

A Software Journey

A Programming Exercise that Grew into a Web App Client and Web Service

Intro

Note: this is a fake project, whose purpose is to showcase some knowledge of various software tools and technologies. I suppose it could also be used as a tutorial, given more work on it.

The goal with these series of articles (and related source code) is to document an idea that began as just a simple command-line app, and move and grow it into a service that was exposed as:

- Re-usable component(service) (Spring framework , pure Java config, JPA / Hibernate, MySQL)
- Web Service (SOAP/XML) (JAX-WS)
- Web Service (REST) (XML & JSON) (JAX-RS)
- Complete Web app – (Bootstrap, JS, HTML5, CSS, JSP, Spring MVC) directly using the internal component (described first)
- Web app clients using the various Web Services as a backend
- TODO: Secure the above apps

If you continue this series of articles to the end, we will touch on all of the following:

[Bootstrap](#) , JSP, Javascript, [Spring MVC 4](#), [Spring IOC](#), [JPA / Hibernate 5](#), [Java 1.8](#), Java Config, WSDL (top-down), WADL, JAX-WS, JAX-RS, XML, JSON, concurrency, [MySQL](#), [Cygwin](#), [Maven](#), Junit, [Eclipse IDE](#), and more.

I was given a programming assignment. Here it is:

"Given a single number (either it is the radius or length of an edge) , write a program that will calculate the area of a circle, a square, and an equilateral triangle. Make the program multi-threaded to handle many calculations."

Sounds simple, and uninteresting. I used it as starting point for much more.

I will not explain everything because so many have already done so.

Another note - these articles are more of a story about a software development journey, that happen to contain some **how-tos**, so I tend to explain something at the point when it is needed (with some exceptions). This means you may not find, for example, all my explanations about Maven, or JUnit, in one section. They are probably strewn throughout the article series (again - when needed).

Let's dive right in.

Initial Setup

(This article may seem a bit drawn-out or slow to some readers. Hopefully we can speed things up in upcoming articles of this series)

Install JDK

The first thing we need to do is download and install Java JDK 1.8.

I chose to install everything right at my C:\ drive in windows, and not the usual (C:\Program Files ...blah blah), just to make it easier - not dealing with spaces on the command line, and to reduce path length.

Since I love Linux/Unix but didn't want to worry about re-imaging my laptop or dual-booting, or running a virtual machine, I went with Cygwin. If you want to use the Windows command line, you are on your own for this series of articles.

So I chose to install the JDK portion at C:\jdk1.8.0_101. Another good thing about this location (C:\) is that I can install different versions of the JDK (and JRE).

The JRE installation is not required for our purposes.

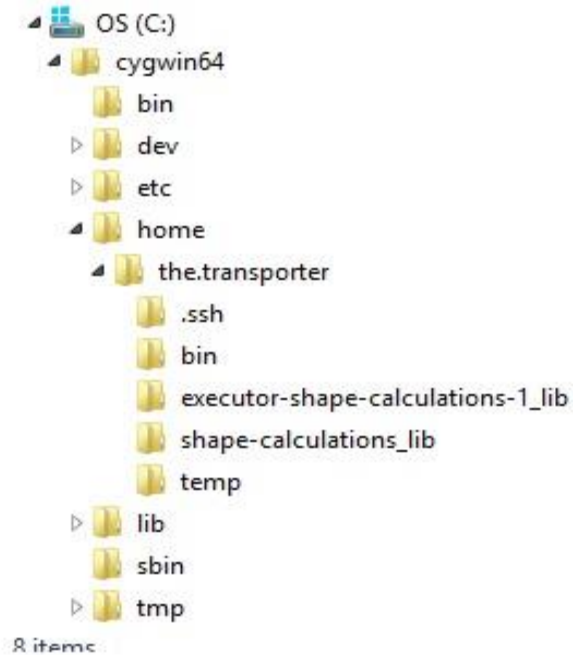
The next question of the Java installer asked was where to put the JRE.

I chose C:\Java\jre.1.8.0_101.

Install Cygwin

The initial installation shouldn't take all that long. There are plenty of articles on the net about installing / setting up cygwin. I will only touch on some points of cygwin.

After installation, you should see something like this:



Your computer user name would be in the place of 'the.transporter'. (don't judge me) :D

You probably won't have a local 'bin' folder yet.

If you open a cygwin terminal window, it will have dropped you right at

C:\cygwin64\home\<user>. But what you should see on the screen will look different.

Let's walk through each step in the screenshot below...

```
the.transporter@yomamita ~  
$ pwd  
/home/the.transporter  
  
the.transporter@yomamita ~  
$ ls -l  
total 16  
-rwxr-xr-x 1 the.transporter None 11383 Aug 7 03:18 _viminfo  
drwxr-xr-x+ 1 the.transporter None 0 Aug 25 02:49 bin  
drwxr-xr-x+ 1 the.transporter None 0 Jun 20 14:48 temp  
  
the.transporter@yomamita ~  
$ touch an-empty-test-file.txt  
  
the.transporter@yomamita ~  
$ ls -l  
total 16  
-rwxr-xr-x 1 the.transporter None 11383 Aug 7 03:18 _viminfo  
-rw-r--r-- 1 the.transporter None 0 Aug 25 03:01 an-empty-test-file.txt  
drwxr-xr-x+ 1 the.transporter None 0 Aug 25 02:49 bin  
drwxr-xr-x+ 1 the.transporter None 0 Jun 20 14:48 temp  
  
the.transporter@yomamita ~  
$ cd /cygdrive/c/cygwin64/home/the.transporter/  
  
the.transporter@yomamita /cygdrive/c/cygwin64/home/the.transporter  
$ pwd  
/cygdrive/c/cygwin64/home/the.transporter  
  
the.transporter@yomamita /cygdrive/c/cygwin64/home/the.transporter  
$ ls -l  
total 16  
-rwxr-xr-x 1 the.transporter None 11383 Aug 7 03:18 _viminfo  
-rw-r--r-- 1 the.transporter None 0 Aug 25 03:01 an-empty-test-file.txt  
drwxr-xr-x+ 1 the.transporter None 0 Aug 25 02:49 bin  
drwxr-xr-x+ 1 the.transporter None 0 Jun 20 14:48 temp  
  
the.transporter@yomamita /cygdrive/c/cygwin64/home/the.transporter  
$
```

Unix/Linux has a concept of a user's HOME directory. A short-cut way to display and deal with that is the "~". That's what you see after the end of the green prompt. Means we're currently in our "home" directory.

The "pwd" command means print working directory. It just means to show what is our current location.

The "ls -l" command does a long listing of the current directory contents. Like Windows' "dir". Once we know what is in our directory, we do a little test to prove something...

The "touch" can create a new (empty) file.

Repeating the "ls -l", we now see the new file, along with whatever else was there.

The "cd" command means **change directory** (just like in Windows) to whatever path is typed. I chose to start from the root(top) directory, which is (C:\) or (/cygdrive/c).

The cygwin bash command shell can auto-complete for you when you are typing valid paths or commands, by just hitting the <TAB> key a few times as you type.

Now, if we enter: "/cygdrive/c/cygwin64/home/<user>", and then do another "pwd", it would seem that we are elsewhere, from when we started.

But doing another 'ls -l', shows the very same file we 'touch'ed.

So, it proves that the following is all the same location:

```
/home/<user>  <-- typical unix-like path
/cygdrive/c/cygwin64/home/<user>  <-- cygwin-special
C:\cygwin64\home\<user>  <-- Windows
```

Another tip: using the up-arrow at the cygwin terminal, will show you a history of commands you have entered. You can also search for a command in the history, but since I have converted my command-line to act like a single-line vi editor, I have forgotten how to do a normal bash search. Hint: to convert to vi command-line mode, enter 'set -o vi'.

Verify JDK / JRE

We want to test our JDK / JRE installation.

At the cygwin terminal, start entering (and auto-completing) the path to the `jdk javac` compiler executable.

```
"cygdrive/c/jdk1.8.0_101/bin/javac.exe"<ENTER>
```

You should see a menu of options.

```
the.transporter@yomamita ~
```

```
$ /cygdrive/c/jdk1.8.0_101/bin/javac.exe
```

```
Usage: javac <options> <source files>
```

where possible options include:

```
-g Genera ..... (blah ..blah..blah)
```

We won't really be using the `javac` executable from the command-line, but we will be using `java`, so we check it ...

```
the.transporter@yomamita ~
```

```
$ /cygdrive/c/jdk1.8.0_101/bin/java.exe
```

```
Usage: java [-options] class [args...]
```

(to execute a class)

```
or java [-options] ..... (blah ... blah ... blah)
```

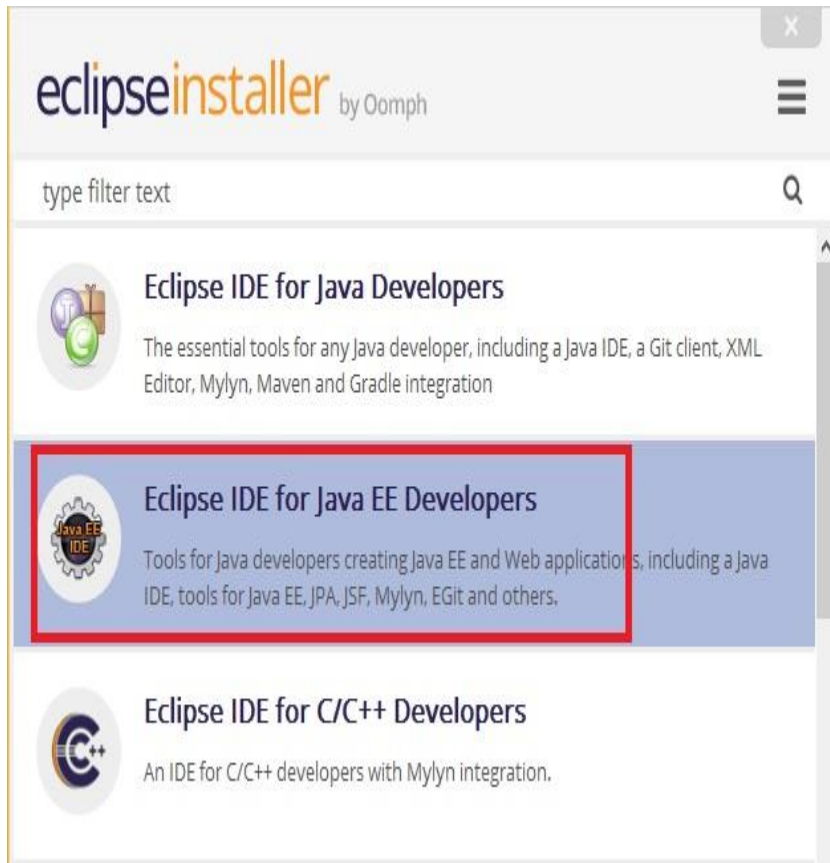
If you got anything else, there may have been a problem with the JDK installation.

We will come back to this topic a bit later. Let's move on.

Setup Eclipse IDE

I used to eschew IDEs. A command-line, editor (vi) and API web pages, home-grown build / deploy scripts die-hard. IDEs were for wussies. But I have to say they've come a long way. And - with eclipse, I can 'have-my-cake-and-eat-it-too' by a plugin ([vwrapper](#)) that mimics enough key-stroke codes of my favorite editor, and I still get all of the benefits of Eclipse.

For this project, it is Eclipse (Neon). I also used Spring's STS, but decided to stick with Eclipse. Others will have their preferences. For my installation, I selected the one for Java EE Developers.



Note, when I first launched Eclipse, I did NOT accept the suggested location for the workspace (where all projects will be kept). I again made a workspace folder right off the C:\ root.

We need to do a little house-cleaning before launching into a software project.

Java Compiler / Runtime

Go to Window->Preferences->Java->Compiler. On the right, you should see the JDK Compliance level set to 1.8. If you have an older Eclipse, that may not be the case.

Go to Windows->Preferences->Java->Installed JREs. Most likely, if you see anything under 'Installed JREs', it is a JRE. We need to change that.

At far right of dialog, click 'Add...'; (another dialog window), then 'Standard VM'; then 'Next>', (another dialog window), click on 'Directory...', (another dialog), and navigate to and select:

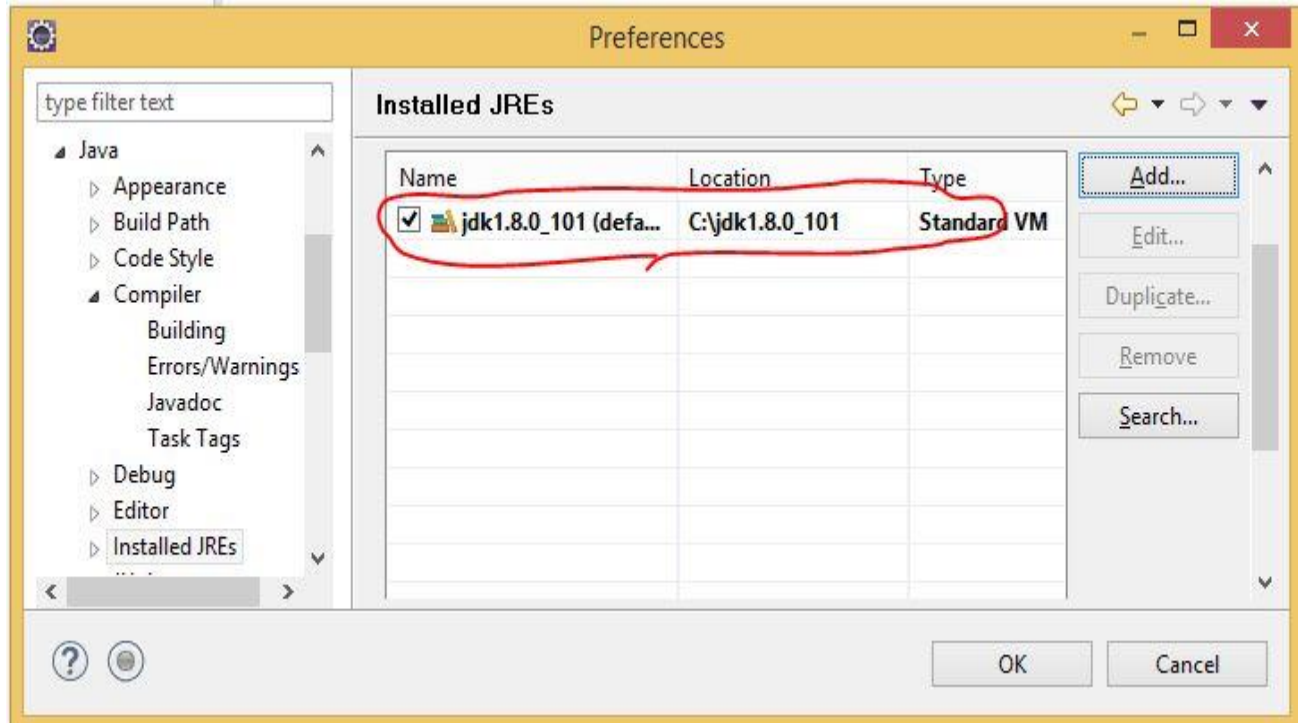
C:\jdk1.8.0_101

Click 'Ok'.

For newbies, this could be a bit confusing, since the dialog window seems to be asking about a JRE, but we selected the JDK.

Click 'Finish'.

Make sure the JDK/JRE selected for use is the one we just added:



Click 'Ok'.

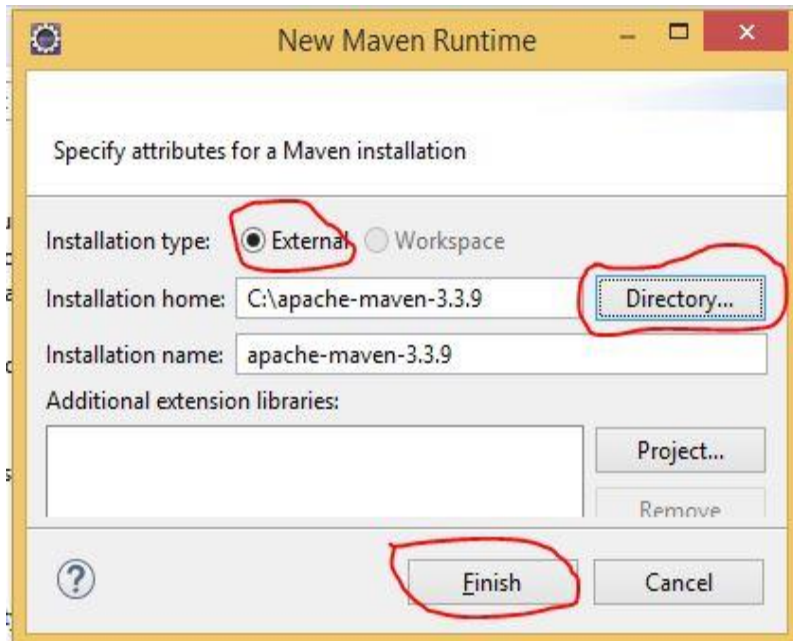
Setup Maven

Eclipse comes with an embedded Maven, and it also expects to place your local repository of downloaded jars into an .m2 folder that is in your Windows (not Cygwin) user location. I don't want to use the embedded Maven, and I don't want to have .m2 where it is, so...

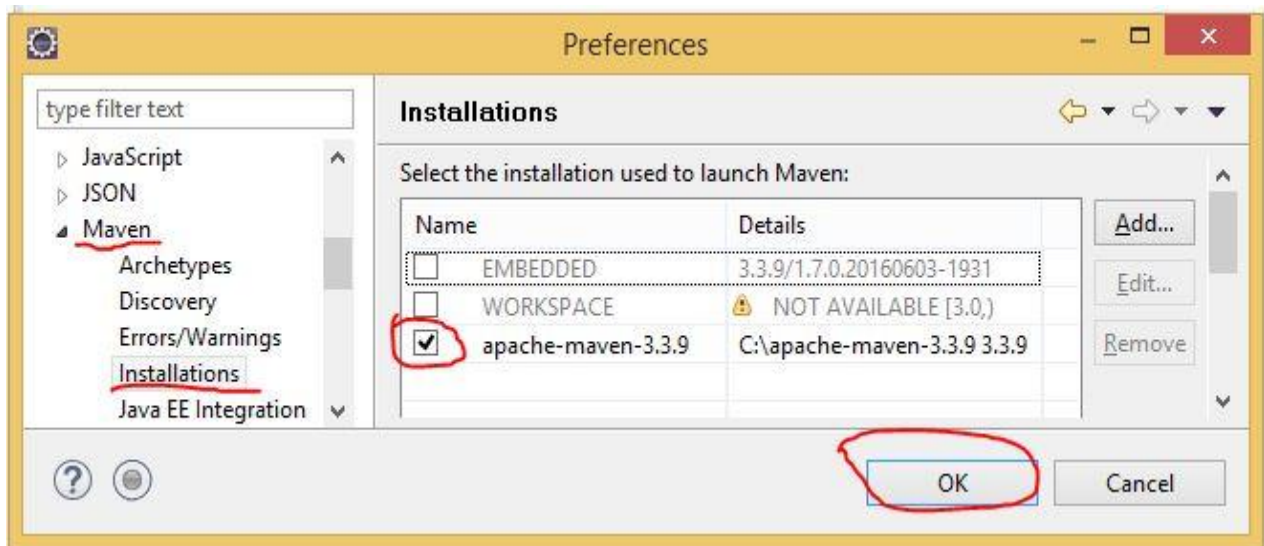
I downloaded Maven from <http://maven.apache.org/download.cgi> and unzipped it to (C:\).

Next, in Eclipse, go to Windows->Preferences->Maven->Installations.

At far right, click 'Add...'. Then do...



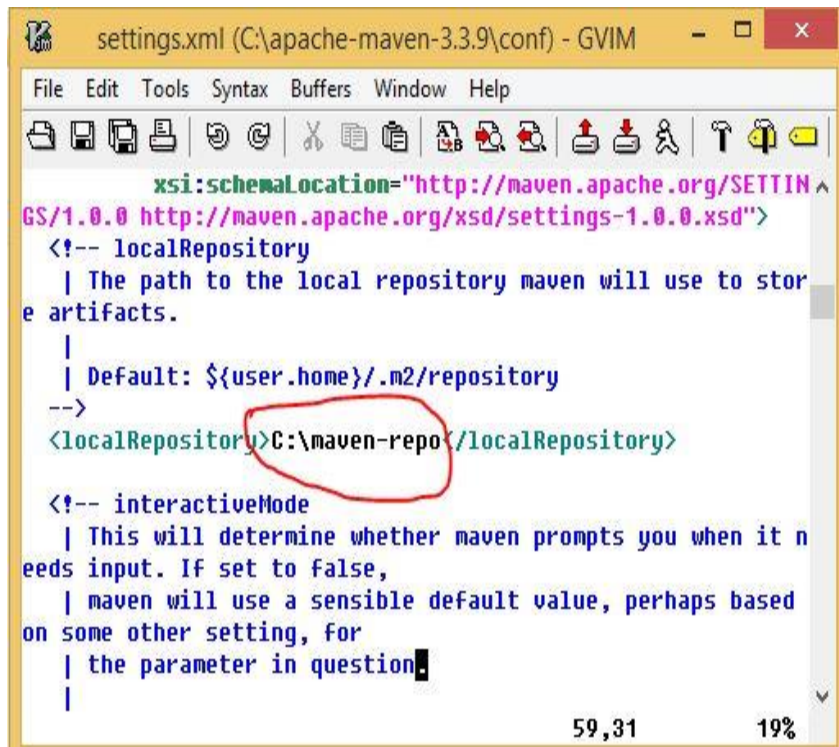
Once you click 'Finish', make sure you select the one you just added:



Next, in my file explorer, I created a new folder: C:\maven-repository.

Next, I went to my external maven installation (C:\apache-maven-3.3.9), and in the 'conf' folder I edited the 'settings.xml' file. I want to tell Maven to my local repository (C:\maven-repository).

Side note: I love the [gVim editor](#). You can go crazy with all the things it will do. Here I used it to edit the 'settings.xml' file.

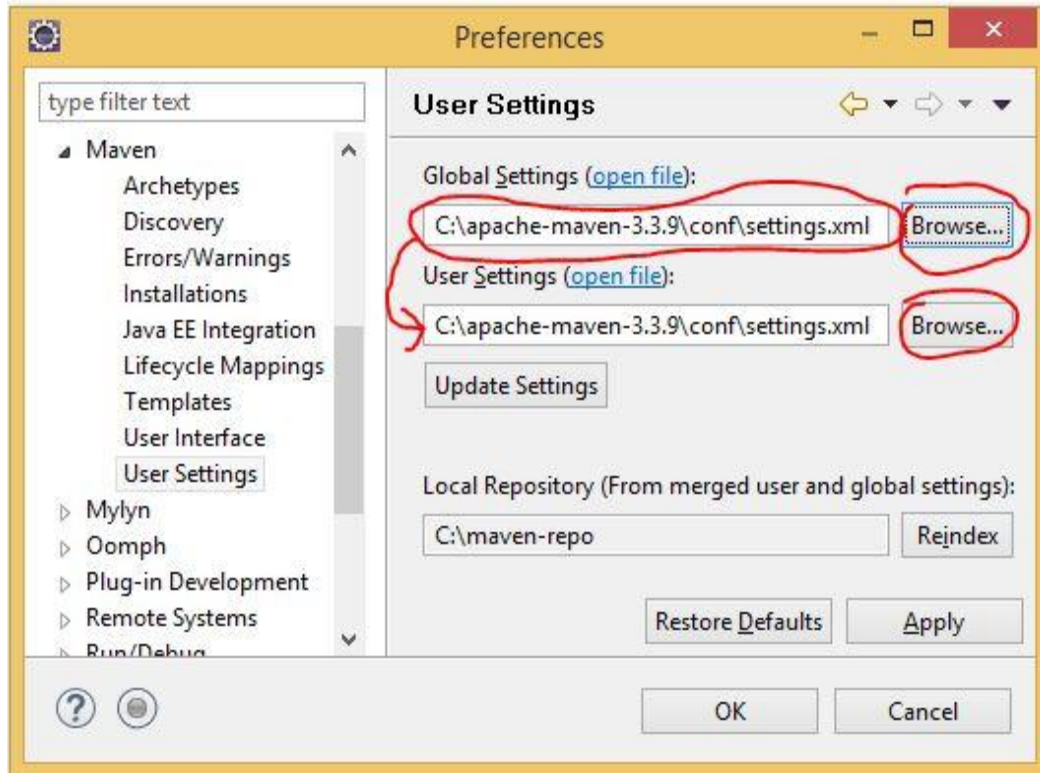


```
settings.xml (C:\apache-maven-3.3.9\conf) - GVIM
File Edit Tools Syntax Buffers Window Help
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd"
<!-- localRepository
| The path to the local repository maven will use to store artifacts.
|
| Default: ${user.home}/.m2/repository
-->
<localRepository>C:\maven-repo</localRepository>

<!-- interactiveMode
| This will determine whether maven prompts you when it needs input. If set to false,
| maven will use a sensible default value, perhaps based on some other setting, for
| the parameter in question
|
```

Going back in Eclipse, you want to go to Windows->Preferences->Maven->User Settings.

At far right, next to 'Global Settings:' box, click 'Browse...' and navigate and select that settings file. Repeat for 'User Settings:'. (see screenshot below)



Next, go to Windows->Preferences->Maven->User Interface, and click the check-box: 'Open XML page in the POM editor by default'

Ok, I think we are ready to begin doing some software development. (finally!)

Continued in next article....