Project Guidelines

@version@ (@date@)

by Ted Husted, modified by Dieter Wimberger

Note:

These project guidelines have been adapted from the Jakarta Project Guidelines (as well as their working proposal), which you can find at:
http://jakarta.apache.org/site/guidelines.html
http://jakarta.apache.org/site/proposal.html
Modifications were required to adapt the wording to a single non-Jakarta project (i.e. not subproject, and not subject of supervision by the PMC).
The material is copyright by the jakarta project of the Apache Foundation and is published with permission from the PMC.

1. Overview

This document defines the guidelines of the project. It includes definitions of the various categories of membership, who is able to vote, how conflicts are resolved by voting and the procedures to follow for proposing and making changes to the codebases of the project.

- Roles and Responsibilities
  Defines the recognized roles in the project
  - Users
  - Contributors
  - Committers
  - Release Manager
- Communication
  Defines how users and developers communicate
  - Announcement Lists
  - User Lists
  - Developer Lists
  - Commit Lists
- Decision Making
  Defines how action items are proposed and voted on.
  - Action Item
    - Long Term Plans
    - Short Term Plans
    - Release Plan

Copyright © All rights reserved.

    - Release Testing
    - Public Release
    - Showstoppers
    - Product Changes
  - Voting
    - Action Item Votes
    - Voting Caveats
    - Other Items
    - Voting on Other Matters
  - Branches
- Source Repository

  Defines how the Project's source code is organized and developed.
  - License
  - Status Files
  - Branches
  - Changes
  - Patches

2. Roles and Responsibilities

The roles and responsibilities that people can assume in the project are based on merit. Everybody can help no matter what their role. Those who have been long term or valuable contributors to the project obtain the right to vote and commit directly to the source repository.

2.1. Users

Users are the people who use the products of this project. People in this role aren't contributing code, but they are using the products, reporting bugs, making feature requests, helping other users, extending an online FAQ and such. This is by far the most important category of people as, without users, there is no reason for the project.
When users start to contribute code or documentation patches, they become contributors.

2.2. Contributors

Contributors are the people who write code or documentation patches or contribute positively to the project in other ways. A volunteer's contribution is always recognized. In source code, all volunteers who contribute to a source file may add their name to the list of authors for that file.

2.3. Committers

Copyright © All rights reserved.

Contributors who give frequent and valuable contributions to the project can have their status promoted to that of a "Committer". A committer has write access to the source code repository and gains voting rights allowing them to affect the future of the project.
In order for a Contributor to become a Committer, another Committer can nominate that Contributor or the Contributor can ask for it. Once a Contributor is nominated, all of the Committers of this project will vote. If there are at least 3 positive votes and no negative votes, the Contributor is converted into a Committer and given write access to the source code repository for the project.
Before receiving write access, a Committer must also affirm that they have read and understood these guidelines, and agree to abide by their terms, as they may be revised from time to time.

At times, Committers may go inactive for a variety of reasons. A Committer that has been inactive for 6 months or more may lose his or her status as a Committer.

A list of some of our current Committers can be found in our project credits.

2.4. Release Manager

Product releases will be managed by a selected committer. A release manager should actively manage the release process, watch the deadlines, release and make announcements.

3. Communications

The project obtains its strength from the communication of its members. In order for members to easily communicate with each other, the project has mailing lists as well as discussion fora.
These lists, with the exception of the announcement lists, are not moderated and anybody is more than welcome to join them. However, you must be subscribed to post to a list. To reduce the bandwidth demands on everyone, mail should not contain attachments. It is recommended that you place interesting material (such as patches) either within the body of the message or provide a URL for retrieval.
To join the mailing lists, see our Mailing List page.

The project's list fall into the following categories:

Announcement Lists
Announcement lists are very low traffic designed to communicate important information, such as releases of a product of this project, to a wide audience.

User Lists

Copyright © All rights reserved.

User lists are for users of a product to converse about such things as configuration and operating of the products of the project.

Developer Lists

Developer lists are for the contributors to the project. On these lists suggestions and comments for code changes are discussed and action items are raised and voted on. For the developer community, these lists are the very center of the project where all the "action" is.

Commit Lists

The commit lists are where all CVS code commit messages are sent. All committers are required to subscribe to this list so that they can stay aware of changes to the repository.

4. Decision Making

All Contributors are encouraged to participate in decisions, but the decision itself is made by those that have Committer status in the project.
In other words, the project is a *"Minimum Threshold Meritocracy"*.

4.1. Action Items

All decisions revolve around *"action items"*. Most action items require a vote to be approved. Votes can either be by majority or by consensus.

Action items include the following:

- Long Term Plans: No vote required.
- Short Term Plans: No vote required.
- Release Plan: Lazy majority vote on each issue, or lazy consensus if issue involves a product change.
- Release Testing: No vote required, once Release Plan is approved.
- Public Release: Majority vote, if no showstoppers are listed in the status file.
- Showstoppers: Lazy consensus until resolved and removed from status file.
- Product Changes: Lazy consensus.

Long Term Plans

Long term plans are simply announcements that group members are working on particular issues related to the Project. These are not voted on, but Committers who do not agree with a particular plan, or think that an alternative plan would be better, are obligated to inform the group of their feelings.

Short Term Plans

Copyright © All rights reserved.

Short term plans are announcements that a volunteer is working on a particular set of documentation or code files with the implication that other volunteers should avoid them or try to coordinate their changes.

Release Plan
A release plan is used to keep all volunteers aware of when a release is desired, who will be the Release Manager, when the repository will be frozen to create a release, and other assorted information to keep volunteers from tripping over each other. Lazy majority decides each issue in a release plan, or lazy consensus if the issue involves a product change.

Release Testing
After a new release is built, it must be tested before being released to the public. After the release plan is approved, the Release Manager will announce that a product is ready for testing.

Public Release
Once the Release Manager determines that testing is complete, and all showstoppers for the release have been resolved, the Release Manager shall call for a vote on the public release. Majority approval is required before the public release can be made. The Committers who approve a public release (vote **+1**) are expected to provide ongoing support for that release while it is current. The Release Manager must summarize the outcome of the vote before the public release becomes final.

Showstoppers
Showstoppers are issues that require a fix be in place before the next public release. They are listed in the status file in order to focus special attention on these problems. An issue becomes a showstopper when it is listed as such in the status file and remains so by lazy consensus.

Product Changes
Changes to the products of the project, including code and documentation, will appear as action items in the status file. All product changes to the currently active repository are subject to lazy consensus.

4.2. Voting

4.2.1. Action Item Votes

The act of voting on an action item carries certain obligations. Voting members are not only stating their opinion, they are also agreeing to help do the work.

Copyright © All rights reserved.

Any Contributor or Committer ("member") may call for an action-item vote on the Developer mailing list. It is preferred that a vote be preceded by a formal proposal offered for discussion purposes. The message announcing a vote should contain a Subject beginning with "[VOTE]", and a distinctive one-line summary corresponding to the action item for the vote.

Each vote on an action item can be made in one of four flavors:

+1	"The action should be performed, and I will help."
+0	"Abstain", "I support the action but I can't help."
-0	"Abstain", "I don't support the action but I can't help with an alternative."
-1	"The action should not be performed and I am offering an explanation or alternative."

Votes cast by the project's committers are considered "binding". When needed, at least 3 binding **+1** votes are required for approval of an action item.

An action item may need one of two types of approval:

1. *Consensus approval:*
   An action requiring consensus approval must receive at least **3 binding +1** votes and **no binding vetos**.
2. *Majority approval:*
   An action requiring majority approval must receive at least **3 binding +1** votes and more **+1** votes than **-1** votes.

Except for a public release, all action items are considered to have lazy approval until somebody votes **-1**, at which point the action item is converted to a formal consensus or majority vote, depending on the type of action item .

Note:

"Lazy" means the action item has immediate tactic approval, and it is not necessary to tally the vote until a **-1** reply is posted. Once a **-1** reply is posted, the vote must be tallied and reported before the action item is considered approved.
All action-item votes are lazy except for a public release vote.

Any member may vote on any action item or related issue. When voting on an action item, the project's committers may also include the word "binding" next to their vote, to simplify a tally if it is needed. All binding vetos **must** also contain an explanation of why the veto is appropriate.

Copyright © All rights reserved.

4.2.2. Voting Caveats

- A **+1** vote regarding product code can only be made binding if the voter has tested the action on their own equipment.
- A binding **+1** vote on a public release means the project's committer agrees to provide ongoing support for that release while it is current.
- An abstention may have detrimental effects if too many people abstain.
- Vetos with no explanation are void. If you disagree with the veto, you should lobby the person who cast the veto. Voters intending to veto an action item should make their opinions known to the group immediately so that the problem can be remedied as early as possible.
- If a committer believes the explanation for a veto is invalid, an affirmation of the veto can be requested. If some other committer does not affirm that the explanation for the veto is valid, the veto shall be void.
- Members who wish to discuss a vote before replying, may open another thread to help avoid premature vetos. Any **+/-1**'s or **+/-0**'s posted to an alternate thread, or any other thread not labeled "[VOTE]", are considered conversational, and **do not** qualify as an valid action-item vote. A "lazy item" remains subject to lazy approval until a valid **-1** reply is posted to the "[VOTE]" thread.

4.2.3. Vote Results

Most action items are subject to lazy approval, and do not require the posting of a formal result. However, any other majority item that receives any **-1** reply (later rescinded or not) must be followed by a "[VOTE-RESULT]" message before the action item is considered approved.

Likewise, any consensus item that receives any binding veto must post a "[VOTE-RESULT]" message summarizing the result, and show that each veto was rescinded, before it is considered approved. In either case, the member who requested the vote should also post the result within 120 hours (5 days).

A Public Release is not considered approved until the Release Manager posts a followup message with the legend "[VOTE-RESULT]" summarizing the replies.

4.2.4. Proposals (and plans)

Proposals are not a formal action item; however, the message offering a proposal should contain a Subject beginning with "[PROPOSAL]".
It is strongly recommended that a proposal be circulated before calling for a formal vote. Often, once members have had the chance to critique a proposal, an updated copy of a proposal can be reposted as the vote document.

Copyright © All rights reserved.

Most other messages posted to the Developer's List usually involve either short-term or long-term plans. Often, a long-term plan will be made in the form of a "[PROPOSAL]". If appropriate, the proposed change or feature may then be entered to the project's STATUS file or TODO list.

4.2.5. Voting on other matters

There are other matters upon which members will vote that do not involve action items. The same general voting structure is used in these cases, except that the "flavors" are taken to mean:

+1	"Yes", "I agree."
+0	"Abstain", "No opinion."
-0	"Abstain", "Unsure."
-1	"No", "I disagree."

4.3. Branches

In any software development project there is a natural tension between revolution and evolution. Many software development teams, open or closed, have a team leader who can declare whether the code base is in evolutionary or revolutionary mode. In a volunteer meritocracy, this type of decision is difficult to make and impossible to enforce. Our meritocracy is fueled by voluntary contributions, and so we must allow everyone to contribute and then base our final product decisions on the contributions we actually receive.

Accordingly, as a matter of project policy, these principles are adopted:

1. Every committer has the right to revolution. Anyone can establish a branch or seperate whiteboard directory in which to experiment with new code seperate from the main trunk. The only responsibility a committer has when they do this is to announce their short and long term plans and allow others who want to help to do so. Committers working on a revolutionary branch have the right to develop their approach free of interference.
2. When a revolution is considered ready for prime time, the committer(s) shall propose a merge to the developers list. At that time, the overall community evaluates whether or not the code is ready to become part of, or to potentially replace, the trunk. Suggestions may be made, changes may be required. Once all issues have been taken care of and the merge is approved, the new code may become the trunk.
3. All development branches should be unversioned. No branch, evolutionary or revolutionary, should have any official version standing. This allows several parallel

Copyright © All rights reserved.

tracks of development to occur with the final authority of what eventually becomes the trunk resting with the entire community of committers.

4. The trunk is the official versioned line of the project. All evolutionary minded people are welcome to work on it to improve it. Evolutionary work is important and should not stop as it is always unclear when any particular revolution will be ready for prime time or whether it will be officially accepted.

5. Source Repository

The project's codebase is maintained in a shared information repository using CVS. Only Committers have write access to this repository. Everyone has read access via anonymous CVS. All Java Language source code in the repository must be written in conformance to the "Code Conventions for the Java Programming Language" as published by Sun.

5.1. License

All source code committed to the Project's repositories must be covered by the project license or contain a copyright and license that allows redistribution under the same conditions as the project license.

5.2. Status

The project's active source code repository contains a file named *STATUS.xml* which is used to keep track of the agenda and plans for work within the repository. The status file includes information about release plans, a summary of code changes committed since the last release, a list of proposed changes that are under discussion, brief notes about items that individual volunteers are working on or want discussion about, and anything else that may be useful to help the group track progress.

5.3. Branches

It is allowed to create a branch for each release cycle, etc. They are expected to merge completely back with the main branch as soon as their release cycle is complete.

A branch is considered "evolutionary" by lazy consensus. Upon any dispute (binding veto), a branch must be converted to "revolutionary" status, and must be assigned a working name that does not imply it will be the next version of the product. Once a release plan for the branch has been approved, and the proposed release is ready for testing, it may be assigned a version name reflecting it is a product release candidate, and merged back with the main branch, as appropriate to the circumstances. Any branch, "evolutionary" or "revolutionary", shall meet the same standard for release approval.

Copyright © All rights reserved.

5.4. Changes

Simple patches to fix bugs can be committed then reviewed. With a commit-then-review process, the committer is trusted to have a high degree of confidence in the change. Doubtful changes, new features, and large scale overhauls need to be discussed before committing them into the repository. Any change that affects the semantics of an existing API function, the size of the program, configuration data formats, or other major areas must receive consensus approval before being committed.

Related changes should be committed as a group, or very closely together. Half complete changes should never be committed to the main branch of a development repository. All code changes must be successfully compiled on the contributor's platform before being committed. The current source code tree for the project should be capable of complete compilation at all times. However, it is sometimes impossible for a contributor on one platform to avoid breaking some other platform when a change is committed. If it is anticipated that a given change will break the build on some other platform, the committer must indicate that in the commit message.

A committed change must be reversed if it is vetoed by one of the voting members and the veto conditions cannot be immediately satisfied by the equivalent of a "bug fix" commit. The veto must be rescinded before the change can be included in any public release.

5.5. Patches

When a specific change to a product is proposed for discussion or voting on the appropriate development mailing list, it should be presented in the form of input to the patch command. When sent to the mailing list, the message should contain a Subject beginning with [PATCH] and a distinctive one-line summary corresponding to the action item for that patch.

Related changes should be committed as a group, or very closely together. Half complete changes should never be committed to the main branch of a development repository. All code changes must be successfully compiled on the contributor's platform before being committed. The current source code tree for the project should be capable of complete compilation at all times. However, it is sometimes impossible for a contributor on one platform to avoid breaking some other platform when a change is committed. If it is anticipated that a given change will break the build on some other platform, the committer must indicate that in the commit message.

The patch should be created by using the diff -u command from the original software file(s) to the modified software file(s). For example:

```
diff -u Main.java.orig Main.java >> patchfile.txt
```


Copyright © All rights reserved.

or

```
cvs diff -u Main.java >> patchfile.txt
```

On Win32 you can use WinCVS for a nice GUI or you can install Cygwin which will enable you to use the bash shell and also installs a lot of other utilities (such as diff and patch) that will turn your PC into a virtual Unix machine.

All patches necessary to address an action item should be concatenated within a single patch message. If later modification to the patch proves necessary, the entire new patch should be posted and not just the difference between the two patches.

Copyright © All rights reserved.