

GEP 1 - Groovy Enhancement Proposal

Metadata

Number:	GEP-1
Title:	Groovy Enhancement Proposal
Version:	1
Type:	Informational
Status:	Draft
Leader:	Guillaume Laforge
Created:	2009-03-26
Last modification:	2009-03-26

Abstract: What is a GEP?

GEP stands for **Groovy Enhancement Proposal**. A GEP is a document providing information to the Groovy development team and user community describing a new feature, an enhancement, or a change to [Groovy](#), the language, its APIs, its integration, or its infrastructure. Whenever such an addition or change is significant and deserves a detailed discussion on its rationale, its design, or its impact on the project, the writing of a GEP should be considered.

The concept of the GEP was freely inspired from [Python's PEP](#) (Python Enhancement Proposal).

Rationale: Why a GEP?

For non-trivial, complex or strategic features, discussions on mailing-lists are difficult to lead and follow, and often don't help reach a consensus. Writing a proper document explaining the design and implications of said feature allows both the originator of the idea and the community at large to have a chance to provide interesting and useful feedback and helps better understanding the rationale, the design decisions, the impact of the proposal.

What's in a GEP?

A GEP is:

- led by a "**Leader**" who is responsible for the writing and progress of the proposal
- assigned a unique "**Number**"
- has got a "**Title**" describing succinctly its intent
- has a "**Type**"
- has a "**Status**" giving information on its progress
- has a "**Version**" number indicating its current revision
- gives a "**Last modification**" date

- gives a "**Creation**" date
- features an "**Abstract**" explaining the intent of the GEP
- gives a "**Rationale**" for this enhancement

The type of a GEP can be of the following:

- "**Informational**": if the GEP provides some information or guidance on a topic related to Groovy (this GEP is of such type, as well as the list of all existing GEPs)
- "**Feature**": if the GEP is about the implementation of a new feature, enhancement or a change in Groovy
- "**Process**": if the GEP describes a process related to the Groovy project (examples: the Groovy release process, how to enroll new committers, etc.)

If the GEP is a **Feature** GEP, it should also:

- define a "**Target**" Groovy version for its integration
- provide a "**Reference implementation**" properly covered by unit tests and commented (JavaDoc comments as well as inline comments) so that the Groovy community can play with the enhancement and provide useful feedback
- detail the "**Impact**" on Groovy, especially in terms of backward-compatibility

The "Status" of a GEP can be:

- "**Draft**": when a GEP is currently in the writing and is in discussion but hasn't yet reached a state ready for inclusion in Groovy
- "**Accepted**": when the draft GEP, as is, is in a state which doesn't mandate any additional modification, is ready to be integrated into Groovy and has been accepted by the development team, following the usual [Cod ehaus Manifesto](#) principles
- "**Rejected**": when consensus emerges or a development team decision has been made that deems the proposal should not be integrated into Groovy
- "**Final**": when the GEP has been integrated into a released version of Groovy and has been properly documented in the Groovy wiki

Beyond all these metadata, as appropriate, a GEP should also:

- give pointers to existing mailing-list discussions (through Nabble or Markmail)
- list existing papers or documentations related to this feature that provides additional material for understanding the concepts or implementation difficulties
- provide samples showing how the feature should be used and how idiomatic Groovy the solution is

The general workflow

The general workflow of a Groovy Enhancement Proposal is as follows:

- The inception of a GEP generally stems from a discussion on the Groovy mailing-lists about a new feature or change when the Groovy Despot and the development team deem necessary to properly document and articulate this new idea, for further discussion and analysis. Generally, simple bug fixes and minor enhancements don't require a full-blown GEP. A GEP is started solely with the agreement of the Groovy Despot.
- A unique **Number** is assigned by the Groovy Despot and a meaningful "Title" is created for this GEP.
- A proposed **Target** Groovy version is proposed for the inclusion of this GEP.
- A **Leader** for this effort (usually the instigator of the GEP or the lead developer of it) takes care of the creation of the GEP, of writing a detailed document proposing this enhancement, and eventually of providing a reference implementation as a branch or through a patch for the targeted Groovy version.
- Once the development team and the "Leader" are happy with the status of the GEP, a decision should be made to act the integration of this GEP into Groovy and a JIRA issue for this task should be created. As per

the Codehaus Manifesto rules, the Groovy Despot has the last say on the acceptance of the GEP as a **Final** status GEP.

- The integration of the GEP in Groovy is **Final** once proper documentation is added to the Groovy wiki and that the reference implementation has reached maturity and provides a good test suite covering the feature.
- If the GEP doesn't fulfill all the requirements of the GEP process, the GEP can be **Rejected**.

References

Mailing-list discussions

- [\[groovy-dev\] Groovy DevCon #3](#)

JIRA issues

- [GROOVY-1709: GEP: Groovy Enhancement Proposal](#)

Useful links

- [Python's PEP](#)
- The [Codehaus Manifesto](#) principles