

# best practices - jesse's general approach

## primer

I would like to advocate a perhaps philosophical approach to the general best practices of maven2 projects.

There are basically two approaches that we have available to us in giving **meaning** to project layout that can be used in some combination.

- 1) nesting into directories
- 2) naming directories

IMO, one of the strongest themes that maven2 presents is the small discretely compiled/prepared artifacts centralized through various repositories. Whereas other build systems like make and ant leverage directory structure in the organizing of source and compiled dependencies, and then *backup* a directory and bundle the subdirectories into some new shared library/jar/j2ee artifact, maven2 treats all of these artifacts as logical equals, all woven together via the repository.

With the existing rule of thumb that one project produces one artifact, all we would really be accomplishing by formalizing some specific j2ee project structure is choosing a layout where independent artifact generating subprojects would be mounted...and what does doing that exercise really give us?

## my thoughts on how maven-components came to be

If I could crawl into the minds of Jason, Brett and folks I think I would see that this idea of nesting directories to give an implied meaning/purpose to subprojects was a factor in the layout.

The layout of maven-components is very flat, the maven-project, maven-model, maven-plugin-api, maven-profile projects are all given implicit equal standing by existing at the same top lvl directory. Even though things like maven-profile could logically be nested inside of maven-project, it is really only a artifact dependency to maven-project so it doesn't need to be nested below maven-project..just as a dependency.

So that leaves me with the following as a general rule of thumb when organizing a maven2 based project.

- all artifacts share equal standing at the root of the project
- if 5 or more of a particular functionality or packaging type exist in the root of the project, nest it in a subdirectory named after the functionality or packaging type.
- for non-jar packaging artifacts, end the name with the type of artifact -ear, -war, -ejb, etc.

## a general rule of thumb applied to j2ee

```
dropit
dropit-logging
dropit-config
dropit-interfaces-api
dropit-frontend-servlet
dropit-backdoor-servlet
dropit-war
dropit-ear
```

For a simple project like this, there is no need to venture into lots of convoluted subdirectories.

However, if you have 5 or more servlets, nest them in a servlets directory.

```
dropit
dropit-logging
dropit-config
dropit-interfaces-api
servlets/dropit-frontend-servlet
servlets/dropit-backdoor-servlet
servlets/dropit-fun-servlet
servlets/dropit-silly-servlet
servlets/dropit-humor-servlet
dropit-war
dropit-ear
```

and the same thing would hold for the ejbs, wars, ears, whatever...the only exception being artifacts that compile to be jars...then would all exist at the root of the project directory unless it really makes sense to nest components that have maybe 5 or more of a particular thing.

## conclusion

Following an approach like this there is no need to really dig into specialized archetypes for meta level project layouts. The archetype system we currently have in place is for project layouts based around the type of artifact that will be generated. They give people a template to work off of for making an *artifact* work with their build.

Instead of trying to identify some meta project layout that weaves these artifacts together, we can continue fleshing out artifacts based around different types of plugins, even mojo plugins, like the wsdl2java and sablecc plugins. All of these archetypes are basically used at the root of project and we just follow the bulleted rules above for naming conventions and nesting.