



Canterbury legacy application framework

1	<b>Contents</b>	
2	Flatpak . . . . .	2
3	Canterbury . . . . .	3
4	<b>Comparison</b>	<b>3</b>
5	Applications concept . . . . .	4
6	Application layout . . . . .	4
7	Application entry points . . . . .	4
8	Application metadata . . . . .	4
9	Bundle spec . . . . .	4
10	Permissions . . . . .	5
11	Preferences and persistence . . . . .	5
12	Containerisation . . . . .	5
13	Large data sharing . . . . .	5
14	Dialogs and notifications . . . . .	5
15	Launch applications and services . . . . .	6
16	Launch pre-configured default apps at start-up (Launcher / Global popup / Status Bar) . . . . .	6
17	AppArmor . . . . .	6
18	Headless agents . . . . .	7
19	System agents . . . . .	7
20	Multiple entry points . . . . .	7
21	Application manager D-Bus interface . . . . .	8
22	Audio management . . . . .	8
23	Hard Keys . . . . .	9
24	Preference application launching . . . . .	9
25	Out-of-memory handling . . . . .	9
26	Bandwidth prioritization . . . . .	9
27	App store . . . . .	9
28	Manage launched application windows using the Window Manager . .	10
29	Notifies application whether they are in background or foreground . .	10
30	Maintain an application stack . . . . .	10
31	Store Last User Mode (LUM) information periodically and restore LUM on start-up . . . . .	10
32		
33		
34	<b>Conclusions</b>	<b>10</b>
35	Apertis currently ships with a custom application framework based on the Can- terbury app manager which is in the process of being phased out in favor of upstream components like Flatpak, see the <a href="#">application framework</a> <sup>1</sup> document for more details.	
36		
37		
38		
39	Flatpak and Canterbury cover the core tasks of an application framework:	
40	• packaging	
41	• distribution	

---

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

- 42 • sandboxing

43 When Canterbury was designed Flatpak didn't exist and the available technolo-  
44 gies were quite different from what is in today's usage, so it's now time to  
45 reconsider our approach.

## 46 Flatpak

- 47 • upstream, large community
- 48 • mature, proven on the field
- 49 • uses Linux containers to isolate the filesystem view from the application
- 50 • sandbox based on Linux containers and seccomp
- 51 • uses AppStream and .desktop files to encode metadata about the applica-  
52 tion
- 53 • backed by OSTree
- 54 • shared runtimes decouple libraries on the host from libraries depended by  
55 applications, changes on the host won't break applications
- 56 • deduplicates files across applications, runtimes and the host OSTree-based  
57 system
- 58 • SDK runtimes decouple development from the host
- 59 • growing IDE support (GNOME Builder, Eclipse)
- 60 • standardized D-Bus based portals for privileged operations
- 61 • transparent support for portals already available in the most widespread  
62 toolkits (Qt/GTK/etc.)
- 63 • large userbase
- 64 • available out-of-the-box on the most widespread distributions (De-  
65 bian/Ubuntu/Fedora/Red Hat/Suse/etc.)
- 66 • well documented
- 67 • additional permissions are managed through high level entries in the ap-  
68 plication manifest
- 69 • sandboxed with seccomp
- 70 • mature OTA mechanism for applications
- 71 • user-facing app store available upstream
- 72 • the upstream app-store, FlatHub, can be deployed for Apertis, or the  
73 experimental Magento app-store could be adapted
- 74 • enables third-party applications (Sublime Text, Visual Studio Code, etc.)  
75 to be run on the SDK with no effort

## 76 Canterbury

- 77 • Apertis specific, no community
- 78 • not proven on the field
- 79 • pre-dates Linux containers availability, does not use them
- 80 • sandbox based on AppArmor
- 81 • uses AppStream and .desktop files to encode metadata about the applica-  
82 tion

- 83 • backed by OSTree
- 84 • applications use libraries from the host, no decoupling
- 85 • no concept of runtimes
- 86 • no deduplications
- 87 • limited IDE support (Eclipse)
- 88 • very sparsely documented
- 89 • security constraints expressed via low-level AppArmor profiles, no higher-
- 90 level permission system
- 91 • no seccomp sandbox
- 92 • OTA mechanism for applications and agents at the prototype stage (Bosch-
- 93 only, not available in Apertis)
- 94 • user-facing app store at the prototype stage (Bosch-only, not available in
- 95 Apertis)
- 96 • there's an experimental Magento-based app-store, not currently available
- 97 in Apertis

## 98 Comparison

99 Since Apertis is meant to adopt upstream solutions whenever possible it is nat-  
100 ural for us to adopt Flatpak, but to do so the gaps that need to be filled must  
101 be evaluated.

102 The two systems are very different and for this reason no transparent compatibil-  
103 ity can be provided, but thanks to the modular approach in Apertis Canterbury  
104 can be kept available in the repositories even if the reference setup will use Flat-  
105 pak.

106 Since the two systems share many underlying technologies (D-Bus, OSTree,  
107 etc.) their performance are comparable. The additional use of control groups  
108 in Flatpak doesn't add any noticeable overhead. Flatpak consists of just an  
109 executable setting up the environment and does not require an always-running  
110 daemon as Canterbury does, so there may be a negligible memory saving.

## 111 Applications concept

112 The legacy Apertis application framework already defined the concept of appli-  
113 cation bundles. The new application framework defines the wanted format used  
114 for the bundle as being Flatpak.

## 115 Application layout

116 The application layout remains compatible with the legacy application frame-  
117 work, note that the layout is relative to the `/app/` folder inside of the Flatpak.

## 118 **Application entry points**

119 As the [entry points](#)<sup>2</sup> were defined using the standard specification from  
120 FreeDesktop.org, they remain compatible with the new Apertis application  
121 framework and are exposed by the flatpak executable to the system when  
122 necessary.

123 Desktop file should be updated to use Flatpak instead of Canterbury to launch  
124 the application, e.g. replacing

```
125 Exec=@bindir@/eye app-name @app_id@ play-mode stop url NULL
```

126 by

```
127 Exec=flatpak run app-name @app_id@ play-mode stop url NULL
```

## 128 **Application metadata**

129 The application metadata were specified using the AppStream FreeDesktop.org  
130 specification and remains the main metadata specification for Flatpak.

## 131 **Bundle spec**

132 The latest Canterbury application bundle specification has been largely based  
133 on the Flatpak one, in a initial effort to align Canterbury with recent upstream  
134 technologies:

- 135 • the binary format is the exactly same;
- 136 • in both cases AppStream is used for the bundle metadata;
- 137 • entrypoints are defined with `.desktop` files both in Canterbury and Flat-  
138 pak;
- 139 • installation paths differ since Canterbury requires an unique installation  
140 path while Flatpak relies on containers to put different contents on the  
141 same path for each application, but from a practical point of view the  
142 difference is purely cosmetic.

## 143 **Permissions**

144 No high level support for application permission has been implemented in Can-  
145 terbury, application access to resources was exclusively based on writing dedi-  
146 cated [AppArmor profiles](#)<sup>3</sup> for each applications and carefully reviewing them.

147 Flatpak instead lets application authors specify in the application manifest a set  
148 of special high-level permissions. The Flatpak approach has been analysed in  
149 more detail in the original [permissions](#)<sup>4</sup> document which already described the

---

<sup>2</sup><https://martyn.pages.apertis.org/apertis-website/architecture/bundle-spec/#entry-points>

<sup>3</sup><https://martyn.pages.apertis.org/apertis-website/architecture/bundle-spec/#apparmor-profile>

<sup>4</sup><https://martyn.pages.apertis.org/apertis-website/concepts/permissions/>

150 use-cases for the permissions mechanism in the context of the Apertis application  
151 framework.

## 152 **Preferences and persistence**

153 The Apertis application framework satisfies the requirements of the legacy ap-  
154 plication framework. The only missing part is that application rollback is not  
155 able to revert the user-data to a previous state.

## 156 **Containerisation**

157 Canterbury pre-dates the maturity of containerization in Linux (cgroups and  
158 namespaces) and it does not make use of it.

159 Flatpak is instead heavily based on containers, providing much stronger isolation  
160 capabilities.

## 161 **Large data sharing**

162 The Apertis application framework allows to share data using the standard  
163 mechanisms as described by the FreeDesktop.org Desktop File specification.  
164 Any D-Bus enabled sharing service can be used when specifying the right in-  
165 terface in the Flatpak manifest. It is no more possible to register a service by  
166 putting a file into `/var/lib/apertis_extensions/applications` at installation time  
167 as the files are installed into a different path for each bundle.

## 168 **Dialogs and notifications**

169 The Apertis application framework is also using the [Notification Specification](#)<sup>5</sup>  
170 and allows to reuse the same interface without any breakage.

171 The dialog abstraction for the legacy application framework has never been  
172 implemented as its design is subject to many questions.

## 173 **Launch applications and services**

174 As Flatpak is well-integrated into existing environments and uses the same tech-  
175 nology and protocols for its foundations, there is no expected problems with  
176 Flatpak here.

## 177 **Launch pre-configured default apps at start-up (Launcher 178 / Global popup / Status Bar)**

179 The work has already been started as show by this [upstream request](#)<sup>6</sup> for this  
180 feature making it a small gap to fill.

---

<sup>5</sup><https://people.gnome.org/~mccann/docs/notification-spec/notification-spec-latest.html>

<sup>6</sup><https://github.com/flatpak/flatpak/issues/118>

## 181 AppArmor

182 Currently Apertis depends heavily on AppArmor to constrain services and ap-  
183 plications: it is used to restrict filesystem access and mutually authenticate  
184 applications in a secure way when communicating over D-Bus.

185 AppArmor is currently used in Apertis for two different purposes:

- 186 • access constraints
- 187 • secure identification of D-Bus peers

188 While Flatpak has no support for AppArmor out of the box and adding it is  
189 not on the roadmap so far, the first use case is already covered by the use of  
190 Linux cgroups and namespaces which provide more flexibility than AppArmor.  
191 Flatpak also ships a D-Bus proxy to manage access policies at the D-Bus level,  
192 since that needs a finer control than cgroups and namespaces can provide.

193 The higher-level access constraints implemented by Flatpak are much easier and  
194 secure to be used by application authors than the low-level AppArmor policy  
195 language currently used by Apertis. In that sense, the adoption of Flatpak would  
196 be aligned to the plan to provide an higher-level access constraints mechanism  
197 to application authors and shield them from the AppArmor policy language.

198 Flatpak also includes the concept of “portals” to provide restricted access to  
199 resources to unprivileged applications, either by applying system-specific policies  
200 or by requiring user interaction. For instance, applications don’t have access to  
201 user files, and file opening is handled via a privileged portal that ensure that  
202 applications can only access files users have given their consent to.

203 The second use of AppArmor is something very few applications at the moment  
204 use, and portals seem well suited to replace its known usages:

- 205 • Canterbury itself uses it to control applications: this is managed by Flat-  
206 pak by using cgroups
- 207 • Newport (download manager) uses it to securely identify its clients: creat-  
208 ing a dedicated Flatpak portal would address the use-case with no reliance  
209 on AppArmor
- 210 • Frome (magento app-store client) uses it to only let the `org.apertis.Mildenhall.Setting`  
211 system application talk to it: a dedicated Flatpak portal seem appropriate  
212 here as well
- 213 • Beckfoot (network management service) uses it to talk with `org.apertis.Mildenhall.StatusBar`,  
214 but Beckfoot itself has been declared obsolete long ago in {T3626} and  
215 the existing [org.freedesktop.portal.Notification](https://flatpak.github.io/xdg-desktop-portal/portal-docs.html#gdbus-org.freedesktop.portal.Notification)<sup>7</sup> could be used instead.

---

<sup>7</sup><https://flatpak.github.io/xdg-desktop-portal/portal-docs.html#gdbus-org.freedesktop.portal.Notification>

## 216 **Headless agents**

217 Flatpak focuses on graphical application on the user session bus: nothing in its  
218 design prevents its usage for headless agents and some testing didn't show any  
219 significant issue, but some rough edges are expected.

220 Some one-time effort may be needed to consolidate this use-case in Flatpak.

## 221 **System agents**

222 Canterbury can only manage user-level applications and agents, and it doesn't  
223 currently have support for agents meant to be accessed on the system bus by  
224 different users.

225 Flatpak is not suited for system agents as well and focuses on the user session.  
226 Upstream explicitly considers system agents a non-usecase and working in this  
227 direction would produce a significant delta that would significantly impact the  
228 maintenance burden.

229 Flatpak apps run in an environment that can never exercise capabilities  
230 (`CAP_SYS_ADMIN`, `CAP_NET_ADMIN` etc.) or transition between uids, so some system  
231 services will not be possible to implement. System services that could run as  
232 an unprivileged system-level uid and don't do anything inherently privileged,  
233 like downloading files and putting them in a centralized location where all  
234 users can access them, should work. System services that need to be root to do  
235 inherently privileged things, like ConnMan/BlueZ, won't.

236 systemd "portable services", perhaps deployed using OSTree, might be a rea-  
237 sonable solution for system agents. They are very new and not yet considered  
238 stable, but are specifically meant for this purpose.

## 239 **Multiple entry points**

240 Canterbury supports multiple entry points in a single app-bundle, and Flatpak  
241 should support more than one desktop file which, as in Canterbury, are the  
242 implementation of entry points.

## 243 **Application manager D-Bus interface**

244 Canterbury exports an obsoleted D-Bus interface with a set of largely unrelated  
245 methods to:

- 246 • let application register themselves
- 247 • communicate to applications their new application state (show, hide,  
248 paused, off)
- 249 • hide global popups
- 250 • get the currently active application
- 251 • get the application that is currently using the audio source
- 252 • find out if the currently active application needs an Internet connection



253 Tracking the application that is currently “active” and hiding popups are tasks  
254 that should be handled by the compositor. The other interfaces are considered  
255 problematic as well.

256 Canterbury-core, the version of Canterbury for headless systems, already doesn’t  
257 ship the application manager interface so there’s no contingent need to reimplement  
258 it.

## 259 **Audio management**

260 The legacy application framework was built around PulseAudio.

261 Canterbury provides a custom audio manager which was already considered obso-  
262 leted and a [different design](#)<sup>8</sup> was proposed some time ago on top of PulseAu-  
263 dio.

264 With the need of more containment into the framework, the Apertis application  
265 framework is meant to use PipeWire as a replacement for PulseAudio. The  
266 intent for PipeWire is to be a drop-in replacement for PulseAudio during the  
267 transition period. PipeWire also provides a sink and source GStreamer element  
268 to replace their PulseAudio counterparts.

269 PipeWire is designed to let an external policy engine dictate how the audio  
270 should be routed and also provide proper security controls to restrict untrusted  
271 applications: for this reason AGL plans to use it as the foundation for their  
272 upcoming audio management solution, and Collabora is involved to ensure the  
273 embedded use-cases are covered.

274 An alternative which is largely in use is the GENIVI AudioManager, which can  
275 be used with Flatpak as well.

276 Canterbury-core, the version of Canterbury for headless systems, already doesn’t  
277 ship the audio manager so there’s no contingent need to reimplement it.

## 278 **Hard Keys**

279 Canterbury provides a D-Bus interface for handling hard-keys by communicating  
280 with the compositor over private interfaces. This is considered obsolete and  
281 hard-key handling should happen in the compositor directly.

282 Canterbury-core, the version of Canterbury for headless systems, already doesn’t  
283 ship the hard key interface so there’s no contingent need to reimplement it.

## 284 **Preference application launching**

285 Canterbury provides a D-Bus interface to let applications launch the preference  
286 manager to edit their preferences rather than providing their own interface.

---

<sup>8</sup><https://martyn.pages.apertis.org/apertis-website/concepts/audio-management/>

287 This also requires support in the preference manager, which is not currently  
288 implemented.

289 Canterbury-core, the version of Canterbury for headless systems, already doesn't  
290 ship the preference launcher interface so there's no contingent need to reimplement  
291 it.

## 292 **Out-of-memory handling**

293 When memory pressure is detected Canterbury tries to kill applications not  
294 currently visible. The private API between Canterbury and the Mildenhall  
295 compositor and the implementation were already known to be problematic and  
296 were considered to be needing a significant rework in any case, possibly to move  
297 them to a dedicated module.

298 The module dedicated to the prioritization of applications in case of memory  
299 pressure can then be implemented to work with Flatpak applications seamlessly.

## 300 **Bandwidth prioritization**

301 Canterbury provides an experimental bandwidth prioritization system that is  
302 known to be problematic and has been considered obsolete, see {T4043} for  
303 details. No similar mechanism is available in Flatpak.

## 304 **App store**

305 There's an experimental Magento-based app-store for Canterbury, but it is not  
306 yet available in Apertis. Flatpak has its own upstream app store, FlatHub,  
307 which is Open Source and can be self-hosted. It doesn't currently implement  
308 payments in any form. Possible options here are either publishing the Magento-  
309 based code and adapting it to work with Flatpak with a limited amount of  
310 changes but higher maintenance costs, or contribute on the implementation of  
311 payment methods on FlatHub, with a higher one-time cost but likely lower  
312 on-going maintenance requirements.

## 313 **Manage launched application windows using the Window 314 Manager**

315 This was deprecated since Apertis 17.09. Canterbury uses private interfaces  
316 with the compositor to:

- 317 • show/hide splashscreens, but WM should be able to display splashscreens  
318 on its own without involving the application manager
- 319 • learn which application is being displayed to manage the “back” stack,  
320 but the WM is better positioned to handle the “back” stack on its own
- 321 • inform the WM that the Last User Mode is being set up, but it appears  
322 that the compositor takes no special action in that case

323 **Notifies application whether they are in background or**  
324 **foreground**

325 This is not part of canterbury-core and has been deprecated since Apertis 17.09.  
326 In a single fullscreen window scenario this can be handled by tracking whether  
327 the application has the focus or not. In the case multiple applications are visible  
328 at the same time, such as in the normal desktop case, the “background” status  
329 can be misleading since applications can still be partially visible. Wayland  
330 provides the frame clock to throttle the rendering of application windows which  
331 are not visible.

332 **Maintain an application stack**

333 Canterbury maintains a stack of applications to provide an Android-like back  
334 button. This feature should be implemented by the compositor to avoid layering  
335 violation. This is not part of canterbury-core as well and deprecated since  
336 Apertis 17.09.

337 **Store Last User Mode (LUM) information periodically and**  
338 **restore LUM on start-up**

339 This is not part of canterbury-core, and was deprecated since 17.09. Canterbury  
340 saves the currently running applications, “back” stack and the selected audio  
341 output in order to restore them on reboot.

342 The compositor should handle the saving and restoration of the application  
343 stack and the audio manager should save and restore the selected audio output  
344 without involving the application manager.

345 **Conclusions**

- 346 • No major gaps have been identified between Canterbury and Flatpak
- 347 • Flatpak has an very active upstream community and widespread adoption
- 348 • Most of the Canterbury APIs not related to app-management have been  
349 formally deprecated since Apertis 17.09
- 350 • Providing compatibility between the two would be a very big undertaking  
351 with unclear benefits, so it’s actively discouraged and existing applications  
352 needs to be ported explicitly
- 353 • HMI applications will need to be reimplemented in any case as Mildenhall  
354 is not a viable solution for product teams
- 355 • The Canterbury application framework will remain available in Apertis  
356 as an option at least until the new application framework has matured  
357 enough and reference applications are available for it, and product teams  
358 will be able to choose one or the other depending on their specific needs