Apache Forrest project guidelines

Table of contents

This document provides the guidelines under which the Apache Forrest project operates. It defines the roles and responsibilities, who may vote, how voting works, how conflicts are resolved, etc. Apache Forrest is a project of the Apache Software Foundation (ASF) which is a non-profit corporation. As with all such organistions there are some procedures to be followed. The ASF website explains the operation and background of the ASF. These project guidelines supplement that ASF documentation. Normally these guidelines are not needed - the project just gets on with its day-to-day operation - but they enable all people to understand how the project operates.

1. The mission of Apache Forrest

The generation of aggregated multi-channel documentation, maintaining a separation of content and presentation.

2. The Apache Way

Forrest is typical of Apache projects, in that it operates under a set of principles known collectively as the "Apache Way". This facilitates open collaborative development, with respect for others. For more information about how Apache projects operate, please refer to the ASF foundation and ASF developer sections of the ASF website, including the ASF ByLaws and the How it works document, the FAQs about the Foundation, and the Incubator project.

3. Roles and responsibilities

The meritocracy enables various roles as defined in the How it works document.

user | developer | committer | PMC member | ASF member

The current Apache Forrest committers and PMC members are listed.

4. Project Management Committee (PMC)

The Apache Forrest project was established in January 2002 and became a top-level project in May 2004. The Project Management Committee (PMC) was created by a resolution of the board of the Apache Software Foundation. See explanation of the role of the PMC in that resolution and also the ASF Bylaws and How-it-works.

The responsibilities of the PMC include:

- Be familiar with these project guidelines, and the ASF Bylaws, and with the ASF documentation and procedures in general.
- Keep oversight of the commit log messages and ensure that the codebase does not have copyright and license issues.
- Resolve license disputes regarding products of the project, including other supporting software that is re-distributed.
- Decide what is distributed as products of the project. In particular all releases must be approved by the PMC.
- Guide the direction of the project.
- Strive for and help to facilitate a harmonious productive community.
- Nominate new PMC members and committers.

- Maintain the project's shared resources, including the codebase repository, mailing lists, websites.
- Speak on behalf of the project.
- Maintain these and other guidelines of the project.

The PMC does have a private mailing list on which it can discuss certain issues. However this list is rarely used and every effort is made to conduct all discussion on the public mailing lists.

Membership of the PMC is by invitation only and must receive consensus approval of the active PMC members.

A PMC member is considered "emeritus" by their own declaration or by not contributing in any form to the project for over six months. An emeritus member may request reinstatement to the PMC. Such reinstatement is subject to consensus approval of the active PMC members. Membership of the PMC can be revoked by unanimous consensus of all active PMC members (other than the member in question).

The chair of the PMC is appointed by the Board and is an officer of the ASF (Vice President). The chair has primary responsibility to the Board, and has the power to establish rules and procedures for the day-to-day management of the communities for which the PMC is responsible, including the composition of the PMC itself. The chair reports to the board every three months about the status of the project. The PMC may consider the position of PMC chair annually and may recommend a new chair to the board. Ultimately, however, it is the board's responsibility who it chooses to appoint as the PMC chair. See further explanation of the role of the chair in the ASF Bylaws and the PMC FAQ

4.1. Quarterly reports to ASF Board

Every three months, it is the responsibility of our PMC chair to send a report to the ASF Board. This is mainly concerned with the status of our community, but can also include the technical progress. Further details are in the "committers" SVN in the /board/ directory.

The minutes are available for each board meeting. Our reporting schedule is: Feb, May, Aug, Nov.

4.2. Electing new committers and PMC members

We conduct the vote on the private PMC mailing list to enable a frank discussion. In most cases we will be inviting people to go straight from developer to PMC member, i.e. they simultaneously become committer and PMC member. However, there may be extraordinary cases where we want limited work-related commit access (not also a PMC member). This will be resolved during the discussion and vote. Notes about this process are in the "committers" svn in the pmc/forrest/ directory.

5. Decision making

Different types of decisions require different forms of approval. For example, the previous section describes several decisions which require "consensus approval". This section defines how voting is performed, the types of approval, and which types of decision require which type of approval.

Most day-to-day operations do not require explicit voting - just get on and do the work. See the "Lazy approval" type described below.

5.1. Voting

Certain actions and decisions regarding the project are made by votes on the project development mailing list. Where necessary, PMC voting may take place on the private PMC mailing list.

Votes are clearly indicated by subject line starting with [VOTE]. Discussion and proposal should have happened prior to the vote. Voting is carried out by replying to the vote mail. See voting procedure below. Votes are expressed using one of the following symbols:

+1	"Yes," "Agree," or "the action should be performed." In general, this vote also indicates a willingness on the behalf of the voter to assist with "making it happen".
+0	This vote indicates a willingness for the action under consideration to go ahead. The voter, however will not be able to help.
-0	This vote indicates that the voter does not, in general, agree with the proposed action but is not concerned enough to prevent the action going ahead.
-1	This is a negative vote. On issues where consensus is required, this vote counts as a veto. All vetoes must contain an explanation of why the veto is appropriate. Vetoes with no explanation are void. It may also be appropriate for a -1 vote to include an alternative course of action.
abstain	People can abstain from voting. They can either remain silent or express their reason.

All participants in the project are encouraged to show their preference for a particular action by voting. When the votes are tallied, only the votes of PMC members are binding. Non-binding votes are still useful to enable everyone to understand the perception of an action by the wider community.

Voting can also be applied to changes made to the project codebase. These typically take the form of a veto (-1) in reply to the commit message sent when the commit is made.

5.2. Types of approval

Different actions require different types of approval:

Consensus approval	Consensus approval requires 3 binding +1 votes and no binding vetoes.
Lazy majority	A lazy majority vote requires 3 binding +1 votes and more binding +1 votes than -1 votes.
Lazy approval	An action with lazy approval is implicitly allowed unless a -1 vote is received, at which time, depending on the type of action, either lazy majority or consensus approval must be obtained.
2/3 majority	Some actions require a 2/3 majority of active PMC members. Such actions typically affect the foundation of the project (e.g. adopting a new codebase to replace an existing product). The higher threshold is designed to ensure such

	changes are strongly supported. To pass this vote requires at least 2/3 of binding vote holders to vote +1
Unanimous consensus	All voters with binding votes must vote and there can be no binding vetoes (-1).

5.3. Vetoes

A valid veto cannot be over-ruled, it can only be withdrawn by its issuer. Any veto must be accompanied by reasoning and be prepared to defend it.

The validity of a veto, if challenged, can be confirmed by anyone who has a binding vote. This does not necessarily signify agreement with the veto - merely that the veto is valid. In case of disputes about whether a veto is valid, then opinion of the PMC chair is final.

If you disagree with a valid veto, then you must engage the person casting the veto to further discuss the issues. The vetoer is obliged to vote early and to then work with the community to resolve the matter.

If a veto is not withdrawn, the action that has been vetoed must be reversed in a timely manner.

5.4. Actions

This section describes the various actions which are undertaken within the project, the corresponding approval required for that action, and those who have binding votes over the action.

Action	Description	Approval	Binding Votes
Code change	A change made to a codebase of the project by a committer. This includes source code, documentation, website content, etc.	Lazy approval	Active PMC members
Release plan	Defines the timetable and actions for a release.	Lazy majority	Active PMC members
Product release	When a release of one of the project's products is ready, a vote is required to accept the release as an official release of the project.	Lazy majority	Active PMC members
Adoption of new codebase	When the codebase for an existing, released product is to be replaced with an alternative codebase. If such a vote fails to gain approval, the existing code base will continue. This also covers the creation of	2/3 majority	Active PMC members

	new sub-projects within the project.		
New committer	When a new committer is proposed for the project.	Consensus approval	Active PMC members
New PMC member	When a new member is proposed for the PMC.	Consensus approval	Active PMC members
Reinstate emeritus member	An emeritus PMC member can be reinstated.	Consensus approval	Active PMC members (excluding the member in question)
Committer removal	When removal of commit privileges is sought.	Unanimous consensus	Active PMC members (excluding the committer in question if a member of the PMC)
PMC member removal	When removal of a PMC member is sought. See also section 6.5 of the ASF Bylaws whereby the ASF Board may remove a PMC member.	Unanimous consensus	Active PMC members (excluding the member in question)

5.5. Voting timeframes

Votes are normally open for a period of one week to allow all active voters time to consider the vote. If the vote has not achieved a quorum, then it can be extended for another week. If still no quorum, then the vote fails, and would need to be raised again later. Votes relating to code changes are not subject to a strict timetable, but should be made as timely as possible.

5.6. Voting procedure

Discussion about the topic would have already happened in a [Proposal] email thread to express the issues and opinions. The [Vote] thread is to ratify the proposal.

The instigator sends the Vote email to the dev mailing list. Describe the issue with no ambiguity and in a positive sense. Define the date and time for the end of the vote period.

Votes are expressed by replying email using the voting symbols defined above. Voters can change their vote during the timeframe. At the end of the vote period, the instigator tallies the number of final votes and reports the results.

5.7. Ultimatum and breakdown

For breakdown situations and those requiring unanimous consensus, if this consensus cannot be reached within the extended timeframe, then the Board expects the chair to act as the officer of the Foundation and make the ultimate decision.

6. Communication channels

The primary mechanism for communication is the mailing lists. Anyone can participate, no matter what their time zone. A reliable searchable archive of past discussion is built. Oversight is enabled. Many eyes ensures that the project evolves in a consistent direction.

All decisions are made on the "dev" mailing list.

The main channel for user support is the "user" mailing list. As is usual with mailing lists, be prepared to wait for an answer.

Occasionally we will use other communication channels such as IRC. These are used only for a specific purpose and are not permanently available. This policy ensures that solutions are available in the mailing list archives and enables people to respond at whatever time that they choose. Permanent IRC channels are poor from a community-building point-of-view, as they tend to create time-zone based cliques. So we don't.

Similarly, private discussions are discouraged. The rest of the community would not benefit from the understanding that is developed. Off-list discussions put too much load on overworked volunteers.

7. Code management

The term "patch" has two meanings: Developers provide a set of changes via our Issue Tracker marked for inclusion, which will be applied by a committer. Committers apply their own work directly, but it is still essentially a patch.

We use the Commit-then-review method for decision-making about code changes. Please refer to that glossary definition. Note that it does not preclude the committer from making changes to patches prior to their commit, nor mean that the committer can be careless. Rather it is a policy for decision-making.

There is an important issue where both developers and committers need to pay special attention: "licenses". We must not introduce licensing conditions that go beyond the terms of the Apache License. If such issues do creep in to our repository, then we must work as quickly as possible to address it and definitely before the next release.

There are some other problem areas: What should a committer do if the patch is sloppy, containing inconsistent whitespace and other code formatting, which mean that actual changes are not easily visible in the svn diff messages. What if there is poorly constructed (yet working) xml or java code? What if the new functionality is beyond the scope of the project? What if there is a better way to do the task? What if the patch will break the build, thereby preventing other developers from working and causing an unstable trunk?

The committer has various options: ask the developer to resubmit the patch; change the patch to fix the problems prior to committing; discuss the issues on the dev list; commit it and then draw attention to the issues so that the rest of the community can review and fix it. A combination of these options would appear to be the best approach. Please aim to not break the build, or introduce license problems, or make noisy changes that obscure the real differences.

Committers should not be afraid to add changes that still need attention. This enables prompt patch application and eases the load on the individual committer. An interesting side-effect is that it encourages community growth.