



Applications

Contents

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

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

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

To avoid ambiguity, this document will avoid using “application” as a jargon term. Instead, we use two distinct terms for separate concepts that could informally be referred to as applications: *graphical programs*, and *application bundles*. See [Terminology](#).

42 Apertis is a multiuser system; see the [multiuser](https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/multiuser/)¹ design document for more on the
43 specifics of the multiuser experience and the division of responsibilities between
44 middleware and HMI elements.

45 Traditional package managers are unfit for applications

46 Apertis relies heavily on a traditional packaging system to compose the base OS.
47 However, it does not rely on it to distribute the composed system as it is not
48 a good fit for the use-cases Apertis addresses, see [system updates and rollback](https://apertis-website-0b3586.pages.apertis.org/concepts/platform/system-updates-and-rollback/)²
49 for more details. Similarly, a traditional packaging system is not a good fit for
50 applications in Apertis since:

- 51 • Apertis relies on an immutable base OS to implement a robust update
52 mechanism, see [system updates and rollback](https://apertis-website-0b3586.pages.apertis.org/concepts/platform/system-updates-and-rollback/)³ for more details. This means
53 that a traditional package manager is not used to distribute updates on
54 the field and that the writable application storage should be kept separate
55 from the read-only base OS.
- 56 • Application bundles don't depend on each other –this simplifies depen-
57 dency management in modern package management systems specializing
58 in support for applications.
- 59 • Much of the complexity in application bundle handling (DRM, rollbacks,
60 communicating security “permissions” to the user) is not part of traditional
61 package management tools, and is not interesting to the upstream tool
62 maintainers.
- 63 • Applications can have conflicting dependencies which can't be shipped as
64 part of the base OS and should be somehow bundled with the application
65 itself.

66 Thus, Apertis has chosen [Flatpak](https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_framework/)⁴ as the system for installing and managing
67 application bundles (see [application framework](https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_framework/)⁵).

68 Terminology

69 Graphical program

70 A *graphical program* is a program with its own UI drawing surface, managed
71 by the system's window manager. This matches the sense with which “appli-

¹https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/multiuser/

²<https://apertis-website-0b3586.pages.apertis.org/concepts/platform/system-updates-and-rollback/>

³<https://apertis-website-0b3586.pages.apertis.org/concepts/platform/system-updates-and-rollback/>

⁴<https://flatpak.org/>

⁵https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_framework/

72 cation”is traditionally used on desktop/laptop operating systems, for instance
73 referring to Notepad or to Microsoft Word.

74 **Bundle**

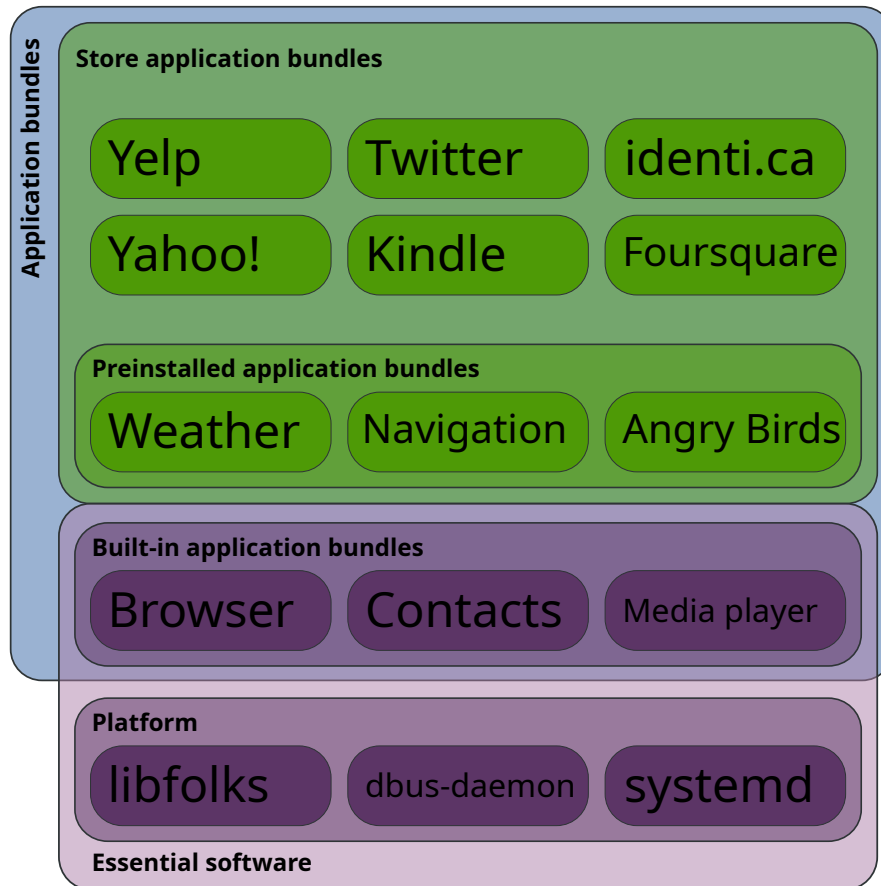
75 A *bundle* or *application bundle* is a group of functionally related components
76 (be they services, data, or programs), installed as a unit. This matches the sense
77 with which “app”is typically used on mobile platforms such as Android and iOS;
78 for example, we would say that an Android .apk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

82 **Store account**

83 The **Digital rights management** section discusses *store accounts*, anticipated
84 to have a role analogous to Google Play accounts on Android or Apple Store
85 accounts on iOS. If these accounts exist, we recommend against using the term
86 “user”for them, since that would be easily confused with the users found in the
87 Multiuser design document; it is not necessarily true that every user has access
88 to a store account, or that every store account corresponds to only one user.

89 **Software Categories**

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



93

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

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

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

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

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

117 **Pre-installed Applications**

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

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

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

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

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

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

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

143 **Case Study: a navigation application, how would it work?** The navi-
144 gation application was singled out as a case that has requirements and features
145 that intersect those of built-in applications and those of store applications. On
146 the one hand, the navigation application is core functionality, which means it
147 should be part of the system. On the other hand, it should be possible to make

148 the application extensible or upgradable, enabling the selling of updated maps,
149 for instance.

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

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

163 **Responsibilities of the Application Store**

164 The application store will be responsible for hosting a developer's signed appli-
165 cation bundles. Special "SDK" system images will provide software development
166 tools and allow the installation of unsigned packages, but the normal "target"
167 system image will not allow the installation of packages that don't contain a
168 valid store signature.

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

174 However, pre-publication screening of applications will represent a significant
175 time commitment, as even minor changes to applications must undergo thor-
176 ough testing. High priority security fixes from developers may need to be given
177 a higher priority for review and publication, and the priority of application up-
178 dates may need to be considered individually. System updates will correspond
179 to the busiest periods for both internal and external developers, and the appli-
180 cation store will experience significant pressure at these times.

181 **Identifying applications**

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

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

- 189 • the name of the application bundle, being the [Flatpak app-id](#)⁶ or the name
190 discussed in [Application bundle metadata](#)⁷
- 191 • the D-Bus well-known name or names taken by the program(s) in the
192 bundle, for instance via GLib's GApplication interface
- 193 • the name(s) of the freedesktop.org .desktop file(s) associated with the
194 program(s), if they have them
- 195 • the name of the systemd user service (.service file) associated with the
196 program(s), if they have them

197 [Flatpak aligns these according to the following system](#)⁸:

- 198 • The *bundle ID* is a case-sensitive string matching the syntactic rules for
199 a D-Bus interface name, i.e. two or more components separated by dots,
200 with each component being a traditional C identifier (one or more ASCII
201 letters, digits, or underscores, starting with a non-digit).

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

- 207 • Application authors should be strongly encouraged to use a DNS name
208 that they control, with its components reversed (and adjusted to follow
209 the syntactic rules if necessary), as the initial components of the bundle
210 ID. For instance, the owners of collabora.com and 7-zip.org might choose
211 to publish `com.collabora.MyUtility` and `org._7_zip.Decompressor`, respec-
212 tively. This convention originated in the Java world and is also used for
213 Android application packages, Tizen applications, D-Bus names, GNOME
214 applications and so on.
- 215 • App-store curators should not allow the publication of a bundle whose
216 name is a prefix of a bundle by a different developer, or a bundle that is
217 in the essential software set. App-store curators do not necessarily need
218 to verify domain name ownership in advance, but if a dispute arises, the
219 app-store curator should resolve it in favour of the owner of the relevant
220 domain name.
- 221 • Well-known namespaces used by platform components (such as `aper-`
222 `tis.org`, `freedesktop.org`, `gnome.org`, `gtk.org`) should be restricted to app
223 bundles associated with the relevant projects. Example projects provided

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

⁷https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_framework/application-bundle-metadata/

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

224 in SDK documentation should use the names that are reserved for
225 examples (see [RFC2606](http://www.rfc-editor.org/info/rfc2606)⁹), such as example.com, but app-store curators
226 should not publish bundles that use such names.

- 227 • Programs in a bundle may use the D-Bus well-known name correspond-
228 ing to the bundle ID, or any D-Bus well-known name for which the
229 bundle ID is a prefix. For instance, the `org.apertis.MyUtility` bundle
230 could include programs that take the bus names `org.apertis.MyUtility`,
231 `org.apertis.MyUtility.UI` and/or `org.apertis.MyUtility.Agent`.
- 232 • If a program has a freedesktop.org .desktop file, its name should be the
233 program's D-Bus well-known name followed by `.desktop`, for example
234 `org.apertis.MyUtility.UI.desktop`.

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

237 Application Releasing Process

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

- 241 • Verify that the application's bundle ID does not collide with any bundle
242 by a different publisher (in the sense that neither is a prefix of the other).
- 243 • Make the application available at the store.

244 Application Installation Tracking

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

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

252 The application store can only know which applications have been downloaded
253 for use in a specific vehicle –with no guarantee of a persistent Internet connection,
254 the store has no way to know whether the application has really been installed
255 or subsequently uninstalled. The store also can't reliably track what version of
256 an application is installed.

257 If an application is downloaded on a computer with a web browser (presumably
258 for installation via external media), the store shouldn't assume it was actually
259 installed anywhere. Only applications installed directly to the device should be
260 logged as installed. When the user logs in to the store (or the device logs into

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

261 the store with the users credentials to check for updates), the list of installed
262 packages can be synchronized.

263 If an application is installed from a USB storage device the application manager
264 could write a synchronization file back to the device that could subsequently be
265 uploaded back to the application store from a web browser. Care should be
266 taken to ensure these files can't be used by malicious users to steal applications
267 -the store should check that the applications listed in the synchronization file
268 have been legitimately purchased by the user and the file's contents should be
269 discarded if they have not.

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

273 **Digital Rights Management**

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

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

279 The installation could be locked in the following ways:

- 280 • Locked to a specific device ID -it will only install on a specific Apertis
281 unit.
- 282 • Locked to a specific ID of a larger system containing the device running
283 Apertis. For instance, in the automotive use case, this could be locked to
284 a specific vehicle ID. The Apertis unit will refuse to install the application
285 if the vehicle ID does not match the ID embedded in the downloaded
286 application package.
- 287 • Locked to a customer ID -It will only install for a specific person, as
288 represented by their store account - presumably a store account must be
289 present and logged in for this to work. The store account is assumed
290 to be analogous to an Apple Store or Google Play account: as noted in
291 **Terminology**, we recommend avoiding the term "user" here, since a store
292 account does not necessarily correspond 1:1 to the "users" discussed in the
293 Multiuser design document.

294 Any "and" combination of these 3 locks could also be used. For example, an
295 application bundle may only be installable to a specific device in a specific
296 vehicle (in other words, locked to vehicle ID and device ID) -if the Apertis
297 unit is placed in another vehicle, or the vehicle's Apertis unit is replaced, the
298 application bundle would not be installable.

299 Conversely, rights could also be combined with the “or” operator, such as allow-
300 ing an application bundle to be installed if either the correct Apertis unit is
301 used, or the correct vehicle. Collabora recommends these combinations not be
302 implemented. Most of the combinations provided by “or” aren’t obviously useful.

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

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

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

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

320 Permissions

321 Applications can perform many functions on a variety of user data. They may
322 access interfaces that read data (such as contacts, network state, or the users
323 location), write data, or perform actions that can cost the user money (like send-
324 ing SMS). As an example, the Android operating system has a comprehensive
325 [manifest](http://developer.android.com/reference/android/Manifest.permission.html)¹⁰ that govern access to a wide array of functionality.

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

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

330 Ideally, users would be able to accept a subset of permission, but there are some
331 difficulties in allowing this:

- 332 • A huge testing burden is placed on the application developer if they can’t
333 t rely on the requested permissions. They must test their applications in
334 all possible configurations.

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

¹¹https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/permissions/

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

To mitigate some of these problems, an application author may assume that the permissions listed in the [Flatpak manifests](#)¹² will be available. More sensitive permissions are available via runtime requests to the external [XDG portals](#)¹³ running on the host system; the user may reject access to these permissions at their discretion.

Some permissions may prove to be more of an annoyance than helpful to the user. It may be worth considering having some permission acceptance governed by system settings, and only directly query the user if a permission is “important” (such as sending SMS).

Data Storage

Applications will have access to several types of writable application storage, as per [the official Flatpak guidance on XDG base directories](#)¹⁴. In addition, an application can request permission to access general storage locations such as the user’s home or documents folder.

Extending Storage Capabilities

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

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

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

If it is critical that applications (or other trusted data, such as navigation maps) be run off of removable storage, allowing the system to “format”the device before use, deleting all data already on the card and replacing it with an encrypted

¹²<https://docs.flatpak.org/en/latest/manifests.html>

¹³<https://flatpak.github.io/xdg-desktop-portal/portal-docs.html>

¹⁴<https://docs.flatpak.org/en/latest/conventions.html#xdg-base-directories>

370 BTRFS filesystem would allow a secure method of placing application storage
371 on the device.

372 The dm-crypt [LUKS¹⁵](#) system would be used to encrypt the storage device
373 using a random binary key file. These key files would be generated at the time
374 the external storage device is formatted and stored along with the device serial
375 number. One way to generate a key file would be to read an appropriate number
376 of bytes (such as 32) from `/dev/random`.

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

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

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

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

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

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

401 **Application Management**

402 Applications will be distributed by the application store as “application bundles”
403 containing programs and services that can be launched in a variety of ways.

404 All communication with the application store will take place over a
405 secure HTTPS connection.

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

406 The metadata in this bundle provides information about the application such
407 as it's user friendly name, services it needs from the system (such as querying
408 the GPS) and the permissions it needs from the user.

409 **Store Applications**

410 **Acquisition** Applications will be made available through the application
411 store's corresponding [Flatpak repository](#)¹⁶.

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

417 **Installation** If an application is being installed directly from the store, an
418 icon will be displayed in the launcher while the download and installation takes
419 place will now be acquired from the application store.

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

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

- 426 • Eventually storage caches are filled and begin writing out causing a dra-
427 matic slowdown in apparent installation speed for larger applications.
- 428 • Decompression speed may vary for different OSTree objects that are part
429 of the same application.
- 430 • Some of the required files to be downloaded may already be available
431 locally, but this will not be known until the file's digest is determined.

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

435 **Upgrades** If configured with a suitable Internet connection, the system will
436 periodically check whether upgrades are available for any store applications that
437 have been installed. Apertis will provide its vehicle ID to the application store
438 and the application store will reply with a list of the most recent versions of the
439 applications authorized for the vehicle. If Apertis has had software installed or
440 removed without an Internet connection, the list of installed applications will
441 be synchronized with the store at this time.

¹⁶<https://docs.flatpak.org/en/latest/repositories.html>

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

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

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

453 **Removal** When a user removes the application, any personal settings and
454 caches required by the application will be automatically removed –files the ap-
455 plication has stored in general storage will be left behind.

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

460 **Roll-back** Apertis may utilize [Flatpak functionality](https://docs.flatpak.org/en/latest/tips-and-tricks.html#downgrading)¹⁷ to implement a per-
461 application rollback system that allows an end user to revert to the last installed
462 version of an application (that is, a single previously installed version will be
463 kept when an upgrade is performed).

464 This rollback paradigm has some interesting quirks:

- 465 • If a user rolls back an application installed system-wide, all other users of
466 that application will also be rolled back.
- 467 • As some software updates may contain critical security fixes, an ever grow-
468 ing blacklist will have to be maintained to prevent a user from rolling back
469 to potentially dangerous versions.
- 470 • A rollback will not modify the application’s data. Thus, if the data has
471 been modified by the newer version, it would then be up to the application
472 to determine how to handle the changes when loading the data in the
473 previous version.
- 474 • Developers will have no control over what software versions their cus-
475 tomers are using, making long term support very difficult. They may
476 receive bug reports for bugs already fixed in newer versions of the soft-
477 ware.

¹⁷<https://docs.flatpak.org/en/latest/tips-and-tricks.html#downgrading>

- 478 • Old versions of applications may break if they interact with online services
479 that changed their protocols, or if Apertis APIs are deprecated.
 - 480 • The effect of a system rollback on installed applications is unclear. If an
481 application has been upgraded twice since the last system update and a full
482 system rollback occurs it is possible for applications to have no launchable
483 version installed.
 - 484 • In some cases an application rollback may not even be possible if the old
485 version of the application is not capable of running on the current version
486 of the system.
- 487 After application rollback, launching the application now will use the previously
488 installed version, with all user data left untouched, in identical state as before
489 the rollback.

490 License Agreements

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

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

498 Such licenses generally disclose information regarding the use of data collected
499 by an application or related services, define acceptable usage of the application
500 or services by a user, or discuss the warranty and culpability of the application
501 provider.

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

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

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

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

516 Some licenses, such as the GPL, inform the user of their rights to obtain a copy
517 of the source code of the software. Licenses like this should be made available
518 to the user, but don't necessarily need to be displayed to the user unless the
519 user explicitly requests the information.

520 **Application Run Time Life-Cycle**

521 The middleware will assist UI components in launching and managing applica-
522 tions on Apertis. Application bundles can provide executable files (programs)
523 to be launched via different mechanisms, some of them user visible (perhaps as
524 icons on the desktop or home screen that will launch a graphical program), and
525 some of them implicit (like a connection manager for the Telepathy framework,
526 or a graphical program that does not appear in menus but is launched in order
527 to handle a particular request).

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

534 The approach will be similar to that traditional desktop Linux; an application
535 is fully closed upon restart or when requested, and resuming state is entirely
536 within its own responsibility.

537 **Start**

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

- 540 • Direct launch - application bundles may contain an image to be displayed
541 in the application launcher, which will launch a suitable graphical program.
542 The name and icon shown in the application launcher is part of the [entry
543 point metadata](#)¹⁸.
- 544 • By data type association - The content-type (MIME type) of data is used
545 to select the appropriate application to handle the request. Applications
546 will provide a list of content-types (if any) that they handle in the [en-
547 try point metadata](#)¹⁹; activating the application with the corresponding
548 content type will launch the corresponding graphical program.
- 549 • An application can request the ability to launch persistent non GUI pro-
550 cesses that provide a background component for applications. These can
551 also be launched automatically at boot time. The ability for an application

¹⁸https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_framework/application-entry-points/

¹⁹https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_framework/application-entry-points/

552 to request this is conditional upon the user's choice, and the application
553 is responsible the situation that would arise if background or auto-start
554 support were rejected.

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

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

561 During pre-publication review by the app store, careful attention should be paid
562 to application bundles that wish to use background services, and the resource
563 consumption of the services. The concept does not scale –it creates a system
564 where the number of installed application bundles can dramatically affect run-
565 time performance as well as system boot-up time.

566 Background Operation

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

573 Background tasks may also be performed by long running background processes.
574 These run for the duration of the user's session, and are never explicitly termi-
575 nated unless the user revokes the background permissions or the system is low
576 on resources.

577 Graphical programs will be notified by the compositor when they lose focus and
578 are relegated to background status –the response to this notification is appli-
579 cation dependent. If it has no need to perform processing in the background,
580 It may save its current state and self-terminate, or it may remain idle until re-
581 focused. Some graphical programs will continue to operate in the background
582 –for example, a navigation application might remain active in the background
583 and continue to give turn-by-turn instructions.

584 Graphical programs that need to perform tasks in the background will have to
585 request for [permissions](https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/permissions/)²⁰. Ideally they should be designed with a split between
586 foreground and background components (perhaps using a graphical program for
587 the user interface and a background service for such work) instead.

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

²⁰https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/permissions/

590 **End**

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

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

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

603 It should be noted that state saving is difficult to implement, and much of the
604 work is the responsibility of the application writer. While Apertis can provide
605 functions for handling the incoming signal and storing state data, the hardest
606 part is determining exactly what application state needs to be saved in order
607 for the application to exit and restart in exactly the same way it had been
608 previously running.

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

614 However, some applications may react to changes in system state –such as net-
615 work connectivity. One method of preventing applications from reacting to
616 D-Bus messages, system state changes, and other signaling is to use `SIGSTOP` to
617 halt an application's processing. The application becomes responsible for han-
618 dling whatever arises after `SIGCONT` causes it to resume processing (such as a
619 flood of events or network timeouts).

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

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

629 It's possible that a viewer application may exit and the file it was viewing be
630 deleted before the application's next start, making it impossible to completely

631 restore the previous application state. Applications will be responsible for han-
632 dling such situations gracefully.

633 Resource Usage

634 To make better use of the available memory, it's recommended that applications
635 listen to the cgroup notification [memory.usage_in_bytes](#)²¹ and when it gets
636 close to the limit for applications, start reducing the size of any caches they
637 hold in main memory. It may be good to do this inside the SDK and provide
638 applications with a [GLib object](#)²² that will notify them.

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

644 The recommended mechanism for monitoring available disk space is for a dae-
645 mon running in the user session to call `statvfs (2)` periodically on each mount
646 point and notify applications with a D-Bus signal. Example code can be found
647 [in the GNOME project](#)²³ which uses a similar approach (polling every 60 sec-
648 onds).

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

652 Applications not Written for *Apertis*

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

657 References

658 This document references the following external sources of information:

- 659 • [XDG Base Directory Specification](#)²⁴
- 660 • [Apertis System Updates and Rollback](#)²⁵ design

²¹<https://www.kernel.org/doc/Documentation/cgroup-v1/memory.txt>

²²https://gitlab.gnome.org/GNOME/glib/merge_requests/1005

²³<http://git.gnome.org/browse/gnome-settings-daemon/tree/plugins/housekeeping/gsd-disk-space.c#n693>

²⁴<https://specifications.freedesktop.org/basedir-spec/latest/>

²⁵<https://apertis-website-0b3586.pages.apertis.org/concepts/platform/system-updates-and-rollback/>

- 661 • Apertis [Multiuser](#)²⁶ design
- 662 • Apertis [Supported API](#)²⁷ design
- 663 • Apertis [Preferences and Persistence](#)²⁸ design
- 664 • Eastlake 3rd, D. and A. Panitz, “Reserved Top Level DNS Names”, BCP
665 32, RFC 2606, DOI 10.17487/RFC2606, June 1999 ([http://www.rfc-](http://www.rfc-editor.org/info/rfc2606)
666 [editor.org/info/rfc2606](http://www.rfc-editor.org/info/rfc2606))

²⁶https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/multiuser/

²⁷https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_customization/supported-api/

²⁸https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_customization/preferences-and-persistence/