

GDC 3 report

Groovy DevCon #3 report

Introduction

The third Groovy Developer Conference took place in Paris, in Sun's offices, on the 29th and the 30th of January 2007. This event was kindly sponsored by the Codehaus Foundation and the room booked by Alexis Moussine-Pouchkine.

Were present:

- Guillaume Laforge (Groovy Project Manager)
- Jochen Theodorou
- Dierk König (author of Groovy in Action)
- John Wilson
- Jeremy Rayner
- Graeme Rocher (Grails Project Lead)
- Guillaume Alléon
- Emmanuel Bernard (from JBoss)

During this meeting, we followed the agenda outlined [there](#).

The meeting has been productive and the resulting decisions will be explained in the following sections.

Naming scheme

First of all, we have decided to keep the current naming / number scheme of betas and RCs. The number of betas and RCs hasn't been decided yet.

The next major milestone will be 1.1 (and not 2.0 as we might have supposed). We are aiming at releasing this 1.1 before the end of 2007, ideally in Q3. The next versions will thus be:

- 1.1-beta-1
- 1.1-beta-2
- 1.1-beta-3
- 1.1-rc-1
- 1.1-rc-2
- 1.1 (final release)

We might however provide some 1.x.y releases in case some critical bugs are found that can't wait for the next major milestones.

Branching scheme

So far, we haven't felt a strong need to create branches. We will continue to work on SVN trunk to add new features as well as for fixing bugs. This will not complicate our work for dealing with different branches.

"GEP": Groovy Extension Proposal

For further enhancements / additions / changes proposals, we should formalize them in the form of a dedicated document. Discussing such proposals is often tedious on mailing-lists because we often miss the "big picture" or the tricky details, and the discussion might finish in a dead end.

The proposal should provide some context of why such an enhancement is provided, explain the way it solves a certain problematic, code samples, corner cases, ideas for the implementation or even a prototype implementation as far as possible.

The details of such a template for proposals should be provided soon.

Sources and tests reorganization and distribution

We are going to reorganize the sources and tests to separate:

- Regression tests from 1.0 from new tests, or from TCK related tests
- Language-related classes, from library-related ones

This also means we should separate more cleanly:

- The core language classes
- The library classes
- The module content

Core languages classes are everything related to the core concepts of the language as well as the GDK. Library classes contain classes related to various APIs like JMX, Swing, and such. While module content usually require additional dependencies than the ones of the JDK and also usually follow their own release cycle.

Separating sources also means creating different artifact deliverables:

- groovy-core (including "jarjar-ed" ASM and Antlr)
- groovy-gdk

Having a small groovy-core library is particularly interesting for those wishing to integrate Groovy in their Java applications.

Regarding the tests, we should separate the 1.0 tests in a dedicated folder, while creating new folders for including "Groovy in Action" tests, new tests for 1.1 (with tests specific to Java 1.5 features), as well as tests for the TCK of JSR-241.

The ASL 2 license has to be applied and enforced everywhere -- instead of copying the old BSD license again and again.

Documentation

Despite the great progresses made on improving, adding new content, and reorganizing the documentation, more care should be provided to the online documentation in the future. Newcomers should easily find their way through the website and should be able to get started quickly. Non-documented features should be identified and reported so that actions can be taken to improve those areas.

A particular care should also be provided to improve the JavaDoc of the core Groovy classes.

Build system

Our Maven 1 build system is very fragile and is too complicated to make it evolve properly. Having suffered various pains with it in the past that we don't want to enumerate here, and as we haven't got enough feedback from Maven 2 projects for complex builds like Groovy's, we decided to take the simplest possible path by choosing Ant as our new build system. We will progressively create the most simplistic build that can possibly exist to build the artifacts and pass the test cases, as well as providing reporting capabilities (test reports, coverage) for integration in our

Continuous Integration process. Users will also be able to very easily build Groovy themselves with raw Ant without having to install anything like Maven or other build tool.

Other facilities such as creating the distributions can potentially be handled differently (Gant or other).

Status of JSR-241

The deliverables of JSR-241 consists of:

- GLS: the Groovy Language Specification (the specification and the formal grammar)
- RI: the Reference Implementation
- TCK: the Test Compatibility Kit

The RI has been released in the form of Groovy 1.0.

The formal grammar of the GLS has been generated out of the Antlr grammar.

The TCK need to be forked from our test suites, with many additional tests added.

The specification in itself will be generated automatically out of the commented test cases and test suites from the TCK.

Tooling support

The Abstract Syntax Tree will be improved to provide better support for tools, especially IDEs, and the "groovydoc" documentation tool. And all help possible should be provided to teams working on IDE plugins.

"groovydoc" should as a first step be able to generate the JavaDoc documentation from the Groovy classes, but should eventually be able to generate documentation for both Java and Groovy classes so that inter-links can be created. More thoughts and discussions might happen to see whether solutions could be found to document classes with dynamic behaviors, like Builders or MetaClasses.

The Groovy console and shell will be revamped to become even more user-friendly.

Syntax enhancements and new features

The grammar will be cleaned up slightly to remove some artifacts of ideas brushed during the past JSR meetings but which have never been implemented or have been rejected after discussions.

Annotation support

Groovy 1.1 will provide annotation support and will integrate seamlessly with Java 5's annotations.

Despite Groovy supporting this particular Java 5 feature, the rest of the project will always stay compatible with bytecode for 1.4 JVMs, except for classes annotated which will be compiled as bytecode for 1.5 JVMs.

Annotated classes will only be able to be compiled when the compiler is run on a JVM 1.5, while the rest of Groovy classes will always be able to compile and run both on 1.4 and 1.5 and beyond.

Specific test suites and probably DGM methods requiring a JVM 1.5 will be separated and will not prevent anyone from using Groovy on a JVM 1.4.

It is important that Groovy stays compatible with 1.4 as much as possible so that Groovy can still be used in corporate environment that have not made the switch yet to more recent versions of their JVMs.

Enum support

Enum will be supported, at least partially.

We may reserve ourselves the right to not provide all the extensibility provided by Java Enums (particularly the capability to define behavior on Enums).

Old for loop

While we usually avoid providing users with too many choices for implementing a given thing, we listened to newcomers wishing to be able to copy and paste some Java code containing "old for loops". Groovy 1.1 will add the ability to use classical for loops as provided by Java or C#.

ExpandoMetaClass

The ExpandoMetaClass pioneered by the Grails project will be introduced in Groovy itself. By default, those MetaClasses should not be extensible by default as they can be dangerous in case of concurrent access and shared environments.

Additionally, a new convention will be introduced to define a script customizing various MetaClasses. For instance, a script in the magic package: `groovy.runtime.metaclass.script` can hold a script using ExpandoMetaClasses to enhance other classes.

Date / time / duration support

The experimental date / time / duration support of the GData module will be back in the core so as to provide a nice syntax for dealing with date and time handling. This will be based on the Java calendar class, as we don't want to add an additional dependency of the core on Joda-Time despite its great merits. But the day JSR-310 finds its way into the JDK, we might certainly upgrade our implementation to using this API, as the underlying implementation should be transparent to the users.

Named-parameters with bracket omissions

While it is possible to call methods without parentheses for top-level statements, it only works for methods with normal arguments, but not with methods taking named-arguments (map literal). To provide even more readability and expressivity, we decided to also allow to omit parentheses in that case, to help with the definition of Domain-Specific Languages.

Multiple assignments

We consider adding the possibility of doing multiple assignments. Methods returning lists could spread the elements of their list to multiple variables at a time. This mechanism may even be used later in more complex scenarios like retrieving groups from a regular expression.

Follow-up

This section contains some additional information on what has happened and been decided till the meeting.

Minimal generics support

Groovy will sports minimal generics support so that declarations like `List<Employee>` adds the relevant reflective bytecode information needed by frameworks like JPA.

Map and Closure coercion extension

Make map and closure coercion work also for classes and abstract classes, and not only for interfaces.

"Elvis" operator for default values

```
c = foo ?: bar // equivalent to c = foo ?  
foo : bar
```