Module Descriptor Generator (maven plugin)



Your Rating:  Results:  14 rates

GREY Proposal for generating Magnolia module descriptors based on Maven POM files GREY

Currently, we maintain our module descriptor xml's by hand. We use a couple of properties to ease the process a little (`${project.name}`, `${project.description}` and `${project.version}`), but it's still quite a hassle to keep the dependencies in synch, for instance.

We can't have a 1:1 mapping between the pom files and the module descriptor either:

1. dependencies in our module descriptor use ranges, as opposed to fixed build dependencies in our pom files
2. we often have optional dependencies in our module descriptors, which are not marked as such in our pom files
3. there are elements in the module descriptor we can't have in the pom (properties, repositories, module class, version handler, etc)

Ideas for the above:
1) In fact, in Maven, <version>x.y.z</version> isn't a strict version requirement, but just a preference indication. The strict version would be x.y.z. Additionally, ranges are also supported in Maven, with a syntax that's much more complete than ours (and afaik matches the OSGi one). If we had support for that syntax in our module descriptors, and we would improve our dependency declarations in our pom files, we could generate these easily.

2) We could simply start using the <optional> tag in maven dependencies too. As far as I can tell, this should be a perfect match for what we use optional dependencies in module descriptors for.

3) At least 2 possibilities for this issue: we could split the module descriptor in 2 files (this part maintained manually, the rest generated), or merge the generated bits into the manually maintained ones.
---> plugin can inject what's missing in the existing file

Reference for version ranges: http://www.sonatype.com/books/mvnref-book/reference/pom-relationships-sect-version-ranges.html http://www.google.com/search?q=maven+dependency+range

3 more little obstacles

- difference between artifactId and module-name
- what maven-dependencies do we consider to be magnolia-dependencies (maybe using the groupId as a default differentiator + possible configuration of the plugin)
- slight overlap with "module downloader" - the idea with the module downloader was to use the maven infrastructure (pom dependencies) for publishing modules, downloading them, and resolving dependencies (the maven index can give this information without having to download the jar and read our module descriptor inside it) - but at that point, we might also consider using the pom (which ends up inside the module's jar in any case) to resolve module dependencies within our module mechanism too. (and not only for the module downloader part)