



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 . . . . .	16
19	System Extensions . . . . .	18
20	License Agreements . . . . .	20
21	Application Run Time Life-Cycle . . . . .	21
22	Start . . . . .	21
23	Background Operation . . . . .	23
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](#)<sup>1</sup> 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](#)<sup>2</sup>.  
51 The custom legacy framework is in the process of being replaced with an evolu-  
52 tion based upstream components, see [application framework](#)<sup>3</sup>.

### 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](#)<sup>4</sup>  
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](#)<sup>5</sup> 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.

---

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

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

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

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

<sup>5</sup><https://martyn.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

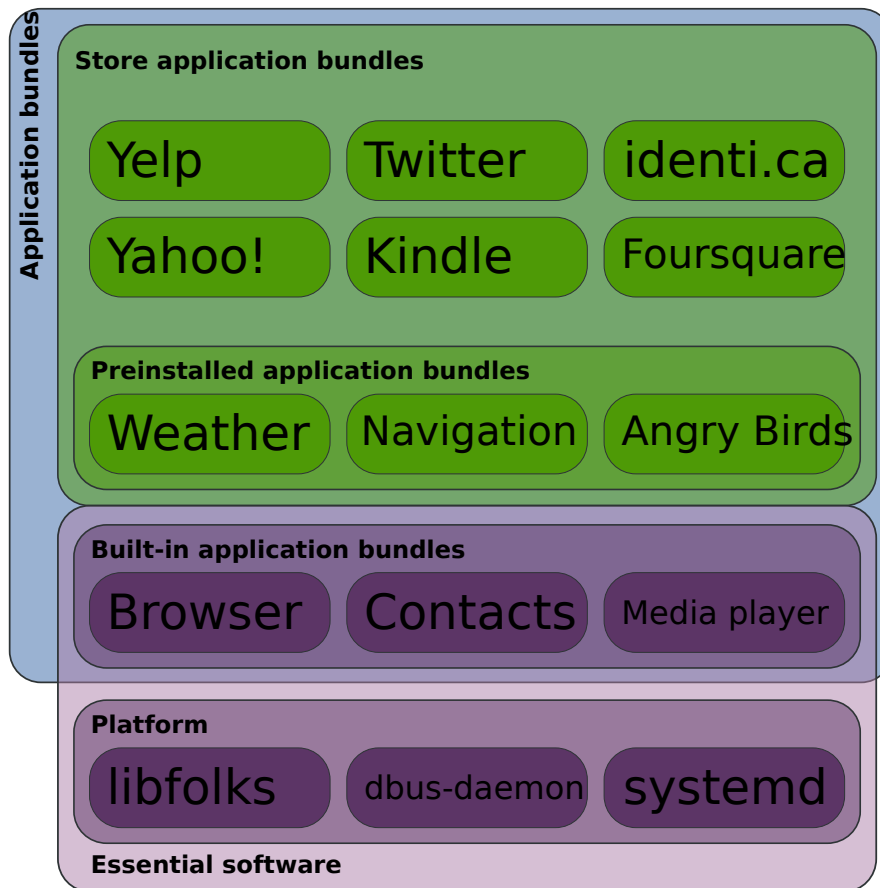
82 A *bundle* or *application bundle* is a group of functionally related components  
83 (be they services, data, or programs), installed as a unit. This matches the sense  
84 with which “app” is typically used on mobile platforms such as Android and iOS;  
85 for example, we would say that an Android .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 dows 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](#)<sup>6</sup> or the name  
217       discussed in [Application bundle metadata](#)<sup>7</sup>
- 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](#)<sup>8</sup>:

---

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

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

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



- 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,  
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)<sup>9</sup>), 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`.

---

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

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.

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 This was implemented in `libcanterbury-platform`.

### 309 **Application Releasing Process**

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

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

### 319 **Application Installation Tracking**

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

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

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

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

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

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

## 348 **Digital Rights Management**

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

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

354 The installation could be locked in the following ways:

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

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

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

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

381 Note that these are all install time checks, and if a device is moved to another  
382 vehicle after successfully installing a bundle, it may result in running an app  
383 somewhere that an application developer or OEM didn't intend it to be run. In

384 order to prevent this from happening, it would be more reliable to do launch-  
385 time testing of the applications.

386 The store would generate a file to be bundled with the application that listed the  
387 launch criteria, and the application manager would check those criteria before  
388 launching the application for use.

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

## 394 Permissions

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

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

403 See the [Permissions concept design](#)<sup>11</sup> for further details.

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

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

- 409 • Some of the permissions will be controlled by an AppArmor profile gen-  
410 erated by the application store. The user is merely accepting the profile,  
411 actually changing it would not be trivial.
- 412 • AppArmor profiles are per-application, not per user. AppArmor profiles  
413 would need to be changed on user switch if different users required different  
414 permission configurations for the same applications.
- 415 • A huge testing burden is placed on the application developer if they can't  
416 rely on the requested permissions. They must test their applications in  
417 all possible configurations.

---

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

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

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

- 418 • The permissions may be required for the application developer’s business  
419 model – be that network permissions for displaying advertising, or GPS  
420 information for crowd sourcing traffic information. Allowing the user to  
421 restrict permissions in these situations would be unfair to the developer.

422 To mitigate some of these problems, there must be two kinds of permissions:  
423 required and optional. Required permissions are those that can’t be removed  
424 from an application – such as anything granted by the AppArmor profile. If a  
425 user chooses to deny a required permission, an application can not run.

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

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

## 442 Data Storage

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

- 446 • Application User – for settings and any other per-user files. In the event  
447 of an application rollback, files in this area are rolled back with their  
448 associated application.
- 449 • Application Everyone – for data that is rolled back with an application  
450 but isn’t tied to a user account – such as voice samples or map data.
- 451 • Cache – for easily recreated data. If the system is low on storage space,  
452 it may reclaim cache space for applications that aren’t currently running.  
453 Caches will be cleared on update and rollback instead of being stored by  
454 the rollback system.
- 455 • Shared – This area’s contents are not touched by the application manage-  
456 ment framework for any reason. They are also not subject to any form of  
457 rollback. This area is intended for storage of videos, music, photos and

458 other data in standard formats that aren't tied to a single application,  
459 analogous to `/sdcard` on Android devices. Since Apertis space may be lim-  
460 ited, and since it is thought that users will usually want to share media  
461 between accounts, the data in this storage area will be accessible by all  
462 users. More details on this are available in the [multiuser](#)<sup>13</sup> design.

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

### 465 **Extending Storage Capabilities**

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

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

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

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

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

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

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

---

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

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

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

495 potentially copy and distribute applications.

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

498 If this is extended to multiple SD cards, difficulties arise in deciding which  
499 storage device to install an application to. Either configuration options will  
500 need to be added to control this, or the device with the greatest free space at  
501 the time of installation can be selected.

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

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

## 513 **Application Management**

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

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

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

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

## 529 **Store Applications**

### 530 **Acquisition**

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



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

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

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

## 546 **Installation**

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

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

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

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

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

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

## 566 **Upgrades**

567 If configured with a suitable Internet connection, the system will periodically  
568 check whether upgrades are available for any store applications that have been  
569 installed. Apertis will provide its vehicle ID to the application store and the

570 application store will reply with a list of the most recent versions of the ap-  
571 plications authorized for the vehicle. If Apertis has had software installed or  
572 removed without an Internet connection, the list of installed applications will  
573 be synchronized with the store at this time.

574 Some users may voice concerns over the store’s tracking of all the installed  
575 packages on their Apertis. It may be worth mentioning in a “privacy policy”  
576 exactly what the data will be used for.

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

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

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

## 591 **Removal**

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

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

## 600 **Roll-back**

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

605 This rollback paradigm has some interesting quirks:

- 606 • If a user rolls back an application, all other users of that application will  
607 also be rolled back. This allows one user to delete some of another user’s  
608 settings and personal data.

- 609 • As some software updates may contain critical security fixes, an ever growing  
610 blacklist will have to be maintained to prevent a user from rolling back  
611 to potentially dangerous versions.
- 612 • Developers will have no control over what software versions their cus-  
613 tomers are using, making long term support very difficult. They may  
614 receive bug reports for bugs already fixed in newer versions of the soft-  
615 ware.
- 616 • Old versions of applications may break if they interact with online services  
617 that changed their protocols, or if Apertis APIs are deprecated.
- 618 • Application developers have to think very carefully about what data goes  
619 into application storage (and is subject to rollbacks) and general storage  
620 (which isn't). In reality, application developers will likely pay very lit-  
621 tle attention to this distinction and the application store will carry this  
622 burden.
- 623 • The effect of a system rollback on installed applications is unclear. If an  
624 application has been upgraded twice since the last system update and a full  
625 system rollback occurs it is possible for applications to have no launchable  
626 version installed.
- 627 • In some cases an application rollback may not even be possible if the old  
628 version of the application is not capable of running on the current version  
629 of the system.
- 630 • Settings management tools like GSettings directly manipulate application  
631 setting data and don't currently support the rollback system.

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

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

### 639 **System Extensions**

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

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

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

651 The application bundle metadata will have a list of known extension "types",  
652 and extension components inside an application bundle will be handled differ-  
653 ently based on the specified type. There is no comprehensive list of extension  
654 types, but "Telepathy connection manager" and "theme" will be the commonly  
655 used examples in this document.

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

## 661 **Installation**

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

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

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

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

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

## 680 **Upgrades**

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

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

## 688 **Removal**

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

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

## 693 **Rollback**

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

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

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

## 701 **License Agreements**

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

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

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

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

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

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

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

727 Some licenses, such as the GPL, inform the user of their rights to obtain a copy  
728 of the source code of the software. Licenses like this should be made available  
729 to the user, but don’t necessarily need to be displayed to the user unless the  
730 user explicitly requests the information.

## 731 **Application Run Time Life-Cycle**

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

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

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

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

## 752 **Start**

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

- 755 • Direct launch – application bundles may contain an image or widget to be  
756 displayed in the application launcher, which will launch a suitable graphi-  
757 cal program. The name and icon shown in the application launcher is part  
758 of the [entry point metadata](#)<sup>16</sup>.

---

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

- 759 • By data type association - The content-type (MIME type) of data is used  
760 to select the appropriate application to handle the request. Applications  
761 will provide a list of content-types (if any) that they handle in the [en-  
762 try point metadata](#)<sup>17</sup>; activating the application with the corresponding  
763 content type will launch the corresponding graphical program.
- 764 • Sharing back-ends – Applications may define sharing capabilities that al-  
765 low other applications to launch them and send a receiver-limited amount  
766 of data. Again, activating an application in this way would normally start  
767 a corresponding graphical program.
- 768 • Agents – persistent non GUI processes that provide a background com-  
769 ponent for applications. These will be launched automatically at boot  
770 time or immediately after application installation. An application bundle  
771 will contain at most a single agent, and the [permissions] will include an  
772 “agent” permission to allow users to know they’re installing an application  
773 that uses one.

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

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

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

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

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

## 791 **Background Operation**

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

---

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

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

802 Graphical programs will be notified with a D-Bus message when they lose focus  
803 and are relegated to background status – the response to this notification is ap-  
804 plication dependent. If it has no need to perform processing in the background,  
805 It may save its current state and self-terminate, or it may remain idle until re-  
806 focused. Some graphical programs will continue to operate in the background  
807 – for example, a navigation application might remain active in the background  
808 and continue to give turn-by-turn instructions.

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

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

## 816 **End**

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

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

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

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

835 There is no standard Linux API for saving application state. POSIX defines  
836 `SIGSTOP` and `SIGCONT` signals for pausing and resuming programs, but these signals  
837 don’t remove applications from memory or provide any sort of persistence over a



838 system restart. Since they're unblockable by applications, the application may  
839 be interrupted at any time with no opportunity to do any sort of clean-up.

840 However, some applications may react to changes in system state – such as  
841 network connectivity. One method of preventing applications from reacting to  
842 D-Bus messages, system state changes, and other signaling is to use `SIGSTOP`  
843 to halt an application's processing. The application becomes responsible for  
844 handling whatever arises after `SIGCONT` causes it to resume processing (such as a  
845 flood of events or network timeouts).

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

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

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

## 859 **State Saving Convenience APIs**

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

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

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

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

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

873 Maemo provided, through `libosso`<sup>18</sup> a very simple state saving API. It expected

---

<sup>18</sup>[http://maemo.org/api\\_refs/5.0/5.0-final/libosso/group\\_\\_Statesave.html](http://maemo.org/api_refs/5.0/5.0-final/libosso/group__Statesave.html)

874 relevant application state be contained within a single contiguous memory re-  
875 gion, and provided a call that would write out this single memory area to some  
876 abstract storage area that persists across reboots. On startup, an application  
877 would attempt to re-read that memory, and if no pre-existing state was present,  
878 would start over.

## 879 **Frozen**

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

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

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

## 895 **Resource Usage**

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

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

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

---

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

<sup>20</sup>[https://gitlab.gnome.org/GNOME/glib/merge\\_requests/1005](https://gitlab.gnome.org/GNOME/glib/merge_requests/1005)

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

910 onds).

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

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

### 917 **Applications not Written for *Apertis***

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

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

### 926 **Responsibilities of the Application Manager**

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

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

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

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

944 Giving the application manager control over content-type-based launches can  
945 allow it to restrict the usage of content-type-based launching. The manager

---

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

946 would be able to deny certain applications the ability to launch handlers for  
947 specific data types (perhaps the web browser should never be allowed to launch  
948 a handler for a certain data type), filter the handlers available to an application  
949 to only allow trusted built-in applications to be used, or allow a system upgrade  
950 to blacklist a store application’s handler while waiting for a fix from a third  
951 party.

952 The application launcher is itself a built-in application, and as such its storage  
953 is governed by system rollbacks. In the event of a system rollback, all of the  
954 launcher’s settings (icon placement, for example) will be automatically reset to  
955 the state they were in just prior to the last system upgrade.

## 956 References

957 This document references the following external sources of information:

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

---

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

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

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

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

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