



Software distribution and updates

1	<b>Contents</b>	
2	<b>Terminology</b>	<b>2</b>
3	Application and services . . . . .	2
4	Base operating system . . . . .	2
5	Bundles . . . . .	2
6	Software distribution . . . . .	3
7	Software updates . . . . .	3
8	<b>Operator-driven use cases</b>	<b>3</b>
9	Building access control devices . . . . .	3
10	Robotic lawn mower . . . . .	4
11	<b>User-driven use cases</b>	<b>4</b>
12	Infotainment system . . . . .	5
13	Power and measuring tools . . . . .	6
14	<b>Non-use-cases</b>	<b>6</b>
15	<b>Requirements</b>	<b>6</b>
16	Conditional software deployment based on business rules . . . . .	6
17	Configurable access rights to user data and system resources . . . . .	7
18	Consistent state across devices . . . . .	7
19	Independent release and update of application domains . . . . .	7
20	Operator-driven software distribution and updates . . . . .	7
21	Protecting the fleet from software deployment issues . . . . .	7
22	Resilience to distribution and update failures . . . . .	7
23	Resilience to user operation . . . . .	7
24	Software inventory . . . . .	7
25	Tampering protection . . . . .	8
26	Unwanted changes to the software stack . . . . .	8
27	Updates rollback . . . . .	8
28	User-driven software distribution . . . . .	8
29	<b>High level features</b>	<b>8</b>
30	Immutable software stack . . . . .	8
31	Atomic updates . . . . .	9
32	Separation between system and application domains . . . . .	9
33	Deployment management . . . . .	9
34	<b>Existing systems</b>	<b>10</b>
35	OSTree for base operating system . . . . .	10
36	Flatpak and Docker for applications . . . . .	11
37	Eclipse hawkBit . . . . .	11
38	Microsoft Azure IoT Edge . . . . .	11
39	<b>Appstore</b>	<b>12</b>

40	Curridge . . . . .	12
41	Flathub . . . . .	13
42	<b>Summary of recommendations</b>	<b>13</b>
43	<b>Reference: System updates and rollback</b>	<b>13</b>

44 Apertis is a mature platform that is compatible with modern and flexible solu-  
 45 tions for software distribution and software update. This document describes  
 46 user-driven and operator-driven use cases, explores the challenges of each use  
 47 case to extract requirements, and finally propose building blocks for software  
 48 distribution and software update.

## 49 Terminology

### 50 Application and services

51 Application and services are loosely defined terms that indicate single functional  
 52 entities from the perspective of end users. However each application may be  
 53 composed of more than one component:

- 54 • [system services](#)<sup>1</sup>
- 55 • [user services](#)<sup>2</sup>
- 56 • [graphical programs](#)<sup>3</sup>

57 From the perspectives of software updates and software distribution applications  
 58 and services can be deployed as part of the base operating system or separately  
 59 as [bundles](#)<sup>4</sup>.

### 60 Base operating system

61 The base operating system is the core component of the software stack. It  
 62 includes the kernel, and basic userspace tools and libraries such as process man-  
 63 ager, connectivity services, and update manager. Additional components like  
 64 an application manager may be part of the base OS, depending on the intended  
 65 usage.

### 66 Bundles

67 A bundle or “application bundle” refers to a unit that represent all the com-  
 68 ponents of an Application or service. Comparing to mobile phones a bundle is  
 69 similar to a phone “app”, and we would say that an Android .apk file contains  
 70 a bundle. Some systems refer to this concept as a package, but that term is

---

<sup>1</sup><https://em.pages.apertis.org/apertis-website/glossary/#system-service>

<sup>2</sup><https://em.pages.apertis.org/apertis-website/glossary/#user-service>

<sup>3</sup><https://em.pages.apertis.org/apertis-website/glossary/#graphical-program>

<sup>4</sup><https://em.pages.apertis.org/apertis-website/glossary/#application-bundle>

71 strongly associated with dpkg/apt (.deb) packages in Debian-derived systems,  
72 and it only partially captures the concept of a bundle.

73 The granularity is usually different between packages and bundles. Installing  
74 an application using packages is likely to involve multiple packages, while the  
75 bundle approach in our context goes in the direction of a single monolithic  
76 bundle that contains all components of an application. A bundle, unlike a  
77 package, offers atomic updates, rollback, insulation from the base operating  
78 system, insulation from other applications and configurable run time permissions  
79 for user data and system resources.

80 Docker images, Flatpak bundles, and Snaps are all examples of application  
81 bundles.

## 82 **Software distribution**

83 Software distribution is the process of delivering software to users and devices.  
84 It usually refers to the distribution of binaries of software to be installed or up-  
85 dated. However software distribution is more than a transport layer for packages  
86 as it can include authorization, inventory, and deployment management.

## 87 **Software updates**

88 The most common goals of an update are fixing bugs, removing security vul-  
89 nerabilities, and adding new features to already installed software. Updating  
90 a software component may also involve updating the chain of dependencies of  
91 that software component.

## 92 **Operator-driven use cases**

93 The operator is an entity with the responsibility of ensuring that the devices  
94 operate within pre-defined specifications. A device can have more than one oper-  
95 ator such as the manufacturer and the owner of the devices, and the operators,  
96 and not the device user, have powers to install, remove and update software on  
97 the devices.

## 98 **Building access control devices**

99 Access control is used to restrict access to a particular place, building, room, or  
100 resource. To gain access an individual generally needs to be given permission  
101 to enter by someone who already has authorization.

102 Automated building access rely on control devices to authenticate identity and  
103 to control physical locks. These devices use a variety of authentication methods  
104 such as smart cards, biometric data, and passwords, and can control access  
105 devices such as doors, gates and turnstile.

106 For most use cases, building access control devices are only the interface for  
107 more complex systems that include secure networks and servers. Building access  
108 control devices collect authentication data and send it to a server. The server  
109 then decides if the physical access should be granted, and send commands back  
110 to the device for informing the user and for controlling the lock.

111 Building access control devices have a critical mission. Failing to grant access  
112 to authorized personnel or granting access to unauthorized personnel can have  
113 serious consequences that can go beyond financial losses. Mission critical de-  
114 vices have strict reliability and security requirements, which include protection  
115 against tampering, resilience to user operation, and resilience to minor failures  
116 on the devices.

117 Both the manufacturer and owner may operate a large fleet of building ac-  
118 cess control devices. Large fleets are vulnerable to unintended changes on the  
119 software stack as it can introduce reliability and security issues. Low severity  
120 variability issues can be solved remotely, but high severity issues require manual  
121 intervention on each affected device.

122 Another problem for large fleet of devices is software deployment. Updates and  
123 new features should be deployed to devices on the field with minimal risk of  
124 rendering devices unusable. Operators require information about the software  
125 stack(installed software, version, etc) of each device to make decisions about  
126 how and when to do software deployment.

127 Device manufacturers offer on-demand development services. A new feature is  
128 developed for a customer and then is deployed only to the devices of that spe-  
129 cific customer. Delivering the custom features requires conditional deployment  
130 capabilities based on business rules such as device owner and service level.

## 131 **Robotic lawn mower**

132 A robotic lawn mower is an electric autonomous robot that cuts lawn grass in  
133 a pre-determined area. Common features of robotic lawn mowers include find-  
134 ing the recharging base automatically, avoiding obstacles, and using advanced  
135 algorithms to cover the working area efficiently.

136 High-end robotic lawn mower are connected to the cloud to allow the owner  
137 to configure and control the unit using a convenient web interface. The owner,  
138 acting as the operator, uses a website to configure the schedule and settings of  
139 the mower such as the cutting height. Some models also allow the operator to  
140 remote control the mower.

141 Connected robotic lawn mowers receive over-the-air updates that are installed  
142 when the mower is not in use, respecting the schedule that was configured by  
143 the operator.

## 144 **User-driven use cases**

145 There are two categories of user-driven use cases. The first one is built on top  
146 of operator-driven use cases. In this category the device allow users to install  
147 and remove optional applications, but keeps the operator in control of system  
148 updates and system applications. In the second category the device is left under  
149 full control of the user, without any operator involvement.

## 150 **Infotainment system**

151 An infotainment system is usually an interface between users and a vehicle show-  
152 ing information about the vehicle and allowing the user to configure options such  
153 as interior lights and air conditioning. An infotainment system also provides  
154 additional functionality such as navigation, connectivity with the user's phone,  
155 music, Internet browser, and allows the user to install and remove applications.

156 An infotainment system can offer a personalized set of features for different  
157 models of vehicles and for different users. Premium features and applications  
158 are only available for owners of premium models of the vehicle and for users  
159 willing to pay for them.

160 The life cycle of an infotainment system can go beyond a decade, creating a  
161 challenging scenario for support and maintenance of the software stack. The  
162 vast majority of software components used in an infotainment system have a  
163 much smaller release cycle, with more than one release per year being common.

164 Releases are important for software components because only the latest releases  
165 receive security and bug fixes. Failing to keep the software stack using fairly  
166 recent components results in an infotainment system with bugs and security  
167 vulnerabilities.

168 On the other hand, as users interact with infotainment systems while driving,  
169 these devices are heavily regulated. The device requires an expensive certi-  
170 fication process before deployment, and software updates are also subject to  
171 certification. So while updates are important for bug and security fixes, the  
172 structure and costs of certification of changes makes pressure against too fre-  
173 quent updates.

174 Another important actor in the infotainment ecosystem is the application de-  
175 veloper. Empowering the application developer results in greater availability of  
176 applications and in faster availability of updates. Having more applications is  
177 a competitive advantage for the infotainment system, as users may prefer the  
178 infotainment system that has more installable applications.

179 Application developers need to be able to target as many different infotainment  
180 products as possible without being tied to the release cycle of each specific  
181 product. In other words, it is important for the developer to be as close as  
182 possible to have a single application that runs without changes in different  
183 infotainment systems and in different releases of infotainment systems.

184 This is particularly challenging as the very long lifecycle of infotainment prod-  
185 ucts means that there are significant differences in the kind and versions of  
186 components shipped as part of the base operating system of different products.  
187 As such an application developer should be capable of releasing and updating  
188 applications independently from the base operating system, and should be able  
189 to conveniently create bundles that are optimized for a modern development  
190 flow.

191 The physical deployment characteristics of infotainment systems also complicate  
192 maintenance and updates. An unrecoverable failure due to an over-the-air up-  
193 date may force vehicle owners to pay a visit to the closest service center making  
194 customers unhappy, and potentially causing significant financial loss when the  
195 problem affects tens of thousands of vehicles.

196 And finally resilience to user operation is also a challenge to infotainment sys-  
197 tems. Users should not be able to render the device inoperative, or make the  
198 device to operate outside its design specifications by continuous use, by changing  
199 configurations, or by installing/uninstalling applications.

## 200 **Power and measuring tools**

201 Power tools are electrically driven tools such as drills and grinders, with most  
202 models being powered by batteries. Measuring tools are electronic devices for  
203 measuring, or helping the user to measure, physical properties of the environ-  
204 ment. Examples of measuring tools are wall scanners, thermo cameras, and  
205 laser measures.

206 Connected power and measuring tools can receive over-the-air updates and offer  
207 a convenient interface for the user to adjust operating parameters and to see  
208 the device status. The user can choose between a web interface and a mobile  
209 phone application to interact with power and measuring tools.

## 210 **Non-use-cases**

- 211 • Product development: during product development developers need to  
212 privilege flexibility over robustness. However robustness is of primary  
213 importance in production environment, and as such flexibility to ease de-  
214 velopment is not a use case.
- 215 • Workstations: while the mechanism described here are valuable on work-  
216 stations as well, they are not the focus of this document.

## 217 **Requirements**

### 218 **Conditional software deployment based on business rules**

219 It should be possible to restrict the selection of software components that users  
220 and operators can install, remove and update based on business rules such as  
221 payment, customer, service level, and market segment.

222 It should also be possible for the operator to configure the deployment to ad-  
223 here to business rules such as available time slots for maintenance, and to split  
224 complex deployments in batches.

### 225 **Configurable access rights to user data and system re- 226 sources**

227 Applications should have limited and configurable access to system resources  
228 and user data. For example, applications should not be capable of taking screen  
229 shots, and the music player should have access to only specific files and folders.

### 230 **Consistent state across devices**

231 Maintaining a large fleet of devices requires the software stack of each device to  
232 be in a known state. Devices in unknown state are challenging to maintain and  
233 may present reliability and security issues.

### 234 **Independent release and update of application domains**

235 It should be possible to release and update application domains independently  
236 from the base operating system.

### 237 **Operator-driven software distribution and updates**

238 On operator-driven use cases, the operator should be capable of controlling the  
239 software distribution and update of large fleets of devices.

### 240 **Protecting the fleet from software deployment issues**

241 There should be mechanisms in place to prevent software distribution and soft-  
242 ware update issues, such as an update that renders the devices unusable, to  
243 affect the entire fleet of devices.

### 244 **Resilience to distribution and update failures**

245 Minor problems such as an update failure due to download problem caused by  
246 a network issue on the device side should not render the device inoperative and  
247 should recover automatically without intervention.



## 248 **Resilience to user operation**

249 User actions including installing and removing optional applications should not  
250 render the device inoperative, or make the device to operate outside its design  
251 specifications.

## 252 **Software inventory**

253 Operators require software inventory information such as installed software,  
254 and software version to make decisions about how and when to do software  
255 deployment. As an example when a security vulnerability is discovered, having  
256 an overview of how many devices are affected is important to determine the  
257 severity of the vulnerability, and to plan a response.

## 258 **Tampering protection**

259 Mission critical devices and devices subject to regulation require protection  
260 against unauthorized modification. Users should not be allowed to modify the  
261 devices to operate outside its design specifications.

## 262 **Unwanted changes to the software stack**

263 A common method of attacking a device consists in changing software that is  
264 installed or installing malicious components. Preventing unwanted changes on  
265 the software stack, and preventing non-authorized software to be installed elim-  
266 inates an important attack vector: attacks that require changes to the software  
267 stack.

## 268 **Updates rollback**

269 Software updates should be reversible, and allow to rollback to a previous work-  
270 ing state. This requirement applies to system software and applications.

## 271 **User-driven software distribution**

272 The user should be capable of installing and removing software components on  
273 user-driven use cases.

## 274 **High level features**

275 Before describing existing solutions it is necessary to group the requirements  
276 in features that are implemented by these solutions. One requirement may  
277 be related to more than one feature such as the requirement *Consistent state*  
278 *across devices* being related to the features *Immutable software stack* and *Atomic*  
279 *updates*.

## 280 **Immutable software stack**

- 281 • Related requirements: *Consistent state across devices, Resilience to user*  
282 *operation, Tampering protection, Unwanted changes to the software stack*

283 One solution to address these requirements is to make the base operating system  
284 and the application domains immutable.

## 285 **Atomic updates**

- 286 • Related requirements: *Consistent state across devices, Protecting the fleet*  
287 *from software deployment issues, Resilience to distribution and update*  
288 *failures, Updates rollback*

289 Updates on traditional package-based Linux distributions are prone to errors.  
290 An update usually involves multiple packages, and each package update can fail  
291 in ways that are not trivial to automatically recover from. After a failure on  
292 a package-based update, the limited rollback functionality is not guaranteed to  
293 revert the problem, leading to manual intervention.

294 A robust approach for updates that are capable of reliable rollbacks is called  
295 atomic updates. Atomic updates perform the file operations in a staging area,  
296 and the changes are only committed if the update is successful. When a failure  
297 occurs during an update, the changes are not committed and do not affect the  
298 file system.

299 However the benefits of reliable rollbacks are limited to changes made to the  
300 filesystem. Changes that are not file operations, such as updating the bootloader  
301 are not guaranteed to rollback gracefully.

## 302 **Separation between system and application domains**

- 303 • Related requirements: *Conditional software deployment based on business*  
304 *rules, Configurable access rights to user data and system resource, Con-*  
305 *sistent state across devices, Independent release and update of application*  
306 *domains, Resilience to distribution and update failures, User-driven soft-*  
307 *ware deployment*

308 These requirements are related to separating the base operating system from  
309 application domains in regards to software distribution, software updates, and  
310 execution environment.

311 Separating base operating system from application domains allow product teams  
312 to develop their products with greater independence, and offers more flexibility  
313 on how application domains are deployed, updated and executed.

## 314 **Deployment management**

- 315 • Related requirements: *Conditional software deployment based on business*  
316 *rules, Consistent state across devices, Operator-driven software distribu-*

317            *tion and updates, Protecting the fleet from software deployment issues,*  
318            *Resilience to distribution and update failures, Software inventory*

319 Software distribution is more than a transport layer for packages, it includes  
320 authorization, inventory, and deployment management. The software distribu-  
321 tion infrastructure for traditional tools such as `apt-get` basically consists of static  
322 content providers that were designed to replace the previous method based on  
323 CDs and DVDs.

324 This infrastructure works well for transporting packages over the network, but  
325 it lacks features to implement business rules such as customer, payment, and  
326 hardware profile. On large fleets of operator-driven use cases, the operator need  
327 control over the deployment of updates and new features. It is responsibility of  
328 the operator to run the deployment in conformity to business rules to for exam-  
329 ple schedule a reboot in an appropriate moment, and to divide the deployment  
330 in batches.

331 The main component of a deployment management solutions is usually the back-  
332 end infrastructure that interfaces with agents running on the devices. A com-  
333 mon goal to deployment management is to offer easy and flexible rollout of  
334 software with monitoring of progress which is essential for large fleets.

## 335 Existing systems

### 336 OSTree for base operating system

337 OSTree implements for the base operating system *Immutable software stack* and  
338 *Atomic updates*. It also offers the underlying framework to allow *Separation*  
339 *between system and application domains*.

340 OSTree is a feature-rich deployment and update mechanism for files and directo-  
341 ries in Linux. It offers transactional upgrades and rollback, is capable of replicat-  
342 ing content incrementally over HTTP, support multiple parallel bootable root  
343 filesystems, and have flexible support for multiple branches and repositories.

344 As mentioned earlier, rolling out updates using package management tools such  
345 as `apt-get` is prone to a high degree of variability. Each update involves multiple  
346 packages, and each package update can fail on file operations and on scripts.  
347 Current package management systems have only limited roll back capability(See  
348 [apt-btrfs-snapshot](https://github.com/skorokithakis/apt-btrfs-snapshot)<sup>5</sup>) meaning that a failure during a package update can leave  
349 the system in an unknown state making it challenging to secure and maintain.

350 Failures during an OSTree atomic update are not committed, meaning that  
351 a failed update have no effect on the running system. If an OSTree atomic  
352 update completes successfully but introduces software issues, rolling back to  
353 the previous working version is guaranteed to work.

---

<sup>5</sup><https://github.com/skorokithakis/apt-btrfs-snapshot>

354 However OSTree does not directly address the needs of application domains. For  
355 software distribution and update of application domains we recommend using  
356 either Flatpak or Docker.

## 357 **Flatpak and Docker for applications**

358 Both Flatpak and Docker implement for applications *Immutable software stack*,  
359 *Atomic updates*, and *Separation between system and application domains*. One  
360 requirement that is also addressed by both is *Configurable access rights to user*  
361 *data and system resource*.

362 Both Flatpak and Docker are mature and feature rich solutions for applica-  
363 tion distribution and update. They offer decoupling from the system, give the  
364 application developer greater freedom, give the user greater control, and run  
365 applications insulated from the system and from other applications. These are  
366 advantages when compared to more conventional packaging and distribution  
367 systems such as dpkg and apt-get.

368 Flatpak purposely focuses on user-level applications and services, or in applica-  
369 tions with a GUI, such as the ones to be used on a infotainment system. Flatpak  
370 applications are shipped in bundles named Flatpaks, and it uses libostree under  
371 the hood to provide OSTree efficiency and robustness to application manage-  
372 ment.

373 Docker is instead better suited for non-graphical applications. Docker ships  
374 containers, and it is a good solution for applications that are developed and  
375 deployed as a collection of loosely coupled services. In some cases some sort of  
376 container orchestration is used with Docker, but orchestration is a topic that  
377 goes beyond the scope of this document.

378 Flatpak and Docker can fulfill similar roles for decoupling applications from the  
379 base OS, and there are use cases for both in Apertis. A case-by-case evaluation  
380 needs to be done to find the most suitable mechanism for each application and  
381 service. As examples, for the infotainment system use case Flatpak is better  
382 suited for the applications the user can install and remove. For the building  
383 access control devices Docker is a better fit for headless applications that collect  
384 identity data and controls locking mechanisms.

## 385 **Eclipse hawkBit**

386 Eclipse hawkBit implements *Deployment management*.

387 Eclipse hawkBit is a back-end framework for deployment management of edge  
388 devices. It can manage both the base OS and applications, and it is relatively  
389 agnostic about the kind of applications used. A preliminary investigation of the  
390 feasibility of the integration of the hawkBit-based Bosch Software Innovations  
391 IoT management suite with Apertis has been done with positive outcome.

## 392 **Microsoft Azure IoT Edge**

393 Eclipse hawkBit implements *Deployment management*.

394 Microsoft Azure IoT Edge is a full hosted suite to manage the deployment of  
395 Docker containers on edge devices and it also offer deployment management  
396 capabilities.

397 A preliminary Apertis image with support for Docker containers has been eval-  
398 uated to explore the feasibility of using Apertis with Microsoft Azure IoT Edge.

## 399 **Appstore**

400 An appstore should meet the requirements: *Conditional software deployment*  
401 *based on business rules, Independent release and update of application domains,*  
402 *Protecting the fleet from software deployment issues, Software inventory, User-*  
403 *driven software deployment.* It should also provide support to the high level  
404 feature *Deployment management* or integrate with an external *Deployment man-*  
405 *agement* solution.

406 An appstore is the interface that allow users to browse, buy, install, remove, and  
407 update applications on their devices. Users interact with an appstore remotely  
408 over a web frontend, and locally over an application on the device.

409 The appstore sits at the highest level layer of software distribution and update  
410 and reflects the decisions made for the lower layers. For example the solution  
411 for bundles and for deployment management highly impact the appstore design.

412 As an interface with the user the appstore verifies user credentials, presents the  
413 software catalog, and processes payment. As an interface with the deployment  
414 management layer the appstore queries the software inventory, and issue soft-  
415 ware distribution commands such as install an application on the user device.

416 Unlike an user, the operator is responsible for the health of a fleet of devices,  
417 and an appstore may not be part of the use case. Instead the operator uses an  
418 interface to change device configuration and to control deployment of updates  
419 and new features.

## 420 **Curridge**

421 Curridge is a custom non-upstream solution based on the Magento web com-  
422 merce framework. At the moment Curridge has only been part of demonstra-  
423 tions done by the RBEI team, but Apertis currently ships a component to  
424 interface with it named Frome.

425 Collabora is not aware of the current feature set, but we expect that it is possible  
426 to adapt Curridge to ship Flatpak bundles. However more information is needed  
427 to compare the feature set with the requirements of an appstore.

428 An alternative path is to extend Curridge to interface with external solutions  
429 such as Flathub and hawkBit. This interfacing could allow Curridge to focus  
430 on the appstore user, and offload other tasks such as deployment management  
431 and bundle compatibility to dedicated components.

## 432 **Flathub**

433 Flathub is the upstream appstore for applications distributed via Flatpak.

434 It provides a validated workflow for third-party application authors to [publish](#)  
435 [their work](#)<sup>6</sup>.

436 Applications can be browsed on FlatHub itself or through the on-device appli-  
437 cations for app management, such as GNOME Software or KDE Discover.

438 Flathub does not support payments at the moment, even though there's up-  
439 stream interest in the feature. It does not provide any remote management  
440 solution.

## 441 **Summary of recommendations**

- 442 • Use OSTree for the base operating system for *Immutable software stack*,  
443 *Atomic updates*, and *Updates rollback*.
- 444 • Use Flatpak or Docker for applications for *Immutable software stack*,  
445 *Atomic updates*, *Separation between system and application domains*, and  
446 *Configurable access rights to user data and system resource*.
- 447 • Use Flathub and Docker registry for storage and content delivery systems
- 448 • For operator-driven management, provide integration with hawkBit and  
449 Microsoft Azure IoT Edge
  - 450 – Open point: should Apertis provide a default hawkBit instance for  
451 testing and guidance for product teams?
- 452 • Evaluate the effort to extend Curridge to interface with Flathub and hawk-  
453 Bit.
  - 454 – Open point: Should Curridge handle deployment management or  
455 offload it to other solution such as hawkBit?
- 456 • For user-driven application management, use Flathub on the back-end,  
457 and either adapt GNOME Software or write a custom GUI application on  
458 top of Flatpak for the on-device user interface
  - 459 – Open point: Should Curridge be adapted to interface with Flathub?

## 460 **Reference: System updates and rollback**

461 The [System updates and rollback](#)<sup>7</sup> document contains details about technologies  
462 that are currently being used for software distribution and software update such

<sup>6</sup><https://github.com/flathub/flathub/wiki/App-Submission#how-to-submit-an-app>

<sup>7</sup><https://em.pages.apertis.org/apertis-website/concepts/system-updates-and-rollback/>

<sup>463</sup> as OSTree. Consider reading `system updates` and `rollback` after having read this  
<sup>464</sup> document.