OpenAMP Governance

# Introduction

(From Stefano’s email:)

I assume that Linaro will provide a template for the governance, which could be a good start. Also, if we like the Linux-style merit-based approach, the Xen Project governance could be a model too (just because it is one of the few Linux-style projects that has a properly documented governance, QEMU and Linux don't have it):

<https://xenproject.org/developers/governance/>

We would need to remove parts we don't need such as the section on sub-projects and the voting system, but the rest seems somewhat applicable.

# Code of Conduct

Just adopt one from another project? Does Linaro have a model for these?

Stefano:

This is extremely difficult to get right without offending anybody, but fortunately there are a few well established examples out there, including:

<https://www.contributor-covenant.org/version/1/4/code-of-conduct.html>

# Consensus and Conflict Resolution

(From Stefano’s email:)

It is good idea to write at least a couple of paragraphs on how we are expected to find consensus and resolve conflicts. Conflicts resolution typically involves a form of voting system, and given that we introduced one at the TSC level, we could document that one for a start. Given that we have so few maintainers, we could avoid getting into conflicts resolutions between maintainers for the moment and just say that the TSC could arbitrate in case of conflicts between community members.

# Strategic/Technical Direction

## Board of Directors

## Steering Committee

# Membership

# Development Process

## Maintainers

(From Stefano’s email:)

One key point is whether it is the TSC that appoints new maintainers or whether it is the existing maintainers that appoint new maintainers. In a Linux-style project, such as the Linux kernel, QEMU, and Xen (probably Yocto too?) it is the existing maintainers that decide based on merit.

The benefit of having existing maintainers decide based on merit is that it encourages a skill-based selection. If the TSC decides, it could encourage a selection based on the employer. However, given the current size and status of OpenAMP, I think the Linux model alone wouldn't work right. We might have to come up with a mixture of both.

## Coding Standards

### C Version (C11 OK?)

### MISRA

### Style guidelines

The OpenAMP uses the Linux kernel coding style and has the Linux kernel checkpatch.pl script as part of its continuous integration process.

## Interface Stability Policy

### API Stability

#### Scope

The scope of the OpenAMP API lifecycle policy is mostly the libraries and supporting code owned by the project. Today that is primarily the open-amp and libmetal libraries but we expect that to expand in the future.

Specifically out of scope would be the APIs internal to the Linux kernel. The Linux kernel has a famous "no stable api" policy. Probably also out of scope would be the kernel / user interface. This interface has a more stringent "don't break the user interface policy" than will be described here.

#### Adding New APIs

TBD

#### Deprecating Existing APIs

If an API is to be removed, it must remain in a deprecated state for 2 year (4 releases). In extraordinary cases such as a critical security issue, the TSC may impose a shorter period.

#### Kernel vs. Library Compatibility

The kernel and library implementations of OpenAMP must be interoperable at all times. Extensions on either side must be downward compatible.

### Wire Protocol Stability

This is a copy/paste from Bill Mills’ email:

The wire protocol describes the interactions of two independent CPUs in an AMP system. The bar for maintaining compatibility at this level is much higher.

Some random thoughts/suggestions:

(I am using "firmware" and "kernel" as stand-ins for the more general cases also)

\* break of compatibility of the protocol level should require TSC approval

\* New kernels should run old firmware

\* New firmware that has extra features with new kernels should still work with old kernels if that is desired

\* We need to know what properties of interaction are guaranteed vs just how the current implementation works today

\*\* We need a protocol document

\*\* Until we do, we have to treat current UPSTREAM model as the defacto standard

\*\* We can't really have protocol compatibility states until we have a protocol document

\*\* Anything currently only in vendor branches is considered Experimental

\* New Protocol features and versions are best if they can be discovered live

\*\* However manual opt-in by both parties is acceptable as a fallback

## Development Workflow

All development will be done using the standard GitHub workflow of pull requests.

### Bug Tracking

Via GitHub issues.

### Commit Process

Pull requests will be merged by the maintainers after review.

### New Feature Development

Proposed new features should be discussed actively on the mailing list and the top features selected for the next release. The Technical Steering Committee can also provide direction on feature development.

### Testing

### Documentation

## Certification Concerns

### Safety

### Security

## Branching and Tagging Strategy

### Development on master

The master branch is where all accepted changes are first committed. The CI loop is primarily focused on this branch.

### Release branches

A release branch is created from the master branch at the time of feature freeze for the release.

### Release Tags

At the time of the release, the release branch will be tagged. This will help enable reproducible builds.

## Releases

### Release Frequency

Releases will be twice yearly, in the spring and fall (usually mid-April and mid-October).

### Naming Conventions

Releases will be named yyyy.mm (e.g. 2019.10).

### Release Process

As the OpenAMP project is rather small as open-source projects go, the release process can be fairly simple. The phases and milestones are listed below. This information should be maintained on the Wiki page for the project.

An alternative approach to what is described below is the Yocto Project development methodology: <https://wiki.yoctoproject.org/wiki/Development_Methodology>

#### Discussion and Development

This is for discussion and development of new features. It starts as soon as the release branch is created. During this period, new features can be committed after the usual review.

#### Feature Freeze

This occurs at about the four-month mark in the release cycle. The release branch is created from the master at this point.

#### Debug

After Feature Freeze, no new features can be committed to the release branch, only bug fixes.

#### Code Freeze

At this time, no further commits may be made to the release branch unless they fix a critical bug found in testing.

#### Release

The release branch is tagged and the code archived and made accessible for download.

#### Release Maintenance

See <https://wiki.yoctoproject.org/wiki/Stable_branch_maintenance>

The primary focus for stable branches is bugfixing, security updates, and making sure that builds on recently released Ubuntu, Fedora, and OpenSUSE distros work. All other kinds of patches (e.g, performance improvements) have a high bar to reach for them to be accepted - i.e, it will have to be very clear they pose little risk to introducing more bugs or stability issues. Anything that breaks APIs or compatibility is not appropriate.

General policies:

* Fixes *must* go into master first unless they are applicable only to the stable branch; if back-porting to an older stable branch, the fix should first be applied to the newer stable branches before being back-ported to the older branch.
  + In order to help make it clear when this policy has been followed cover letters for stable pull requests *must* include information for each patch detailing whether it's a backport, which other branches require the fix and (equally as importantly) when appropriate why the patch isn't required in other maintained branches.
* All CVE (security) patches should be back-ported if at all possible, if a CVE fix is only appropriate to a stable branch the patch submission should detail why this is the case.
* No recipe upgrades unless:
  + The old version is completely broken
  + The new version contains a security patch or other critical bugfix that is too difficult to backport to the version already in the stable branch

As always, the stable branch maintainers' judgement is important when it comes to deciding which fixes are or are not appropriate. If there is doubt, feel free to consult with the overall project maintainer (Richard Purdie <richard.purdie@linuxfoundation.org>).