APERTIS

Contributions

Contents

This guide covers the expectations and processes for Apertis developers wishing to make contributions to the Apertis project and the wider open source ecosystem. These policies should be followed by all developers, including core and third party contributors. A checklist[1] is provided in conjunction with these policies to aid contributors.

TL;DR

Do you want to quickly submit some changes to an Apertis component?

[1]https://www.apertis.org/policies/contribution_checklist/

- Have you tried to submit your changes upstream first? Contributing upstream benefits the community at large and keeps Apertis sustainable. Once changes have been landed upstream, backporting them to the versions shipped with Apertis is usually an expedite process.
- Register and log to the Apertis GitLab instance[2]
- Fork the project you want to patch on the Apertis GitLab instance[3]
- Create commits according to the version control best practices[4]
- Go through the contribution checklist[5]
- Submit the branch with your commits as a Merge Request[6]
- Address any review feedback[7]

Suitability of contributions

Like most open source projects, Apertis requires contributions are submitted via a process (which in the case of Apertis is defined below) to ensure that Apertis continues to meet it's design goals and remain suitable for it's community of users. In addition to design and technical implementation details, the suitability of contributions will be checked to meet requirements in areas such as coding conventions[8] and licensing[9].

Upstream First Policy

Apertis is a fully open source GNU/Linux distribution that carries a lot of components for which it is not the upstream. The goal of upstream first[10] is to minimize the amount of deviation and fragmentation between Apertis components and their upstreams.

Deviation tends to duplicate work and adds a burden on the Apertis developers when it comes to testing and updating to newer versions of upstream components. Also, as the success of Apertis relies on the success of open source in general to accommodate new use cases, it is actively harmful for Apertis to not do its part in moving the state of the art forward.

It is the intention of Apertis to utilize existing open source projects to provide the functionality required, where suitable solutions are available, over the creation of home grown solutions that would fragment the GNU/Linux ecosystem further.

[2]https://gitlab.apertis.org/
[3]https://gitlab.apertis.org/
[4]https://www.apertis.org/guides/app_devel/version_control/
[5]https://www.apertis.org/policies/contribution_checklist/
[6]https://docs.gitlab.com/ee/user/project/merge_requests/getting_started.html
[7]https://docs.gitlab.com/ce/development/code_review.html#having-your-merge-request-reviewed
[8]https://www.apertis.org/policies/coding_conventions/
[9]https://www.apertis.org/policies/license-expectations/
[10]https://www.apertis.org/policies/upstreaming/

This policy should be taken into consideration when submitting contributions to Apertis.

Upstream Early, Upstream Often

One mantra that can be often heard in Open Source communities is “upstream early, upstream often”. The approach that this espouses is to breakdown large changes into smaller chunks, attempting to upstream a minimal implementation before implementing the full breath of planned features.

Each open source community tends to be comprised of many developers, which share some overlap between their goals, but may have very different focuses. It is likely that other developers contributing to the project may have ideas about how the features that you are planning may be better implemented, for example to enable a broader set of use cases to utilise the feature. Submitting an early minimal implementation allows the general approach to be assessed, opinions to be sought and a consensus reached regarding the implementation. As it is likely that some changes will be required, a minimal implementation minimizes the effort required to take feedback into account.

Taking this approach a step further, it can often be instructive to share your intention to implement larger features before starting. Such a conversation might be started by sending an email to the projects devel mailing list[11] saying:

```
Hi,

I'm attempting to use <project> to <task> for my project.

I'm thinking about doing <brief technical overview> to enable this usecase.

I'm open to suggestions should there be a better way to solve this.

Thanks,

<developer>
```

This enables other experienced developers the chance to suggest approaches that may prove to be the most efficient, saving effort in implementation and later in review, or may point to missed existing functionality that can be used to solve a given need without needing substantial development effort.

[11] https://lists.apertis.org/

Extending Apertis

Adding components to Apertis

Apertis welcomes requests for new components to be added to the distribution and can act as a host for projects where required, however the open source focus of Apertis should be kept in mind and any proposed contributions need to both comply with Apertis policies and present a compelling argument for inclusion.

Additional components can be categorised into 3 main groups:

- Existing upstream component available in Debian stable (with suitable version)
- Existing upstream component, not available in debian stable
- New component on gitlab.apertis.org

There is a maintenance effort associated with any components added to Apertis, as any components added will need to be maintained within the Apertis ecosystem. The effort required to maintain these different categories of components are very different. Prepackaged Debian components require a lot less maintenance effort than packaging other existing upstream components. Developing a new component on gitlab.apertis.org requires both the development and packaging/maintenance to be carried out within Apertis, significantly raising the effort required.

When looking for ways to fullfil a requirement there are a number of factors that will increase the probability of a solution being acceptable to Apertis.

- Component already included in Debian stable: As Apertis is based on Debian and already has processes in place to pull updates from this source. The cost of inclusion is dramatically lower than maintaining packages drawn from other sources, as a lot of the required effort to maintain the package is being carried out within the Debian ecosystem.
- Proven actively maintained codebase: Poorly maintained codebases present a risk to Apertis, increasing the chance that serious bugs or security holes will go unnoticed. Picking a solution that has an active user base, a developer community making frequent updates and/or is a mature codebase that has undergone significant “in the field”testing makes the solution more attractive for inclusion in Apertis. It is understood that, whilst extensive, the Debian repositories are not all encompassing, if proposing an existing open source component that isn’t currently provided by Debian, being able to show that it is actively maintained will be important.
- Best solution: In general, there exists more open source solutions than there exists problems. To be in with a good chance of having a component included in Apertis it will be required to explain why the chosen solution represents the best option for Apertis. What is “best”is often nuanced and will be affected by a number of factors, including integra-

tion/overlap with existing components and the size/number of dependencies it has (especially if they aren't currently in Apertis). It may be that whilst a number of existing solutions exist, none of them are a good fit for Apertis. This may suggest a new component is the best solution, though adapting/extending one of the existing solutions should also be considered.

The Apertis distribution is supported by it's members. As previously mentioned, in order to ensure that Apertis remains viable and correctly focused, it is important that any additions to the main Apertis projects[12] are justified and can be shown to fill a specific and real use case. Maintaining the packaging, updating the codebases of which Apertis is comprised and performing testing on supported platforms is a large part of the effort needed to provide Apertis. As a result, it will be necessary to either be able to provide a commitment to support any packages proposed for inclusion in the main Apertis projects or gain such a commitment from an existing member.

The Apertis development team commit to maintaining the packages included in the references images. Packages may be added to the main package repositories but not form part of the reference images. Such packages will be maintained on a best effort basis, that is as long as the effort remains reasonable the Apertis team will attempt to keep the package in a buildable state, however runtime testing will not be performed. Should the package fail to build or runtime issues are reported and significant effort be required to modify the package the original or subsequent users of the package may be approached to help resource fixing the package. Ultimately the package may be removed if a solution can not be found. Likewise, should a different common solution for Apertis be chosen at a later date, the package may be deprecated and subsequently removed.

Proposals for inclusion of new components are expected to be made in the form of a written proposal. Such a proposal should contain the following information:

- Description of the problem which is being addressed
- Why the functionality provided by the proposed component is useful to Apertis and it's audience
- A review of the possible solutions and any advantages and disadvantages that have been identified with them
- Why the proposed solution is thought to present the best way forward, noting the points made above where relevant
- Whether any resources are to be made available to help maintain the component.

Dedicated Project Areas

An alternative to adding packages to the main Apertis project is to apply to have a dedicated project area, where code specific to a given project can be stored. Such an area can be useful for providing components that are highly

[12] https://www.apertis.org/policies/package_maintenance/

specific to a given project and/or as a staging area for modifications to core packages that might later get folded back into the main area, either by changes being submitted to the relevant Apertis component or after changes have been upstreamed[13] to the components main project. A dedicated area will allow a project group to iterate on key components more rapidly as the changes made do not need to work across the various supported hardware platforms. It must be noted that whilst a dedicated project area would allow some requirements with regard to platform support to be ignored, packages in such areas would still be required to comply with other Apertis rules such as open source licensing[14]. It should be expected that the Apertis developers will take a very hands off approach to the maintenance and testing of packages in such areas. If packages in such areas require work, the project maintainers will be contacted. The Apertis maintainers may at their discretion help with minor maintenance tasks should a package be of interest to the Apertis project. Packages that become unmaintained may be removed.

Requests for dedicated project areas are also expected to be made in a form of a written proposal. Such a proposal should contain the following information:

- Description of the project requiring a dedicated project area
- Preferred name to be used to refer to the project
- Expected use of the dedicated area
- Expected lifetime of the project area
- Contact details of project maintainers

Such submissions should be made via the devel mailing list[15].

The submission should be discussed on the mailing list and must be agreed with the Apertis stakeholders.

Extending existing components

Apertis carries a number of packages that have been modified compared to their upstream versions. It is fairly typical for distributions to need to make minor modifications to upstream sources to tailor them to the distribution, Apertis is not different in this regard.

Whilst Apertis does accept changes to existing components, it needs to be acknowledged that this increases the effort required to maintain the package in question. It may be requested that an attempt be made to upstream the changes, in line with the upstream first policy, either to the packages upstream or Debian. More guidance is provided in the upstreaming[16] documentation. If changes are not generally of use or would have a negative impact on the broader Apertis

[13]https://www.apertis.org/policies/upstreaming/
[14]https://www.apertis.org/policies/license-expectations/
[15]https://lists.apertis.org/
[16]https://www.apertis.org/policies/upstreaming/

user base, changes may be required to be carried by the specific project within a dedicated project area.

Adding support for new hardware

One special case of contributions is the support for new hardware, since even if the same general rules apply, some additional considerations need to be taken into account.

When adding new hardware there are two scenarios:

- Adding new hardware to an already supported board: This is the simplest one, consisting in adding or enabling the required drivers in the kernel or in other packages (such as ofono)
- Adding a new board: This requires to provide complete support for the new board.

In order to support new boards, Apertis requires maintainers to provide:

- Hardware pack: A combination of hardware specific packages, such as bootloader, kernel, firmware, as described in New hardware[17]
- Image recipe: A recipe as described in Image building[18] which makes use of the hardware pack.

As mentioned before, Apertis follows the policies of Upstream First and Upstream Early Upstream Often, so hardware packs should be based in upstream packages, like `linux` in Debian. This guarantees the maintainability of such hardware support across time and releases.

Besides a basic support as the one described, boards can become [Reference Hardware]({{ }}). The key value in this approach is the Apertis QA[19] which ensures that automated tests are run in LAVA on daily images and manual testing is done on weekly images. For this to happen, an agreement needs to be reached with the Apertis team.

Adding designs to Apertis

Another way to contribute to Apertis is with design documents. A design document contains the description of all relevant aspects of a feature or of a requirement. The current design documents can be found in the Concepts Designs section[20]. These documents cover topics that have been researched but not necessarily implemented. They should provide a good understanding of the impact of the technology that forms the basis of the concept, what it is, how it works, what are the threat models, the required infrastructure, how it would be integrated with Apertis and anything else that is deemed relevant.

[17]https://www.apertis.org/guides/low_level/enabling_new_hardware/
[18]https://www.apertis.org/guides/image_devel/image_building/
[19]https://www.apertis.org/qa/
[20]https://www.apertis.org/concepts/

Such designs should be updated when implemented to explicitly cover the final implementation and moved to a suitable section of the site, typically the Architecture[21] or Guides[22] section.

Project-wide impact is the metric used to decide if a contribution will be handled as a component or as a design. If the impact of the contribution on the Apertis project goes beyond the additional maintenance effort, it is likely to require a design document before the component contribution.

As an example we will consider a proposal to provide tools and workflows for process automation by including the Robot Framework[23] in the Apertis Universe. The Robot Framework is a generic open source automation framework that can be used for automation of tests and processes. Robot Framework is released under Apache License 2.0[24]. However we do not expect to ship Robot Framework components on Apertis target images.

The first important consideration is the state-of-the-art for addressing the goals of the design. In our example the Robot Framework is preferred due it's maturity, unique and simple to use descriptive language, and it's active development community. However a strong argument in favor of the Robot Framework is it' s user base. Adding the Robot Framework to the Apertis Universe is expected to bring Robot Framework users to Apertis.

The next important consideration are how the design is expected to work and the potential impact on Apertis. The Robot framework has a layered architecture. The top layer is the simple, powerful, and extensible keyword-driven descriptive language for testing and automation. This language resembles a natural language, is quick to develop, is easy to reuse, and is easy to extend. On the bottom layer of the architecture is the item to be tested, or the process to be automated.

The middle layer is what makes the Robot Framework extensible: libraries. A library, in Robot Framework terminology, extends the Robot Framework language with new keywords, and provides the implementation for these new keywords. Each Robot Framework library acts as glue between the high level language and low level details of the item being tested, or of the environment in which the item to be tested is present.

Adding the Robot Framework to the Apertis Universe has potential to impact:

1. Development workflow: Apertis encourages the use of continuous integration and the use of shared infrastructure resources instead of resources that are private to specific developers.
2. Testing Apertis images: Apertis encourages the use of environments that are as close as possible to production environments, meaning that ideally,

[21] https://www.apertis.org/architecture/
[22] https://www.apertis.org/guides/
[23] https://robotframework.org/
[24] http://www.apache.org/licenses/LICENSE-2.0.html

the Apertis images under test are not instrumented for testing, and are only minimally modified.

3. Testing infrastructure: Apertis uses LAVA for deployment of operating system and software in hardware, and for automated testing. The two main constraints are LAVA being asynchronous and non-interactive. While both developers and CI pipelines can submit jobs to LAVA, they cannot interact with a job while it is running. The LAVA workflow is: submit a job, wait for the job to be selected for execution, wait for the job to complete execution, and download test results.

Addressing the benefits of the new design proposal is also important. As mentioned, adding tools and workflows for process automation with the Robot Framework will extend the Apertis projects and we expect to attract more users by doing so. Adding real-world use cases can illustrate the value with a good level of details.

The proposal should also describe how to address the integration with Apertis taking into account the constraints of the Apertis development workflow, of testing Apertis images, and of the Apertis testing infrastructure.

The design proposal can also include a high level description of the estimated work. For example, adding Robot Framework to Apertis will involve developing and/or modifying Robot Framework libraries; and developing a run-time compatibility layer for LAVA to keep testing environments as close as possible to production environments, and to adapt the execution of Robot Framework tests to suit the LAVA constraints.

And finally it could contain a high level implementation plan. In our example, one possible way to integrate Robot Framework is to adopt it in stages:

1. Add Robot Framework to the Apertis SDK to enable developers to use the Robot Framework locally
2. Robot Framework Integration development: Adapt libraries and create the run-time compatibility layer for LAVA
3. Deployment on the Apertis infrastructure

This section describes general topics, but it may not be complete for all designs. Regarding the level of details the design document should be complete enough to describe the design and surrounding problems to developers and project managers, but it is not necessary to describe implementation details.

As a rule of thumb start with a lean design document and submit it for review as early as possible. You can send a new design for review to the same process used for a component contribution[25].

[25]https://www.apertis.org/guides/app_devel/development_process/

Concept Design Document Template

The following template should be used as a guide when writing new concept designs:

```
+++
title = "<document title>"
weight = 100
outputs = [ "html", "pdf-in",]
date = "20xx-xx-xx"
+++

# Introduction

# Terminology and concepts

# Use cases

# Non-use cases

# Requirements

# Existing systems

# Approach

# Evaluation Report

# Recommendation

## Design recommendations

# Alternative designs

# Open questions

## Unresolved design questions

## Unresolved implementation questions

# Risks

# Summary

# Appendix

# References
```

Other important bits

Sign-offs

Like the git project and the Linux kernel, Apertis requires all contributions to be signed off by someone who takes responsibility for the open source licensing of the code being contributed. The aim of this is to create an auditable chain of trust for the licensing of all code in the project.

Each commit which is pushed to the main branches in git **must** have a `Signed-off-by` line, created by passing the `--signoff`/`-s` option to `git commit`. The line must give the real name of the person taking responsibility for that commit, and indicates that they have agreed to the Developer Certificate of Origin[26]. There may be multiple `Signed-off-by` lines for a commit, for example, by the developer who wrote the commit and by the maintainer who reviewed and pushed it:

```
Signed-off-by: Random J Developer <random@developer.example.org>
Signed-off-by: Lucky K Maintainer <lucky@maintainer.example.org>
```

Apertis closely follows the Linux kernel process for sign-offs, which is described in section 11 of the kernel guide to submitting patches[27].

Privileged processes

Pushing commits to gitlab.apertis.org requires commit rights. Whilst commit rights to most repositories are only granted to trusted contributors (see "Getting commit rights"for how to get commit rights) the Apertis GitLab infrastructure is open for registration, enabling anyone to sign up for an account, fork packages into there personal space and submit merge requests (see the development process[28] for more details). All commits must have a `Signed-off-by` line assigning responsibility for their open source licensing.

Some admin steps on the periphery of packaging and releasing new versions of Apertis modules as Debian packages may require access to build.collabora.com (OBS). These are issued separately from commit rights, and are generally not needed for the main development workflows.

Submitting automated test runs on lava.collabora.dev requires CI rights, which are granted similarly to packaging rights. However, CI results may be viewed read-only by anyone.

Getting commit rights

Commit rights (to allow direct pushes to git, and potentially access to the package building system, build.collabora.com) may be granted to trusted third

[26] http://developercertificate.org/
[27] https://www.kernel.org/doc/Documentation/SubmittingPatches
[28] https://www.apertis.org/guides/app_devel/development_process/

party contributors if they regularly contribute to Apertis, with high quality contributions at the discretion of current Apertis maintainers.

Accounts on the Apertis GitLab instance can are available via open registration[29]

By creating an account you signify that you accept the Apertis Privacy Policy[30] and Terms of Use[31]

For access to other Apertis infrastructure, please send an email to `account-requests@apertis.org` including:

- Your full name
- The email address you prefer to be contacted through
- The nickname/account name you wish to be known by on the Apertis GitLab

The role of maintainers

Most Open Source projects have one or more core contributors that take on a managerial role for the project. This group may include the original author(s) of the project and long-term trusted contributors, though in many projects with a longer history, lead of the project may well have been taken on by another knowledgable contributor.

The basic role of a project maintainers is to:

- help set the direction for the project;
- ensure that the projects policies are followed and that the project continues to work towards it's stated objectives;
- review and evaluate contributions for correctness and suitability;
- apply accepted contributions;
- resolve issues (such as bugs and security issues) that arise;
- and ensure the processes required to release new project artifacts are completed.

Larger projects may have many maintainers who specialise in parts of the work that need to be carried out or who have deeper knowledge of specific parts of a larger codebase. For example such maintainers may be in charge of applying these roles to a single component within the Apertis distribution.

The Apertis maintainers are funded by the projects backers, with direction agreed between the maintainers and backers to fulfill the needs of the backers whilst driving the project towards it's stated objectives. Many of the maintainers have a long history with the Apertis project or have come to the project with lots of experience in the area in which they work (such as Debian packaging).

[29]https://gitlab.apertis.org/users/sign_up
[30]https://www.apertis.org/policies/privacy_policy/
[31]https://www.apertis.org/policies/terms_of_use/

The Apertis maintainers are responsible for ensuring that bug and security fixes are applied to the various components of which Apertis is made and for migrating to newer releases of it's upstreams inline with the documented polices. The maintainers then ensure that the source of these components is reliably built into the binaries and images provided, covering the range of architectures and platforms supported by the project.

In addition to tracking updates and fixes from the projects that Apertis uses, the maintainers also review changes that are submitted to the project from contributors. The maintainers actively contribute to the project and submit changes following the same processes that are expected from other contributors. All such changes are reviewed to ensure that they meet the project goals, objectives and policies as well as ensuring the are sound and do not contain any obvious issues.

Whilst some contributors may remain active within the projects community of users and developers for some time, this is a long way from guaranteed. Maintainers must evaluate contributions to ensure that the changes that are being proposed would continue to be maintainable in the absence of the original contributor. As a result the maintainers may reject contributions that otherwise appear to meet the policies if they feel that they would be impossible to maintain or requiring changes to make the contribution more maintainable for the project.

The maintainer is usually taking on the responsibility on behalf of the project to ensure that your changes and modifications continue to be provided by the project, porting them to new versions of packages or ensuring that they remain valid as the project inevitably changes to accommodate new goals or the ever changing computing landscape. As a result accepting changes will transfer this burden from you to the maintainers. You can continue to use the project without needing to actively maintain the changes. As a result the onus is on the contributor to persuade the project of the advantages of the changes, not for the project to be beholden to accept contributions.

Work across releases

The Apertis releases flow[32] sets a strict schedule for development, which should help to plan the work and contributions.

Using development releases, contributors and maintainers work in new features following the policies of Upstream First and Upstream Early Upstream Often. Thanks to this approach, new development releases bring new improvements that can be tested by the community.

After testing new features and bug fixes, a maintainer can propose to backport low impact changes to stable release using `-security`, `-updates` or `-backports`, providing a good rationale for the request and a justification on the low impact

[32] https://www.apertis.org/policies/release-flow/

for a stable release. This process allows maintainers to address issues in stable version while ensuring the reliability of stable releases.

Hardware packs

As described in Adding support for new hardware, hardware packs present a special case which requires additional clarifications.

If the Apertis upstream polices are followed, the additional effort to maintain hardware packs in Apertis should be minimal, as the main support will already be available trough some upstream package. However, integration and QA play a key role.

For hardware not listed as reference[33], Apertis assumes hardware pack maintainers will run regular tests on the devices for all the supported releases to confirm the status. If issues are found during testing, maintainers should report them in the Apertis Issues board[34], so the community is aware of them and can keep track of the latest news.

For reference boards[35] this process is carried out by the Apertis team and the latest status can be tracked using the Apertis QA Report[36] application.

Contribution Template

This section contains a contribution template that illustrates the ideal first email a developer would send for adding a design document to Apertis. This template for the first email contains the description of the design document instead of the design document itself. The idea is to promote involving the Apertis team as early as possible, and ideally before completing the work.

The rationale for this approach is that it is very difficult for an external contributor to understand the impact a contribution can bring to Apertis, and by asking early, the work can be done in ways that are compatible with Apertis and welcome by the Apertis team.

```
From: Your name <your email>
To: devel@lists.apertis.org
Subject: Robot Framework design document

Hi,

I want to contribute to Apertis, and I am sending this email to ask if our
proposal can be added to Apertis. I am sending the email based on the
contribution template I found on the Apertis website, and we are looking
```

[33]https://www.apertis.org/reference_hardware/
[34]https://gitlab.apertis.org/infrastructure/apertis-issues/-/issues
[35]https://www.apertis.org/reference_hardware/
[36]https://qa.apertis.org/

forward for receiving feedback from the Apertis team.

Thank you,

Your name

-- // --

1. Me and my team
I am a developer, I am specialized in embedded devices, and I work in a product team that creates IoT devices with all sorts of environmental sensors and actuators.

2. What is the goal of my proposal
My proposal is for a design document that describes tools and workflows for process automation using the Robot Framework. The Robot Framework is a generic open source automation framework that can be used for automation of tests and processes.

- From our perspective this adds value to the Apertis Universe. Do you agree?

2. State-of-the-art
We prefer the Robot Framework because it is mature, it is simple to use, and because it has an active development community.

While there are other automation frameworks available, they tend to be purpose specific. Examples of purpose specific automation frameworks that we considered include Selenium and JUnit.

3. How does our contribution works?
The Robot framework has a layered architecture. The top layer is the simple, powerful, and extensible keyword-driven descriptive language for testing and automation. This language resembles a natural language, is quick to develop, is easy to reuse, and is easy to extend. On the bottom layer of the architecture is the item to be tested, or the process to be automated.

The middle layer is what makes the Robot Framework extensible: libraries. A library, in Robot Framework terminology, extends the Robot Framework language with new keywords, and provides the implementation for these new keywords. Each Robot Framework library acts as glue between the high level language and low level details of the item being tested, or of the environment in which the item to be tested is present.

4. Potential impact on Apertis?
We are aware there the architecture of the Robot Framework is different from the Archutecture of LAVA. In some cases the Robot Framework accepts human intervention with tests while LAVA expects everything to be automated. While we do not fully understand to which extent this will impact Apertis, we expect that for our design proposal will need to adapt to Apertis and LAVA constraints. Can you help us here?

5. Benefits for Apertis?
The Robot Framework project is active for many years and is used for a variety of use cases. We expect that adding the Robot Framework to the Apertis Universe will bring Robot Framework users to Apertis.

6. What is the license of the main components?
The Robot Framework itself is licensed under the Apache License 2.0, however Robot Framework libraries can use different licenses.

7. The plan to integrate the design into Apertis
Our understanding is that Apertis currently uses LAVA for testing, and that images being tested are as close to production images as possible (almost no testing instrumentation included). We propose to develop and/or modify a few Robot Framework libraries, and to create a run-time compatibility layer for LAVA. We expect that the combination of custom libraries with the run-time compatibility
layer for LAVA will enable us to keep testing environments as close as possible to production environments, and to adapt the execution of Robot Framework tests to suit the Apertis and LAVA constraints.

8. Estimated work to implement the design
Our ballpark estimation to add or modify Robot Framework libraries and to create the run-time compatibility layer for LAVA is of approximatedly 1500 hours of work. But we need your help to fully understand the impact on the Apertis side.

9. High level implementation plan
While we understand our use case and requirements, we would like to receive feedback from other potential users as soon as possible. Our idea is to deploy the Robot Framework in stages to allow early involvement of other users:

- Add Robot Framework to the Apertis SDK to enable developers to use the Robot Framework locally

- Robot Framework Integration development: Adapt libraries and create the run-

```
time
compatibility layer for LAVA

- Deployment on the Apertis infrastructure
```

Frequently asked questions

When is a good time to start offering package updates?

Package updates should be offered as soon as they are stable enough, through the development release available. This will allow testing the new version in the latest daily images and also allow other interested parties to get involved in the development, test the new features and provide feedback. This will also help package maintainers to get early feedback about the changes.

Is it expected that the package maintainer checks the version updates of upcoming releases for

its dependencies?

The Apertis release flow provides different types of releases: development, preview and stable. Developers should push new features to development releases, as this type of release is meant to test new features. After confirming that new features are stable enough, and if the changes have low impact, they can be backported to stable releases.

With this idea in mind, contributors should be aware of the versions of packages in different releases in order to plan possible feature backports.

What happens in case the dependencies are not yet available in the upcoming release, because

the required packages are not fully ported?

If the recommended process is followed, this should not happen, since development is done in the latest development release available. However, it is possible that in newer releases some required dependencies are no longer available. Under these circumstances, the package maintainer should address the issue by either:

- Using an alternative dependency
- Disabling or replacing the functionality that requires the dependency that is not available

In case neither of these options are feasible, unfortunately the package will fail to build and hence won't be available in the release.

What is the latest point in time to deliver the stable version, etc..?

Apertis release flow uses preview releases as the last point to introduce medium impact changes, in order to ensure stability of new stable releases. For stable releases it is expected that only bugfixes are introduced, again to ensure stability, with exceptions handled case by case.

What can a package maintainer expect from the Apertis distribution maintainer in a release flow?

Apertis is a Debian based distribution, and each Apertis release tracks one Debian release, from which it gets the majority of packages. Following this idea, package maintainers can get the relevant information from the resources available:

- Apertis Dashboard: https://infrastructure.pages.apertis.org/dashboard/
- Apertis Daily images: https://images.apertis.org/daily/
- Debian:
  - https://packages.debian.org/stable/
  - https://packages.debian.org/testing/