Using Eclipse with Maven

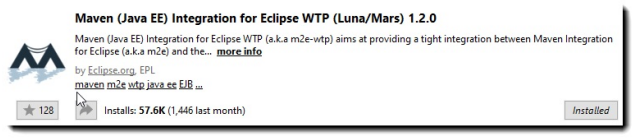
Recommended Reading

- Apache Maven Project documentation
- Baeldung Apache Maven Tutorial
- Jenkov Maven Tutorial
- Technology Conversations - Java Build Tools: Ant vs Maven vs Gradle
- Using Maven within the Eclipse IDE - Tutorial

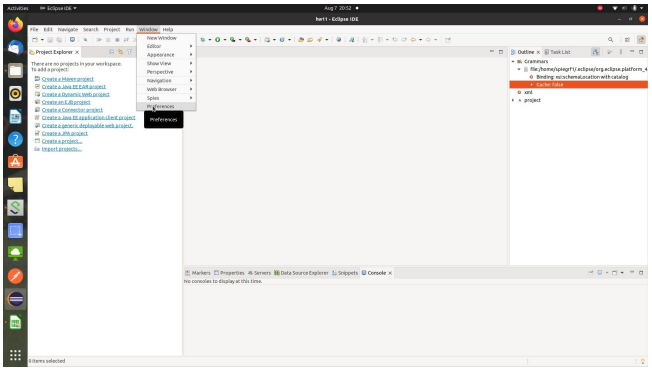
Using Eclipse with Maven

Recent releases of Eclipse for Enterprise Java and Web Developers come with Maven included. However, if you are using an earlier version that does not include the capability you will need to install the M2Eclipse project plugin to add that capability.

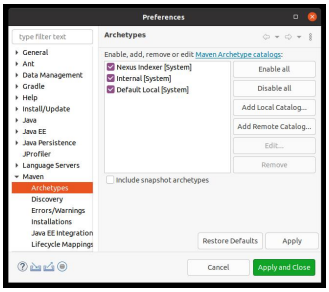
This plugin manages the plugin dependencies and manages the classpath settings within the IDE. Turns out, there is even a POM editor that you can use. Just go to the Eclipse Marketplace under the Help menu in eclipse and download the plugin:



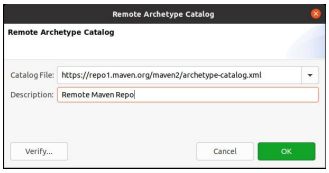
Before setting up your first Maven project in Eclipse, let's make sure that the IDE is configured to access appropriate Maven repositories. Some versions came with it pre-configured, but as of late that may or may not be the case. To set up or at least see if your Eclipse is configured to find remote Maven repositories go to the preferences menu from the Window menu in the main IDE. The following figure illustrates where that can be found.



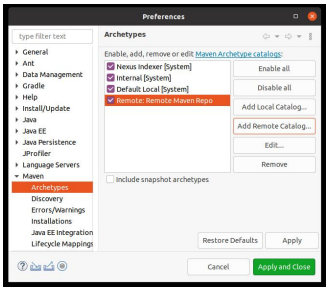
Selecting *Preferences* will result in the following dialog box being displayed:



As illustrated above, open the Maven menu in the panel to the left and select the Archetypes option. That will display the currently configured Maven archetype source locations on your local machine and remote from your machine. If you have an item in the Archetypes list on the right called *Remote*. If so you will want to select it and press the *Edit* button on the right to interogate its setting. If you don't have that (as in the figure) you should press the button *Add Remote Catalog* to bring up the following empty dialog box:

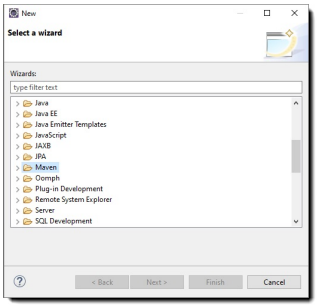


If you are editing the configuration, you should confirm that your Catalog File entry matches the one in the illustration above. Otherwise, enter it if is a new entry. You can enter whatever you would like as the Description as that will be the name of your reference. Then hit the *OK* button.

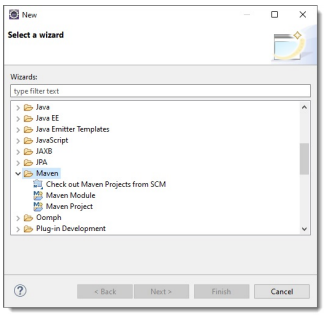


You can now see the added archetype repository configured as Remote. You can press *Apply and Close* to complete your configuration and you are now ready to create your first Maven project.

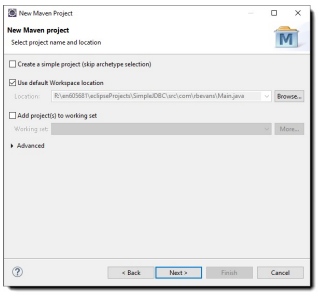
You can create your first Maven project in Eclipse by going to the File->New->Other Menu and you should see the full set of Wizards available:



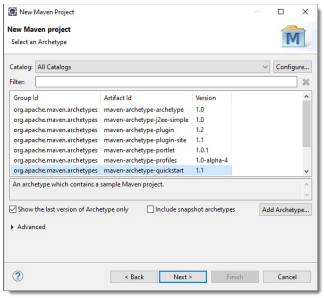
You want to scroll down, Select Maven, then open up that node



Choose Maven Project, then click Next>

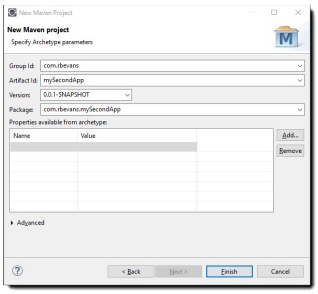


Do NOT select the "Create a simple project" option at this time. Click next and you get a window with some archetypes available. Look! there is the good old quickstart that we just used!

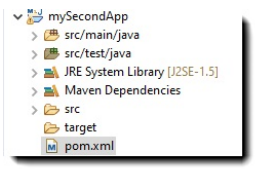


Select quickstart and then click on Next>

Okay, at this point, you should see a dialog with some familiar attributes. As you fill in the first two, the Package field will fill in with a correct package name



Clicking Finish builds the Eclipse project...lets take a look



And if we take a look at the POM file created, we see one, but it is a bit more simple than the CLI version

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.rbevans</groupId>
  <artifactId>mySecondApp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>mySecondApp</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

One interesting thing that Eclipse offers, is that there is a tab that shows the "Effective POM", which we talked about earlier.

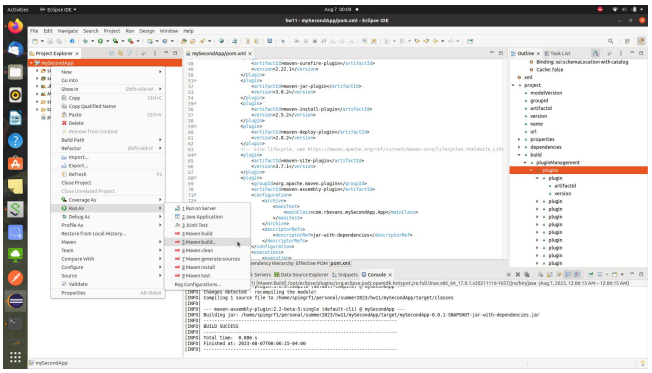
Okay, we have a working POM file at this point from myFirstApp, lets **just grab the assembly plugin that built the runnable JAR and place it in the POM**. I grabbed all of the <build> attributes, and inserted them into the file. You can see the maven assembly-plugin section, note that I had to rename the mainClass entry to match the packaging that Eclipse generated.

mySecondApp/pom.xml | App.java

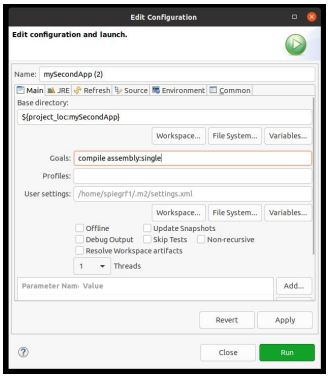
```
      </plugin>
      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>2.8.2</version>
      </plugin>
      <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site_Lifecycle -->
      <plugin>
        <artifactId>maven-site-plugin</artifactId>
        <version>3.7.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>3.0.0</version>
      </plugin>
      <!-- added to take us from the quickstart configuration to a runnable jar -->
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
      <archive>
            <manifest>
              <mainClass>com.rbevans.mySecondApp.App</mainClass>
            </manifest>
      </archive>
      <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
      </descriptorRefs>
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>single</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    <!-- End of mod -->
    </plugins>
  </pluginManagement>
  </build>
</project>
```

Okay. Final Step, lets build this thing! Turns out you can just go up to the Project menu and build it there, instead right click on the mySecondApp project, and then select **Run As**.

You should see this menu come up, click on the **Maven Build ...** choice as illustrated below:



The result of the selection will be the dialog box below.

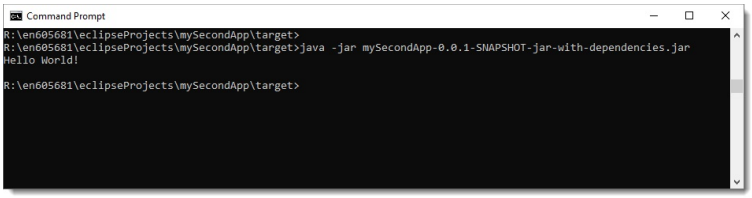


The only thing you have to complete is the *Goals* field. Enter compile assembly:single in the field and then press the *Run* button.

After you select the *Run* button, you should see the project being built in the lower Eclipse window. The first time you run it, you may see more repository "action" but it should end with something like this.

```
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< com.rbevans:mySecondApp >-----------------------
[INFO] Building mySecondApp 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-assembly-plugin:2.2-beta-5:single (default-cli) @ mySecondApp ---
[INFO] Building jar: R:\en605681\eclipseProjects\mySecondApp\target\mySecondApp-0.0.1-SNAPSHOT-jar-with-dependencies.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.650 s
[INFO] Finished at: 2020-02-01T11:56:46-05:00
[INFO] ------------------------------------------------------------------------
```

Now, for your last step, go to a command window and try running the JAR file.



So, on the plus side, you keep your IDE environment and can still use Maven, on the minus side, it is a plugin and may not be updated as quickly as the "raw" Maven CLI release. Maven is basically a CLI based tool, and it is up to you to determine how to integrate it with your development environment.