Setting up Eclipse

Setting up an Eclipse project to build CXF is pretty easy. There are three parts to it:

Required plugins

We use several Eclipse plugins to make building CXF a bit easier

- Checkstyle - we use checkstyle to make sure we have consistent code style as well as to find various types of bugs and other issues. https://checkstyle.sourceforge.io/
- PMD - like Checkstyle, we use PMD to find potential programming problems in the code. Point the Eclipse auto-install thing at https://pmd.github.io/

ℹ While there exist Maven plug-ins for Eclipse, team developer experience has found using them with CXF problematic at best. We recommend importing the CXF source code as Eclipse projects as shown below and/or using Maven externally (i.e., from a command-line window) as discussed on the CXF build page.

To install the plugins:

- Go to

  ```
  Help -> Eclipse Marketplace
  ```

- On the Search tab, enter "Checkstyle" to search for it and install the "Checkstyle Plugin"
- On the Search tab, enter "PMD" to search for PMD. There are two PMD plugins, install the "pmd-eclipse-plugin"

Experimental Alternative: M2Eclipse

Some of us are starting to experiment with using M2Eclipse. See this page for instructions.

Creating a workspace

First check out CXF from Subversion.

To create a workspace, just run from the root directory of the CXF project (see the build page for more detailed information):

```
> mvn -Pfastinstall
> mvn -Psetup.eclipse
```

OR

```
> mvn install -Pfastinstall -Psetup.eclipse
```

This creates a new workspace in "../workspace" for use with CXF.

If you don't want the workspace there, you can run:

```
"mvn -Psetup.eclipse -Declipse.workspace.dir=path/to/workspace"
```

If you don't want the eclipse projects' output directory to be ./target directory (by default) but ./eclipse-classes, you can run:

```
"mvn -Psetup.eclipse -Pset.eclipse.output"
```

What this does is create a workspace and imports our checkstyle rules, the maven 2 repository, code format rules, import order rules, etc... into that workspace. It also goes through each sub-project and creates the . project and .classpath files. This process will take some time. It will down load source jars for most of the dependencies and hook them up in the .classpath file as well. Thus, while coding/debugging, you can trace right into the dependent libraries. While running, you ***WILL*** see a bunch of warnings and such flying by. There are a bunch of jars on ibiblio that do NOT have source jars with them. Thus, you will see warning about those. Those warnings are safely ignorable. As long as it says "BUILD SUCCESSFUL" at the end, you should be OK.

Create the project in Eclipse

- In eclipse, switch to the workspace you created above.
- Go To:

  ```
  File -> Import....
  ```

- Select "Existing Projects into Workspace" and hit Next
- Select root directory: enter the path to your trunk directory and hit Next.
- Select all the subprojects and hit Finish. Eclipse will import and rebuild all the subprojects selected. This will take a while.

That's all there is to it. From eclipse, all the unit tests and system tests should be runnable. However, to build kits/jars and stuff, you still need to use the command line "mvn" stuff.

Importing new projects that depend on CXF projects

With the latest version (2.5) of the maven-eclipse-plugin, when you run "mvn eclipse:eclipse" on a project, if it knows where your workspace is, it will see what projects are already defined and wire them in to the new project instead of pointing at the jars in your ~/.m2/repository dir. Thus, debugging is a lot easier. There are two ways to get it to know where your workspace is:

1. Explicitly on the command line. When running eclipse:eclipse, add -Declipse.workspace=/home/dkulp/working/workspace
2. Update your Maven ~/.m2/settings.xml to have a active profile that always sets these variables. Thus, whenever the eclipse plugin looks for it, it know where the workspace is. In settings.xml, do:

```
...
    <activeProfiles>
        <activeProfile>extra</activeProfile>
    </activeProfiles>
    <profiles>
        <profile>
            <id>extra</id>
            <properties>
                <eclipse.workspace>/home/dkulp/working/workspace<
/eclipse.workspace>

                <eclipse.workspace.dir>/home/dkulp/working/workspace<
/eclipse.workspace.dir>
            </properties>
        </profile>
    </profiles>
...
```

By doing that, you can pretty much run eclipse:eclipse (or -Psetup.eclipse for cxf projects) at any point and it will always wire the new project to depend on the existing projects.

How Does This All Work, Anyway?

If you are wondering about how all this manages to make Eclipse, Maven, Checkstyle, and PMD cooperate, see Connecting Maven, Eclipse, Checkstyle, and PMD.