Versioning and Branching

Versioning

Apache Geode loosely follows Semantic Versioning specification (http://semver.org/) since even on major releases we don't break backward compatibility with previous major version. But most principles still apply, such as the model:

[MAJOR].[MINOR].[MAINTENANCE]

Where:

MAJOR: Releases will have major version increase when there are significant features or functionalities that are considered enough for a major project release. All releases with the same major version number will have API compatibility and those releases will stay stable and in use for a longer period of time. (~1+ year) - Requirements for QA of major releases will include regression and complete performance tests. Voting will be casted on the mailing lists and after consensus a release cycle will be initiated for the major release.

MINOR: Minor releases can contain minor new features and must definitely include significant improvements to current API or components that justify not be configured as *maintenance* changes. Minor releases can also be increased if extensions or sub-projects add new features or are updated somehow.

MAINTENANCE: Maintenance releases can occur more frequently and depend on specific patches introduced (e.g. bug fixes) and their urgency. In general these releases are designed to hot-fixes and bugs.

API Compatibility

Changes to public classes or APIs should take into consideration backward compatibility with at least the previous major version in order to support rolling updates on a cluster.

All changes to the public API must be backwards compatible, with the exception of removing deprecated features. Removing deprecated features also requires discussion. API should be deprecated for a major version before being removed.

Geode supports rolling upgrades on a live system. Message serialization and disk persistence must be backwards compatible with previous versions. This includes algorithmic changes that don't affect the serialization format. Geode has a framework for backwards compatible serialization - see Managing backward-compatibility.

Branching

Apache Geode is heavily based on Git-Flow model [1] and [2] - Based on one of the first threads discussed in the project mailing list.

The main concepts are:

1) The **master** branch is reserved for production-ready code, aka releases.

- No changes are made directly on the master branch.
- Master is merged solely from develop, and only by the "release manager".
- Merges can only be done, when build is successful on all supported platforms, and the tests on all supported platforms have passed.
- In other word, if you want to develop an enhancement for Geode, use develop branch explained below.

2) The **develop** branch is the development mainline. Ongoing work shows up here first.

- develop branch holds the latest code
- It is used for active development thus can be unstable time to time (though unit test is enforced)
- Contributors can develop enhancements on this branch and make patches.

3) Feature branches may be used to isolate interim work and will merge back to develop. Public feature branches should only be used, when multiple people work on a feature. A feature branch is started after the intention is made clear on the mailing lists and must be backed by a JIRA ticket.

4) Release branches are created to allow a stabilization period for a release while not impacting ongoing work on develop. Once a release is stabilized the release branch is merged to master and develop and the release branch shall not be used anymore.

5) Hotfix branches are created for a patch from a release (off master) to support ongoing hot fixes and patches as needed, but after completed MUST be merged back into both master and develop. Master should also receive a tag with an updated maintenance version number.

Branch naming pattern:

- master

- develop

- feature/GEODE-nn

- hotfix/GEODE-nn

- release/VERSION

[1] http://nvie.com/posts/a-successful-git-branching-model/

[2] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow