



Release flow and product lines

1	Contents	
2	Debian release processes	3
3	Process towards a release	4
4	Process after release	5
5	Stable repository	5
6	Security repository	5
7	Stable Proposed Updates repository	6
8	Stable Updates repository	6
9	Backports repository	6
10	Debian release flow conclusions	6
11	Linux kernel release flow	7
12	Process towards a release	7
13	Process after a release	8
14	Linux release flow conclusions	8
15	Apertis release flow	8
16	Flow up to a product release	10
17	Development releases (Q4, Q1, Q2, Q3)	11
18	Preview release (Q4)	11
19	Product release (Q1)	11
20	Process after a product release	12
21	Stable Repository	12
22	Security repository	13
23	Updates repository	13
24	Backports repository	13
25	Dependencies between these repositories	14
26	Example images	14
27	Apertis release flow conclusions	14
28	Release flow for the direct downstreams of Apertis	15
29	Guidelines for product development on top of Apertis and its di-	
30	rect downstreams	16
31	Pre-production guidelines	16
32	Post-production support guidelines	17
33	Product guideline conclusions	18
34	Appendix: Change in release strategy	19
35	Appendix: Distribution “freshness”	19
36	Appendix: Frequently Asked Questions	20
37	What is the effort required to move to a new product release?	20
38	How often security fixes are made available to users?	20
39	Do packages get updated in a published development/preview release?	21

40	Do downstream distributions need to perform a folding?	21
41	Do downstream distributions need to perform a branching?	22

42 Apertis and its direct downstreams are intended as baseline distributions for
 43 further product development, as such it's important to have a clear definition of
 44 what downstreams further down the chain can expect in terms of releases and
 45 support cycles in order to understand how to best use them in their product
 46 development cycles.

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

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

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

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

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

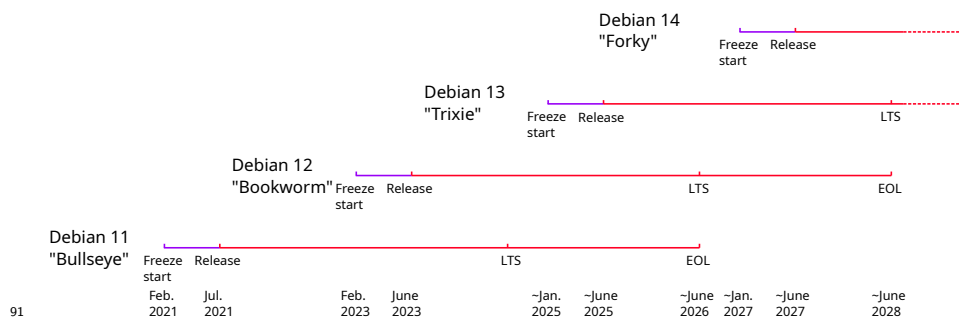
76 Debian release processes

77 Debian aims to do a new major release about every two years. These releases are
 78 *not* time-based, but done when “ready”(defined as having no more issues tagged

79 “release-critical”). Even so, the process is well understood and predictable. For
 80 more information see the [Debian release statistics](https://wiki.debian.org/DebianReleases#Release_statistics)¹

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

86 A new stable Debian release is done roughly every two years. Each release gets
 87 3 years of support before it is taken over by the LTS team which provides other
 88 two years of security support before a release enters end of life (EOL). The
 89 following diagram shows the expected timeline for the current Debian release
 90 and the upcoming releases:



92 Process towards a release

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

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

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

- transition freeze: no updates that need a collection of packages to transi-
 tion into `testing` at once are allowed (e.g. due to ABI breakage);
- soft freeze: no new packages are allowed into testing anymore;

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

109 • full freeze: only updates for release critical issues are allowed.

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

118 Once a release is done the `stable` suite is updated to refer to the new release,
119 while `testing` is changed to refer to the next version (to be code-named `bookworm`
120 at the time of writing).

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

130 **Process after release**

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

135 **Stable repository**

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

138 **Security repository**

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

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

144 **Stable Proposed Updates repository**

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

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

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

156 **Stable Updates repository**

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

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

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

165 **Backports repository**

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

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

173 **Debian release flow conclusions**

174 From a purely downstream perspectives there are various interesting aspects in
175 this process.

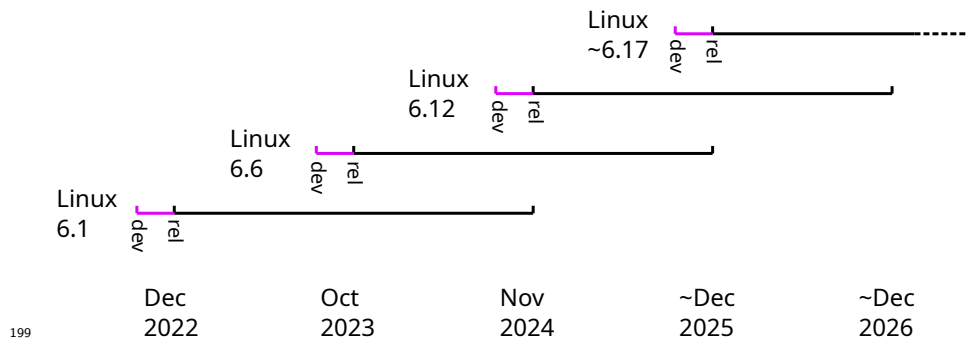
176 In the process going towards a release it's notable that even during the soft and
177 hard freeze periods Debian is already a quite stable baseline as such a rebasing
178 process for an Apertis product release can start when Debian is in freeze as long
179 as there is enough time left before the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

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

Linux kernel release flow

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

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



Process towards a release

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

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

210 **Process after a release**

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

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

217 Currently each last kernel release of the year is expected to be a long term
218 release, supported for at least two years after release. Specific releases may be
219 provided with longer upstream support depending on industry interest. For
220 example the 4.4 kernel is getting a total of 6 years of support mainly due to
221 interest from Android. Similarly the Linux 3.16 kernel is also getting a total of
222 6 years of support as that was the kernel used by the Debian Jessie release. For
223 Linux 4.9 a similar longer cycle is to be expected as that was used in Debian
224 Stretch, however that hasn't been made official thus far and at the time of this
225 writing Linux 4.9 will go EOL in January 2019.

226 **Linux release flow conclusions**

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

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

235 **Apertis release flow**

236 The overall goal is for Apertis to do a yearly product release. These releases
237 will be named after the year of the stable release, in other words the product
238 release targeted at 2024 will be given major version 2024. A product release
239 is intended to both be based on the most recent mainline kernel LTS release
240 and the current Debian stable release. Since Debian releases roughly once every
241 two years, that means that there will typically be two Apertis product releases
242 based on a single Debian stable release. With Linux doing an LTS release on a
243 yearly basis, each Apertis product release will be based on a different (and then
244 current) Linux kernel release.

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

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

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

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

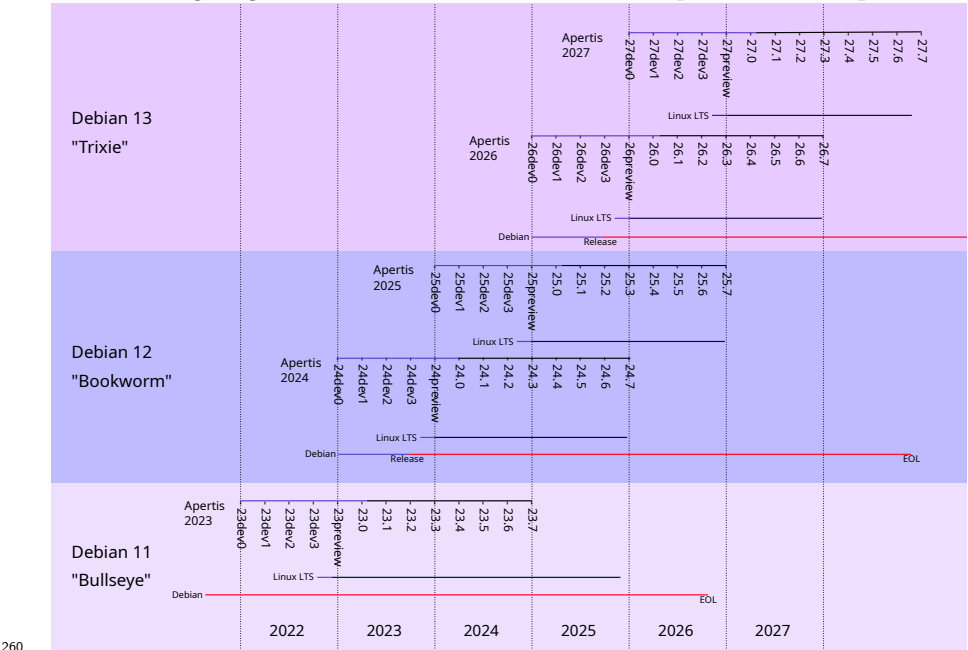
Quarter	Release type	Support
Q4	Release N-1 Preview	Limited, until the Q1 product release
Q4	Release N Development	Limited, until the Q1 development release
Q1	Release N-1 Product	Full support, until 1.75 years after release
Q1	Release N Development	Limited, until the Q2 development release
Q2	Release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

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

Quarter	N-2	N-1	N	N+1	N+2	N+3	v2023	v2024	v2025	v2026	v2027
Q1	.4	.0	dev1				v2023.0	v2024.dev1			
Q2	.5	.1	dev2				v2023.1	v2024.dev2			
Q3	.6	.2	dev3				v2023.2	v2024.dev3			
Q4	.7	.3	pre	dev0			v2023.3	v2024.pre	v2025.dev0		
Q1		.4	.0	dev1			v2023.4	v2024.0	v2025.dev1		
Q2		.5	.1	dev2			v2023.5	v2024.1	v2025.dev2		
Q3		.6	.2	dev3			v2023.6	v2024.2	v2025.dev3		
Q4		.7	.3	pre	dev0		v2023.7	v2024.3	v2025.pre	v2026.dev0	
Q1			.4	.0	dev1			v2024.4	v2025.0	v2026.dev1	
Q2			.5	.1	dev2			v2024.5	v2025.1	v2026.dev2	
Q3			.6	.2	dev3			v2024.6	v2025.2	v2026.dev3	
Q4			.7	.3	pre	dev0		v2024.7	v2025.3	v2026.pre	v2027
Q1				.4	.0	dev1			v2025.4	v2026.0	v2027
Q2				.5	.1	dev2			v2025.5	v2026.1	v2027
Q3				.6	.2	dev3			v2025.6	v2026.2	v2027
Q4				.7	.3	pre			v2025.7	v2026.3	v2027
Q1					.4	.0				v2026.4	v2027

Quarter	N-2	N-1	N	N+1	N+2	N+3	v2023	v2024	v2025	v2026	v2027
Q2					.5	.1				v2026.5	v2027
Q3					.6	.2				v2026.6	v2027
Q4					.7	.3				v2026.7	v2027
Q1						.4					v2027
Q2						.5					v2027
Q3						.6					v2027
Q4						.7					v2027

259 The following diagram shows how this would look for Apertis releases up to 2027:



261 Further details about the various types of release will be given in the following
 262 sections.

263 Flow up to a product release

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

²<https://apertis-website-0b3586.pages.apertis.org/policies/releases/>

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

268 For a development release, everything is allowed as the main focus is develop-
269 ment. These can include bigger changes to the infrastructure as well as to the
270 delivered software stack. At the end of every quarter there is an Apertis de-
271 velopment release: this ensures that there can be ongoing development of the
272 distribution even if the preparation for the next product release has entered a
273 stabilisation phase.

274 Rebasing on the upcoming stable version of Debian can only be done as part of
275 a development release. The rebase can start in a quarter as soon as Debian hits
276 the soft freeze stage.

277 Development releases are versioned as `development number`, with numbering start-
278 ing from 0. The version of the first development release for the 2024 product
279 release would be `Apertis 2024 development 0` or optionally shortened to `v2024dev0`.

280 **Preview release (Q4)**

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

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

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

295 As there is only one preview release for each product release, the version is the
296 major product version followed by preview. For example `Apertis 2024 preview`,
297 which can be shortened to `v2024pre`.

298 **Product release (Q1)**

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

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

305 The major version of the product release is simply the year in which the release
306 is to be done. The minor version starts at 0 and is increased for each later point
307 release. This means the initial product release for 2024 would be `Apertis 2024.0`
308 or simply shortened to `v2024.0`.

309 Process after a product release

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

312 For non-product releases any post-release updates will directly go into the main
313 repository for that specific release. Only fixes to high-impact issues will be
314 published for non-product releases, everything else will only be available in the
315 next release.

316 For product releases a setup similar to Debian is to be used to stage updates
317 before a new point release is done. The repositories used by Apertis are outlined
318 in the following sections.

319 Every quarter a release cycle for every supported release is started with the goal
320 of publishing a new point release. Before the actual point release is published a
321 set of intermediate steps are performed to ensure a reliable process:

- 322 • Soft Feature Freeze: From this point no new features are allowed to the
323 release
- 324 • Hard Feature Freeze / Soft Code Freeze: From this point only bug fixing
325 is allowed, staged updates are folded into the main repository
- 326 • Release Candidate / Hard Code Freeze: From this point no changes are
327 allowed, RC is published for testing
- 328 • Release: Point release is published

329 The last point release is a special case since after three months the staged
330 updates will get folded but no additional point release is published. The overall
331 support period of a product release is thus two years from the `.0` release.

332 Stable Repository

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

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

337 For instance, in Apertis v2024 this maps to:

- 338 • the `apertis/v2024` git branch in the [packaging repositories](https://gitlab.apertis.org/pkg)³
- 339 • the `apertis:v2024:{target,development,sdk,non-free}` OBS repositories
- 340 • the deb `https://repositories.apertis.org/apertis/ v2024 target develop-`
341 `ment sdk non-free` APT source

³<https://gitlab.apertis.org/pkg>

342 Once a point release is published, the updates staged in the repositories de-
343 scribed below get folded in this repository to make them generally available.

344 **Security repository**

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

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

350 For instance, in Apertis v2024 this maps to:

- 351 • the `apertis/v2024-security` git branch in the [packaging repositories](https://gitlab.apertis.org/pkg)⁴
- 352 • the `apertis:v2024:security:{target,development,sdk,non-free}` OBS repos-
353 itories
- 354 • the deb `https://repositories.apertis.org/apertis/ v2024-security target`
355 `development sdk non-free` APT source

356 **Updates repository**

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

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

363 For instance, in Apertis v2024 this maps to:

- 364 • the `apertis/v2024-updates` git branch in the [packaging repositories](https://gitlab.apertis.org/pkg)⁵
- 365 • the `apertis:v2024:updates:{target,development,sdk,non-free}` OBS repos-
366 itories
- 367 • the deb `https://repositories.apertis.org/apertis/ v2024-updates target`
368 `development sdk non-free` APT source

369 **Backports repository**

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

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

375 For instance, in Apertis v2024 this maps to:

⁴<https://gitlab.apertis.org/pkg>

⁵<https://gitlab.apertis.org/pkg>

- the `apertis/v2024-backports` git branch in the [packaging repositories](#)⁶
- the `apertis:v2024:backports:{target,development,sdk,non-free}` OBS repositories
- the deb <https://repositories.apertis.org/apertis/v2024-backports/target/development/sdk/non-free> APT source

Dependencies between these repositories

The main repository is standalone, that means it doesn't depend on any other repository (neither `security` nor `updates` nor `backports`). The `security` repository depends only on the main repository, while the `updates` repository depends on both main and `security` repositories. The `backports` repository depends on all other repositories (`main`, `security` and `updates`).

Example images

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

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

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

A description of the expected release artifacts can be found on the [images](#)⁷ page.

Apertis release flow conclusions

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

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

⁶<https://gitlab.apertis.org/pkg>

⁷<https://apertis-website-0b3586.pages.apertis.org/policies/images/>

Combining all the various types of releases, for a single product release 13 different releases will be done. For example for Apertis 2024 the schedule looks like this:

Quarter	Release	Name	Type
2022Q4	Apertis 2024 development 0	v2024dev0	development
2023Q1	Apertis 2024 development 1	v2024dev1	development
2023Q2	Apertis 2024 development 2	v2024dev2	development
2023Q3	Apertis 2024 development 3	v2024dev3	development
2023Q4	Apertis 2024 preview	v2024pre	preview
2024Q1	Apertis 2024.0	v2024.0	stable release
2024Q2	Apertis 2024.1	v2024.1	stable point release
2024Q3	Apertis 2024.2	v2024.2	stable point release
2024Q4	Apertis 2024.3	v2024.3	stable point release
2025Q1	Apertis 2024.4	v2024.4	stable point release
2025Q2	Apertis 2024.5	v2024.5	stable point release
2025Q3	Apertis 2024.6	v2024.6	stable point release
2025Q4	Apertis 2024.7	v2024.7	stable point release
2026Q1			end of support for v2024

For projects using Apertis (or its direct downstreams) given this schedule there is a rebase window of a year to move to the newer version. Starting from when the preview release of the new version is done (for instance, v2025pre in 2024Q4) until the .7 stable point release of the old version (for instance, v2024.7), which is end of Q4 to end of the next Q4.

Release flow for the direct downstreams of Apertis

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

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

For an direct downstream product release it is expected that in addition to the stable repository the updates and especially security repository are tracked closely, with any updates from Apertis being made available in the direct downstream within a week. A similar time-frame is expected for Apertis point releases.

434 Since Apertis will perform the folding of `updates` and `security` before each re-
435 lease, downstreams will get packages updates in the main repositories during
436 the month previous to the release. This will make the folding process for down-
437 streams simpler, focused only in the deltas from Apertis they carry.

438 **Guidelines for product development on top of** 439 **Apertis and its direct downstreams**

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

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

450 **Pre-production guidelines**

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

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

458 For the final stabilisation phase, the baseline used for Apertis and its direct
459 downstreams should be focused on stability. This means either a preview or the
460 current product release should be used. Care should be taken to ensure that
461 there is still a reasonable window of support for the baseline distribution when
462 production is planned to start. After production has started the guidelines for
463 post-production support should be taken into account.

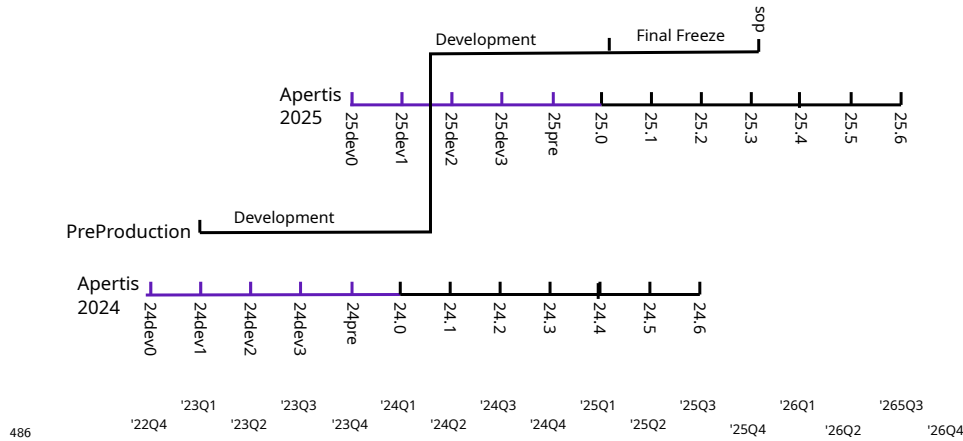
464 For the initial development phase there are two main options:

- 465 • follow the development releases of Apertis or its direct downstreams;
- 466 • follow the product releases of Apertis or its direct downstreams (switching
467 at the preview stage).

468 The first option allows the product development to use the very latest Apertis
469 features and developments on top of the most recent software baseline which
470 will form the basis of the future product release of Apertis or of its direct down-
471 stream, while the second option provides a more stable, but older, baseline al-

lowing the product team to focus on their own software stack. These approaches can be mixed, for example by starting out early product development on the current Apertis (or one of its direct downstreams) development release to take advantage of more recent features, but following that baseline when it becomes the product release instead of moving to the next cycle of development releases. By mixing the approaches in this way the product team has the flexibility of choosing the baseline that best fits their priorities at any given time.

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



Post-production support guidelines

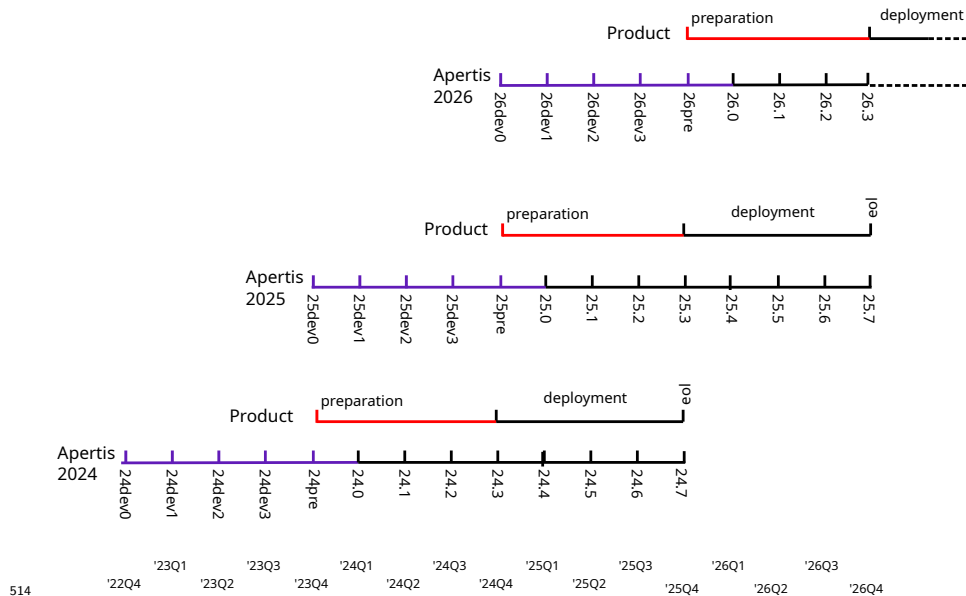
The post production support phase is the phase where the product is out in the market and any software updates are primarily done for the purpose of fixing bug and security issues.

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

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

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

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



Product guideline conclusions

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

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

524 the amount of support that can be provided.

525 **Appendix: Change in release strategy**

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

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

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

534 When the initial concept was set out, Ubuntu would support non-LTS releases
535 for 18 month (one year after the *next* Ubuntu release). Currently however the
536 support for non-LTS releases is only 9 months (3 months after the *next* Ubuntu)
537 release), which is simply too short for supporting product usage even if the
538 product has a very aggressive timeline.

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

542 The rationale for switching from Ubuntu as a baseline to Debian has been out-
543 lined in more detailed in the “[The case for moving to Debian stretch or Ubuntu](#)
544 [18.04](#)”⁸ concept document.

545 **Appendix: Distribution “freshness”**

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

552 This is a consequence from the trade-offs that are being made in the release
553 process to best serve users of Apertis and its direct downstreams, stability and
554 community support are preferred over having the very latest features. In case
555 newer features are required this can either be handled via the backports mech-
556 anism if only needed for specific users or, in case of a feature useful to most
557 users, including a newer version in the next release of Apertis or of its direct
558 downstreams can be considered.

⁸<https://apertis-website-0b3586.pages.apertis.org/architecture/distribution/case-for-moving-to-debian/>

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

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

571 **Appendix: Frequently Asked Questions**

572 **What is the effort required to move to a new product re-** 573 **lease?**

574 While Apertis publishes a product release every year, Debian does a release only
575 once every two years: this means that for each Debian release there will be two
576 Apertis product releases based on it.

577 Moving from an Apertis product release to another based on the same Debian
578 release usually does not require considerable effort: since one of the goals of
579 Apertis is to minimize the deviation from upstream, the vast majority of pack-
580 ages are pulled straight from Debian and the two releases will ship the same
581 versions. Only few components are specific to each releases, the main one being
582 the kernel due to the Apertis policy of tracking the **latest Linux LTS releases**.

583 Moving to a product release with a different Debian baseline often requires more
584 effort since the new baseline brings new major versions of many components and
585 in some cases deprecated components may get removed: an example of this is
586 the removal of the Python 2 interpreter in Debian Bullseye/Apertis v2022 after
587 more than ten years of it being deprecated.

588 **How often security fixes are made available to users?**

589 Apertis pulls security updates from Debian with an automated pipeline and
590 security fixes are quickly made available in the repositories for the in-progress
591 development/preview releases and in the `-security` repositories for the published
592 product releases.

593 In addition, the fixes in the `-security` repositories are folded in the main repos-
594 itory right after a point release for that product release is published, to make
595 them available to the widest audience.

596 This means that users of product releases have two options:

- 597 1. a constant stream of the latest security fixes by subscribing to the -
598 security repositories;
- 599 2. a quarterly stream of updates that get an additional validation step
600 through the QA rounds done for the point releases, by only subscribing
601 to the main repositories.

602 Subscribing to the `-security` repositories is **strongly recommended** in all cases
603 since the risk of regressions is minimal thanks to the upstream validation done
604 by the Debian project.

605 Do packages get updated in a published develop- 606 ment/preview release?

607 Once development/preview releases are published they are generally regarded
608 as immutable, and all new updates are landed in the repositories of the next
609 release.

610 There are exceptions however, in particular for:

- 611 1. security fixes that address vulnerabilities serious enough that are deemed
612 worth fixing even in releases only meant for development and not for pro-
613 duction
- 614 2. fixes addressing packages that fail to build from sources

615 In any case the updates are going to be kept as minimal as possible to minimize
616 the chances of introducing regressions. For instance, such updates do not usually
617 bump the version of the affected component significantly and in the majority of
618 the cases they only involve the addition of a specific patch.

619 In general, the impact of each update needs to be evaluated: for instance the
620 [CVE-2021-44228](https://nvd.nist.gov/vuln/detail/CVE-2021-44228)⁹ Log4J fix required a significant bump of the component's ver-
621 sion, but given the current marginality of the package in the Apertis ecosystem
622 the fix has been landed to all active branches with no further checks. In other
623 cases, where the effective impact may be more significant, the Apertis team
624 may consider rolling out a new point release (for instance, `v2022dev2.1`) after
625 validating it with a full QA round.

626 Do downstream distributions need to perform a folding?

627 Apertis will perform the folding of `security` and `updates` before each point release,
628 saving downstreams of much of the work. However, since downstreams can
629 carry their own changes and have their own custom repositories a folding will
630 be required.

631 Downstreams are encouraged to push changes upstream, which will allow all
632 Apertis users and other downstreams to take advantage of the changes, and in
633 turn will reduce the delta and maintenance cost.

⁹<https://nvd.nist.gov/vuln/detail/CVE-2021-44228>

634 **Do downstream distributions need to perform a branching?**

635 Apertis branches a new development or preview release as previously described
636 to provide a new starting point for the release. This same process should be
637 done by downstreams to follow the Apertis release flow.