Background (#background)

M2Eclipse 0.12 and earlier executed some parts of Maven build lifecycle inside Eclipse and then configured the Eclipse project based on after-execution state collected in MavenProject. This was controlled by many different sets of maven goals - goals when projects were imported,a project configuration changes and workspace full and incremental builds. Some of these goals were configured at workspace level, some in project/.settings. On top of that, there was project-level setting to “skip” maven-compiler-plugin execution.

Unfortunately, this did not work well or not at all for many projects. Probably even worse, it did not always work for many projects, so we had to go through series of refresh/update dependencies/update configuration/rebuild voodoo (or “m2eclipse dance” as some called it) to get projects in a good state. For example MNGECLIPSE-823 (https://issues.sonatype.org/browse/MNGECLIPSE-823) was the most voted issue in M2Eclipse jira and it was a direct manifestation of this “flakiness”.

Most, if not all, such problems were traced back to one of two root causes.

1. Out-of-workspace resource changes made by Maven plugin triggered unexpected workspace builds. This was very indeterministic. In some cases projects appeared to work fine. In some cases, generated/filtered resources would go missing. And in some cases workspace build would go on forever.

2. Various JVM and OS resources leaks by Maven plugins was another common cause of problems.

To solve these long-standing issues, M2Eclipse 1.0 requires explicit instructions what to do with all Maven plugins bound to “interesting” phases (see [M2E interesting lifecycle phases](M2E interesting lifecycle phases “wikilink”)) of a project build lifecycle. We call these instructions “project build lifecycle mapping” or simply “lifecycle mapping” because they define how m2e maps information from project `pom.xml` file to Eclipse workspace project configuration and behaviour during Eclipse workspace build.

Project build lifecycle mapping can be configured in a project’s `pom.xml`, contributed by Eclipse plugins, or defaulted to the commonly used Maven plugins shipped with m2e. We call these “lifecycle mapping metadata sources”. m2e will create error marker like below for all plugin executions that do not have lifecycle mapping in any of the mapping metadata sources.

```
Plugin execution not covered by lifecycle configuration:
org.apache.maven.plugins:maven-antrun-plugin:1.3:run
    (execution: generate-sources-input, phase: generate-sources)
```

M2Eclipse matches plugin executions to actions using combination of plugin groupId, artifactId, version range and goal. There are three basic actions that M2Eclipse can be instructed to do with a plugin execution - ignore, execute and delegate to a project configurator.

Delegate to a Project Configurator (recommended)

A configurator mapping tells M2Eclipse to delegate workspace project configuration mapping for matching plugin execution to an implementation of AbstractProjectConfigurator registered with m2e using projectConfigurators extension point.

In most cases a configurator mapping will be used by M2Eclipse extension developers. See M2E Extension Development (/documentation/m2e-extension-development.html) page for more information about developing M2Eclipse extensions.

Ignore Plugin Goal (#ignore-plugin-goal)

The ignore option, as the name suggests, tells M2Eclipse to silently ignore the plugin execution. Here is sample pom.xml snippet:

```
<pluginManagement>
  <plugins>
    <plugin>
     <groupId>org.eclipse.m2e</groupId>
     <artifactId>lifecycle-mapping</artifactId>
     <version>1.0.0</version>
     <configuration>
       <lifecycleMappingMetadata>
         <pluginExecutions>
           <pluginExecution>
             <pluginExecutionFilter>
               <groupId>some-group-id</groupId>
               <artifactId>some-artifact-id</artifactId>
               <versionRange>[1.0.0,)</versionRange>
               <goals>
                 <goal>some-goal</goal>
               </goals>
             </pluginExecutionFilter>
             <action>
               <ignore/>
             </action>
           </pluginExecution>
         </pluginExecutions>
       </lifecycleMappingMetadata>
     </configuration>
    </plugin>
  </plugins>
</pluginManagement>
```

HINT: M2Eclipse provides a quick-fix associated with “plugin execution not covered” to easily create `<pluginManagement/>` elements like above.

Execute Plugin Goal (#execute-plugin-goal)

The execute option tells m2e to execute the action as part of Eclipse workspace full or incremental build. Beware that M2Eclipse does not provide any safeguards against rogue maven plugins that leak classloaders, modify random files inside workspace or throw nasty exceptions to fail the build. Use this as the last resort and make sure you know what you are doing.

```
<pluginManagement>
  <plugins>
    <plugin>
     <groupId>org.eclipse.m2e</groupId>
     <artifactId>lifecycle-mapping</artifactId>
     <version>1.0.0</version>
     <configuration>
       <lifecycleMappingMetadata>
         <pluginExecutions>
           <pluginExecution>
             <pluginExecutionFilter>
               <groupId>some-group-id</groupId>
               <artifactId>some-artifact-id</artifactId>
               <versionRange>[1.0.0,)</versionRange>
               <goals>
                 <goal>some-goal</goal>
               </goals>
             </pluginExecutionFilter>
             <action>
               <execute>
                 <runOnIncremental>false</runOnIncremental>
               </execute >
             </action>
           </pluginExecution>
         </pluginExecutions>
       </lifecycleMappingMetadata>
     </configuration>
    </plugin>
  </plugins>
</pluginManagement>
```

HINT: Use quick fix to create "ignore" mapping, then replace `<ignore/>` action with `<execute/>`. M2Eclipse 1.3 and newer assume safer runOnIncremental=false by default. It is recommended to always specific desired runOnIncremental value explicitly in lifecycle mapping configuration.

Metadata Source Lookup Order (#metadata-source-lookup-order)

M2Eclipse considers lifecycle mapping metadata sources in the following order:

1. The project's `pom.xml`
2. The parent `pom.xml`, the grand-parent `pom.xml` and so on.
3. M2Eclipse 1.2+ workspace preferences (see below)
4. Installed M2Eclipse extensions (in no particular order)

5. M2Eclipse 1.1+ lifecycle mapping metadata provided by maven plugin (see below)
6. Default lifecycle mapping metadata shipped with M2Eclipse

M2Eclipse uses the first applicable mapping found.

Lifecycle mapping metadata provided by maven plugin (#lifecycle-mapping-metadata-provided-by-maven-plugin)

Starting with m2e 1.1, maven plugin developers are able to provide lifecycle mapping metadata as part of the plugin itself. If present, such mapping metadata will be automatically used by m2e, thus eliminating the need for plugin specific project configurator and/or lifecycle mapping metadata in pom.xml.

[M2E compatible maven plugins](M2E compatible maven plugins "wikilink") wiki page provides more information about developing m2e-compatible maven plugins that do not require external build lifecycle mapping configuration.

Eclipse workspace lifecycle mapping metadata (#eclipse-workspace-lifecycle-mapping-metadata)

Starting with M2Eclipse 1.2, it is now possible to configure lifecycle mapping metadata in m2e workspace preferences. Plugin goals can be ignored at workspace level using new quick-fix, which is available both from `pom.xml` editor and from Problems view. It is also possible to edit lifecycle mapping xml file directly. The file can be opened from Preferences->Maven->LifecycleMappings.

Viewing Effective Lifecycle Mapping (#viewing-effective-lifecycle-mapping)

Starting with M2Eclipse 1.1, it is now possible to see effective lifecycle mapping in Maven->LifecycleMapping project properties.

Lifecycle

Help Improve M2Eclipse Maven Plugin Coverage (#help-improve-m2eclipse-maven-plugin-coverage)

First and foremost, you need to understand the desired behaviour. In most cases this should be limited to IDE usecase, i.e. editing sources and running tests, and not the complete Maven build, so plugin goals that publish build results to a remote repository can be ignored without any adverse side effects, while java source code generation most likely is necessary.

If the desired behaviour is applicable to other Maven projects using the plugin goal, we strongly recommend documenting your findings in m2e bugzilla (https://bugs.eclipse.org/bugs/). Please use “[mojo] plugin-artifact-id:goal support” bugzilla summary and make sure to search for existing records (https://bugs.eclipse.org/bugs/buglist.cgi?query_format=specific&order=relevance+desc&bug_status=__open__&product=m2e&content=mojo). When submitting new request, please provide standalone example project and detailed description of desired behaviour when the project is imported in Eclipse workspace. This will allow other users and interested developers to track popularity of various Maven plugins and schedule implementation work accordingly.

M2E Extension Development (/documentation/m2e-extension-development.html) has pointers how to develop M2Eclipse extensions.

Common Problems (#common-problems)

Some Maven plugins are recognized as problematic and will produce error markers with a text similar to: maven-dependency-plugin (goals “copy-dependencies”, “unpack”) is not supported by M2Eclipse

In version 1.0 there is no quick fix available for this but it is possible to define a lifecycle mapping for the plugin as well (as shown in ignore plugin goal above). Which removes the error marker.

Eclipse 4.2 Adds Default Mapping

If you are using Eclipse 4.2 and have troubles with mapping and won’t put mess into yours pom.xml create new file lifecycle-mapping-metadata.xml configure it in Windows -> Preferences -> Maven -> Lifecycle mapping . (don’t forget press Reload workspace lifecycle mappings metadata after each change of this file!).

If you have multiple Eclipse workspaces and/or work in a team, it is easy to get workspace-level configuration out-of-sync. This is unlikely to cause any confusion for <ignore /> mappings, but for <execute /> and <configurator /> mappings configuration in pom.xml or maven-plugin is strongly recommended.

Here is example based on
eclipse/plugins/org.eclipse.m2e.lifecyclemapping.defaults_1.2.0.20120903-1050.jar/lifecycle-mapping-metadata.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<lifecycleMappingMetadata>
  <pluginExecutions>
    <pluginExecution>
      <pluginExecutionFilter>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>buildnumber-maven-plugin</artifactId>
        <goals>
          <goal>create-timestamp</goal>
        </goals>
        <versionRange>[0.0,)</versionRange>
      </pluginExecutionFilter>
      <action>
        <ignore />
      </action>
    </pluginExecution>

    <pluginExecution>
      <pluginExecutionFilter>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <goals>
          <goal>list</goal>
        </goals>
        <versionRange>[0.0,)</versionRange>
      </pluginExecutionFilter>
      <action>
        <ignore />
      </action>
    </pluginExecution>

    <pluginExecution>
      <pluginExecutionFilter>
        <groupId>org.zeroturnaround</groupId>
        <artifactId>jrebel-maven-plugin</artifactId>
        <goals>
          <goal>generate</goal>
        </goals>
        <versionRange>[0.0,)</versionRange>
      </pluginExecutionFilter>
      <action>
        <ignore />
      </action>
    </pluginExecution>

    <pluginExecution>
      <pluginExecutionFilter>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>gwt-maven-plugin</artifactId>
        <goals>
          <goal>compile</goal>
        </goals>
        <versionRange>[0.0,)</versionRange>
      </pluginExecutionFilter>
      <action>
        <ignore />
      </action>
    </pluginExecution>

    <pluginExecution>
      <pluginExecutionFilter>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <goals>
          <goal>copy-dependencies</goal>
          <goal>unpack</goal>
        </goals>
        <versionRange>[0.0,)</versionRange>
      </pluginExecutionFilter>
      <action>
        <ignore />
      </action>
    </pluginExecution>

  </pluginExecutions>
</lifecycleMappingMetadata>
```