Contribution Checklist

Contents

This document covers the steps that should be taken at the various stages of making a contribution to Apertis with the rationale more fully explained in the policy[1]. It covers both those steps to be taken by the contributor as well as the maintainer(s)[2] accepting the contribution on behalf of Apertis. It is presented in this manner to provide transparency of the steps that are taken and the considerations that are made when accepting a contribution into Apertis, be that a modification of an existing component, addition of a new component or concept design.

The steps required to be taken by a contributor will be marked with `Contributor`, those to be taken by a maintainer on behalf of the Apertis project will be labeled `Maintainer`.

Apertis is utilized by multiple parties, all of whom have a stake in Apertis continuing to meet their own set of requirements. Whilst a proposed change may provide the optimal solution for your use case, the Apertis maintainers will need to consider the impact the change will have on the other users of Apertis too and thus may request changes to the solution to ensure that Apertis continues to well serve all its users.

Depending on the scope and content of the requested change, options may be

[1] https://www.apertis.org/policies/contributions/
[2] https://www.apertis.org/policies/contributions/#the-role-of-maintainers

available to provide a dedicated project area[3] to host party/project specific packages to enable such changes to be made to a project specific version therefore avoiding and impact on the core Apertis offering.

This checklist is broken down into the following stages, with some items broken out per contribution type:

- Pre-development: These are topics that should be addressed prior to any significant work being carried out to avoid pitfalls that may cause a contribution to be rejected.
- Development: Certain factors and considerations should be made during the development of proposed changes to ensure they conform to the projects polices.
- Pre-submission: Final checks that should be made prior to a submission being made.
- Review: Points that should be covered during the review of the contribution.
- Post-submission: These are on-going responsibilities after a change has been accepted.

Pre-development

General

- `Contributor` **Understand licensing requirements**: Ensure that the contribution will be able to be licensed in a manner acceptable to the Apertis project[4].

- `Contributor` **Determine if ongoing support is to be provided**: Apertis is supported with resources and effort by it's core backers. It is the requirements of those who support the project who ultimately control its direction. Whilst simple non-intrusive changes are very welcome, the ability to offer firm commitments to support the project may impact the viability of a proposed substantial change that provides no benefit to the existing maintainers.

- `Contributor` **Identify the value that the proposed changes bring to Apertis**: During review, the Apertis maintainers will consider the value brought to Apertis[5] by any proposed changes. Ensure that such value can be expected before starting development.

- `Contributor` **Ensure proposed changes comply with all existing relevant policies**: The Apertis maintainers will be evaluating whether the proposed changes comply with the full range of policies which govern the Apertis project. Now is a good time, before significant effort is expended,

[3] https://www.apertis.org/policies/contributions/#dedicated-project-areas
[4] https://www.apertis.org/guides/licensing/license-applying/
[5] https://www.apertis.org/policies/contributions/#extending-apertis

to check that the proposed changes align with the Apertis projects policies to guide development and avoid disappointment or wasted effort.

Concept Design

- `Contributor` **Survey the state of art**: The project strives to adapt and expand to new use cases utilizing the approach that provides the best fit for Apertis. Any proposed change to the project should show that alternative have been researched and evaluated.

Development

General

- `Contributor` **Explain what the contribution brings to Apertis**:
  - Code: What does the change do?
  - Component: What is the component, what does it do?
  - Concept Design: What is the goal, how is it expected to work?
- `Contributor` **Any impacted documentation is updated**: This may be able to form part of the same merge request or may need to be part of a separate merge request depending on the repository to which changes are being made. Either way, such changes should be available for review at the same time.

Code

- `Contributor` **Coding conventions**: Ensure that any code conforms with the Apertis coding conventions[6]
- `Contributor` **Changes don't break any supported architecture**: Apertis provides support for a number of reference platforms[7], the changes must work or not be applicable on all applicable architectures and platforms.

Component

- `Contributor` **Components should follow the packaging workflow**: Repositories for newly added components should be structured according to the component structure[8] guide, including providing Apertis'CI pipelines. The pipeline should succeed for all supported architectures. Where a components applicability is limited to specific platforms or architectures, this should be well documented and the components repository configured to reflect this.

[6] https://www.apertis.org/policies/coding_conventions/
[7] https://www.apertis.org/reference_hardware/
[8] https://www.apertis.org/guides/app_devel/component_structure/

Concept Design

- `Contributor` **Follow document template **: The document should utilize the document template[9] as a framework when writing the concept design.
- `Contributor` **Document expected approach to meeting goals**: An outline should be giving a high level overview of the steps that would need to be taken to take Apertis from its current state to the end goal of the concept design.

  The breadth of topics that may need to be covered here will be highly dependent on the goal of the concept document. The document should be detailed enough to clearly describe the design and surrounding problems to developers and project managers, but it is not necessary to describe implementation details.

  Topics that may need to be addressed include changes or impact on:

  - The development workflow
  - CI/CD and testing approach
  - Infrastructure configuration
  - Existing components
  - Support of releases over their lifetimes
  - Long-term maintainability of the project
  - Impact on security and effect on security boundaries
  - Backwards compatibility with existing feature set

  Such topics may require a high degree of familiarity with the project to answer. The Apertis maintainers are open to discussing goals and approaches prior to a concept design being submitted. Discussing and collaborating with the maintainers at an early stage is likely to prove beneficial to the contributor, increasing the likelihood that the submitted design concept will ultimately be accepted.
- `Contributor` **Website integration**: Design concepts and other equivalent documentation changes are submitted as a change to the documentation on the Apertis website and should also be generated by the website CI/CD as a PDF to aid with review. Documents should be formatted in Markdown and follow the relevant guidance[10].

[9] https://www.apertis.org/policies/contributions/#concept-design-document-template
[10] https://gitlab.apertis.org/docs/apertis-website/-/blob/master/README.md

Pre-submission

General

- `Contributor` **The proposed changes should be broken down into 1 or more atomic commits**[11]: The addition of a new concept design may be presented as a single commit, however is likely that many code changes should be broken down into multiple well described logical commits.
- `Contributor` **Document impact of changes on Apertis**: What is the expected outcome in Apertis? What is it adding? What needs to change? Is anything being removed? Is it expected to cause any regressions?

Code

- `Contributor` **Evaluate whether Apertis is the correct place for the contribution**: Is Apertis the most suitable place to submit the proposed changes in line with Apertis' upstreaming policy[12]?
  In some circumstances important fixes may be acceptable for inclusion in Apertis in parallel with efforts being made to upstream elsewhere, so as to reduce the time taken for the changes to reach Apertis'users.

Review

General

- `Contributor` **Address review comments promptly and fully**: It is likely that most submissions will result in feedback, be that requests or questions. It is expected that a resolution should reached for any feedback prior to a submission being accepted.
- `Maintainer` **License suitability**: Does the contribution meet the license expectations[13] and guidelines[14]?
- `Maintainer` **Evaluate the benefits of accepting the contribution**:
  - Does the benefit of accepting the contribution outweigh the cost of maintaining the changes long-term?
  - Does the change fit the long-term goals of the Apertis project?
  - Does the change well with the goals and objectives of the Apertis project?
- `Maintainer` **The goal of the change adequately explained.** For small code changes a well written commit message will suffice. Larger changes

[11] https://www.apertis.org/guides/app_devel/version_control/#guidelines-for-making-commits
[12] https://www.apertis.org/policies/contributions/#extending-existing-components
[13] https://www.apertis.org/policies/license-expectations/
[14] https://www.apertis.org/guides/licensing/license-applying/

should be accompanied by a merge review description covering the entire merge request. For concept documents, the goal should be adequately explained in the document its self.

- `Maintainer` **Evaluate the impact on Apertis**:
  - What is the expected outcome in Apertis?
  - What is it adding?
  - What needs to change?
  - Is anything being removed?
  - Is it expected to cause any regressions?
- `Maintainer` **Are changes broken down into atomic commits**: Good practice should be followed with regards to commit history[15].
- `Maintainer` **Changes pass on all applicable CI/CD pipelines**: The changes do not cause build regressions on any supported architecture.
- `Maintainer` **Evaluate impact across all supported platforms**: Changes should either be applicable to all supported platforms or care should be taken to evaluate that platform specific changes are made in a way that other platforms are not negatively impacted.
- `Maintainer` **Ensure adherence to Apertis policies**: Changes to Apertis should only be accepted if they adhere to all the relevant Apertis policies.

Code

- `Maintainer` **Check coding conventions**: The contribution should conform to the coding conventions[16].
- `Maintainer` **Evaluate whether Apertis is the correct place for contribution**: Is Apertis the most suitable place to submit the proposed changes in line with Apertis' upstreaming policy[17]?

Component

- `Maintainer` **Ensure that the component doesn't duplicate existing core functionality**: Where possible adding multiple components that implement the same functionality should be avoided as this increases maintenance for little appreciable gain. An exception to this policy exists for developer tools, especially editors.
- `Maintainer` **Component implements the required workflow**: The package repository is structured as described in the component structure

[15] https://www.apertis.org/guides/app_devel/version_control/#guidelines-for-making-commits
[16] https://www.apertis.org/policies/coding_conventions/
[17] https://www.apertis.org/policies/contributions/#extending-existing-components

guide[18], is correctly configured and all applicable CI pipelines succeed.

Concept Design

- `Maintainer` **Concept broadly follows concept design template**: New concept designs should follow the design template where possible. Extra sections may be included and sections removed where appropriate and where as strong argument exists to do so.
- `Maintainer` **Ensure an evaluation of the state-of-the-art been performed**:
  - Has a comprehensive review of alternative solutions been performed?
  - Does the proposed solution seem the best fit for Apertis? (This should take the rationale for inclusion into account.)
- `Maintainer` **Is the approach to meeting the goals is sufficiently clear**: Have the impact of the proposed changes been fully considered. Is it understood how it will work.

Post-acceptance

- `Contributor` **Continue to support the changes that have been made**: Where commitments have been made regarding support of changes to the project, these should be honored.
- `Maintainer` **Maintain changes as long as possible**: Changes added to Apertis should be maintained where possible for as long as they are meaningful to the project.

If submitting a large patch set, consider whether it can be broken down into several stages, ensuring that any feedback from reviews of earlier stages are applied to subsequent ones.

As a rule of thumb start with a lean design/change and submit it for review as early as possible.

You can send a new design for review to the same channels used for a component contribution[19].

[18] https://www.apertis.org/guides/app_devel/component_structure/
[19] https://www.apertis.org/guides/app_devel/development_process/