



The next-gen Apertis application framework

1 **Contents**

2 Creating a vibrant ecosystem 2

3 The next-generation Apertis application framework 3

4 Application runtime: Flatpak 4

5 Compositor: libweston 6

6 Audio management: PipeWire and WirePlumber 7

7 Session management: systemd 7

8 Software distribution: hawkBit 8

9 Evaluation 8

10 Focus on the development user experience 12

11 **Legacy Apertis application framework 13**

12 **High level implementation plan for the next-generation Apertis**

13 **application framework 14**

14 Flatpak on the Apertis images 15

15 The Apertis Flatpak application runtime 15

16 Implement a new reference graphical shell/compositor 16

17 Switch to PipeWire for audio management 16

18 AppArmor support 17

19 The app-store 17

20 As a platform, Apertis needs a vibrant ecosystem to thrive, and one of the
 21 foundations of such ecosystem is being friendly to application developers and
 22 product teams. Product teams and application developers are more likely to
 23 choose Apertis if it offers flows for building, shipping, and updating applications
 24 that are convenient, cheap, and that require low maintenance.

25 To reach that goal, a key guideline is to closely align to upstream solutions
 26 that address those needs and integrate them into Apertis, to provide to appli-
 27 cation authors a framework that is made of proven, stable, complete, and well
 28 documented components.

29 The cornerstone of this new approach is the adoption of Flatpak, the modern
 30 application system already officially supported on [more than 20 Linux distribu-](#)
 31 [tions](#)¹, including Ubuntu, Fedora, Red Hat Enterprise, Alpine, Arch, Debian,
 32 ChromeOS, and Raspian.

33 The target audiences of this document are:

- 34 • for *Product Owners* and *Application Developers* this document describes
 35 how the next-generation Apertis application framework creates a reliable
 36 platform with convenient and low maintenance flows for building, deploy-
 37 ing, and updating applications;
- 38 • for *Apertis Developer* this document offers details about the concepts be-
 39 hind the next-generation Apertis application framework and a high level
 40 implementation plan.

¹<https://flatpak.org/setup/>

41 The goals of the next-generation Apertis application framework are:

- 42 • employ state-of-the-art technologies
- 43 • track upstream solutions
- 44 • expand the potential application developers pool
- 45 • leverage existing OSS documentation, tooling and workflows
- 46 • reduce ongoing maintenance efforts

47 The next-generation Apertis application framework is meant to provide a super-
48 set of the features of the legacy application framework and base them on proven
49 upstream OSS components where possible.

50 **Creating a vibrant ecosystem**

51 Successful platforms such as Android and iOS make the convenient availability
52 of applications a strategic tool for adding value to their platforms.

53 To be able to build an adequate number of applications with acceptable quality,
54 the entire platform is designed around convenience for developing, building,
55 deploying, and updating applications.

56 Given the relatively small scale of Apertis when compared to the Android and
57 iOS ecosystems, the best strategy is to align to the larger Linux ecosystem, and
58 Flatpak is the widely adopted solution to the previously listed challenges.

59 However, what makes Flatpak particularly compelling for Apertis is that Flat-
60 pak effectively creates a shared development ecosystem that crosses the distri-
61 bution boundaries: while the fact that being automatically able to run any
62 desktop Flatpak on Apertis is an amazing technological feat, the biggest benefit
63 for Apertis is that by joining the Flatpak ecosystem the skills developers need
64 to learn to develop applications for Apertis become the same as the ones needed
65 to write applications aimed at all the mainstream Linux desktop distributions.
66 This significantly expands the potential developer pool for Apertis, and ensures
67 that the easily available online documentation and workflows to build appli-
68 cations for the main Linux desktop distributions also automatically apply to
69 building applications for Apertis itself.

70 **The next-generation Apertis application framework**

71 The next-generation Apertis application framework is a set of technologies bring-
72 ing applications to the state-of-the-art of security and privacy considerations.

73 With the use of modern tools, the framework is meant to grant to the user strict
74 control over its data. Applications are meant to be run contained, and can talk
75 with each other and with the rest of the system only using dedicated interfaces.

76 The containment is designed to keep the applications on their restricted envi-
77 ronment and prevents to modify the base system in any way without being
78 explicitly granted to do so.

79 Whenever possible, applications have to define upfront their requirements to
80 access privileged resources, be it to share files across application or to get In-
81 ternet access. It is up to the [app store maintainers](#)² to review and ensure that
82 the requested access is sensible before it reaches final users. For other more
83 dynamic privileged resources, authorization can be granted at runtime through
84 explicit user interaction, usually via dedicated interfaces called “portals”.

85 Flatpak provides those guarantees by using the kernel namespacing and control
86 groups subsystems to implement containers similarly as what Docker does. Por-
87 tals are then implemented as D-Bus interfaces that application can invoke to
88 request privileged actions from inside their sandbox.

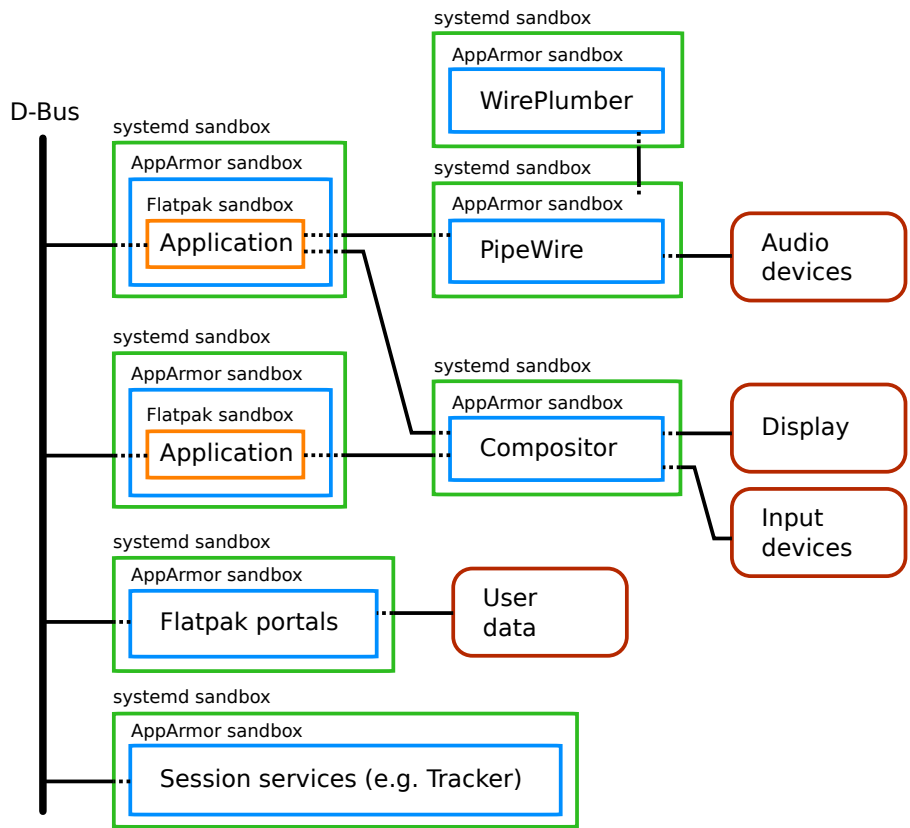
89 Access to the graphical session both to render the application contents and to
90 manage input from users is managed securely by a Wayland compositor.

91 Audio policies are extremely important for Apertis, specially so in automotive
92 environments, and PipeWire provides an excellent foundation to handle those
93 by providing the tools to wire applications to the needed resources in a secure
94 and customizable way.

95 Launching applications, agents, and other services happens through `systemd`,
96 which in charge to run both the system and the user sessions. Systemd provides a
97 [wide set of options to further secure services](#)³, track their resource consumption,
98 ensure their availability, etc.

²<https://sjoerd.pages.apertis.org/apertis-website/policies/contributions/#the-role-of-maintainers>

³<https://gist.github.com/ageis/f5595e59b1cddb1513d1b425a323db04>



99

100 **Application runtime: Flatpak**

101 Flatpak is a framework with the goal of letting developers to deploy and run
 102 their applications on multiple Linux distributions with little effort. To do so,
 103 it decouples the application from the base OS: this decoupling also allow an
 104 application to be deployed with no changes on different variants of the same
 105 base OS, different versions of the same base OS or even be deployed alongside
 106 another application which need an incompatible set libraries.

107 Decoupling the base OS from applications is particularly valuable for Apertis
 108 since it allows applications to be deployed seamlessly over multiple variants while
 109 minimizing the set of components shipped in the base OS.

110 Another interesting effect of the decoupling is that the release cycles of applica-
 111 tions are no longer tied to the one of the base OS: while the latter needs to go
 112 through a longer validation process, applications can release much faster and in
 113 a completely independent way.

114 **Applications as made by the developer**

115 A Flatpak application is a self-contained application based on a runtime, ensuring
116 that the user runs the application the way it has been meant by the developer
117 without depending on what is currently installed on the user machine.

118 **Secure by design**

119 A Flatpak application run confined under a restrictive security sandbox. Updates
120 for the application can be done quickly and atomically or according to
121 any system-wide policy. As Flatpak is vendor-agnostic, it allows ensuring that
122 the applications are genuine by signing the applications and the source store.

123 Flatpak at the moment does not support AppArmor to further confine applica-
124 tions. Since Apertis makes heavy use of AppArmor to protect its service, we
125 plan to add AppArmor support to Flatpak to add another layer of defense to
126 keep applications confined and prevent them from doing unwanted changes to
127 the base operating system.

128 **Privacy**

129 Every application ship with a security profile that describes the requirements
130 of the application and explicit consent from the user is needed to get access to
131 any service not described by the security profile.

132 **Integrated into the environment**

133 Flatpak is providing the latest standards for building applications: using re-
134 versed DNS domain name notation, AppStream and Desktop specifications from
135 FreeDesktop.org developers have a complete control over the metadata of their
136 applications and have the suitable tools to provide rich information describing
137 their application.

138 **Efficient and lightweight**

139 Flatpak is very efficient and doesn't require to spend time configuring a het-
140 erogeneous set of tools to work on a system. With libostree at the heart of
141 Flatpak, cutting-edge technology is used to reduce its footprint by the use con-
142 tent deduplication. The deduplication results in consuming less disk-space and
143 less network bandwidth.

144 **Release at your own pace**

145 Flatpak decouples applications from the underlying Operating System, so that
146 they can follow different release schedules minimizing the impact of conflicting
147 changes: applications in Flatpak rely on basic set of libraries called "frame-
148 works" that shield them from the actual libraries used by the OS. OSTree helps
149 to keep this redundancy under control, minimizing the storage consumption by
150 de-duplicating items in common. Frameworks help to keep the base OS lean and
151 minimal as non-core libraries can be moved closer to the applications that need
152 it, and thus development and validation can happen faster. On the application
153 side, new versions of basic libraries can be used without fearing regressions on
154 other applications, reducing the time to market.

155 **Compositor: libweston**

156 The compositor is the boundary between applications and the actual human-
157 machine interface: it is responsible of mediating access to the screen and to the
158 input devices, guaranteeing that each application only get the input commands
159 directed to it and can't read or interfere with the resources assigned to other
160 applications.

161 The next-generation Apertis application framework continues to rely on the
162 Wayland protocol to let applications talk to the compositor in a secure, efficient,
163 and well-supported way.

164 The compositor is meant to be agnostic of the UI toolkit applications use, and
165 by sticking to the commonly implemented Wayland interfaces it supports the
166 main OSS UI toolkits out of the box, even running at the same time, with no
167 custom code being required on the application side.

168 While applications targeting the next-generation Apertis application framework
169 should work with any compliant Wayland compositor implementing the most
170 common extensions, Apertis plans to provide a reference compositor that aims
171 to be customizable for the different non-desktop use-cases targeted by Apertis.

172 The main requirement for the reference compositor is to be based on `libweston`,
173 as this library is a valuable asset of reusable code for compositors originating
174 from the Weston project.

175 A good starting point for the compositor reference implementation is to use the
176 [agl-compositor](https://gerrit.automotivelinux.org/gerrit/admin/repos/src/agl-compositor)⁴ project because it was purposely built as a reference implemen-
177 tation. Ease of coding was a design goal, and it is expected that both the client
178 shell and the compositor itself are easy to understand and modify. The code
179 base is small, trim, maintained and is currently evolving.

180 Additional features includes support to clients using XDG shell protocol, and an
181 example of a compositor private extension that allows the client shell to provide
182 additional roles to surfaces.

183 Another option for the reference compositor is the [Maynard](https://gitlab.apertis.org/hmi/maynard)⁵ project. Unfortu-
184 nately the project is not currently maintained, and it's internal architecture is
185 outdated: it builds Weston plugins out of tree which was the recommended way
186 before `libweston` existed. The main issue of using Maynard is that because it is
187 not maintained upstream, we would need to maintain it ourselves.

188 **Audio management: PipeWire and WirePlumber**

189 Applications should be able to play sounds and capture the user speech if they
190 desire to do it, but the system need to guarantee that:

- 191 • applications cannot interfere with the audio streams of other applications;

⁴<https://gerrit.automotivelinux.org/gerrit/admin/repos/src/agl-compositor>

⁵<https://gitlab.apertis.org/hmi/maynard>

- 192 • access to the audio captured by microphones is granted only on explicit
193 authorization by the user whenever possible;
194 • on a multi-zone setup like on some cars, sounds are emitted in the zone
195 where the application is displayed;
196 • important messages can be emitted in clear, audible way even if other
197 applications are already playing multimedia contents, by pausing the other
198 streams whenever possible or mixing the streams at different volumes.

199 PipeWire is the current state-of-the-art solution for secure and efficient audio
200 routing. Applications can use it natively, from GStreamer, or via the ALSA and
201 PulseAudio compatibility layers, and it is designed to work well when combined
202 with the Flatpak sandboxing capabilities.

203 Since PipeWire does not include any default policy engine, a separate compo-
204 nent is in charge of setting up the connections between the PipeWire nodes
205 to ensure that the system rules are enforced. The [WirePlumber](#)⁶ project from
206 AGL implements such policy service with goals and restrictions aligned to the
207 ones for Apertis.

208 **Session management: systemd**

209 While not directly exposed to applications, session management is a fundamental
210 part of the application framework with the purpose of:

- 211 • launching applications upon user request from the graphical launcher;
212 • running headless agents;
213 • activating session services needed by applications and agents;
214 • monitor the life-cycle of applications and services;
215 • enforce resource tracking on applications and services.

216 The systemd user session system provides the currently most advanced solution
217 to the above problem space, with the Apertis legacy application framework
218 already making use of it and other mainstream environment like GNOME being
219 in the process of completely switching to systemd to manage their sessions.

220 **Software distribution: hawkBit**

221 For software distribution use-cases Apertis supports Eclipse hawkBit, a domain
222 independent back-end framework for rolling out software updates to constrained
223 edge devices as well as more powerful controllers and gateways connected to IP
224 based networking infrastructure. This software distribution has to be enhanced
225 to gain flatpak support.

226 With Flatpak, bundle repositories can be created and configured as needed, and
227 a single system can fetch applications from multiple repositories at the same
228 time.

⁶<https://gitlab.freedesktop.org/gkiagia/wireplumber>

229 Apertis will offer a reference instance where application can be shared and made
230 available to all the Apertis users, to foster collaboration and to provide a rich
231 set of readily available applications.

232 Downstreams and product teams can set up their own instance to publish ap-
233 plications intended for a more limited audience.

234 The Apertis reference store also builds on top of the Apertis GitLab code hosting
235 services to define a reproducible Continuous Integration workflow to automati-
236 cally build applications from source and publish them to the app store.

237 Once the quality assurance has validated a specific version of an application, an
238 easy way is provided to the developer to publish the Apertis hawkBit instance.

239 To ensure a good quality of service, and to be certain that the service matches the
240 expectations, Apertis core applications may themselves be shipped as Flatpak
241 bundles over the Apertis hawkBit instance.

242 **Evaluation**

243 The next-generation application framework matches all the requirements that
244 have driven the development of the legacy application framework.

245 In particular, in no way the next-generation application framework results in
246 a loss of functionality or features: it instead builds on top of mature, proven
247 technologies to expand what it is possible with the legacy framework, adapting
248 to the evolving state-of-the-art application ecosystem on Linux.

249 The application framework is compliant with the current requirements of the
250 Apertis platform for [system services](#)⁷, [user services](#)⁸, and [graphical programs](#)⁹.
251 It relies heavily on the freedesktop.org specifications that specify where appli-
252 cations can store their data with different guarantees, how their metadata is to
253 be encoded, and how they can best integrate with the system.

254 Flatpak uses `libostree` to implement robust application updates and rollbacks,
255 efficiently using network bandwidth and local storage. Updates are signed and
256 the alternative signing mechanisms developed by Apertis for its system updates
257 can be used to avoid the GPL-3 issues related to the use of GnuPG.

258 The requirement of having a security boundary between applications is ad-
259 dressed by the use of the control group and namespacing kernel subsystems.
260 The use of AppArmor can be introduced to add another layer of defense to the
261 already strong security provisions Flatpak offers. Flatpak also let applications
262 to be installed per-user, increasing the separation on multi-user systems.

263 Application data and settings are stored inside the application sandbox, ensuring
264 that they are stored securely, that they can be managed easily for rollback

⁷<https://sjoerd.pages.apertis.org/apertis-website/glossary/#system-service>

⁸<https://sjoerd.pages.apertis.org/apertis-website/glossary/#user-service>

⁹<https://sjoerd.pages.apertis.org/apertis-website/glossary/#graphical-program>

265 purposes, and that applications are free to chose any mechanisms to manage
266 them.

267 **App bundle contents**

268 The Flatpak [application bundle contents](#)¹⁰ is a well-defined application layout
269 that largely matches the approach used by the legacy application framework,
270 improving over it in particular with the introduction of “frameworks” as a way
271 to decouple the application from the base OS and yet retain efficiency in term
272 of deploying updates affecting multiple applications and in term of storage con-
273 sumption.

274 With the use of Flatpak frameworks any language runtime can be used easily
275 by applications even if the base OS does not ship it.

276 **Data Management**

277 Flatpak applications can use the [XDG Base Directory Specification](#)¹¹ to find
278 the appropriate places to store persistent private data that can’t be accessed by
279 other applications, and temporary cache files that can be deleted by the system
280 to reclaim space.

281 Policies for storage space reclaiming and rollback need to be defined and are to
282 be implemented in dedicated components.

283 **Sandboxing and security**

284 With the use of the control group and namespacing kernel subsystems, Flatpak
285 offers a state-of-the-art approach for containing applications to limit what they
286 can access on the system and to isolate them from each other.

287 The integrity of the application data is guaranteed by the namespaced applica-
288 tion filesystem being mounted read-only, and thus being unmodifiable by the
289 application itself, and by using namespaces to limit the amount of data each
290 application can access.

291 Applications can not see the other installed and running applications and neither
292 can modify them. They also can’t communicate between each other without user
293 consent.

294 **App pemissions**

295 The [Flatpak pemissions](#)¹² system allows to declare in advance any needed per-
296 missions to access sensitive resources like user data or special devices, to be
297 reviewed by app store curators.

298 Additional runtime permissions to access data outside of what the application
299 normally need to use can be granted via explicit user actions, usually via dedi-
300 cated Flatpak portals.

¹⁰<https://github.com/flatpak/flatpak/wiki/Filesystem>

¹¹<https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html>

¹²<http://docs.flatpak.org/en/latest/sandbox-permissions.html>

301 Integration with Flatpak portals to transparently grant applications privileged
302 access on explicit user actions is already available in the main application toolk-
303 its like Qt, GTK, etc.

304 **App launching**

305 Each installed Flatpak application automatically exports its `.desktop` entry
306 point, in a way that any compliant application launcher can automatically list
307 and start the installed Flatpak applications.

308 The applications themselves have to use the [Desktop Entry Specification](#)¹³ to
309 provide the required metadata and entry points.

310 It is possible for applications to explicitly specify that they should not be listed
311 in the launcher, to avoid headless agents polluting the menu.

312 **Document launching**

313 Applications and entry points can specify the media types they handle using the
314 [MIME type handling provisions](#)¹⁴ from the [Desktop Entry Specification](#)¹⁵. The
315 application framework is responsible of making the selected document visible to
316 the associate application and run the application if it wasn't previously running,
317 or queue the queue the file opening on busy systems.

318 **URI launching**

319 With the special `x-scheme-handler` MIME type the same mechanism used for
320 *Document launching* can be used to handle specific URI schemes. In case the
321 URI scheme is a `file`, treat it as launching a local document.

322 **Content selection**

323 Flatpak provides portals to let users explicitly grant access to any of their files
324 without any upfront special permissions being granted to the application. Inte-
325 gration with the file selection portals is already available in the most widespread
326 OSS application toolkits.

327 **Data sharing**

328 Flatpak applications can be granted special permissions to access D-Bus services
329 or filesystem subtrees that can be used to share data across a set of applications.
330 Flatpak also let applications to be activated on-demand via D-Bus, which can
331 be particularly useful for headless agents.

332 **Life cycle management**

333 Each Flatpak sandbox automatically contains all the application processes in
334 a secure and efficient way. The system user session management can add an-

¹³<https://standards.freedesktop.org/desktop-entry-spec/latest/>

¹⁴<https://standards.freedesktop.org/desktop-entry-spec/latest/ar01s10.html>

¹⁵<https://standards.freedesktop.org/desktop-entry-spec/latest/>

335 other layer of control, tracking both application and system services with a
336 homogeneous approach.

337 The compositor can track to which process and thus to which application or
338 service each window belongs to.

339 **Last used context**

340 Applications can store their last status in their private data area and have
341 it available on the next launch, enabling the implementation of the simplest
342 approach purely on the application side with no specific involvement of the
343 application framework.

344 More advanced use cases that may require a deeper involvement of the applica-
345 tion framework needs to be evaluated.

346 **Installation management**

347 Flatpak allows applications to be installed system-wide or per-user, and provides
348 extensive tooling to retrieve contents from remote stores, list local applications,
349 and fetch updates.

350 The use of OSTree to store application contents makes rolling them back simple
351 and efficient. Data is not usually rolled back when rolling back an applica-
352 tion: if use-cases require data rollback it needs to be implemented in dedicated
353 components.

354 Flatpak also provides both efficient online and offline installation mechanisms.

355 **Conditional access**

356 Flatpak lets applications to be installed either system-wide, making them avail-
357 able to every user, or per-user where only user that have explicitly installed an
358 application can access it.

359 However, the latter means that storage is not de-duplicated. Advanced setups
360 may be defined to leverage the de-duplication capabilities of OSTree without
361 automatically sharing installed applications with every user of the system.

362 **UI customization**

363 One of the key values for Apertis is to be aligned with upstream, so the best UI
364 customization strategy is to rely on the upstream theming infrastructure offered
365 by toolkits like GTK.

366 Flatpak can [inject system themes in the containerized runtimes](#)¹⁶ to apply a
367 global theme without changing anything in the applications.

¹⁶<https://blog.tingping.se/2017/05/11/flatpak-theming.html>

368 **Focus on the development user experience**

369 Flatpak provides extensive tooling to give developers a working environment
370 that is easy to setup and use: the framework provides the necessary tools and
371 libraries for developers to create their application and is highly extensible.

372 A key part of delivering the best developer experience is by promoting a default
373 Integrated Development Environment (IDE). As such, GNOME Builder is the
374 current best option for providing an efficient environment bringing Flatpak as a
375 first-class component, integrating with many languages, providing support for
376 Git versioning system, and available on any Linux distribution as it is itself
377 distributed as a Flatpak.

378 As the framework is composed of a set of different tools interacting with each
379 other, it is also possible for the developer to use a classic developer workflow
380 and use the command line to build and install an application. Guaranteeing the
381 same result independently of the machine it is built on and thus allowing fully
382 reproducible builds. The framework itself is built upon existing technology, it
383 will benefit from the broadly available documentation and support of highly
384 heterogeneous build configuration that each application requires.

385 Installing a flatpak application from Flathub only requires a single command,
386 here is an example with Goodvides, an internet radio player application:

```
1 flatpak install flathub io.gitlab.Goodvibes
```

387 The application can then be run by clicking on the desktop icon or simply with:

```
1 flatpak run io.gitlab.Goodvibes
```

388 Each application can be defined using a [standard manifest](#)¹⁷ that describes all
389 the dependencies, their source and how to build them. If a dependency is not
390 in the Apertis framework Runtime, it can be added by the developer itself in
391 the definition file. The libraries aren't shared with the base system, allowing
392 the developer to ship the version of the dependency that matches the needs of
393 the software and not needing to wait for it to be available in the system itself.
394 A set of tools is even available for the developer to build a runtime using the
395 same dependencies that are available on its machine.

396 To illustrate the comprehensive coverage of flatpak regarding the developer ex-
397 perience, here are the few steps to build the Goodvides application that we
398 previously mentioned:

¹⁷<http://docs.flatpak.org/en/latest/manifests.html>

399 1. Getting the manifest describing the dependencies from the original pack-
400 age

```
1 flatpak run --command=cat io.gitlab.Goodvibes /app/manifest.json > io.gitlab.Goodvibes.json
```

401 2. Build the flatpak locally, allowing to install the dependencies from flathub
402 if required

```
1 flatpak-builder --install-deps-from=flathub build-  
dir io.gitlab.Goodvibes.yaml
```

403 That's it, the flatpak is now built

404 3. For testing the result, you can directly use

```
1 flatpak-builder --run build-dir io.gitlab.Goodvibes.yaml goodvibes
```

405 Legacy Apertis application framework

406 Both the new and the legacy Apertis application frameworks are meant to be
407 available at the same time in Apertis as alternative options, the legacy frame-
408 work being shipped on the reference images in the short term. As the new
409 application framework matures, components will get swapped on the reference
410 images, leaving the legacy components available in the archive.

411 See the [canterbury legacy application framework](#)¹⁸ for more details.

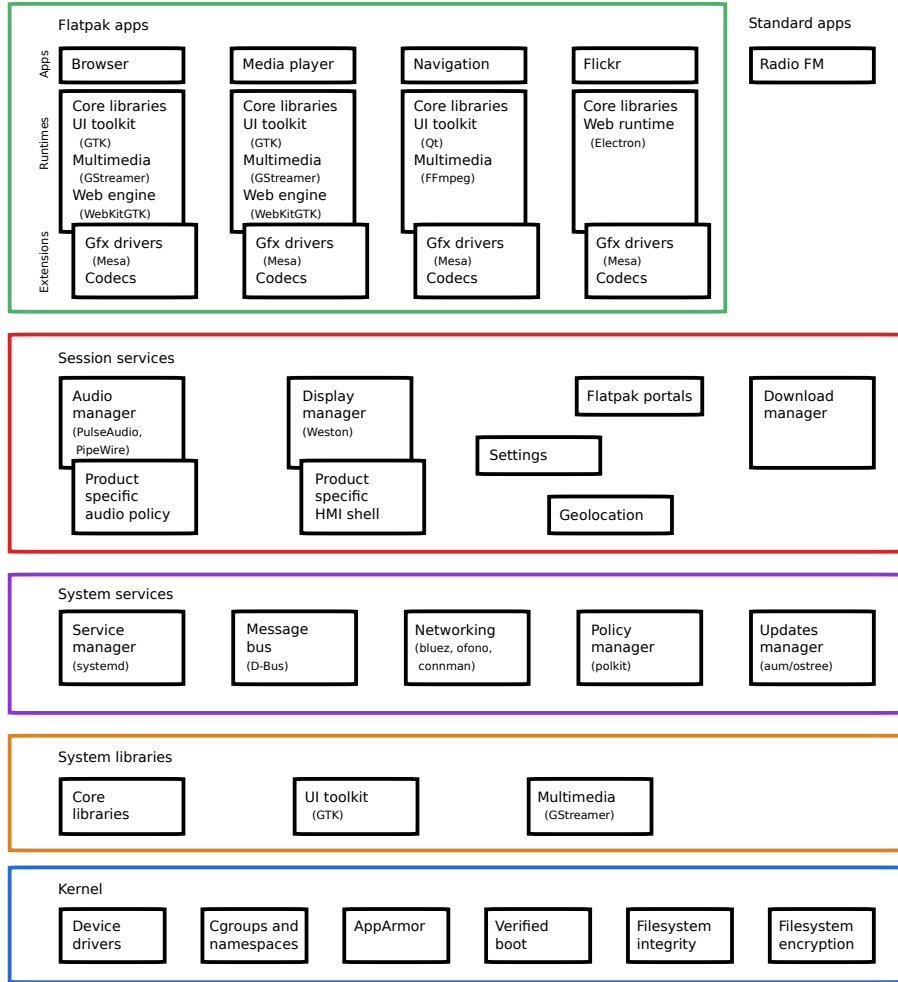
412 High level implementation plan for the next- 413 generation Apertis application framework

414 The transition to the new infrastructure can follow a process to keep the legacy
415 framework fully available during the whole process and ensure that it still con-
416 tinue to work afterwards. Both frameworks will be in the Apertis repositories as
417 mutually exclusive options to be chosen by product teams based on their needs.

418 The new Apertis application framework integrates with the existing QA and
419 testing platform for Apertis.

¹⁸<https://sjoerd.pages.apertis.org/apertis-website/architecture/canterbury-legacy-application-framework/>

420 The implementation will be held within a few different axis that can be devel-
 421 oped in parallel and in the order that might make more sense at the time of the
 422 implementation.



423

424 Flatpak on the Apertis images

425 The goal here is to ensure that all the Flatpak tools and services are working
 426 on the reference Apertis images.

- 427 1. Ensure that all the Flatpak tools are installed by default on the reference
 428 Apertis images:
- 429 • target images have the tools needed to install, update, run, and re-
 430 move Flatpak applications
 - 431 • SDK images also ship the tools needed to create Flatpak bundles

- 432 2. Test that a simple test application like GNOME Calculator can be in-
433 stalled on the reference Apertis images, that it gets displayed normally
434 and that the user interaction is also working.
- 435 3. Test a more complex application like Goodvibes, ensure that the audio
436 playback is working.
- 437 4. Test more complex applications requiring GL rendering (for instance,
438 OpenArena), ensure that the open-source graphical rendering stack works.
439 Testing the proprietary graphical stack is out of scope as it does not pro-
440 vide same levels of functionality and support when compared to the open
441 source stack.
- 442 5. Taking the needs of the product teams into consideration, go through the
443 list of official portals and ensure that they are functional.

444 **The Apertis Flatpak application runtime**

445 The goals here are to create a reference Flatpak runtime for Apertis applications
446 and move all the applications to Flatpak.

447 To avoid bottlenecks, the Flatpak bundles produced in the steps described here
448 can be tested on any non-Apertis platform supporting Flatpak.

- 449 1. Setup [Flatdeb](https://gitlab.collabora.com/smcv/flatdeb)¹⁹ to automate the creation of Flatpak runtimes and Flatpak
450 applications from `.deb` packages using the GitLab Continuous Integration
451 pipelines.
- 452 2. Create a basic Apertis reference runtime aimed at headless agents and
453 without legacy component like Mildehall, built with [Flatdeb](https://gitlab.collabora.com/smcv/flatdeb)²⁰, similar to
454 the FreeDesktop.org SDK.
- 455 3. Create a guide for product teams to create their own applications and
456 runtimes using the Apertis tools.
- 457 4. Create a basic Flatpak runtime to run Mildenhall applications
- 458 5. Convert the sample-apps to Flatpak using the Mildenhall runtime, starting
459 from the simplest ones to the ones requiring the most interaction with the
460 system. Ensure that each porting process is documented.
- 461 6. Coalesce the documentation in a comprehensive guide to convert legacy
462 applications.
- 463 7. Convert more complex Mildenhall legacy applications like Frampton.
- 464 8. Create a legacy-free Apertis reference runtime for GUI applications.
- 465 9. Investigate more modern alternatives to the Mildenhall legacy demo ap-
466 plications and base them on the legacy-free Apertis reference runtimes.

467 **Implement a new reference graphical shell/compositor**

468 This section is about deploying a new graphical shell based on modern compo-
469 nents and avoiding deprecated libraries like Clutter.

¹⁹<https://gitlab.collabora.com/smcv/flatdeb>

²⁰<https://gitlab.collabora.com/smcv/flatdeb>

- 470 1. Begin with a new minimal shell based on the Weston Wayland compositor
471 and make it available on the reference images, to be enabled optionally.
- 472 2. Ensure that legacy Mildenhall applications work properly under the new
473 compositor.
- 474 3. Progressively add features like notifications and an application drawer to
475 discover and launch applications.
- 476 4. Switch the default compositor from the legacy Mildenhall-Compositor to
477 the new one.
- 478 5. Iteratively improve the look and feel of the shell.
- 479 6. Document how the shell can be customized or replaced by product teams
480 while fully re-using the Weston core compositor implementation.

481 **Switch to PipeWire for audio management**

482 The steps described here are about making audio management more secure and
483 flexible on Apertis.

- 484 1. Update the [Apertis audio management](#)²¹ design document to describe the
485 different approach using [PipeWire](#)²² instead of PulseAudio.
- 486 2. Start the work using a basic policy with [WirePlumber](#)²³ from AGL.
- 487 3. Ensure that audio capture is functional using a simple audio player appli-
488 cation.
- 489 4. Ensure that video capture is functional using a simple camera viewer ap-
490 plication.
- 491 5. Ensure that audio playback is functional without PulseAudio, but still
492 default to PulseAudio for audio playback.
- 493 6. Ensure compatibility with applications using the PulseAudio client li-
494 braries to provide a smooth migration.
- 495 7. Switch the default for audio playback to PipeWire.
- 496 8. Progressively refine policies and introduce stream priority handling.
- 497 9. Provide a guide for product teams about customizing the audio manage-
498 ment policies.

499 **AppArmor support**

500 This section focuses on using AppArmor as an additional level of security to
501 constrain applications.

- 502 1. Add a basic AppArmor profile setup to Flatpak to ensure each application
503 runs with its dedicated profile.
- 504 2. Progressively make the application profile more strict.
- 505 3. Customize the AppArmor profile based on the application permissions
506 described in its manifest.

²¹<https://sjoerd.pages.apertis.org/apertis-website/concepts/audio-management/>

²²<https://pipewire.org>

²³<https://gitlab.freedesktop.org/gkiagia/wireplumber>

507 **The app-store**

508 For the user-driven use-case it is key to demonstrate a full workflow that includes
509 an application store.

510 The store and the deployment management service are kept separate:

- 511 • the store is the front-end for the user and is the commercial layer of the
512 system (payments, etc.);
 - 513 • the deployment management service manages the actual installation of
514 the software on the device based on the state of the store, but also dealing
515 with updates that do not go through the store.
- 516 1. Improve the reliability of the Apertis hawkBit instance.
 - 517 2. Plug the Apertis hawkBit instance authentication system to the Apertis
518 user database.
 - 519 3. Extend the application building pipelines to push Apertis apps to hawkBit.
 - 520 4. Extend the hawkBit agent to manage Flatpak applications.
 - 521 5. Create and deploy a simple front-end store for applications, extending an
522 existing e-commerce platform or adopting hawkBit-based solutions like
523 the [Kuksa Appstore](#)²⁴.
 - 524 6. Ensure that the whole app-store workflow is documented and functional
525 to handle user-driven installations and updates via hawkBit.
 - 526 7. Extend the hawkBit agent and other tools to handle the [conditional ac-](#)
527 [cess](#)²⁵ use cases.
 - 528 8. Provide a guide for product teams about deploying their own app-store.

²⁴<https://github.com/eclipse/kuksa.cloud/tree/master/kuksa-appstore>

²⁵https://sjoerd.pages.apertis.org/apertis-website/designs/conditional_access/