Development Ideas -- Hans Dockter -- 2007

For the next version of Gant we want to introduce project tool support a la Maven. Here you can find a discussion in the groovy user mailing list about this topic.

Basically we want to implement three abstractions:

1. Build lifecycle
2. Multiproject builds
3. Dependency handling

I have submitted a framework (see attached Gant04.zip) for lifecycle handling based on an idea Jochen has brought up. The framework is good enough to get an idea and to discuss it, it is by no means complete. There is no support for multiproject builds yet and only the simplest support for dependency handling. There is no integration with the current Gant yet.

Lifecycle

Lifecycle Definition
1.) A list of phases (Jochens approach) or a set of phases related via dependency relations (Gants approach) 2.) Certain actions that are associated with a phase. (The statements of the target or of the methods above, depending on the approach).

(My) Requirements for the lifecycle handling:

1. Easy customization of the lifecycle (which includes an easy reuse of an existing implementation of a phase).
2. Support of multiple lifecycles
    - Even for the standard cases multiple lifecycles make sense (Jar Lifecycle, War lifecycle, EAR lifecycle, ...)
    - Reuse of a customized lifecycle (in particular helpful for multiproject builds)
    - Customization should also be possible by removing or creating new phases. Of course you can squeeze in any custom behavior by customizing existing phases. But this might be not expressive.

We have been discussing two alternative approaches for lifecycle handling. One is based on Gant targets where the lifcycle is established by the dependencies of the targets. The other is to have a lifecycle class with methods implementing a phase of the lifecycle. I have implemented the latter approach.

Examples of build scripts

```
group = 'org.codehaus.groovy.gant.test'
name = 'test1'
version = 1.0
lifecycle = new org.codehaus.groovy.graven.JarLifecycle() // We could
easily add a method like createLifecycle('jar') to make this more
convenient for the standard lifecycles.
lifecycle.dependencies = ['./src/test/libs/commons-io-1.2.jar'] //
Dependency support is very rudimentary. The only objective has been to make
the examples work.
lifecycle.testDependencies = ['./src/test/libs/junit-3.8.2.jar']
```

This is enough to compile, test and create a jar of our test project.

```
group = 'org.codehaus.groovy.gant.test'
name = 'test1'
version = 1.0
lifecycle = new org.codehaus.groovy.graven.JarLifecycle()
lifecycle.dependencies = ['./src/test/libs/commons-io-1.2.jar']
lifecycle.testDependencies = ['./src/test/libs/junit-3.8.2.jar']

packaging = {
   lifecycle.packaging()
   new File(currentDir, 'customLifecycle').createNewFile()
}
```

This is one way of customizing the lifecycle. It is a non reusable customization applied directly in the build script. If there is a closure named like a lifecycle phase, this closure gets called instead of the lifecycle method. The other way of using a non standard lifecycle is: creating a new lifecycle from scratch, inheriting from an existing lifecycle, etc...

One implementation detail is how to define an order for the methods of a lifecycle class. One possibility would be, that the lifecycle method calls it predecessor directly. This has two disadvantages:

1. You can't customize an existing lifecycle class by adding a new phase.
2. You can't call a lifecycle method in isolation. I have no use case yet, where this is a problem. But I'm pretty sure they will come up.

My approach is to use a list which defines the lifecycle phases. Each name of the list corresponds to a lifecycle method name. If the build is started via `gant compile` for example, all lifecycle methos are called that correspond to entries in this list up to the compile phase.

Using the framework

Right now the framework is not capable of being used from the command line. The only user right now are the integration tests which call the main method of the Gant04 class and passes it the basedir and the buildscript path.

I might be a good idea to have first a look at /Gant04/src/test/integTests/org/codehaus/groovy/graven/BuildTest.groovy to best understand the framework.

Conclusions

The framework is a prototype. There are many things that can be improved. The main question is whether we want to implement the lifecycle handling in such a manner or not. I personally think this is a good approach. And I think this complements what Gant offers right now. In a build there are always lifecycle related action and non lifecycle related actions. For the latter the current Gant approach is a good way to implement them.