed without an Internet connection, the list of installed applications will
572 be synchronized with the store at this time.

573 Some users may voice concerns over the store’s tracking of all the installed
574 packages on their Apertis. It may be worth mentioning in a “privacy policy”

575 exactly what the data will be used for.

576 If no Internet connection is available, the user can still supply a newer version
577 of an application on a USB device to start an upgrade. They can acquire ap-
578 plication bundles from the store web page, which will provide the latest version
579 of applications for download. Old application versions will not be available
580 through the store.

581 Since the application store attempts to track installed applications, it could
582 notify a user by e-mail when updates are available, or show a list of updated
583 application when the user logs in to the store.

584 In order to allow application rollbacks to rollback the user data associated with
585 an application, all running instances of an application will have to be closed prior
586 to starting an upgrade for data coherency reasons. The user will be unable to
587 launch an instance of the application during the upgrade process. The system
588 won't recognize services, handlers, or launchable components of the application
589 until the final phase of installation is complete.

590 **Removal**

591 When a user removes the application, any personal settings and caches required
592 by the application will be automatically removed along with any user specific
593 data stored for rollback purposes – files the application has stored in general
594 storage will be left behind.

595 Removing a third-party music player shouldn't delete the user's music collection,
596 but it should delete any configuration information specific to that player. For
597 this to work properly, application developers need to be careful to store data in
598 the appropriate locations.

599 **Roll-back**

600 Apertis may have a per-application rollback system that allows an end user to
601 revert to the last installed version of an application (that is, a single previously
602 installed version will be kept when an upgrade is performed), with all their
603 settings and data in exactly the state it had been the last time they used it.

604 This rollback paradigm has some interesting quirks:

- 605 • If a user rolls back an application, all other users of that application will
606 also be rolled back. This allows one user to delete some of another user's
607 settings and personal data.
- 608 • As some software updates may contain critical security fixes, an ever grow-
609 ing blacklist will have to be maintained to prevent a user from rolling back
610 to potentially dangerous versions.
- 611 • Developers will have no control over what software versions their cus-
612 tomers are using, making long term support very difficult. They may

613 receive bug reports for bugs already fixed in newer versions of the soft-
614 ware.

- 615 • Old versions of applications may break if they interact with online services
616 that changed their protocols, or if Apertis APIs are deprecated.
- 617 • Application developers have to think very carefully about what data goes
618 into application storage (and is subject to rollbacks) and general storage
619 (which isn't). In reality, application developers will likely pay very lit-
620 tle attention to this distinction and the application store will carry this
621 burden.
- 622 • The effect of a system rollback on installed applications is unclear. If an
623 application has been upgraded twice since the last system update and a full
624 system rollback occurs it is possible for applications to have no launchable
625 version installed.
- 626 • In some cases an application rollback may not even be possible if the old
627 version of the application is not capable of running on the current version
628 of the system.
- 629 • Settings management tools like GSettings directly manipulate application
630 setting data and don't currently support the rollback system.

631 The settings problems can be mitigated by using the persistence API from the
632 SDK when writing applications, allowing Apertis to hide the complexity from
633 the application developer. Each application should have its own database of
634 settings instead of using a single system-wide database.

635 After application rollback, launching the application now will use the previously
636 installed version with all settings and private data in the state they were before
637 the upgrade.

638 **System Extensions**

639 In the context of Apertis, system extensions may refer to themes and skins
640 which provide global user interface changes, or plug-ins for existing frameworks
641 that aren't intended for extension by regular application developers.

642 Generally speaking, these will be purchasable add-ons that don't fit into the
643 category of "application", and are instead additions to basic system function-
644 ality. Examples include downloadable content that radically changes the visual
645 appearance of all applications under Apertis, or a plug-in that integrates Skype
646 with the contacts and communications software.

647 While these don't fit the standard role of an application, they are still made
648 available as bundles through the application store, and their installation is still
649 handled by the application manager.

650 The application bundle metadata will have a list of known extension "types",
651 and extension components inside an application bundle will be handled differ-

652 ently based on the specified type. There is no comprehensive list of extension
653 types, but “Telepathy connection manager” and “theme” will be the commonly
654 used examples in this document.

655 System extensions, being outside the realm of regular application developers,
656 are entitled to make assumptions about available libraries and frameworks that
657 applications are not. This makes rolling them back independently complicated,
658 and some simplifications are made by disallowing manual rollback of extensions,
659 and only rolling them back automatically with a system rollback.

660 **Installation**

661 There will be no difference between an application bundle or a “system extension
662 bundle” - and it may even be desirable to deploy an application with supporting
663 system extensions from the same bundle.

664 Most of the process for installing a bundle with system extensions will be no
665 different than the usual application installation process. However, the “applica-
666 tion specific metadata” configuration will include exporting files in the system
667 extension directories.

668 Depending on the extension, the newly installed extension may not be functional
669 until daemons are restarted, or programs rescan plug-in directories.

670 Determining what needs to be restarted can be difficult, and could be different
671 depending on what other system extensions have been installed. For simple
672 add-ons like themes, or Telepathy connection managers, no restarts or re-loads
673 should be required, so no special effort needs to be made.

674 For more invasive system extensions, the application manager can decide based
675 on the extension type in the application bundle metadata whether the function-
676 ality requires that the system be restarted. The user should be informed during
677 installation that new features will only be present next time they start their
678 vehicle.

679 **Upgrades**

680 There may be additional steps required based on the extension type – for ex-
681 ample, if a theme is being upgraded, the application manager should check if it
682 is the theme currently being used to render GUI elements. If it is, the system
683 may need to switch to a default theme before the upgrade begins, and switch
684 back after the upgrade finishes.

685 Apart from any extension type specific steps performed by the application man-
686 ager, the upgrade process will be exactly as described in [Upgrades](#).

687 **Removal**

688 Once again, the process only deviates from [Removal](#) by performing any specific
689 actions required by the extension type before following the standard procedure.

690 As an example, if the extension is a theme, the system should ensure it is not
691 currently in use before beginning the usual removal process.

692 **Rollback**

693 Like regular applications, a system rollback will automatically rollback system
694 extension components.

695 An intentional rollback will only need special steps at the start of the process,
696 dependent on the type of extension being handled.

697 Since system extensions are likely to be low level components, it may be a good
698 idea to disallow rolling them back in order to ensure important bug fixes can be
699 deployed.

700 **License Agreements**

701 Collabora does not have legal expertise in these matters, and any
702 authoritative information – especially if financial damages may be
703 involved – should be supplied by the appropriate legal advisers.

704 Each application may have its own license agreements, privacy policies, or other
705 stipulations a user must accept before they can use the application. Different
706 OEMs may have different requirements, and the legal requirements governing
707 the contents of these documents may vary from country to country.

708 Such licenses generally disclose information regarding the use of data collected
709 by an application or related services, define acceptable usage of the application
710 or services by a user, or discuss the warranty and culpability of the application
711 provider.

712 Regardless of content, Apertis should make all reasonable efforts to ensure a user
713 has agreed to the appropriate agreements before they may use an application.
714 The first step to accomplishing this goal is to require a user accept the license
715 agreement before downloading an application from the store.

716 As this only requires a single user to accept the agreement, and does nothing for
717 built-in applications, it is an incomplete solution. Requiring acceptance of the
718 license terms when an application is installed, or when it is enabled for a user's
719 account, would increase the likelihood that a user has agreed to the appropriate
720 license.

721 If license terms change between releases, it might be advisable to ask users
722 to accept the license terms on the first launch after an application update or
723 rollback as well.

724 Ultimately, there is no guarantee that the person using a Apertis account is the
725 person that agreed to an application's license.

726 Some licenses, such as the GPL, inform the user of their rights to obtain a copy
727 of the source code of the software. Licenses like this should be made available

728 to the user, but don't necessarily need to be displayed to the user unless the
729 user explicitly requests the information.

730 **Application Run Time Life-Cycle**

731 The middleware will assist UI components in launching and managing applica-
732 tions on Apertis. Application bundles can provide executable files (programs)
733 to be launched via different mechanisms, some of them user visible (perhaps
734 as icons in the application manager that will launch a graphical program), and
735 some of them implicit (like a connection manager for the Telepathy framework,
736 or a graphical program that does not appear in menus but is launched in order
737 to handle a particular request).

738 On a traditional Linux desktop, a graphical program doesn't generally make a
739 distinction between foreground and background operation, though it may watch
740 for certain events (input focus, window occlusion) that could be used to monitor
741 that status. Some mobile operating systems (Android, iOS) hide the details of
742 background operation from the user, some (WebOS, Meego) allow the user to
743 interact with background applications more directly.

744 The approach will be similar to that of Android and iOS – whether an applica-
745 tion (graphical program) is actually running is hidden from the user. The user
746 may either launch new applications or press the “back” button to return the
747 last running application.

748 From the user's perspective, applications will be persistent. When a user comes
749 back to an application they've previously used, it will be in the same state they
750 left it – even if the vehicle has been turned off and restarted.

751 **Start**

752 There are multiple ways in which a program associated with an application
753 bundle, whether graphical or not, can be started by Apertis:

- 754 • Direct launch – application bundles may contain an image or widget to be
755 displayed in the application launcher, which will launch a suitable graphi-
756 cal program. The name and icon shown in the application launcher is part
757 of the [entry point metadata](#)¹⁶.
- 758 • By data type association - The content-type (MIME type) of data is used
759 to select the appropriate application to handle the request. Applications
760 will provide a list of content-types (if any) that they handle in the [en-
761 try point metadata](#)¹⁷; activating the application with the corresponding
762 content type will launch the corresponding graphical program.
- 763 • Sharing back-ends – Applications may define sharing capabilities that al-
764 low other applications to launch them and send a receiver-limited amount

¹⁶<https://em.pages.apertis.org/apertis-website/concepts/application-entry-points/>

¹⁷<https://em.pages.apertis.org/apertis-website/concepts/application-entry-points/>

765 of data. Again, activating an application in this way would normally start
766 a corresponding graphical program.

- 767 • Agents – persistent non GUI processes that provide a background com-
768 ponent for applications. These will be launched automatically at boot
769 time or immediately after application installation. An application bundle
770 will contain at most a single agent, and the [permissions] will include an
771 “agent” permission to allow users to know they’re installing an application
772 that uses one.

773 We refer to the programs that are launched in these ways as *entry points*.

774 Collabora feels that the first three types of launch should be under the control
775 of the application manager. See [Responsibilities of the application manager](#)

776 Another method of launching processes is present – D-Bus activation. If a D-Bus
777 client attempts to use a known-name for a service that isn’t currently running,
778 D-Bus will search its configuration files for an appropriate handler to launch.
779 This sort of activation is more useful for system level developers, and won’t be
780 used to launch graphical programs.

781 During pre-publication review by the app store, careful attention should be paid
782 to application bundles that wish to use agents, and the resource consumption of
783 the agents. The concept does not scale – it creates a system where the number
784 of installed application bundles can dramatically affect run-time performance
785 as well as system boot-up time.

786 When a program in an application bundle is started by the application manager,
787 in certain circumstances the manager will need to take extra steps. For the first
788 launch of a built-in application, or the first launch after one has been updated,
789 a subvolume will need to be created for storing user data.

790 **Background Operation**

791 More than one graphical program may be running at the same time, but the
792 user can only directly interact with a limited number of graphical programs at
793 any instant. For example, 1/3 of the screen may be giving driving directions
794 while the other 2/3 of the screen displays an e-mail application. Concurrently, in
795 the background, a music player may be running while several RSS feed readers
796 are periodically updating.

797 Background tasks may also be performed by long running agents. Agents run
798 for the duration of the user’s session, and are only terminated if the system
799 needs to unmount an application’s subvolume - either to shut down the system
800 or to upgrade or uninstall the application.

801 Graphical programs will be notified with a D-Bus message when they lose focus
802 and are relegated to background status – the response to this notification is ap-
803 plication dependent. If it has no need to perform processing in the background,

804 It may save its current state and self-terminate, or it may remain idle until re-
805 focused. Some graphical programs will continue to operate in the background
806 – for example, a navigation application might remain active in the background
807 and continue to give turn-by-turn instructions.

808 Graphical programs that need to perform tasks in the background will have to
809 set the “background” permission in their [permissions]. Ideally they should be
810 designed with a split between foreground and background components (perhaps
811 using a graphical program for the user interface and an agent for the background
812 part) instead.

813 If a background graphical program wishes to be focused, it can use the standard
814 method for requesting that a window be brought to the foreground.

815 **End**

816 Applications written for Apertis have persistent state, so from a user’s perspec-
817 tive they never end. Apertis still needs to be able to terminate applications to
818 manage resources, perform user switching, or prepare for shutdown.

819 Programs – either graphical or not – may be sent a signal by the middleware at
820 any time requesting that they save their state and exit. Even if the application
821 bundle has the background permission, its processes may still be signaled to
822 save its state in the case of a system shut-down or a user switch.

823 To prevent an application that doesn’t respond to the state saving request from
824 delaying a system shutdown or interfering with the system’s ability to manage
825 memory, processes will be given a limited amount of time (5 seconds) to save
826 their state before termination. Applications that don’t need to save state should
827 simply exit in response to this signal.

828 It should be noted that state saving is difficult to implement, and much of
829 the work is the responsibility of the application writer. While Apertis can
830 provide functions for handling the incoming signal and storing state data (See
831 [State saving convenience APIs](#)), the hardest part is determining exactly what
832 application state needs to be saved in order for the application to exit and
833 restart in exactly the same way it had been previously running.

834 There is no standard Linux API for saving application state. POSIX defines
835 `SIGSTOP` and `SIGCONT` signals for pausing and resuming programs, but these signals
836 don’t remove applications from memory or provide any sort of persistence over a
837 system restart. Since they’re unblockable by applications, the application may
838 be interrupted at any time with no opportunity to do any sort of clean-up.

839 However, some applications may react to changes in system state – such as
840 network connectivity. One method of preventing applications from reacting to
841 D-Bus messages, system state changes, and other signaling is to use `SIGSTOP`
842 to halt an application’s processing. The application becomes responsible for

843 handling whatever arises after `SIGCONT` causes it to resume processing (such as a
844 flood of events or network timeouts).

845 Automatically saving the complete state of an application is essentially impos-
846 sible - even if the entire memory contents are saved, the application may have
847 open files, or open connections on remote servers, or it may have configured
848 hardware like the GPU or a Bluetooth device.

849 For a web browser the state might be as simple as a URL and display position
850 within the page, and the page will be reloaded and redisplayed when the browser
851 is re-launched. However, if the user was in the middle of watching a streaming
852 video from a service that requires a log-in, the amount of information that needs
853 to be retained is larger and has potential security ramifications.

854 It's possible that a viewer application may exit and the file it was viewing be
855 deleted before the application's next start, making it impossible to completely
856 restore the previous application state. Applications will be responsible for han-
857 dling such situations gracefully.

858 **State Saving Convenience APIs**

859 As state saving is a difficult problem for developers, it seems appropriate for
860 Apertis to provide API to assist in performing the task accurately.

861 A minimal C language API for state saving could be developed consisting of:

- 862 • A way to register a callback for a D-Bus signal that requests a save of
863 state information.
- 864 • Functions to atomically serialize data structures to application storage.
- 865 • Functions to read previously serialized data structures into memory.
- 866 • Functions to clear previously saved state.
- 867 • Documentation and sample code for using the API.

868 This API's usage won't be mandatory for application developers.

869 If it is intended that users have control over which apps save state and which
870 merely close on exit, this API could also provide the code to handle those
871 configuration options.

872 Maemo provided, through [libosso](#)¹⁸ a very simple state saving API. It expected
873 relevant application state be contained within a single contiguous memory re-
874 gion, and provided a call that would write out this single memory area to some
875 abstract storage area that persists across reboots. On startup, an application
876 would attempt to re-read that memory, and if no pre-existing state was present,
877 would start over.

¹⁸http://maemo.org/api_refs/5.0/5.0-final/libosso/group__Statesave.html

878 **Frozen**

879 Interest has been expressed in creating a state with less resource consumption
880 than background-operation yet still having faster start-up times than ending
881 the process and saving state.

882 This is a very difficult problem to solve without application intervention – simply
883 dumping the memory contents of a process to long term storage won't be enough
884 to restore the application. File descriptors and network connections are tracked
885 by the kernel and would need to be re-established on process restart. The
886 network connections are especially problematic as the remote end would be
887 unaware of what was happening.

888 Having applications involved in the process may allow some form of task sus-
889 pension that reduces the perceived start-up time of a “frozen” application. In
890 response to a signal (presumably over D-Bus) from the application manager, a
891 running application could free easily re-created data, close down network con-
892 nections and remain dormant until the application manager gave it a signal to
893 resume regular operation.

894 **Resource Usage**

895 To make better use of the available memory, it's recommended that applications
896 listen to the cgroup notification [memory.usage_in_bytes](#)¹⁹ and when it gets
897 close to the limit for applications, start reducing the size of any caches they
898 hold in main memory. It may be good to do this inside the SDK and provide
899 applications with a [GLib object](#)²⁰ that will notify them.

900 In order to reduce the chances that the system will find itself in a situation
901 where lack of disk space is problematic, it is recommended that available disk
902 space is monitored and applications notified so they can react and modify their
903 behavior accordingly. Applications may chose to delete unused files, delete or
904 reduce cache files or purge old data from their databases.

905 The recommended mechanism for monitoring available disk space is for a dae-
906 mon running in the user session to call `statvfs (2)` periodically on each mount
907 point and notify applications with a D-Bus signal. Example code can be found
908 [in the GNOME project](#)²¹ which uses a similar approach (polling every 60 sec-
909 onds).

910 In case applications cannot be trusted to properly delete non-essential files, a
911 possibility would be for them to state in their [application bundle metadata](#)²²
912 where such files will be stored, so the system can delete them when needed.

¹⁹<https://www.kernel.org/doc/Documentation/cgroup-v1/memory.txt>

²⁰https://gitlab.gnome.org/GNOME/glib/merge_requests/1005

²¹<http://git.gnome.org/browse/gnome-settings-daemon/tree/plugins/housekeeping/gsd-disk-space.c#n693>

²²<https://em.pages.apertis.org/apertis-website/concepts/application-bundle-metadata/>

913 In order to make sure that malfunctioning applications cannot cause disruption
914 by filling filesystems, it would be required that each application writes to a
915 separate filesystem.

916 **Applications not Written for *Apertis***

917 It may be desirable to run applications (such as Google Earth) that were not
918 written for Apertis. These applications won't understand any custom signals
919 or APIs that Apertis provides, providing yet another reason to minimize those
920 and stick to upstream solutions as much as possible.

921 Non-Apertis applications should be treated as if they have the background per-
922 mission – they should not be killed unless the system is extremely low on re-
923 sources, or they will provide an inconsistent user experience when they don't
924 save state like a native Apertis application.

925 **Responsibilities of the Application Manager**

926 Application life-cycle is dictated by the application manager. When the user
927 interacts with an icon or a link, the application manager is responsible for
928 launching the appropriate application.

929 Collabora recommends that the application manager also be responsible for
930 content-type-based (MIME-type-based) launching. The manager could provide
931 a D-Bus interface through which it can be asked to launch an appropriate ap-
932 plication for a specific content type. A list of available handlers and their
933 invocation details would be gathered from the application entry points.

934 If multiple applications are capable of handling the same content-types, the
935 user may wish to have a way to select which one takes precedence. One pos-
936 sible solution is to have the application manager provide a dialog any time an
937 ambiguous content-type launch is required, allowing the users to choose their
938 preferred handler, with a check-box that can be set to remember the selection.
939 This is how the Android operating system handles this situation.

940 There are security concerns when allowing content-type-based application
941 launching – it can potentially lead to an untrusted source (like a web page)
942 being capable of launching a store application with a known security bug.

943 Giving the application manager control over content-type-based launches can
944 allow it to restrict the usage of content-type-based launching. The manager
945 would be able to deny certain applications the ability to launch handlers for
946 specific data types (perhaps the web browser should never be allowed to launch
947 a handler for a certain data type), filter the handlers available to an application
948 to only allow trusted built-in applications to be used, or allow a system upgrade
949 to blacklist a store application's handler while waiting for a fix from a third
950 party.

951 The application launcher is itself a built-in application, and as such its storage

952 is governed by system rollbacks. In the event of a system rollback, all of the
953 launcher’s settings (icon placement, for example) will be automatically reset to
954 the state they were in just prior to the last system upgrade.

955 **References**

956 This document references the following external sources of information:

- 957 • [XDG Base Directory Specification](#)²³
- 958 • [Apertis System Updates and Rollback](#)²⁴ design
- 959 • [Apertis Multiuser](#)²⁵ design
- 960 • [Apertis Supported API](#)²⁶ design
- 961 • [Apertis Preferences and Persistence](#)²⁷ design
- 962 • Eastlake 3rd, D. and A. Panitz, “Reserved Top Level DNS Names”,
963 BCP 32, RFC 2606, DOI 10.17487/RFC2606, June 1999 ([http://www.rfc-
editor.org/info/rfc2606](http://www.rfc-
964 editor.org/info/rfc2606))

²³<https://specifications.freedesktop.org/basedir-spec/latest/>

²⁴<https://em.pages.apertis.org/apertis-website/concepts/system-updates-and-rollback/>

²⁵<https://em.pages.apertis.org/apertis-website/concepts/multiuser/>

²⁶<https://em.pages.apertis.org/apertis-website/concepts/supported-api/>

²⁷<https://em.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/>