Release flow and product lines

Contents

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

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

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

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

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

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

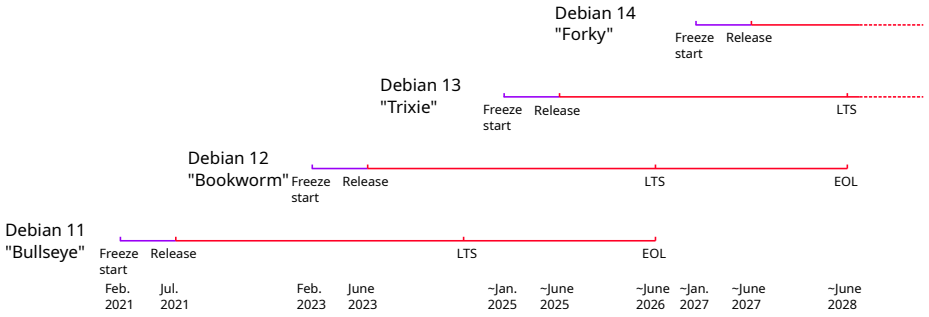
Debian release processes

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

"release-critical"). Even so, the process is well understood and predictable. For more information see the Debian release statistics[1]

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

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



Process towards a release

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

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

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

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

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

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

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

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

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

Process after release

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

Stable repository

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

Security repository

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

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

Stable Proposed Updates repository

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

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

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

Stable Updates repository

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

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

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

Backports repository

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

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

Debian release flow conclusions

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

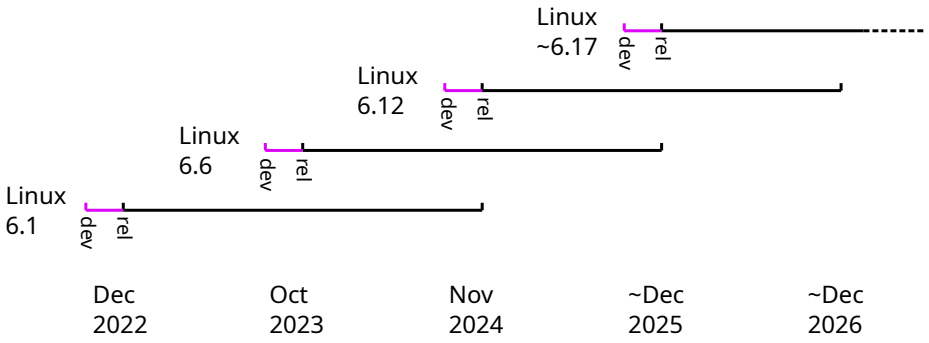
In the process going towards a release it's notable that even during the soft and hard freeze periods Debian is already a quite stable baseline as such a rebasing process for an Apertis product release can start when Debian is in freeze as long as there is enough time left before the product release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

Process after a release

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

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

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

Linux release flow conclusions

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

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

Apertis release flow

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

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

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

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

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

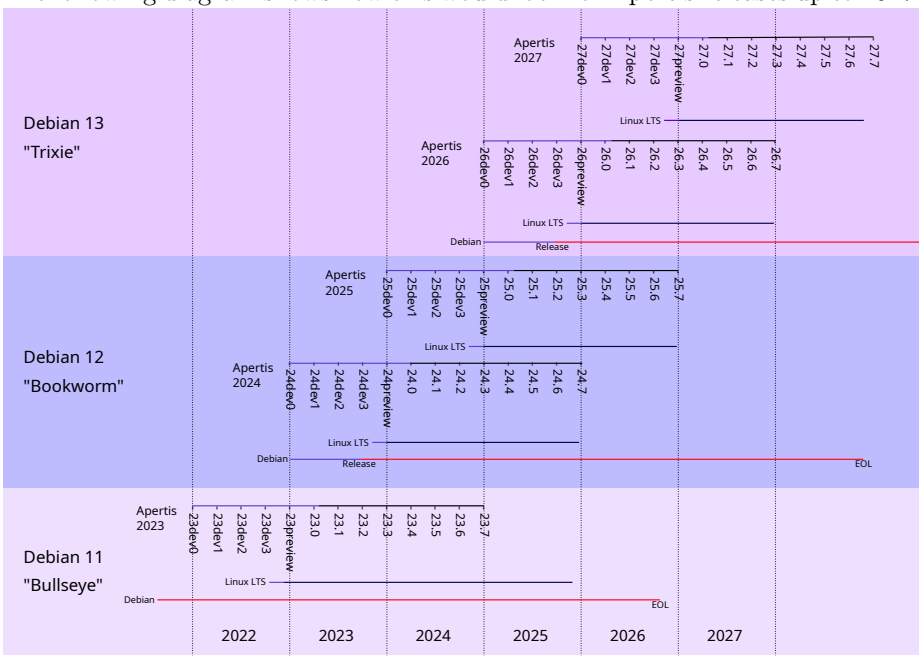
Quarter	Release type	Support
Q4	Release N-1 Preview	Limited, until the Q1 product release
Q4	Release N Development	Limited, until the Q1 development release
Q1	Release N-1 Product	Full support, until 1.75 years after release
Q1	Release N Development	Limited, until the Q2 development release
Q2	Release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

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

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



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

Flow up to a product release

The main flow towards a quarterly release will remain the same as it now, which is documented on the Apertis Release schedule[2] page. However, depending on the type of release the focus may differ.

[2] https://www.apertis.org/policies/releases/

Development releases (Q4, Q1, Q2, Q3)

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

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

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

Preview release (Q4)

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

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

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

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

Product release (Q1)

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

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

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

Process after a product release

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

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

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

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

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

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

Stable Repository

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

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

For instance, in Apertis v2024 this maps to:

- the `apertis/v2024` git branch in the packaging repositories[3]
- the `apertis:v2024:{target,development,sdk,non-free}` OBS repositories
- the `deb https://repositories.apertis.org/apertis/ v2024 target development sdk non-free` APT source

[3] https://gitlab.apertis.org/pkg

Once a point release is published, the updates staged in the repositories described below get folded in this repository to make them generally available.

Security repository

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

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

For instance, in Apertis v2024 this maps to:

- the `apertis/v2024-security` git branch in the packaging repositories[4]
- the `apertis:v2024:security:{target,development,sdk,non-free}` OBS repositories
- the `deb https://repositories.apertis.org/apertis/ v2024-security target development sdk non-free` APT source

Updates repository

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

Downstreams are recommended to include this repository but it's not mandatory.

For instance, in Apertis v2024 this maps to:

- the `apertis/v2024-updates` git branch in the packaging repositories[5]
- the `apertis:v2024:updates:{target,development,sdk,non-free}` OBS repositories
- the `deb https://repositories.apertis.org/apertis/ v2024-updates target development sdk non-free` APT source

Backports repository

This repository has backports of packages which are of special interest to downstreams but where not suitable for inclusion into the product release.

Unless specific agreements have been made, the packages available in this repository are for experimentation use only and are not supported as part of the produce release.

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://www.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.

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

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

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

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

Pre-production guidelines

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

Typically this phase consists of a period of heavy development (potentially interleaved with short stabilisation periods), followed by a potentially longer final stabilisation period before entering production.

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

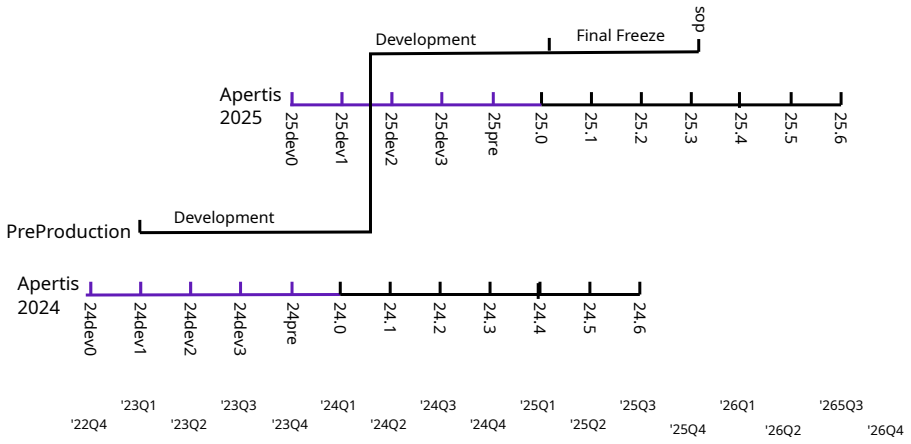
For the initial development phase there are two main options:

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

The first option allows the product development to use the very latest Apertis features and developments on top of the most recent software baseline which will form the basis of the future product release of Apertis or of its direct downstream, 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

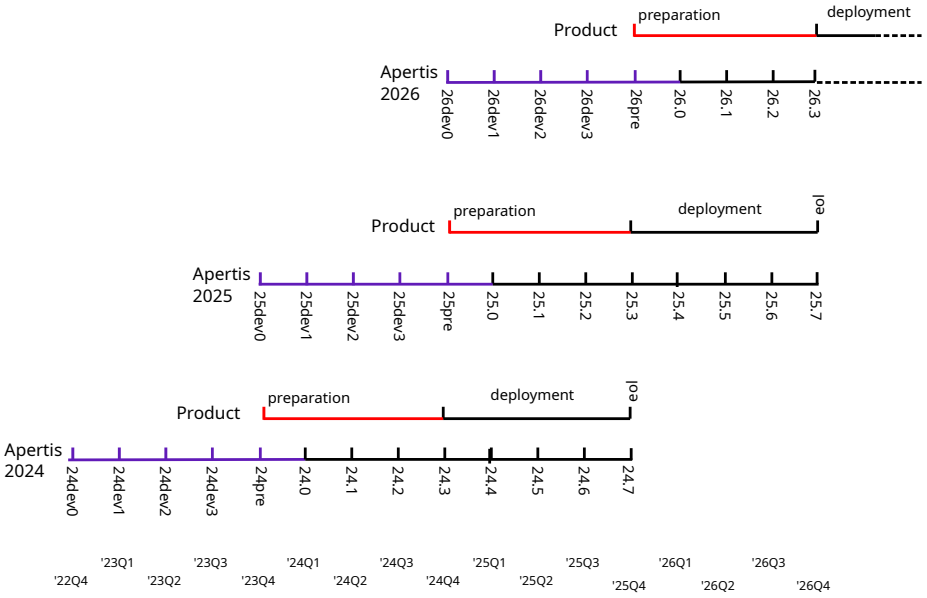
The post production support phase is the phase where the product is out in the market and any software updates are primarily done for the purpose of fixing bug and security issues.

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

the amount of support that can be provided.

Appendix: Change in release strategy

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

Conceptually there are two big changes with this new concept:

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

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

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

The rationale for switching from Ubuntu as a baseline to Debian has been outlined in more detailed in the "The case for moving to Debian stretch or Ubuntu 18.04"[8] concept document.

Appendix: Distribution "freshness"

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

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

[8] https://www.apertis.org/architecture/distribution/case-for-moving-to-debian/

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

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

Appendix: Frequently Asked Questions

What is the effort required to move to a new product release?

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

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

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

How often security fixes are made available to users?

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

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

This means that users of product releases have two options:

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

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

Do packages get updated in a published development/preview release?

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

There are exceptions however, in particular for:

1. security fixes that address vunerabilities serious enough that are deemed worth fixing even in releases only meant for development and not for production
2. fixes addressing packages that fail to build from sources

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

In general, the impact of each update needs to be evaluated: for instance the CVE-2021-44228[9] Log4J fix required a significant bump of the component's version, bug given the current marginality of the package in the Apertis ecosystem the fix has been landed to all active branches with no further checks. In other cases, where the effective impact may be more significant, the Apertis team may consider rolling out a new point release (for instance, `v2022dev2.1`) after validating it with a full QA round.

Do downstream distributions need to perform a folding?

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

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

[9] https://nvd.nist.gov/vuln/detail/CVE-2021-44228

Do downstream distributions need to perform a branching?

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