Using the Eclipse Java compiler (ecj) in Maven builds

The Eclipse Java Compiler (ecj) has a lot of advantages over the standard javac compiler. It is fast, and it has way more warnings and errors that can be configured, improving code quality. One of the most interesting things in the compiler is the addition of null types inside the compiler: by annotating your code with @Nullable and @NotNull annotations you can force the Eclipse compiler to check null accesses at compile time instead of runtime. When applied rigorously this teaches you to code way more safe (by preventing null values) and it prevents NPE exceptions during testing or production.

To use the Eclipse Compiler inside Maven is not too hard, but there is a lot of misinformation and old information on the Internets which causes a lot of confusion. This article is meant to set things straight.

To make Eclipse use the ecj compiler you need to use the plexus-compiler-eclipse plugin and **nothing else**. A typical configuration would be the following:

```
<pluginManagement>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
        </plugin>

        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.7.0</version>
            <configuration>
                <compilerId>eclipse</compilerId>
                <source>${source.jdk.version}</source>
                <target>${target.jdk.version}</target>
                <!-- Passing arguments is a trainwreck, see https://issues.apache.org/jira/browse/MCOMPILER-123
-->
                <compilerArguments>
                    <properties>${project.basedir}/.settings/org.eclipse.jdt.core.prefs</properties>
                </compilerArguments>
                <showWarnings>true</showWarnings>
                <showDeprecation>true</showDeprecation>
            </configuration>

            <dependencies>
                <dependency>
                    <groupId>org.codehaus.plexus</groupId>
                    <artifactId>plexus-compiler-eclipse</artifactId>
                    <version>2.8.3</version>
                </dependency>

                <dependency>
                    <groupId>org.eclipse.jdt</groupId>
                    <artifactId>ecj</artifactId>
                    <version>3.13.101</version>
                </dependency>
            </dependencies>
        </plugin>
</pluginManagement>
```

Put this part in either the pluginManagement or the build section of your parent/root pom.

Let's explain the different parts.

The maven-compiler-plugin needs to be of a recent version. The source and target parameters define the versions of java to use for source code and bytecode, and are usually the same.

Passing arguments to the compiler is an utter trainwreck. See the separate section on that below here. In this example I use the properties setting which allows me to provide detailed settings on which errors and warnings I want to have when compiling things. By using the ${project.basedir} variable inside the parameter I have these settings per project: every project is required to have a .settings/org.eclipse.jdt.core.prefs file present (which is by happy chance the location where the Eclipse IDE leaves its compiler settings).

The dependency on plexus-codehaus-eclipse defines the plugin that knows *how* to run the Eclipse compiler. The 2.8.3 version was the latest at the time of writing but this version has a few issues. Version 2.8.4 should come with a rewritten interface to the compiler which fixes a lot of issues, but this version is still in the works at the time of writing. You can find details on the plugin here, so progress can be followed on new releases/code changes.

The other important dependency is the org.eclipse.jdt:ecj dependency: this one specifies **the exact version of the ecj compiler to use**. You should always specify it because otherwise build stability will suffer when the plugin decides to use another version of the compiler one day before you have a big release 😷

From the list of releases one might be able to find a version number and then check this maven repository for something that looks like it. But this repository only contains the old versions. When you need a more recent release you should apparently look here at this one - this is where Eclipse currently pushes its versions. This newer repository does away with the easily recognizable version numbers of the earlier one; it uses version numbers like 3.1x.x as seen above. Eclipse usually has a major release once every year plus one or two fix releases in between. The second part in the 3.13.x number corresponds to the internal versioning used inside the Eclipse Platform project for releases. It is hard to get by a list but at least these are known:

Version	Eclipse compiler version	Compiler version
3.13.0	Oxygen Release	4.7
3.13.50	Oxygen 1A	4.7.1a
3.13.100	Oxygen R2	4.7.2

The version always starts with 3, the 13 is more or less the "year" of the release. So when 13 is Oxygen (2017, 4.7) 14 will probably be Photon (2018, 4.8).

Versions of the plexus-compiler-eclipse plugin

Before 2.8.4

Versions before 2.8.4 of the plexus-compiler-plugin used an internal API to start the Eclipse compiler. This causes a lot of things not to work that well, as this internal API, for instance, does not interpret the usual command line parameters of the ecj compiler. This makes it quite hard to use, and some things are not supported. The following is a list of restrictions:

- Annotations processing is not implemented. Any configuration is silently ignored.
- Adding specific parameters by using the <compilerArguments> tag is hard as there are multiple problems with the implementation:
    - The compiler mojo seems to add dashes to all parameters entered here. The internal API used by this version of the plugin, however, needs parameters without dashes. So the plugin removes them again.
    - As the parameters here are not really command line ecj parameters it is hard to know which ones to use: look at the Compiler.java class and CompilerOptions.java classes inside Eclipse's source code for details.
    - The plugin DOES accept some parameters there, but these are interpreted by the plugin itself and then "translated" to the internal api.

This plugin accepts the following parameters in the <compilerArguments> tag:

- <properties>filename</properties>: defines a properties file that will be passed to the -properties parameter of the compiler. Examples of this file's format can be found by looking at the file .settings/org.eclipse.jdt.core.prefs in an Eclipse project: this file stores the compiler's configuration. It contains settings for warnings, errors and informational messages plus compiler compliance settings.
- <errorsAsWarnings>whatever</errorsAsWarnings>. When this is valid the plugin will ignore any error that is generated by the compiler and report them as warnings. Of course compilation still failed so depending on the error a .class file might have been written/updated or not. This gets handled by the plugin itself: it just changes all errors to warnings and tells the world that compilation worked.

From 2.8.4 (in the works, not yet released)

Version 2.8.4 of the plexus-compiler-eclipse plugin has been mostly rewritten. It now uses the public API of the ECJ compiler which more or less is the ECJ compiler itself. This for instance means that everything that ECJ can do (like annotations processing) the plugin can now do too, and parameters entered in the <compilerArguments> tag are now passed to the compiler, which means you should be able to use ecj's help page to find out interesting parameters to add.

Like the previous version this version also requires you to remove the '-' from all parameter names; the dash is automagically added again before the parameter name is added to the ecj command line.

This version supports annotation processing as defined by Maven; by adding the required parts to the compilation blob you can have your annotation processors run. For example:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>${maven-compiler-plugin.version}</version>
    <configuration>
        <annotationProcessors>
            <annotationProcessor>db.annotationprocessing.EntityAnnotationProcessor</annotationProcessor>
        </annotationProcessors>
        <annotationProcessorPaths>
            <dependency>
                <groupId>to.etc.domui</groupId>
                <artifactId>property-annotations-processor</artifactId>
                <version>1.2-SNAPSHOT</version>
            </dependency>
        </annotationProcessorPaths>
    </configuration>

    <dependencies>
        <dependency>
            <groupId>to.etc.domui</groupId>
            <artifactId>property-annotations-processor</artifactId>
            <version>1.2-SNAPSHOT</version>
        </dependency>
    </dependencies>
</plugin>
```

This part may seem incomplete because there is no reference to the plexus-compiler-eclipse plugin at all, but remember that in Maven that configuration *inherits*: the parent POM in this case contained the part above, and this just adds a bit of configuration for this POM's project only.