FrequentlyAskedQuestions

descriptor maintainance

Q: How does Gump know the details about our project?

A: The gump descriptor for your project can be held either in the Gump CVS or in your project's source repository. See the details gump object model.

Q: what should I do if a project its build output changes (for example from one to multiple jars or one to multiple maven projects)?

A: you should update the gump descriptors for the project to reflect the new situation. If this change somehow impacts your project dependees, which are listed for you on

http://lsd.student.utwente.nl/gump/${module}/${project}.html#Project+Dependees

then you should probably send them an e-mail explaining the change. Chances are, the project will not only need to update its gump descriptor, but it will also want to change some other aspects of its own build system (for example, update the dependency list in maven's project.xml, or change the jars they keep in CVS).

Q: how do I split a single gump project definition into multiple projects definitions?

A: create the descriptors for the new projects, and remove the descriptor for the old project. You can keep the old project around as a sort of "delegate" for a while, if you want. For example, if you've split the project commons-foo into two, commons-foo-api and commons-foo-impl, you could change the commons-foo descriptor to something like

```
<!-- don't use this one!!!
   Migrate to depending on project-api and project-impl instead... -->
<project name="commons-foo">
  <depend project="commons-foo-api" inherit="jars"/>
  <depend project="commons-foo-impl" inherit="jars"/>
</project>
```

Q: How do I match my Maven IDs to Gump IDs?

A: while efforts are underway to synchronize these identifiers, there are several incompatibilities. Please refer to the MavenId page to match your IDs, or help the effort by adding discovered incompatibilites.

Gump debugging

Q: can we get access to the build outputs of the gump run?

A: sorry, but no. We can't simply publish the full gump build tree, because that would be "distribution" of a whole lot of jars and tarballs. That, we cannot do without having humans check the quality and validity of those files, and, in the case of ASF projects, without proper authorities (ie, a PMC) decide the distributable is fit for distribution. What we *can* do sometimes is give people shell access to one or more of the machines that run gump. But you'll understand we can't give everyone shell access.

Q: why didn't gump run last night?

A: probably because someone killed it halfway through. Gump currently has no dedicated hardware on which to run (but a machine is underway), so it runs shared with other tasks on people's home computers and servers. It could also be because one of the gump developers broke gump badly, and introduced an error into a vital part of the system. If you want to be sure, just send an e-mail and ask.

Q: my project hasn't built for months because of prerequisite failures!!! What should I do?

A: get in contact with the developers of your prerequisite that is failing to build and ask them to do something about it. If the problem is a mismatch between the project's build system and the gump descriptor, you'll often have access to the descriptor, and you can change that yourself (still get in contact with the developers and tell them that you changed things, and encourage them to try and do that themselves next time). Another, dirty, measure to consider is to check a jar into your project's cvs module, and depend on that. This is bad, because you are removing all the benefits of continuous integration: you won't be aware of the changes to other projects that will break your own. Only do this is a temporary measure, and only do it when there is no other way to "debug" your gump problems.

Uncategorized

Q: Which version of Ant (or JUnit or log4j or...) does Gump use?

A: The latest from cvs (as of the instant of the Gump run).

Q: Gump is building the latest version of everything? That could never work, could it? How can you debug anything when you have so many variables to deal with?

A: It isn't as hard as it seems. Getting it to work initially was done by incrementally adding one project at a time. Keeping it working is only requires a small amount of effort over a long period of time. When builds are "mostly" clean and frequent, isolating problems is fairly easy. Rollback projects one by one until the problem disappears, and then reapply changes within that project one by one until the problem reappears. I've used this technique numerous times to produce test cases and patches to projects that I know virtually nothing about their internal design.

Q: I've just got a "nag" e-mail about a failure...what should I do about it?

A: This question deserves it's own page.

Q: I've got a fix, how do I get Gump to pick it up?

A: Check it into cvs.

Q: I checked in a fix, how do I know if Gump picked it up?

A: On the build log, click on the link marked cvs. It will tell you when the latest cvs update or checkout was performed and what source files were obtained in the process.

Q: How do I get nagging to stop (or start)?

A: If there is consensus on your project, send an e-mail to general@gump.apache.org letting us know what the outcome was, and it will be respected.

Q: My project depends on a specific version of a dependency, how can I get Gump to use that version?

A: you can check a jar into your project's cvs module, and depend on that. This is bad, because you are removing all the benefits of continuous integration: you won't be aware of the changes to other projects that will break your own. Only do this is a temporary measure, and only do it when there is no other way to "debug" your gump problems. If you do this, expect some raised eyebrows...the gump community will want you to make sure this really is a temporary measure. That said, here's how you do it.

- Check a jar into your project's CVS module. Example: the jar is in the cvs module commons, for subproject foo, named commons/foo/lib /commons-bar-1.4-dev.jar
- Create a project descriptor for that jar. In this case, you need to probably need to add the descriptor to the descriptor for the commons module, commons.xml. It looks pretty much like a normal descriptor, except that there's no <ant/> or <maven/> tag:

```
<!-- !!!don't use this one!!!
     This is a temporary measure until commons-bar's build system refactoring
     is complete, on which John Doe and Mary Poppins are working... -->
<project name="commons-foo-commons-bar-1.4-dev">
    <package>org.apache.commons.bar</package>
    <jar name="foo/lib/commons-bar-1.4-dev.jar"/>
</project>
```

Q: I've got a change I would like to make to the project definition for a project I'm a committer on...how do I get the change to be made?

A: The gump cvs repository is open to all ASF committers. Simply check it out and commit your changes. It is highly recommended that you build the "gen" target first of the provided build.xml in order to verify that your changes are syntatically correct. Don't be scared... this takes only a minute and only requires that you have ant and JDK 1.4. (or ant + xerces + xalan + JDK 1.2 or later).

Q: How do I get a project to be added to Gump? Must it be an Apache project?

A: Send an e-mail (preferably a patch) to general@gump.apache.org. Being Apache related (i.e, used by or a consumer of other Apache projects) helps, but is not required. What is important is that your code be publicly accessible via cvs and that it be buildable in an OS independent manner.

Q: Shouldn't the project definitions for a project be kept within the project itself?

A: That's supported too, as long as your project supports viewcvs or equivalent. Simply put the URL of the project definition in the gump profile. However, if you are an ASF committer, why not consider leaving it in the gump cvs repository, that way others can also help out too?