



Release flow and product lines

1	Contents	
2	Debian release processes	2
3	Process towards a release	3
4	Process after release	4
5	Stable repository	4
6	Security repository	4
7	Stable Proposed Updates repository	4
8	Stable Updates repository	5
9	Backports repository	5
10	Debian release flow conclusions	5
11	Linux kernel release flow	6
12	Process towards a release	6
13	Process after a release	7
14	Linux release flow conclusions	7
15	Apertis release flow	7
16	Flow up to a product release	9
17	Development releases (Q4, Q1, Q2, Q3)	9
18	Preview release (Q4)	10
19	Product release (Q1)	10
20	Process after a product release	10
21	Stable Repository	11
22	Security repository	11
23	Updates repository	11
24	Backports repository	11
25	Example images	11
26	Apertis release flow conclusions	12
27	Release flow for the direct downstreams of Apertis	13
28	Guidelines for product development on top of Apertis and its di-	
29	rect downstreams	13
30	Pre-production guidelines	13
31	Post-production support guidelines	15
32	Product guideline conclusions	16
33	Appendix: Change in release strategy	16
34	Appendix: Distribution “freshness”	17
35	Apertis and its direct downstreams are intended as baseline distributions for	
36	further product development, as such it’s important to have a clear definition of	
37	what downstreams further down the chain can expect in terms of releases and	
38	support cycles in order to understand how to best use them in their product	
39	development cycles.	

40 The release cycles of Apertis and its direct downstreams are split up in two big
41 phases: a development phase, containing various development releases followed
42 by a product phase which contains various stable point releases. As it is typical,
43 the development phase is where new features are introduced and prepared, with
44 each development release having only a relatively short support time, while
45 during the product phase the focus is on stability, which comes with a longer
46 support cycle, no new feature and only updates for important bugfixes and
47 security issues.

48 This document sets out to define a well-defined process for both the development
49 and production phases of Apertis and its direct downstreams, while ensuring the
50 software taken from upstreams is recent and well-supported. More specifically
51 this process is trying to balance various trade-offs when integrating from com-
52 munity supported upstreams:

- 53 • support baseline versions that also have community support (to prevent
54 the situation where, for instance, Apertis would need to provide full secu-
55 rity support for the base distribution and/or the Linux kernel);
- 56 • ensure there is a reasonable window for users of Apertis and its direct
57 downstreams to rebase on top of a new on version while the older baseline
58 is still supported;
- 59 • limit the amount of simultaneously supported releases to minimize the
60 overall effort.

61 In all cases it should be noted that support timelines documented here are the
62 expected default timelines: given enough interest particular support cycles can
63 be extended to fit the needs of downstreams.

64 For the Apertis releases there are two important upstream projects that need to
65 be taken into account: the Debian project, which is the main upstream distri-
66 bution for Apertis, and the mainline Linux kernel. These will be further looked
67 at first, including the impact of their release process on generic downstreams
68 before looking at Apertis specifically.

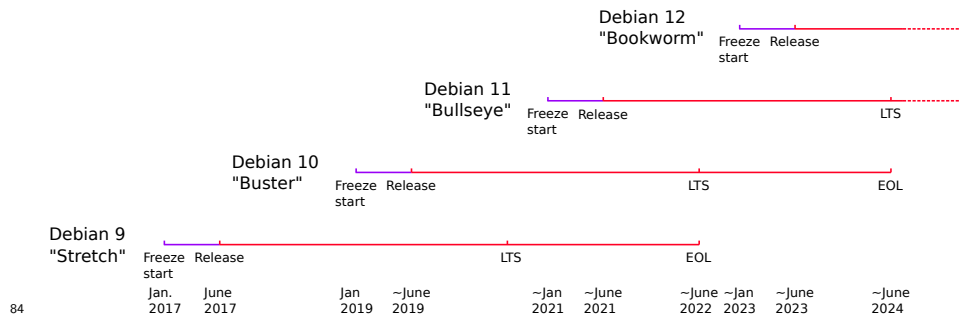
69 Debian release processes

70 Debian aims to do a new major release about every two years. These releases are
71 *not* time-based, but done when “ready” (defined as having no more issues tagged
72 “release-critical”). Even so, the process is well understood and predictable. For
73 more information see the [Debian release statistics](https://wiki.debian.org/DebianReleases#Release_statistics)¹

74 For a downstream there are two important processes to understand. The first
75 one to understand is the process towards a release which impacts when down-
76 stream rebasing should start. The second one being the maintenance process
77 of a stable release, which impacts how to handle security and bugfixes coming
78 from Debian to the downstream.

¹https://wiki.debian.org/DebianReleases#Release_statistics

79 A new stable Debian release is done roughly every two years. Each release gets
 80 3 years of support before it is taken over by the LTS team which provides other
 81 two years of security support before a release enters end of life (EOL). The
 82 following diagram shows the expected timeline for the current Debian release
 83 and the upcoming releases:



85 Process towards a release

86 Debian’s development is done in a suite called `unstable` (code-named `sid`). De-
 87 velopers directly upload packages into this suite. Once updated, packages stay
 88 in the `unstable` suite for some time (typically 10 days) and then they automati-
 89 cally get promoted to the `testing` suite as long as no release-critical bugs were
 90 found (and no other sanity check failed). The `testing` suite has the code-name
 91 of the *next* planned Debian release, at the time of this writing this is `buster`.

92 The idea behind the `unstable` to `testing` progression is to ensure that during
 93 Debian development there is a version available that is shielded from the most
 94 serious regressions and can thus be used by a wider audience for testing and
 95 dogfooding. However among Debian developers it is common to directly run
 96 `unstable` on a day to day basis.

97 To go from the “normal” development to a new release a freeze process is used.
 98 Specifically the `testing` suite is frozen in various stages:

- 99 • transition freeze: no updates that need a collection of packages to transi-
 100 tion into `testing` at once are allowed (e.g. due to ABI breakage);
- 101 • soft freeze: no new packages are allowed into testing anymore;
- 102 • full freeze: only updates for release critical issues are allowed.

103 Typically this process takes around 7 months (plus/minus two months) to com-
 104 plete, with the transition freeze and soft freeze each taking about 1 month while
 105 the full freeze takes the remainder of the time. Even with the `testing` suite being
 106 held in a pretty stable state the final freeze takes this amount of time due to
 107 the sheer size of Debian, due to the big increase in user testing once the freeze
 108 begins and due to all the work that needs to be completed before release, such
 109 as finalising the documentation, installers, etc. The end-result is a new stable
 110 release of a very high-quality Linux distribution.

111 Once a release is done the `stable` suite is updated to refer to the new release,
112 while `testing` is changed to refer to the next version (to be code-named `bullseye`
113 at the time of writing).

114 From the perspective of a downstream distribution such as Apertis it is impor-
115 tant to note that even if during the Debian freeze there will be some amount of
116 outstanding release-critical bugs, only a subset of them will impact the down-
117 streams use-case. As such, if scheduling allows, it is recommended to start
118 rebasing on top of a *next* Debian stable release while Debian itself is in either
119 soft or hard freeze. This has the added benefit that the downstream distribution
120 will already pre-test the upcoming Debian release, with the potential of being
121 able to fix high-priority issues in Debian proper even before its release, thus
122 lowering the delta maintained in the downstream distribution.

123 **Process after release**

124 Once a release has been done, the newly released distribution will follow Debian's
125 stable processes. Debian tends to do point release once every two months to
126 include fixes for the latest security issues and high priority bugs. This process
127 is handled through various different package repositories.

128 **Stable repository**

129 This is the main repository with the full current *released* version of Debian.
130 After release this repository only gets updated when a point releases happens.

131 **Security repository**

132 This repository contains security updates on top of the current point release.
133 The security repositories are managed by the Debian Security team, using their
134 own dedicated infrastructure.

135 As can be expected, security updates are meant to be deployed by users as soon
136 as possible.

137 **Stable Proposed Updates repository**

138 This repository is meant for *proposed* updates to the next point release. The
139 purpose of this repository is to have a way of testing updates before they are
140 included into the next point release.

141 Only packages with issues tagged release-critical will be included in this repos-
142 itory, including both bugfixes and security fixes. Do note that packages with
143 security fixes are immediately published in the security repository for consump-
144 tion by end-user and the inclusion in the proposed update repository is purely
145 so that they can be included as part of the next point release.

146 The set of packages that actually end up in the point release is manually re-
147 viewed and selected by the Debian Stable Release maintainers, thus there is no
148 guarantee that packages in this repository will be part of the next point release.

149 **Stable Updates repository**

150 The `stable-updates` repository exists for updates proposed to stable which are
151 high urgency or time-sensitive and thus should be generally available to users
152 before the next point release. Typical examples of packages landing here are
153 updates to timezone data, virus scanners and high impact/low risk bugfixes.

154 All packages here will also be available in proposed updates and are only allowed
155 into this repository on a case-by-case basis.

156 As with security updates this repository is meant to be used by all the users of
157 a Debian stable release.

158 **Backports repository**

159 The backports repository contains packages taken from the *next* Debian release
160 (specifically from the testing suite) and rebuilt against the current Debian stable
161 release. Backports allow users to upgrade specific interesting packages to newer
162 versions while keeping the remainder of their system running the stable release.

163 However, while backports will have seen a minimal amount of testing, the pack-
164 ages are provided on an as-is basis with no guarantee of stability. As such it's
165 recommended to only cherry-pick the package one needs from this repository.

166 **Debian release flow conclusions**

167 From a purely downstream perspectives there are various interesting aspects in
168 this process.

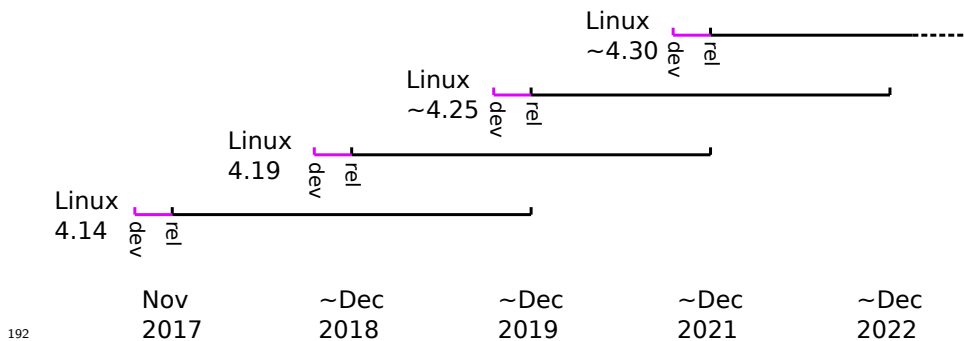
169 In the process going towards a release it's notable that even during the soft and
170 hard freeze periods Debian is already a quite stable baseline as such a rebasing
171 process for an Apertis product release can start when Debian is in freeze as long
172 as there is enough time left before the product release (around 8 to 9 months).

173 After a Debian release there are clear repositories that a downstream should
174 focus upon, namely those in the “stable updates” and “security” repositories, as
175 well as updates included in point releases. The “stable proposed updates” can
176 mostly be ignored on a day to day basis but gives interesting insights in what
177 can be expected from the next point release. Finally the backports repository
178 should in general not be used unless a downstream has a high interest in versions
179 of a package newer than what is available in the stable release. However, in that
180 case extra effort should be put in place to track security issues and other bugfixes
181 for that package as Debian only provides it on a best-effort basis without the
182 usual guarantees.

183 **Linux kernel release flow**

184 Apertis is following the Linux kernel LTS releases to ensure it includes modern
185 features and support for recent hardware. As such it's important to also look
186 at the release flow of the Linux kernel itself and its impact. Linux sees a new
187 major release about every 2 months, which typically is only supported until the
188 next major release happens. However once a year there is a long-term support
189 release which is supported for 2 years.

190 The following diagram shows the expected timelines for the current and next
191 expected Linux long term stable releases.



193 **Process towards a release**

194 The kernel stabilisation process has two big phases: after every release there
195 is a two week *merge window* in which all the various changes lined up by the
196 various subsystem maintainers are pulled in the main tree. At the end of this
197 two-week period the first release-candidate (rc1) is released and the merge win-
198 dows is closed. Afterwards only patches fixing bugs and security issues will be
199 integrated, with a new release candidate coming out every week.

200 Typically 7 or 8 release candidates will be released in each cycle followed by a
201 final release, which means a new stable version of Linux release every 9 to 10
202 weeks.

203 **Process after a release**

204 After each Linux release further maintenance is done in the stable git tree. These
205 trees will only get further bug and security fixes, with releases being done on
206 an as-needed basis. The support time depends on the specific release which fall
207 in two categories:

- 208 • normal release, only supported until the next release;
- 209 • long term release, typically supported for two years.

210 Currently each last kernel release of the year is expected to be a long term
211 release, supported for at least two years after release. Specific releases may

212 be provided with longer upstream support depending on industry interest. For
213 example the 4.4 kernel is getting a total of 6 years of support mainly due to
214 interest from Android. Similarly the Linux 3.16 kernel is also getting a total of
215 6 years of support as that was the kernel used by the Debian Jessie release. For
216 Linux 4.9 a similar longer cycle is to be expected as that was used in Debian
217 Stretch, however that hasn't been made official thus far and at the time of this
218 writing Linux 4.9 will go EOL in January 2019.

219 **Linux release flow conclusions**

220 For usage in Apertis product releases only long term releases are suitable. As
221 there is a yearly LTS release of Linux with only a 2 year support cycle, it is
222 recommended to ensure each yearly release of Apertis has the latest Linux LTS
223 support. This ensures both support for recent hardware as well as having a
224 reasonable security support window.

225 If downstream projects require a longer support period for a specific kernel
226 release then it's recommended to align with other long term support efforts
227 instead, depending on requirements.

228 **Apertis release flow**

229 The overall goal is for Apertis to do a yearly product release. These releases
230 will be named after the year of the stable release, in other words the product
231 release targetted at 2020 will be given major version 2020. A product release
232 is intended to both be based on the most recent mainline kernel LTS release
233 and the current Debian stable release. Since Debian releases roughly once every
234 two years, that means that there will typically be two Apertis product releases
235 based on a single Debian stable release. With Linux doing an LTS release on a
236 yearly basis, each Apertis product release will be based on a different (and then
237 current) Linux kernel release.

238 To move to a yearly product release cycle the recommendation is to keep the
239 current quarterly releases, but rather than treating all the releases equally as
240 is today have releases with specific purposes depending on where in the yearly
241 cycle the releases are for a specific product release.

242 The final product release is planned to occur at the end of Q1 every year, both
243 to avoid the impact of the major holiday periods (Christmas/new-year and
244 european summer) as well as releasing close to the Linux kernel LTS release to
245 maximize the use of its support cycle. Once a product release is published, it
246 will continue to get updates for bug and security fixes, with a point release every
247 quarter for the whole duration of the support period.

248 The standard support period for Apertis is 7 quarters. In other words from the
249 initial release at the end of Q1 until the end of the *next* year.

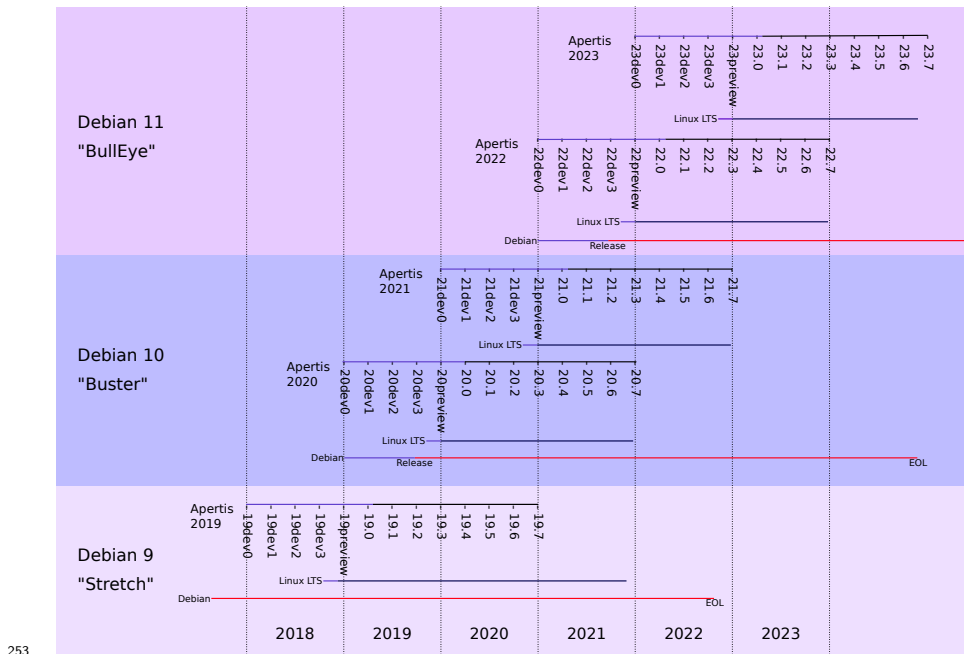
250 The various types of releases per quarter (without point releases) would be:

Quarter	Release type	Support
Q4	Release N-1 Preview	Limited, until the Q1 product release
Q4	Release N Development	Limited, until the Q1 development release
Q1	Release N-1 Product	Full support, until 1.75 years after release
Q1	Release N Development	Limited, until the Q2 development release
Q2	Release N Development	Limited, until the Q3 development release
Q3	Release N Development	Limited, until the Q4 development release
Q4	Release N Preview	Limited, until the Q1 product release
Q4	Release N+1 Development	Limited, until the Q1 development release
Q1	Release N Product	Full support, until 1.75 years after release
Q1	Release N+1 Development	Limited, until the Q2 development release

251 For each quarter the releases would be (with some examples):

Quarter	N-2	N-1	N	N+1	v2020	v2021	v2022	v2023
Q4	.3	pre	dev0		v2020.3	v2021.pre	v2022.dev0	
Q1	.4	.0	dev1		v2020.4	v2021.0	v2022.dev1	
Q2	.5	.1	dev2		v2020.5	v2021.1	v2022.dev2	
Q3	.6	.2	dev3		v2020.6	v2021.2	v2022.dev3	
Q4	.7	.3	pre	dev0	v2020.7	v2021.3	v2022.pre	v2023.dev0
Q1		.4	.0	dev1		v2021.4	v2022.0	v2023.dev1
Q2		.5	.1	dev2		v2021.5	v2022.1	v2023.dev2
Q3		.6	.2	dev3		v2021.6	v2022.2	v2023.dev3

252 The following diagram shows how this would look for Apertis releases up to 2023:



253
 254 Further details about the various types of release will be given in the following
 255 sections.

256 **Flow up to a product release**

257 The main flow towards a quarterly release will remain the same as it now, which
 258 is documented on the [Apertis Release schedule²](#) page. However, depending on
 259 the type of release the focus may differ.

260 **Development releases (Q4, Q1, Q2, Q3)**

261 For a development release, everything is allowed as the main focus is develop-
 262 ment. These can include bigger changes to the infrastructure as well as to the
 263 delivered software stack. At the end of every quarter there is an Apertis devel-
 264 opment release: this ensures that there can be ongoing development of the
 265 distribution even if the preparation for the next product release has entered a
 266 stabilisation phase.

267 Rebased on the upcoming stable version of Debian can only be done as part of
 268 a development release. The rebase can start in a quarter as soon as Debian hits
 269 the soft freeze stage.

270 Development releases are versioned as `development number`, with numbering start-
 271 ing from 0. The version of the first development release for the 2020 product
 272 release would be `Apertis 2020 development 0` or optionally shortened to `v2020dev0`.

²<https://martyn.pages.apertis.org/apertis-website/policies/releases/>

273 **Preview release (Q4)**

274 The goal of a preview release is to provide a preview of what will be the final
275 product release for further testing and validation by downstreams. As such a
276 preview release should achieve a high level of stability: this means that during a
277 preview release cycle only non-disruptive software or infrastructure updates will
278 be allowed. Similarly, new features can only be introduced if they pose a low
279 risk on existing functionality and do not have an impact on the overall platform
280 stability.

281 During the preparation of a preview release extra focus should be given to
282 bugfixing and testing.

283 One important exception to the above considerations is to be made: preview
284 releases should be released with the new Linux kernel LTS (either the final
285 release or a release candidate) to ensure the product release will be done with
286 the most recent LTS Linux kernel, maximising the overlap with the 2 year stable
287 support period offered.

288 As there is only one preview release for each product release, the version is the
289 major product version followed by preview. For example `Apertis 2020 preview`,
290 which can be shortened to `v2020pre`.

291 **Product release (Q1)**

292 As can be expected the focus of the product release quarter is to deliver a high-
293 quality release which can be supported for a longer period. For this release only
294 security fixes, bugfixes and updates to the stable kernel release or updates from
295 the Debian stable release.

296 New features should not be included during this quarter as it's unlikely there
297 will be enough time for them to fully mature.

298 The major version of the product release is simply the year in which the release
299 is to be done. The minor version starts at 0 and is increased for each later point
300 release. This means the initial product release for 2020 would be `Apertis 2020.0`
301 or simply shortened to `v2020.0`.

302 **Process after a product release**

303 After a release has been done, for each of them there is an expected support life
304 depending on the type of release as outlined above.

305 For non-product release any post-release updates will directly go into the main
306 repository for that specific release. For product releases a setup similar to
307 Debian is to be used to stage updates before a new point release is done. The
308 repositories used by Apertis are outlined in the following sections.

309 **Stable Repository**

310 This is the main repository with the full *released* version. This repository only
311 gets updated at point releases.

312 Point release will be done every three months. All downstreams are expected
313 to pull directly from the stable repository.

314 **Security repository**

315 For security issues a dedicated security repository is used. This repository is
316 only used with updated packages including security fixes.

317 This repository should be pulled directly by all downstreams and any updates
318 rolled out at high priority. Updates from the Debian security repository will
319 always be included in this repository.

320 **Updates repository**

321 This repository includes updated packages to be included in the next Apertis
322 point release. Only packages with high priority bugfixes are allowed into this
323 repository. Updated packages from the Debian stable-updates and point releases
324 will be automatically included.

325 Downstreams are recommended to include this repository but it's not manda-
326 tory.

327 **Backports repository**

328 This repository has backports of packages which are of special interest to down-
329 streams but where not suitable for inclusion into the product release.

330 Unless specific agreements have been made, the packages available in this repos-
331 itory are for experimentation use only and are not supported as part of the
332 produce release.

333 **Example images**

334 Apertis includes a big collection of packages which can be used in a variety
335 of system use-cases. As it is impossible to test all combinations of packages,
336 Apertis provides a set of example images for each type of system which has been
337 validated by the Apertis project. While other use-cases can be supported there
338 cannot be a strict guarantee that Apertis is fit for purpose for those as it hasn't
339 been validated in that situation.

340 Furthermore, as these Apertis images are meant as examples for product use-
341 case they can include demonstration quality software, which is not intended nor
342 has been validated to form the basis of a product.

343 To clarify what is expected to be supported for each Apertis product release
344 documentation will be provided to explain what the scope of each example
345 image is, which use-cases it validates and which part of the software stack are
346 fully supported for product usage.

347 A description of the expected release artifacts can be found on the [images](#)³ page.

348 **Apertis release flow conclusions**

349 The above sections outline a process for Apertis to both generate and support
350 yearly product releases. They ensure that Apertis releases are always based on
351 recent but mature upstream software. Each product release will include the
352 very latest Linux LTS kernel as well as the current Debian stable release.

353 What was intentionally not covered is how to manage forward looking devel-
354 opment during the non-development cycles as this is separate from the release
355 flow. However there is no real blocker for doing development not intended to
356 be part of the product release, deliverables can be delivered for instance via the
357 backports repository or by other means to be defined further.

358 Combining all the various types of releases, for a single product release 13 dif-
359 ferent releases will be done. For example for Apertis 2021 the schedule looks
360 like this:

Quarter	Release	Name	Type
2019Q4	Apertis 2021 development 0	v2021dev0	development
2020Q1	Apertis 2021 development 1	v2021dev1	development
2020Q2	Apertis 2021 development 2	v2021dev2	development
2020Q3	Apertis 2021 development 3	v2021dev3	development
2020Q4	Apertis 2021 preview	v2021pre	preview
2021Q1	Apertis 2021.0	v2021.0	stable release
2021Q2	Apertis 2021.1	v2021.1	stable point release
2021Q3	Apertis 2021.2	v2021.2	stable point release
2021Q4	Apertis 2021.3	v2021.3	stable point release
2022Q1	Apertis 2021.4	v2021.4	stable point release
2022Q2	Apertis 2021.5	v2021.5	stable point release
2022Q3	Apertis 2021.6	v2021.6	stable point release
2022Q4	Apertis 2021.7	v2021.7	stable point release

361 For projects using Apertis (or its direct downstreams) given this schedule there
362 is a rebase window of a year to move to the newer version. Starting from when
363 the preview release of the new version is done (for instance, v2022pre in 2021Q4)
364 until the .7 stable point release of the old version (for instance, v2021.7), which
365 is end of Q4 to end of the next Q4.

³<https://martyn.pages.apertis.org/apertis-website/policies/images/>

366 **Release flow for the direct downstreams of Aper-** 367 **tis**

368 The release cycle of the direct downstreams of Apertis is expected to follow the
369 same process as that of Apertis. In other words throughout the year the direct
370 downstreams of will do two development releases based on top of the Apertis
371 development release, one preview release and a final product release.

372 It is expected that the respective direct downstream releases will be done within
373 a month from the quarterly Apertis release and will be made available to the
374 downstreams further down the chain in that time frame.

375 For an direct downstream product release it is expected that in addition to
376 the `stable` repository the `updates` and especially `security` repository are tracked
377 closely, with any updates from Apertis being made available in the direct down-
378 stream within a week. A similar time-frame is expected for Apertis point re-
379 leases.

380 **Guidelines for product development on top of** 381 **Apertis and its direct downstreams**

382 To make the best use of Apertis in product development it is recommended to
383 take the release timelines of Apertis and its direct downstreams into account
384 when creating a product release roadmap. Since Apertis and its direct down-
385 streams have a cadence of a new release once a year, users are driven to the same
386 cadence by default. Given that the overlap of stable releases for two subsequent
387 product releases is three quarters, users have a full year to rebase their work
388 once the preview release for the next product release is published.

389 The details about the use of Apertis and its direct downstreams will depend
390 on the phase of the project, in particular whether it is in the pre-production
391 development phase or in the post-production support phase.

392 **Pre-production guidelines**

393 The pre-production phase is the phase before a new major version of software
394 goes into production. This can either before the product starts its production
395 or when a new major software update is planned to be rolled out to products
396 already in the field.

397 Typically this phase consists of a period of heavy development (potentially in-
398 terleaved with short stabilisation periods), followed by a potentially longer final
399 stabilisation period before entering production.

400 For the final stabilisation phase, the baseline used for Apertis and its direct
401 downstreams should be focused on stability. This means either a preview or the
402 current product release should be used. Care should be taken to ensure that

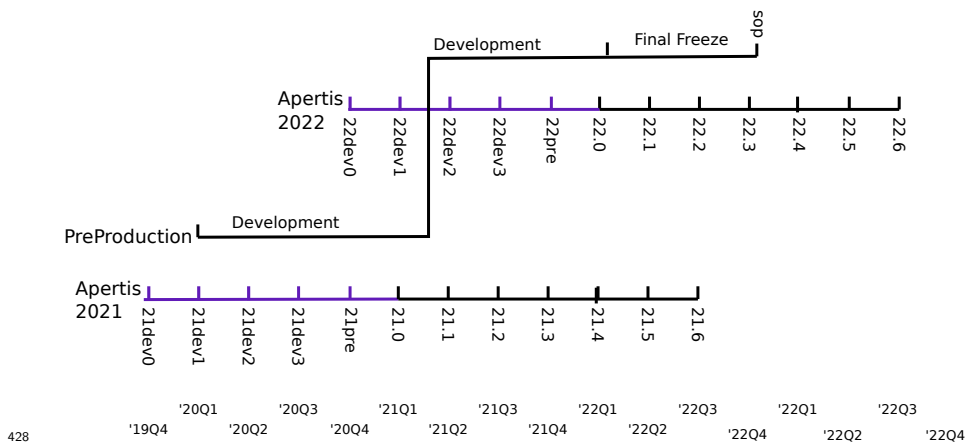
403 there is still a reasonable window of support for the baseline distribution when
 404 production is planned to start. After production has started the guidelines for
 405 post-production support should be taken into account.

406 For the initial development phase there are two main options:

- 407 • follow the development releases of Apertis or its direct downstreams;
- 408 • follow the product releases of Apertis or its direct downstreams (switching
 409 at the preview stage).

410 The first option allows the product development to use the very latest Apertis
 411 features and developments on top of the most recent software baseline which
 412 will form the basis of the future product release of Apertis or of its direct down-
 413 stream, while the second option provides a more stable, but older, baseline al-
 414 lowing the product team to focus on their own software stack. These approaches
 415 can be mixed, for example by starting out early product development on the
 416 current Apertis (or one of its direct downstreams) development release to take
 417 advantage of more recent features, but following that baseline when it becomes
 418 the product release instead of moving to the next cycle of development releases.
 419 By mixing the approaches in this way the product team has the flexibility of
 420 choosing the baseline that best fits their priorities at any given time.

421 The following diagram shows an example of such a mixed development: develop-
 422 ment starts on top of the then current Apertis development release and is
 423 rebased early onto the next development versions of Apertis such that the prod-
 424 ucts final 9 month freeze before SOP coincides with the product-line release
 425 of the Apertis it's based on. If a product is based on a direct downstream of
 426 Apertis, then the chart would be nearly identical, replacing the Apertis labels
 427 with the name of the direct downstream.



429 **Post-production support guidelines**

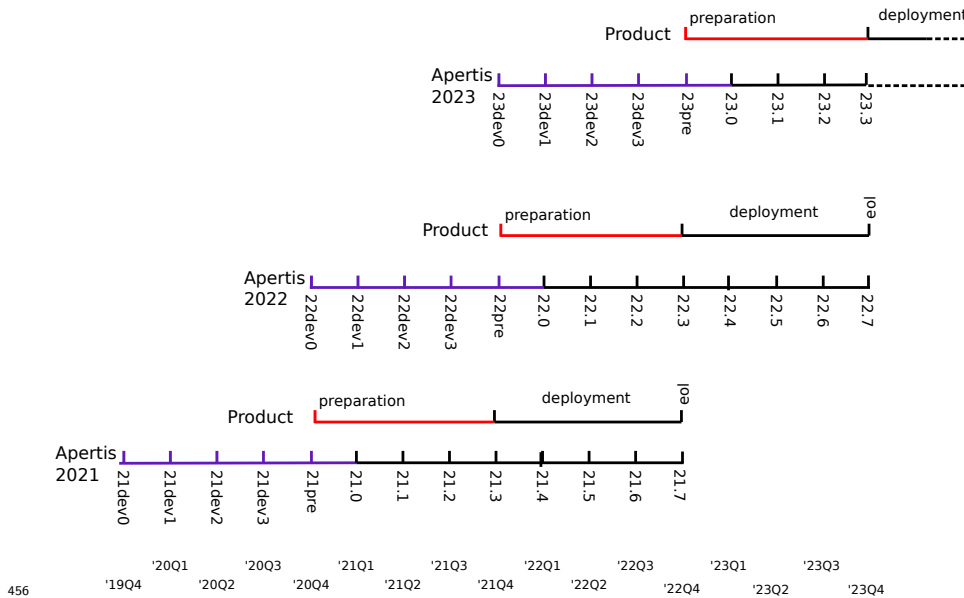
430 The post production support phase is the phase where the product is out in the
431 market and any software updates are primarily done for the purpose of fixing
432 bug and security issues.

433 In this phase it's assumed that the release into the field has been done based on
434 a product release of Apertis or of one of its direct downstreams. The product
435 team is expected to track Apertis security fixes as they become available through
436 the security repository of Apertis or its direct downstream as well as new point
437 releases (containing both security and bug fixes).

438 It is up to the product team to further select and test these updates for their
439 product and schedule software updates that work best for their schedule, with
440 the recommendation to update devices in the field as quickly as possible espe-
441 cially in the case of high impact security fixes.

442 When a new release of Apertis or of its direct downstream comes out the prod-
443 uct team is expected to update to this new version before the support for the
444 previous Apertis release comes to an end. It is typically recommended to start
445 the work to rebase on the new version of Apertis or of its direct downstream
446 when the preview release becomes available as the focus for Apertis is very much
447 on stability at that point.

448 The following diagram shows an example of such a flow, where the product
449 begins the preparation for deploying an update based on the new Apertis version
450 at the time of the preview release and targets deployment in the field when the
451 old Apertis release support ends, which gives a window of a full year to do the
452 necessary preparation and validation before deploying an update into the field.
453 If a product is based on a direct downstream of Apertis, then the chart would
454 be nearly identical, replacing the Apertis labels with the name of the direct
455 downstream.



457 **Product guideline conclusions**

458 As can be seen in the previous sections Apertis and its direct downstreams try
 459 to give product teams flexibility to use Apertis as they see fit for their needs
 460 within the constraints imposed by the support timelines.

461 It should be noted however that these timelines are not set in stone: if there are
 462 business cases for having specific releases of Apertis or of its direct downstreams
 463 supported for an extended period then this is in principle possible. However it
 464 should be noted that Apertis and its direct downstreams in turn have constraints
 465 from its upstreams to be able to rely on community support, which may limit
 466 the amount of support that can be provided.

467 **Appendix: Change in release strategy**

468 This release flow concept is a departure from the initial concept for Apertis,
 469 which would rebase on every new Ubuntu releases (once every 6 months). This
 470 resulted in two releases for every Ubuntu version, where in one quarter the
 471 project would rebase on the new Ubuntu release, and in the following quarter
 472 it would continue on that baseline with further updates and improvements.

473 Conceptually there are two big changes with this new concept:

- 474 • switch to a longer supported distribution release;
- 475 • switch from Ubuntu as a baseline to Debian.

476 When the initial concept was set out, Ubuntu would support non-LTS releases

477 for 18 month (one year after the *next* Ubuntu release). Currently however
478 the support for non-TLS releases is only 9 months (3 months after the *next*
479 Ubuntu) release), which is simply too short for supporting product usage even
480 if the product has a very aggressive timeline.

481 This means that to fit the trade-offs/constraints mentioned in the introduction
482 a switch has to be made to releases with a longer support term, which in both
483 Ubuntu and Debian cases are released every 2 years, with 5 years of support.

484 The rationale for switching from Ubuntu as a baseline to Debian has been out-
485 lined in more detailed in the “[The case for moving to Debian stretch or Ubuntu](#)
486 [18.04](#)”⁴ concept document.

487 **Appendix: Distribution “freshness”**

488 A side-effect of the switch to distributions with a longer support cycle is that
489 there are fewer updates on top of the baseline. As such the software available
490 in the distribution can be older than the latest and greatest from upstream or
491 more recent distribution releases (for instance, older than what it is available
492 in normal Ubuntu releases), which also means that not all the latest features
493 might be available.

494 This is a consequence from the trade-offs that are being made in the release
495 process to best serve users of Apertis and its direct downstreams, stability and
496 community support are preferred over having the very latests features. In case
497 newer features are required this can either be handled via the backports mech-
498 anism if only needed for specific users or, in case of a feature useful to most
499 users, including a newer version in the next release of Apertis or of its direct
500 downstreams can be considered.

501 A practical example of this happening is the way the Linux kernel is handled, as
502 support for recent hardware devices is considered important for a wide variety
503 of users (especially during the early product phases). However this does mean
504 a reduced community kernel support timeline when compared to a distribution
505 kernel, so in situations where an update is considered, care should be taken to
506 evaluate the trade-offs with respect to effort costs.

507 Overall, with this release flow the latency for new updates to components from
508 a newer distribution is at most two years. This is under the assumption that
509 users looking for newer features are still in early development and are using the
510 preview releases of Apertis or of its direct downstreams and at that stage not
511 yet the product release. Generally this is seen as a reasonable trade-off for most
512 components.

⁴<https://martyn.pages.apertis.org/apertis-website/architecture/case-for-moving-to-debian/>