**Reclamation Pisces Web Services Implementation**

Steve Malers, Open Water Foundation, 2016-01-06

Contents

[Introduction 2](#_Toc439907847)

[Eclipse Development Workspace Background 2](#_Toc439907848)

[Clone GitHub Repository 2](#_Toc439907849)

[Create Eclipse Development Workspace 3](#_Toc439907850)

[Import Projects into Eclipse 4](#_Toc439907851)

[Configure Eclipse Workspace 7](#_Toc439907852)

[Run Tomcat 7](#_Toc439907853)

[Deployment 8](#_Toc439907854)

[Appendix A – Developer Environment Software Installation 9](#_Toc439907855)

[Install Software for Development Environment ― MySQL, Java SDK, Tomcat, Eclipse Java EE IDE 10](#_Toc439907856)

[Install MySQL and Pisces Test Database 10](#_Toc439907857)

[Install Java Software 12](#_Toc439907858)

[Install Tomcat Software 17](#_Toc439907859)

[Install Eclipse IDE for Java EE Developers 18](#_Toc439907860)

[Appendix B – Setup of Eclipse Workspace and Software Project 24](#_Toc439907861)

[Eclipse Background 24](#_Toc439907862)

[Background on GitHub Repository, Eclipse Workspace, and Project 24](#_Toc439907863)

[Create Empty GitHub and Git Repository 26](#_Toc439907864)

[Create a New Eclipse Workspace and Initialize Maven Project in Git Working Files 27](#_Toc439907865)

[Add Maven Dependencies 31](#_Toc439907866)

[Define Tomcat Server to Use 39](#_Toc439907867)

[Add Java Code to Project for Web Services 43](#_Toc439907868)

[Configure Hibernate 44](#_Toc439907869)

[Configure Tomcat 44](#_Toc439907870)

[Configure Eclipse 44](#_Toc439907871)

[Configure Logging 46](#_Toc439907872)

[Run Web Services 51](#_Toc439907873)

[Test Configuration 51](#_Toc439907874)

[Git Configuration and Best Practices 51](#_Toc439907875)

# Introduction

This document describes how to set up the development environment for and perform development tasks for the Reclamation Pisces REST web services. The initial development has been performed by the Open Water Foundation (OWF). The implementation of web services uses Java technologies in order to leverage widely used open source components that are portable between operating systems. Additional background information is provided in specific sections to illustrate technical concepts necessary to understand the implementation. The instructions are provided for Windows 7 computers; however, the instructions should generally apply to Linux if the software developer prefers to use that operating system for development.

Appendix A – describes how to install necessary software development tools – this must be done before attempting to set up a development environment

Appendix B – describes how to configure the initial Eclipse workspace and project (Eclipse is used as the development interface)

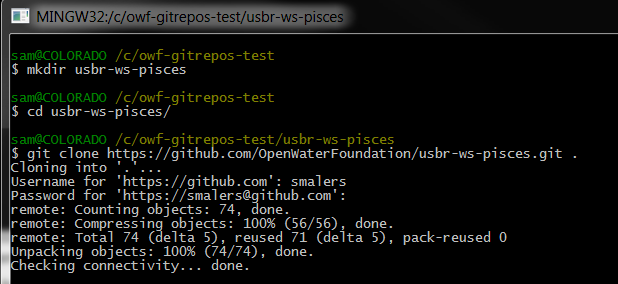
# Eclipse Development Workspace Background

The software is hosted as an Open Water Foundation GitHub repository (private as of January 6, 2016, although it may be made public).

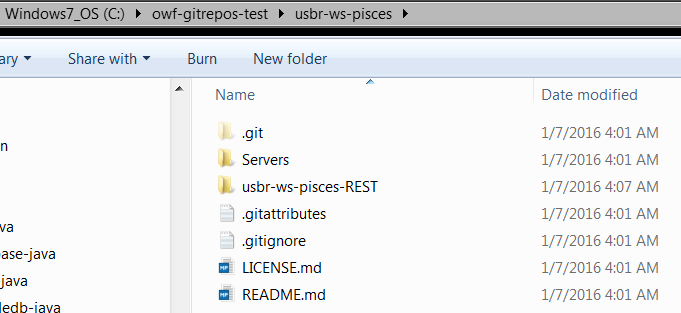
Eclipse development uses the concept of workspaces. Appendix B provides background on Eclipse terminology and file organization. The main concept is that Eclipse uses a “workspace” to provide a development sandbox and one or more “projects” can be managed under the workspace. The workspace corresponds to a developer’s preference in tools and development environment settings. For example, it would be possible to work on this project using NetBeans or a different environment. The projects under the workspace contain the software files and website configuration and should be development-environment-agnostic as much as possible. To isolate the cross-pollination of development environment configuration, it is recommended that for this project a new workspace is created to hold the project files. Additionally, project files should be aligned with the Git/GitHub repository folder, again, to allow flexibility in development. Consequently, the information below describes how to set up an Eclipse workspace that uses project files in a Git working folder location separate from the workspace folder (although linked).

# Clone GitHub Repository

The files in the GitHub repository include the software files and Tomcat web service configuration. A single repository includes the usbr-ws-pisces web service code, and the “Services” folder that contains Tomcat configuration for a local host computer, for development, each as an Eclipse project. To clone the repository to the local computer, determine a suitable location for local Git repositories and execute a command similar to the following. For example, use Git Bash on Windows, which checks out files to have CRLF as newlines (see Git Configuration section in Appendix B for why this is important).



The folders will then be as follows:

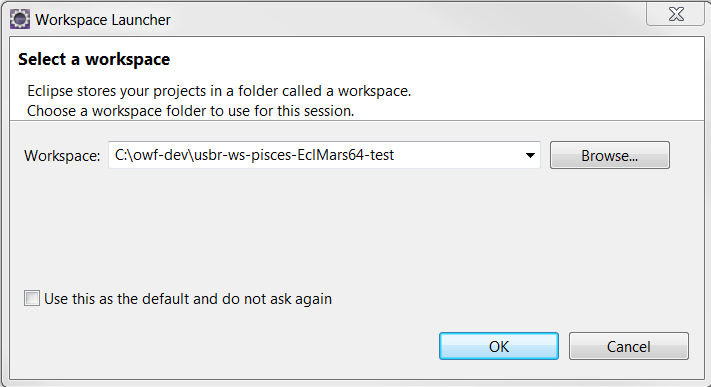


# Create Eclipse Development Workspace

For this work, use the Eclipse version that is recommended in Appendix A. It is OK to install and use different versions of Eclipse on the same computer as long as the correct one is used for this software project.

The Eclipse workspace will hold a reference to the project files and plugin files. The project is configured to store dynamic web content files under the plugin files and the workspace can be large (>100 MB) due to all of the Jar files from software packages. This is one reason why the project files in the repository are kept separate.

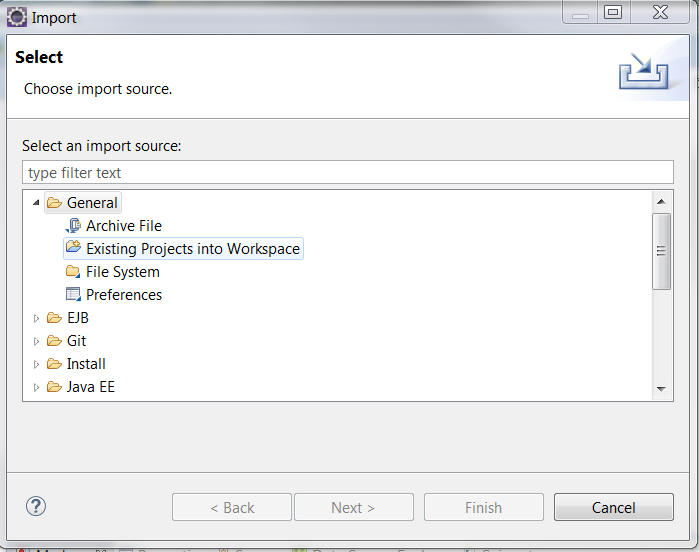
To create a workspace, open Eclipse and when prompted to select a workspace open a workspace similar to the following (the naming convention is arbitrary – pick something consistent with the local organizational or personal conventions):



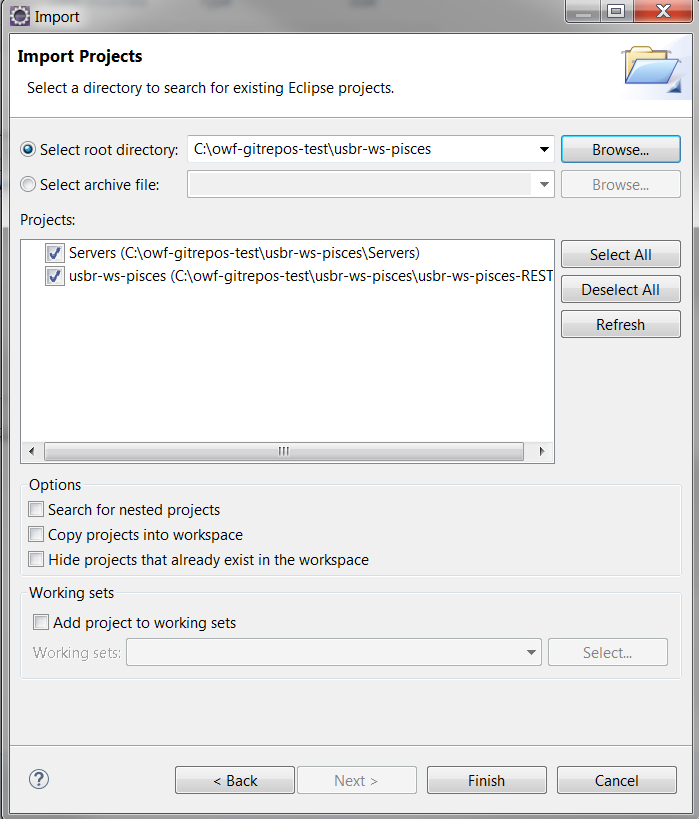
Eclipse will open with an empty workspace (no projects listed under the Project Explorer).

# Import Projects into Eclipse

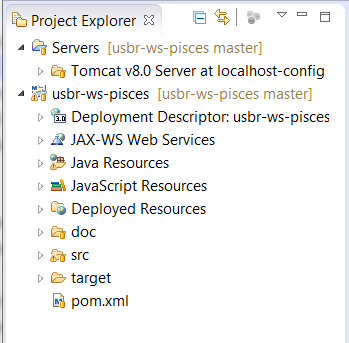
The software files in the Git working folder contain Eclipse projects, as indicated by the .project files under the “Services” and “usbr-ws-pisces-REST” folders. These projects need to be imported into the workspace. To do so, use the File / Import tool as shown below:



Select the workspace folder as shown below. Note that the checkbox “Copy projects into workspace” is NOT checked. This will allow the project files to live in the Git working files separate from the workspace folder.



This results in 2 projects in Eclipse:

****

Interestingly, when the Servers tab is viewed in Eclipse (if necessary use Window / Show View / Servers) it says that no servers are defined. Click on the link to define and select Apache Tomcat 8 and it then recognizes the “Servers” project from the GitHub checkout.

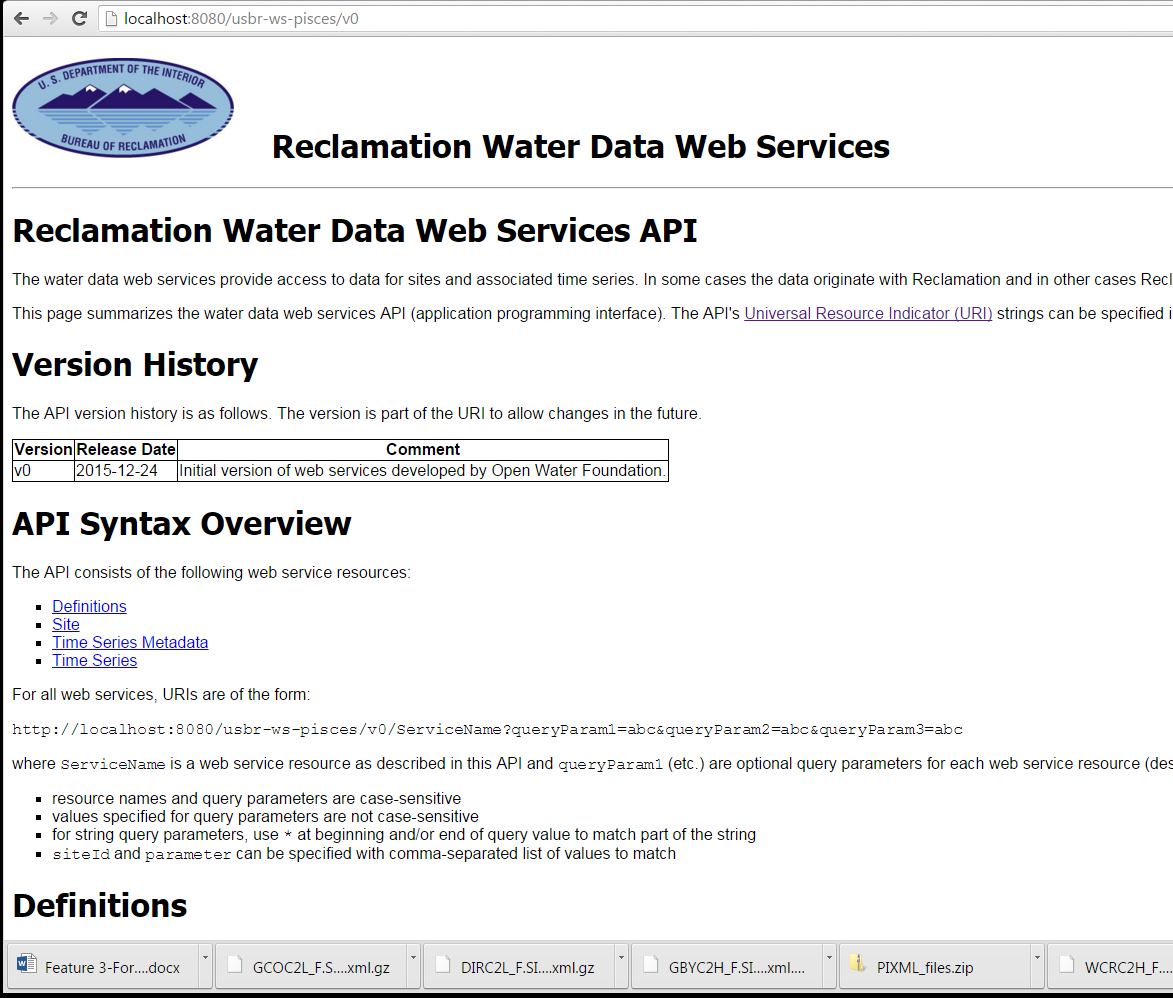
# Configure Eclipse Workspace

Because each new workspace essentially has default settings, need to configure the Eclipse workspace as per Appendix B, in particular:

* Set the runtime configuration to include logging information (if desired – otherwise default will be used).
* Right-click on “Tomcat v8.0 Server at localhost” and select “Add and Remove” – transfer the “usbr-ws-pisces” resource to the “Configured” area.

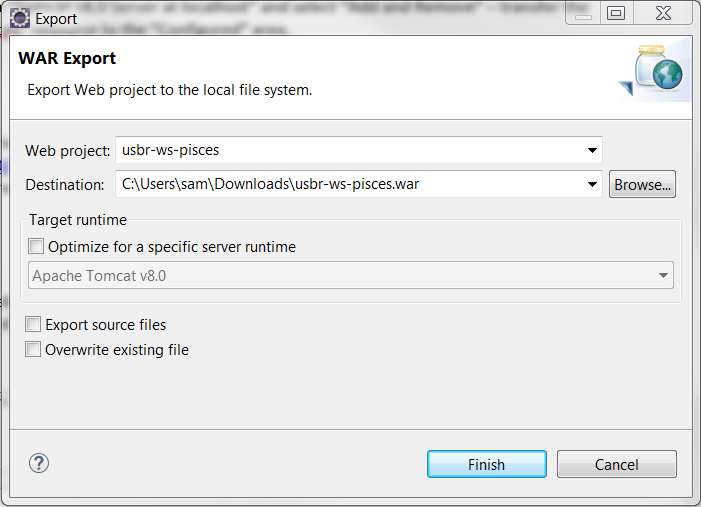
# Run Tomcat

The above configuration should result in the ability to open a browser and view the landing page at <http://localhost:8080/usbr-ws-pisces/v0>, and from there to access other web services. If not, use Appendix B to help troubleshoot. The landing page should look like the following:



# Deployment

The above ran the web services from the development environment. To deploy, right-click on the “usbr-ws-pisces” project and select Export / WAR File, which will essentially create a “web tar” file that contains the website. Ideally the WAR file could be created in C:\Program Files\tomcat8\webapps\usbr-ws-pisces.war but permissions may not be granted. In this case, save to a location that is allowed and then copy to the above location.



In Eclipse, stop the server that is running. Then run the installed Tomcat with C:\Program Files\tomcat8\bin\start from a command shell as administrator. Tomcat will automatically unzip the war file into expanded contents. If installed properly, accessing <http://localhost:8080/usbr-ws-pisces/v0> with a browser should show the web service landing page.

# Appendix A – Developer Environment Software Installation

This appendix describes the steps performed to install development environment software. An attempt was made to use the latest stable software in order to utilize the latest functionality and also ensure that the work would be relevant for the future as much as possible.

The steps in this appendix will generally need to be performed once by the software developer. Suitable modifications to software versions can be made as long as compatibility issues are dealt with.

One software component that is not included is the Git client. It is assumed that the developer is familiar with Git and has installed client software on the development computer.

# Install Software for Development Environment ― MySQL, Java SDK, Tomcat, Eclipse Java EE IDE

The MySQL database is used by Reclamation for the Pisces database and provides the data management solution.

Eclipse and related plug-ins is typically released once per year. The open source Eclipse Java EE IDE provides an interactive developer environment (IDE) for Java. The EE version of Eclipse includes a number of bundled plug-ins that are used with website development.

The Java standard development kit (SDK) is used, NOT the Java EE version. The Java EE version is basically the Glassfish servlet engine (still need to install Java SDK for development). Because the Tomcat engine is used, there is no need to install the Java EE version.

Tomcat is an open source servlet engine that runs Java web server applications. It can also provide content via HTTP but is typically only used to run Java programs (for example, use Apache for other content). Tomcat version 8 is used to take advantage of recent enhancements.

The following installation sequence is in the order necessary to avoid warnings. If software is installed in a different order issues may result.

## Install MySQL and Pisces Test Database

MySQL version 5.6 was downloaded and installed from: <http://dev.mysql.com/downloads/>.

Install the 64-bit version of the software using the instructions: <https://dev.mysql.com/doc/refman/5.6/en/windows-install-archive.html>

Create an options file as per: <https://dev.mysql.com/doc/refman/5.6/en/windows-create-option-file.html>

In the install folder copy my-default.ini to my.ini. Make the following changes (note for Windows that forward slashes are OK):

* 1. Remove the comment at the top about a template since it is a copy
  2. Set basedir = C:/Program Files/MySQL/MySQL Server 5.6
  3. Set datadir = C:/MySQL/data (copy the “data” folder from the MySQL install to this)
  4. Set port = 3306 (the default for MySQL)
  5. Do not set port (in 5.6 there is a documented issue that this always gets set to 1: <http://dev.mysql.com/doc/refman/5.6/en/replication-options.html>)
  6. Leave other settings commented out to use defaults.

Select the MySQL server type: See: <https://dev.mysql.com/doc/refman/5.6/en/windows-select-server.html>

The above basically says to use mysqld for an optimized binary and mysql-debug for a version compiled with full debugging. Use the optimized version unless there is a reason not to.

See: <https://dev.mysql.com/doc/refman/5.6/en/windows-server-first-start.html>

Since the software was installed with a zip file, start it a command shell window with:

“C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld”

Allow access if prompted.

Configure the root password:

See: <https://dev.mysql.com/doc/refman/5.6/en/default-privileges.html>

The new installation apparently has a root account with no password. To set the root password do:

mysql –u root

mysql> SET PASSWORD FOR ‘root’@’localhost’ = PASSWORD(‘insert-password-here’);

Now the following shows entries: Select User, Host, Password from mysql.user;

To facilitate starting the database, create a batch file to be run when starting:

rem Start the Pisces detabase

"C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld"

Similarly, create a batch file to be run when stopping (this will prompt for the root password):

rem Stop the Pisces detabase

"C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqladmin" -u root -p shutdown

Run MySQL as root:

mysql –u root –p (interactively enter password)

Create the database:

mysql> create database timeseries;

Next, restore the database from the dump file. For the following, change to the folder where the dump file is and put in the path to the mysql program since not in the path. This

“C:\Program Files\MySQL\MySQL Server 5.6\bin\mysql” –u root –pThePassword timeseries < Dump20150819.sql

Add a read-only user that will be used for the web services. Execute the following:

mysql> use timeseries;

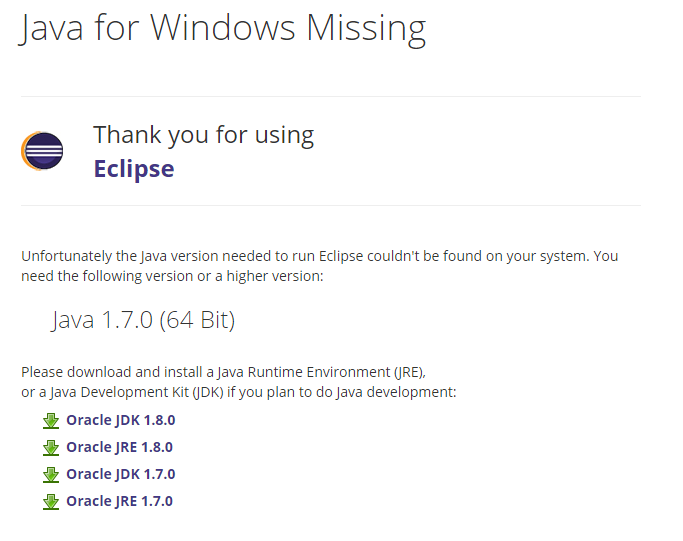
mysql> create user 'app\_user'@'%' identified by 'guest';

mysql> grant select on \*.\* to 'app\_user'@'%';

(may have to do the above with ‘app\_user’@’localhost’ as well).

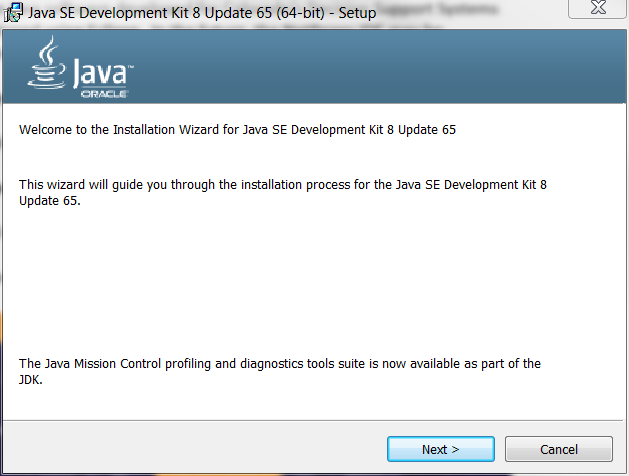
## Install Java Software

If you try to install Eclipse (as per next section) without having the proper version of Java installed, you may see the following:

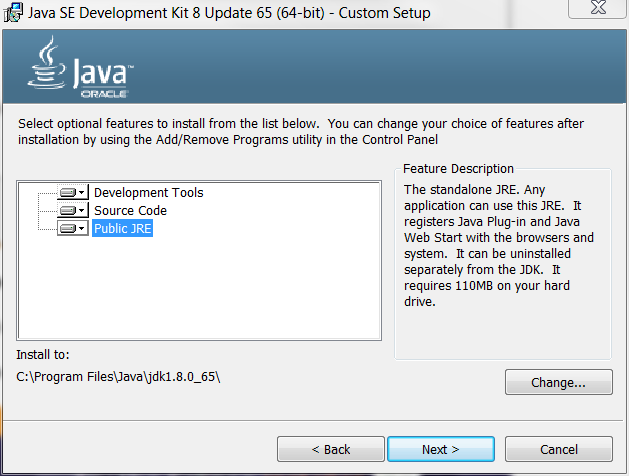


This may be true if to date only 32-bit Java has been installed on the development computer. Java 32-bit resides in C:\Program Files (x86)\Java. Java 64-bit resides in C:\Program Files\Java.

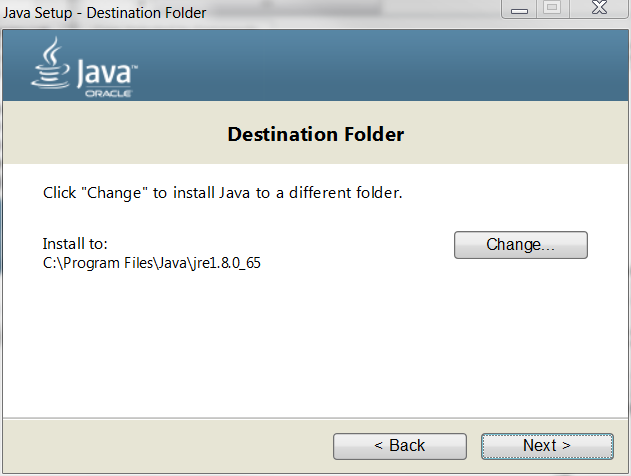
If not already installed, download and install JRE SE 1.8 since this is supported on CentOS and is certified for Windows 10, which are both environments that need to be supported going forward.



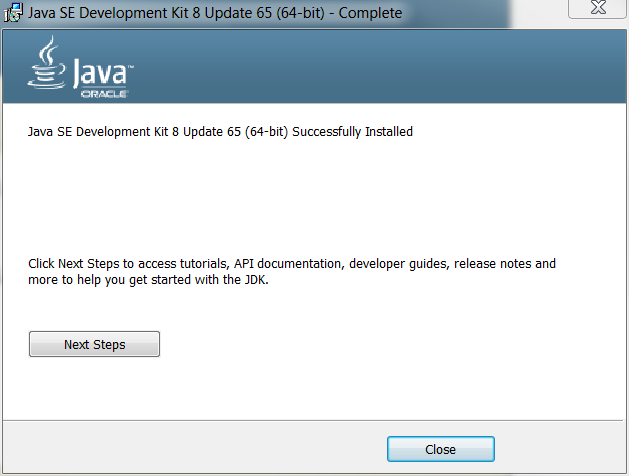
Press **Next >**



Press **Next >** with no changes.



Press **Next >** with no changes.

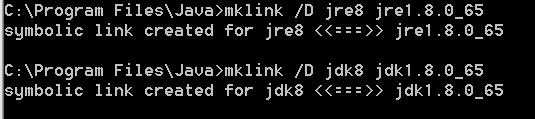


Press **Close**. This results in the following in C:\Proram Files\Java:

jdk1.8.0\_65

jre1.8.0\_65

To simplify configuration, create shortcuts by running a command shell as administrator. This is required because doing the “right-click” create shortcut menu does not appear to use the /D option.



## Install Tomcat Software

Useful links:

* http://tecadmin.net/install-tomcat-8-on-centos-rhel-and-ubuntu/#

Download Tomcat 8 from tomcat.apache.org/download-80.cgi. Version 8 is used because it is compatible with Java 8 and provides useful new features. For these instructions, a Windows 64 computer is used so download the 64-bit Windows zip. For example, install the files to:

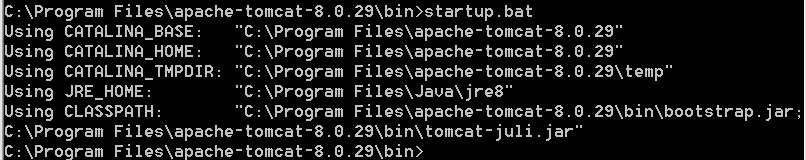
C:\Program Files\apache-tomcat-8.0.29

As per the RUNNING.txt file, create a setenv.bat file in the bin folder with the following contents, pointing to the Java 8 version installed as per the previous section.

set "JRE\_HOME=%ProgramFiles%\Java\jre8"

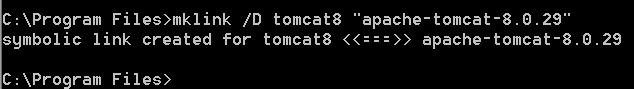
exit /b 0

In a command shell run the setenv.bat script and then run startup.bat in the bin folder.



A command shell window should be shown with “Tomcat” as the title. Open <http://localhost:8080> in a web browser and it should show the Tomcat local home page. This page will not be used going forward because the project website will be used instead.

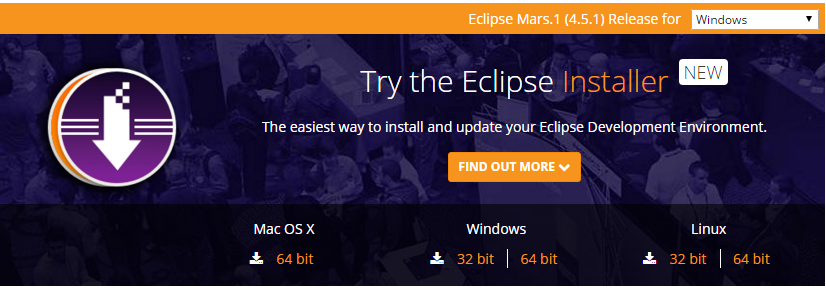
Similar to Java, it is useful to create a symbolic link to simplify access of Tomcat software without the long version. Do the following using a command shell that was opened as administrator. This link can be removed and recreated later if a newer version of Tomcat is installed.



## Install Eclipse IDE for Java EE Developers

The Windows version of Eclipse for Java can be downloaded from the following site:

https://eclipse.org/downloads/

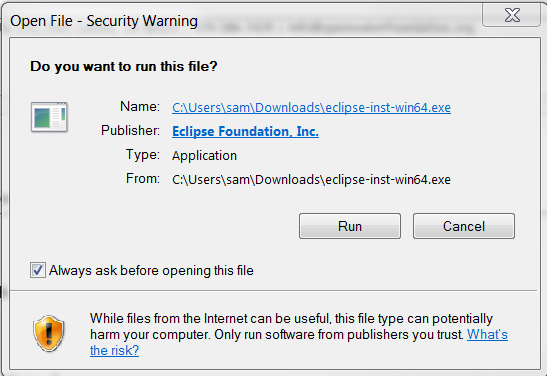


Although it is possible to download a zip file, the new Eclipse Installer was used. Because the focus of initial Java EE development is for web services to be run on a 64-bit Linux machine, download the 64-bit Windows development environment (development will occur on Windows and then deploy to Linux). Other downloads can be used as needed.

The installer filename is as follows:

eclipse-inst-win64.exe

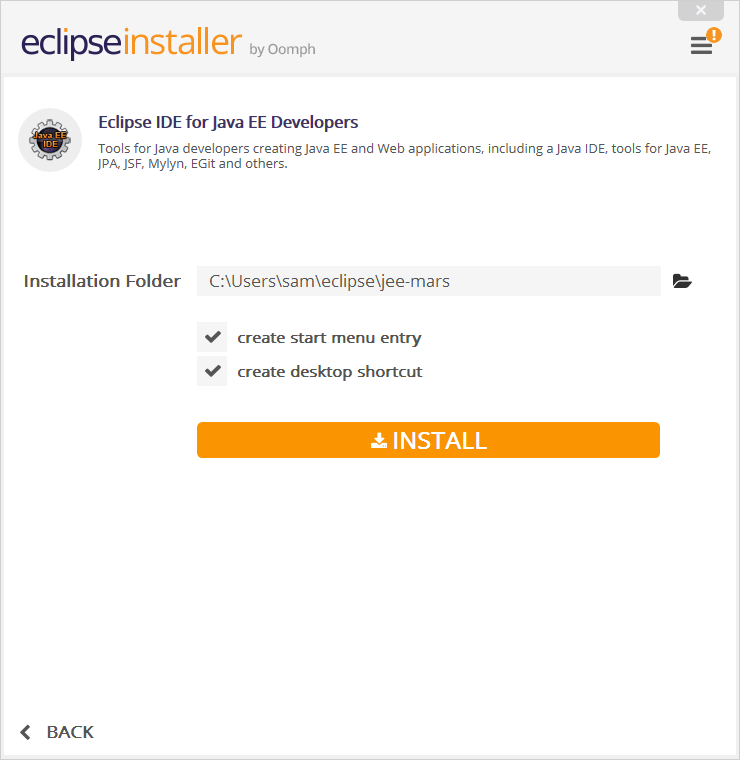
Run it as administrator.



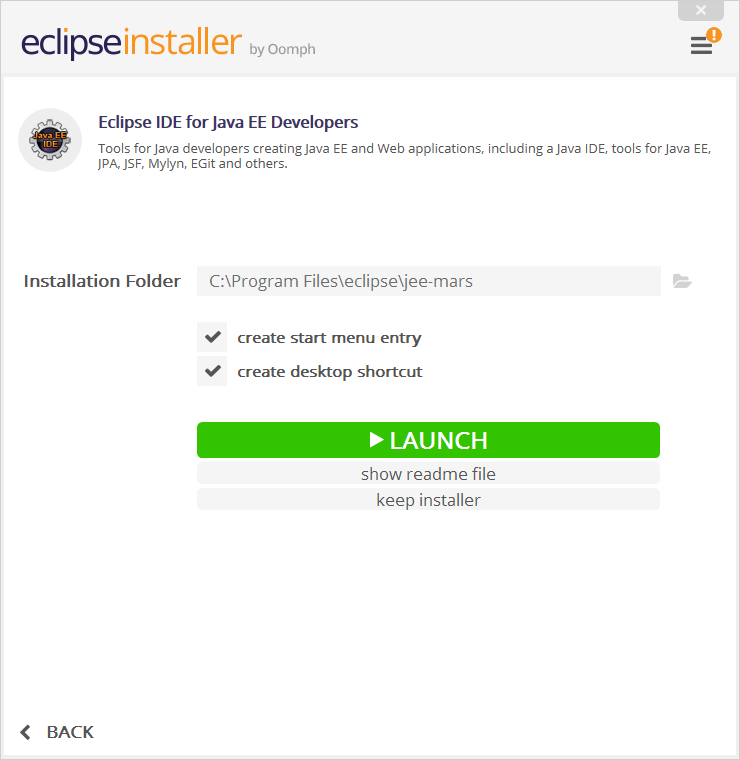
Press **Run** to run.



The installer will try to install under the users home folder as shown below:



Change to a shared location to simplify comparison with previous installs as shown below.



A desktop shortcut will have been created by the installer as follows:



To avoid confusion, change the name to include the Java version and 64, as follows:



Additionally, edit the shortcut properties as follows.

General tab: Eclipse Java8 JEE Mars 64

Shortcut tab – “Target”: "C:\Program Files\eclipse\jee-mars\eclipse\eclipse.exe" –vm “C:\Program Files\Java\jdk8\bin\javaw.exe” -vmargs -Xmx1024M

* -vm is related to installing and configuring Java
* -vmargs sets the maximum memory to use for Eclipse

Shortcut tab – “Start in”: "C:\Program Files\eclipse\jee-mars\eclipse"

Similarly, right click on the “Eclipse JEE Mars” start menu and edit properties to set the menu to “Eclipse Java8 JEE Mars 64” for clarity in case other Eclipse versions are installed. This is an issue if different software development projects require different tools.

At this point, Eclipse, Java, and Tomcat are installed and configured to start development.

# Appendix B – Setup of Eclipse Workspace and Software Project

This appendix describes how the Eclipse software project was originally set up in order to initialize development. It is assumed that developer environment software was installed as per Appendix A. This only needs to be done once by the original developer and then the project can be checked out for development, as per the main body of this document. The following defines a working software project. However, a number of trial and error steps occurred during setup and initial development. Highlights below indicate lessons learned that may prove helpful in the future. Once the steps in this appendix have been executed, then development can occur as per Appendix A and the main section of this document.

## Eclipse Background

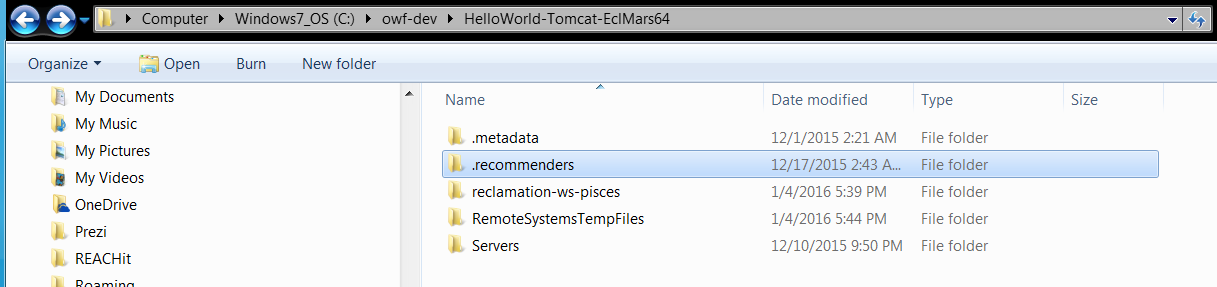
Eclipse and its plugins use four main locations for files:

1. Software installation (C:\Program Files\eclipse\...) – generally don’t need to modify other than start-up configuration (start menu or eclipse.ini file under the software installation folder), for example to specify virtual machine to use to run Eclipse.
2. Workspace (C:\SomeFolder\... folder with name selected by developer) – indicated by .metadata sub-folder, which includes Eclipse software configuration and plugin files for the workspace configuration – generally let Eclipse software and plugins manage these files but sometimes need to manually edit if user interface is limited
   1. The .metadata folder contains a folder .plugins\org.eclipse.core.resources\.projects that lists projects managed within the workspace
   2. Various binary files contain configuration information for projects in the workspace, including the path to project files, which may exist outside of the workspace folder
   3. Files may be copied to \*.snap file when Eclipse is running as a snapshot
3. Project (for example corresponding to the Git repository) – software files indicated by .project sub-folder (these tend to be somewhat agnostic of the above configurations so that developers can check out files with different versions and configurations of Eclipse, and even other developer environments). These files should be in revision control.
   1. A Dynamic Web Project
4. Plugin files – may exist under the user’s home folder, such as the Maven repository files under the .m2 folder (for Maven version 2). The location of such files is dependent on the plugin.

Of the above, only the Project files should be stored in revision control such as Git/GitHub. Workspace files are large and different developers can have different development tools. Eclipse allows a workspace to use project files in a separate location than the workspace. This is desirable because it allows only the project files to be placed under revision control – it would be a mess if different developer’s Eclipse configuration interfered with each other.

## Background on GitHub Repository, Eclipse Workspace, and Project

The initial version of the web services project was created in a workspace folder that also included the project files and was not saved to GitHub. This approach was taken because Maven was used for project dependencies and it was desired to see what folders Eclipse would create. The following was the result:



The folders are as follows:

.metadata

– Eclipse workspace

.recommenders

– created to help with code completion, dynamic

reclamation-ws-pisces

– the Eclipse project for the web services

RemoteSystemsTempFiles

– Eclipse dynamic files, not needed

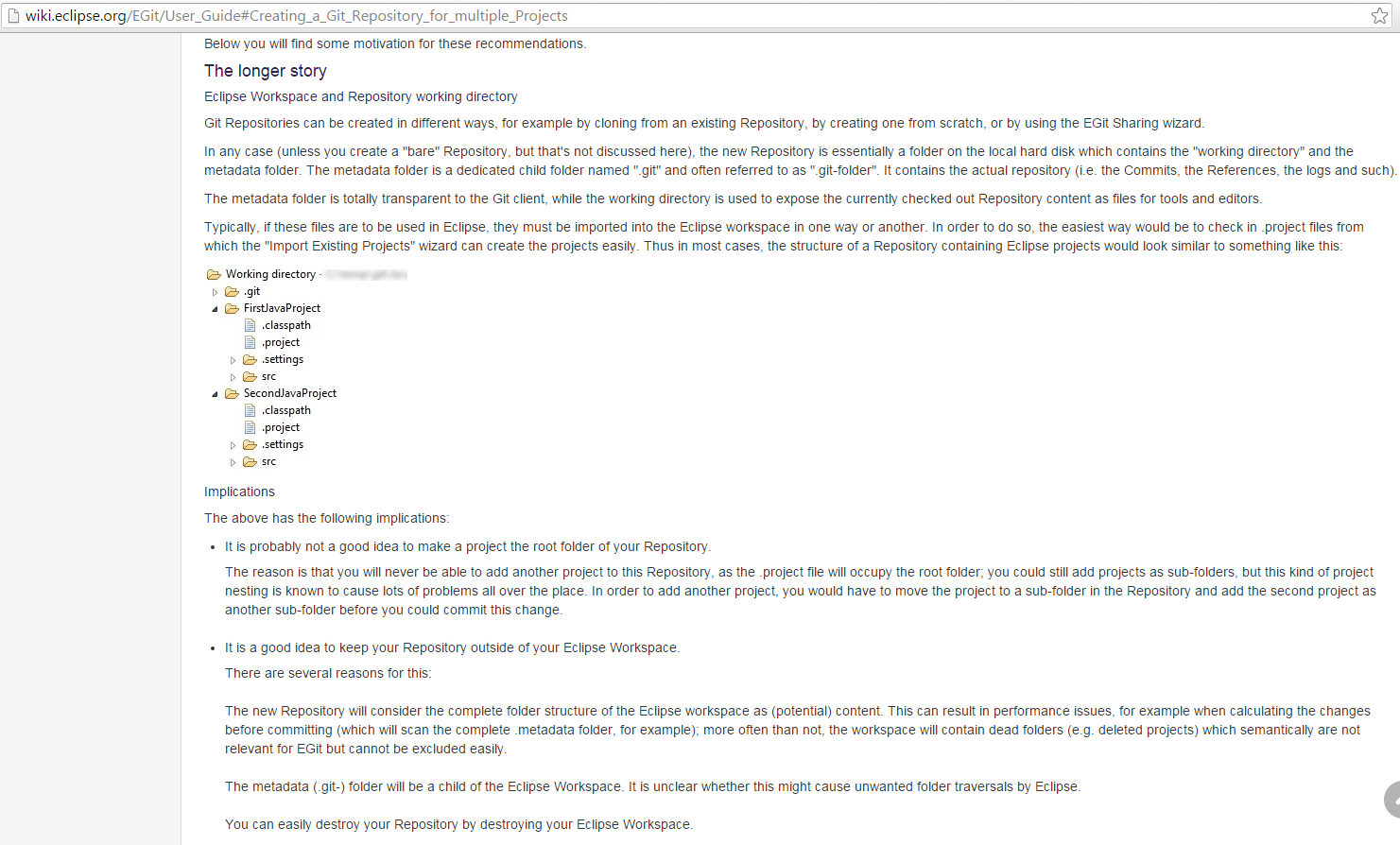
Servers

– configured Tomcat servers, used to start up Tomcat for use in Eclipse, might be considered part of the project, but can also vary by developer computer so need to configure as part of the environment – below will see that Eclipse makes it difficult to NOT include in the workspace folder so had to adjust so it would be in the repository (desirable so that Tomcat configurations are standardized if possible)

The end goal is to have:

1. Eclipse workspace configured for the project, but not including the project files. The Eclipse workspace files for the initial version were over 100 MB and include Tomcat working files.
2. GitHub repository and working directory that corresponds to the project, with appropriate .gitignore configuration
3. Additional environment configuration as appropriate for the development environment. One question is… where will the “Servers” folder live, in the workspace?

However, another option is to have a GitHub repository that includes multiple Eclipse projects, including the web services and the Servers project. This is consistent with Eclipse recommendations:

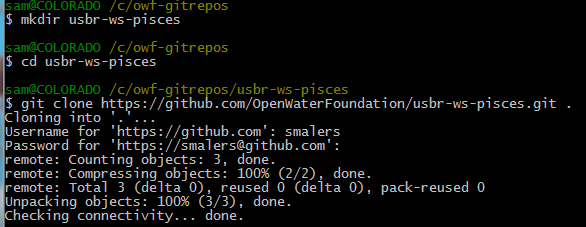


This is the approach that will be taken for setup of the final GitHub repository and Eclipse configuration.

## Create Empty GitHub and Git Repository

On the OWF GitHub site, create an empty private repository: usbr-ws-pisces

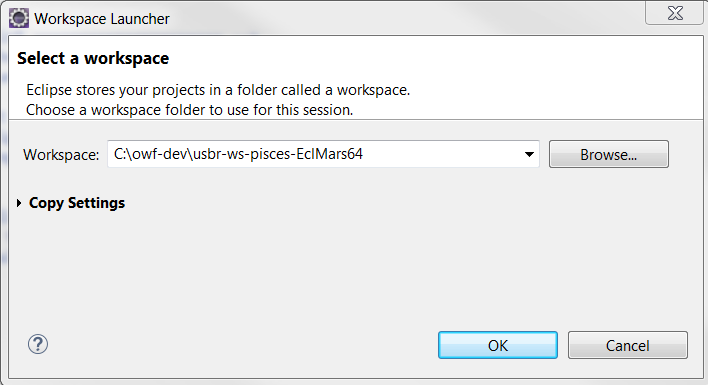
Clone the empty repository using Git Bash. The convention was to check out to a folder C:\owf-gitrepos, which contains OWF Git repositories.



The above steps could also be done with Eclipse but the above is more straightforward.

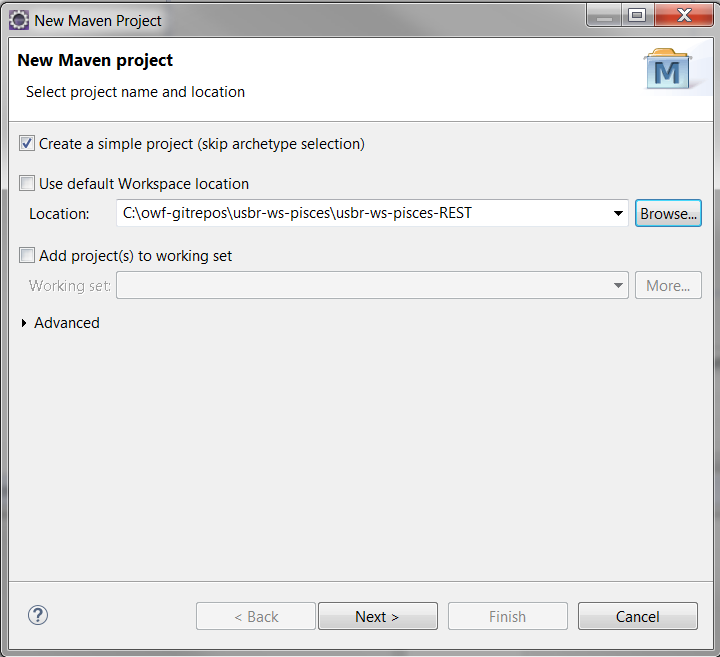
## Create a New Eclipse Workspace and Initialize Maven Project in Git Working Files

Create a new Eclipse workspace in Eclipse. Create a new folder for the workspace as indicated below:

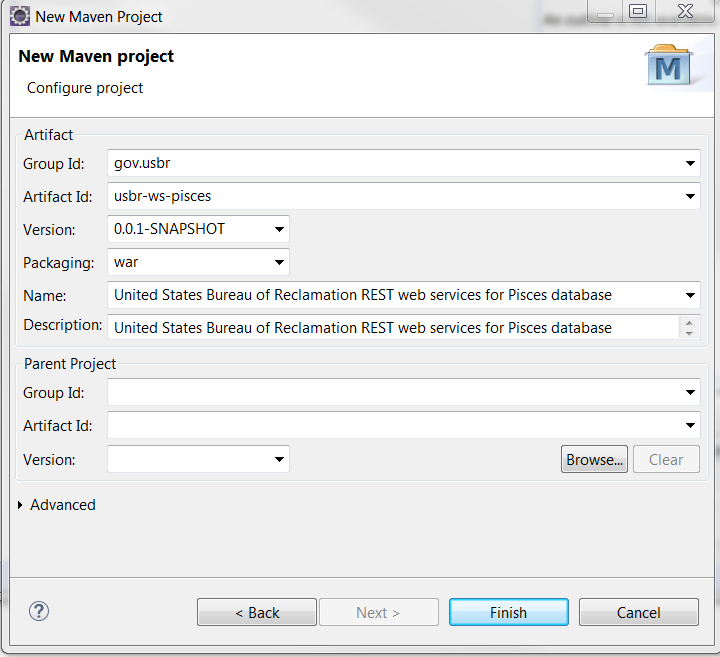


The C:\owf-dev folder is used for development workspaces but is separate from the Git repositories. The folder name indicates the software project and the version of Eclipse that is being used. This is helpful if other development environments need to be used for some reason.

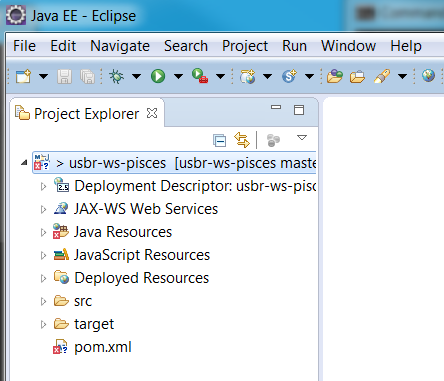
Start a new Maven project using File / New / Maven Project and specify the project folder as follows:



Press ***Next >*** and then enter information as below:

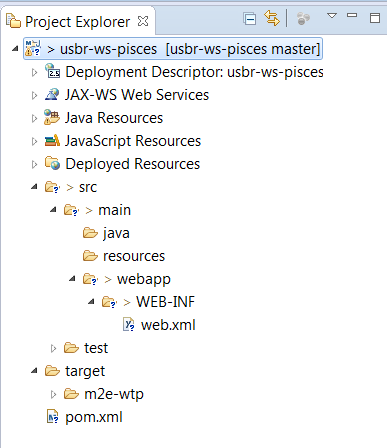


Press ***Finish***. The resulting output looks as follows:



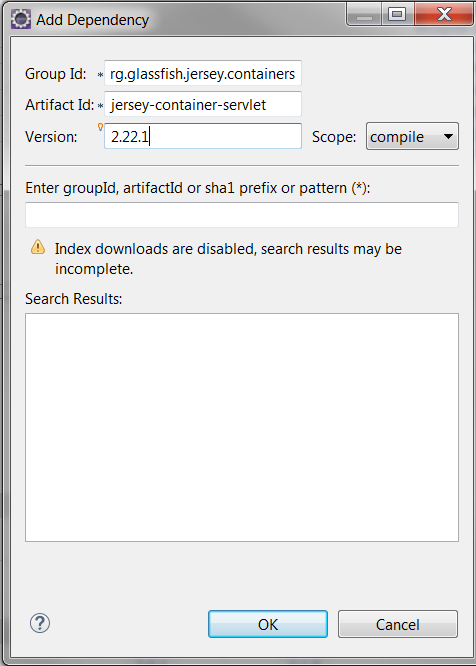
Note that Eclipse has recognized that the project files are managed in a Git repository and is showing the repository name as [usbr-ws-pisces master]. Clicking on “pom.xml” and then using the “pom.xml” viewing tab provides access to the error message. The pom.xml file is used by Maven to indicate external Jar file dependencies.

From previous experience, the solution to the error is to right-click on the project “usbr-ws-pisces” and select Java EE Tools / Generate Deployment Descriptor Stub. This auto-generates the empty web.xml file and other web files as shown below:



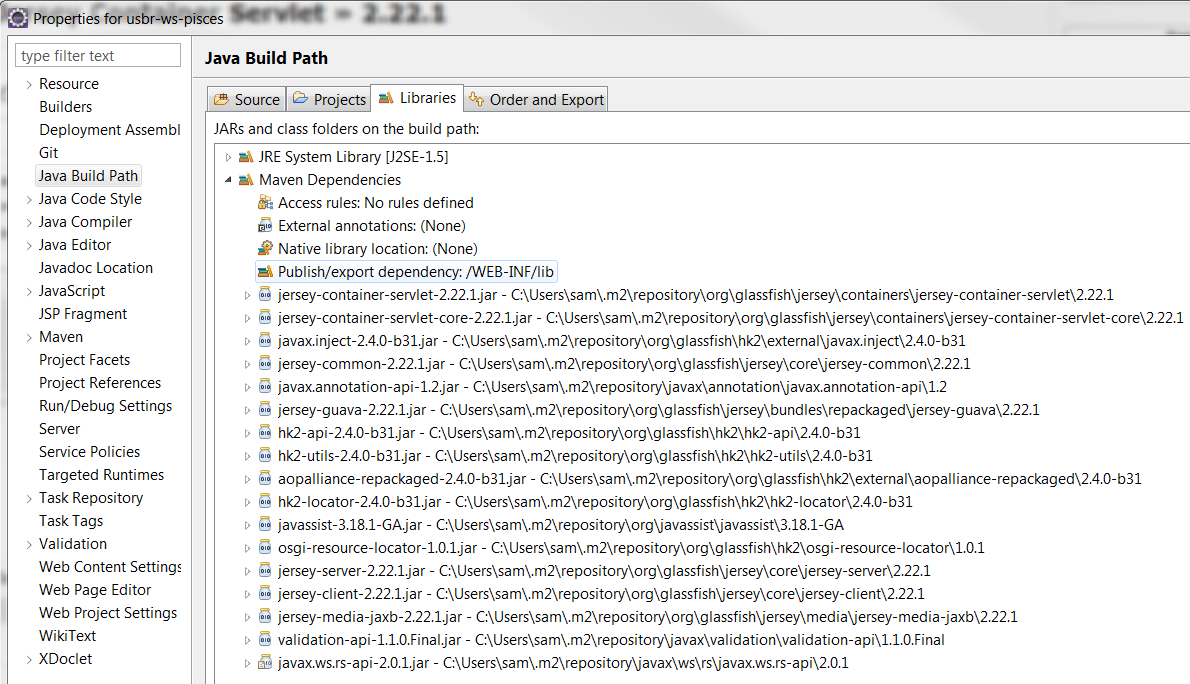
## Add Maven Dependencies

From experience, the following Maven dependencies need to be added. In the initial version these were added incrementally but in this clean version they are added at once here. Right click on the project and then Maven / Add Dependency. The following is to add the Jersey REST web services:



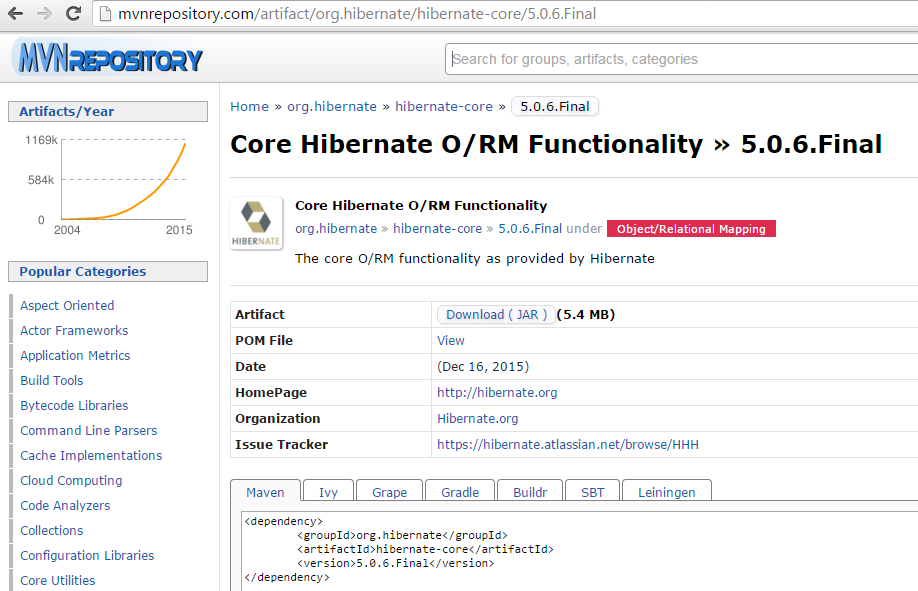
Press OK. This results in:

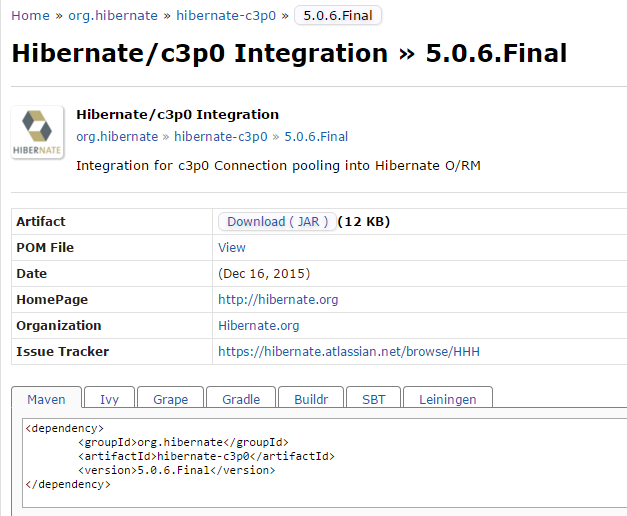
* the pom.xml file being updated with dependency
* the C:\Users\xxxx\.m2 folder being updated with jar files downloaded from the central repository
* the Java build path for the project is updated to include the Maven dependencies as shown below – note that these Jar files are not included in the project files and therefore not the repository; however, the Maven dependency list is included in the project files and repository

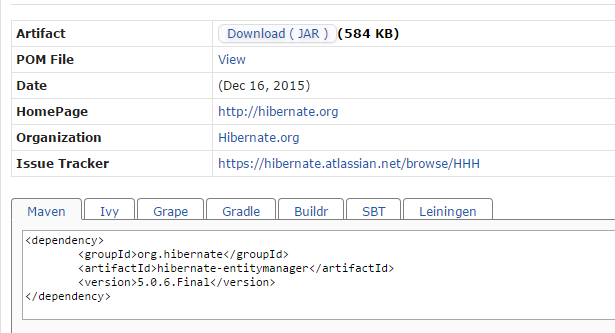


The following are the Maven Central repository dependencies that should be added similarly (first is for Jersey, then Hibernate for database persistence [core and JPA support, needed for some of the features that were tried], MySQL database driver, Jackson for JSON serialization, OGC WaterML 2.0, and Logback logging). The WaterML 2.0 dependencies appear to cause some issues with the project, which are dealt with below, perhaps because of additional dependencies.





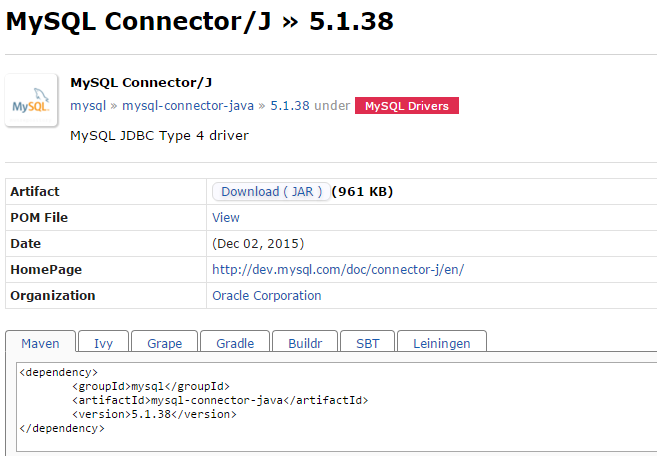




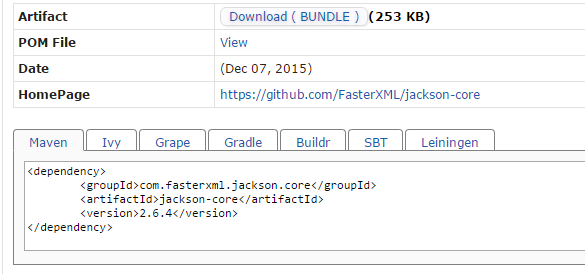
See: <http://stackoverflow.com/questions/18125294/hibernate-4-javax-transaction-systemexception-error>

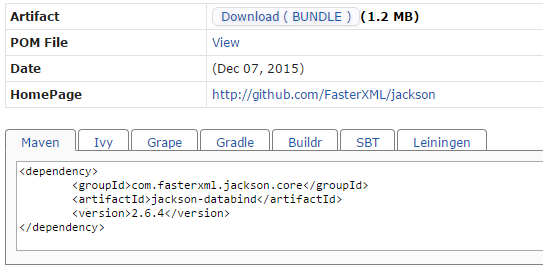
So add the following dependency:





See: https://github.com/FasterXML/jackson-docs/wiki/Using-Jackson2-with-Maven







For WaterML 2.0 see:

<http://www.ogcnetwork.net/jaxb4ogc>

http://confluence.highsource.org/display/OGCS/Schemas



For logging, see the following (SLF4J and logback is chosen as the logging framework binding, although Tomcat uses java.util.logging and JULI internally… more on logging configuration below): http://www.slf4j.org/manual.html

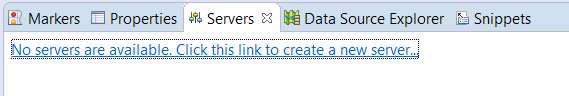


## Define Tomcat Server to Use

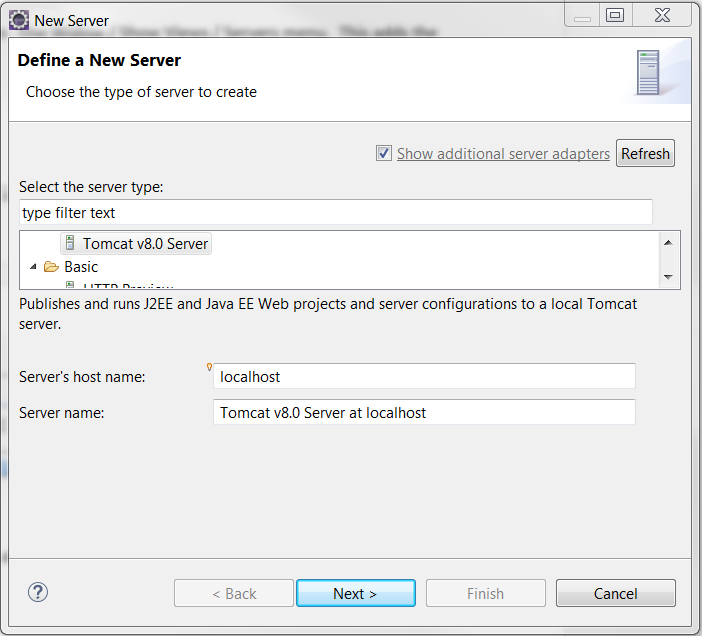
Tomcat was installed separately from Eclipse EE IDE. There are various ways to configure how to use Tomcat, including:

1. Having Eclipse automatically deploy to the installed Tomcat file location (probably a bad idea because it is nice to test in the development environment and then deploy via a “war” file)
2. Having Eclipse use a temporary location in the development environment (this is what is used - files are a bit buried but makes sense once it is figured out)
3. Specify a custom location (seems like more trouble than it is worth if the second option works).

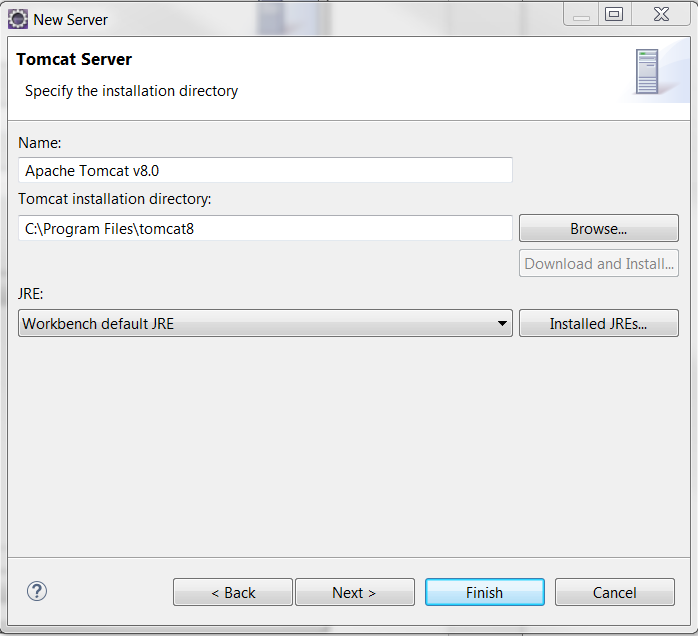
Servers are not shown by default so use the Window / Show Views / Servers menu, which results in the following:



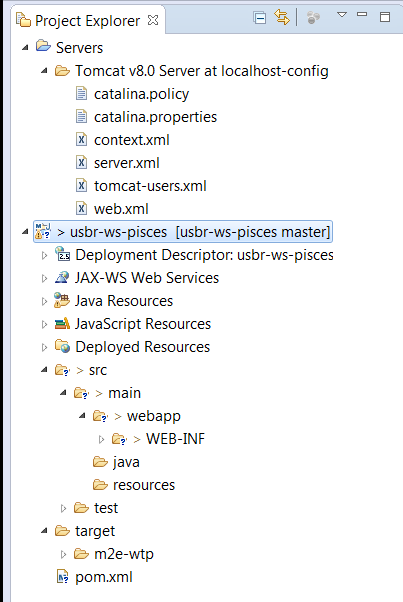
Click on the link and fill out the dialog as follows:



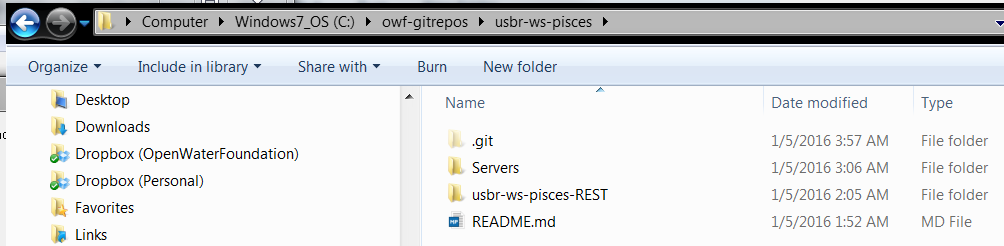
Then specify the Tomcat location in the following dialog. Note that the more generic “tomcat8” folder is specified, which is a symbolic link that was defined when Tomcat was installed (an OWF recommendation). This allows the configuration to update Tomcat by changing the symbolic link.



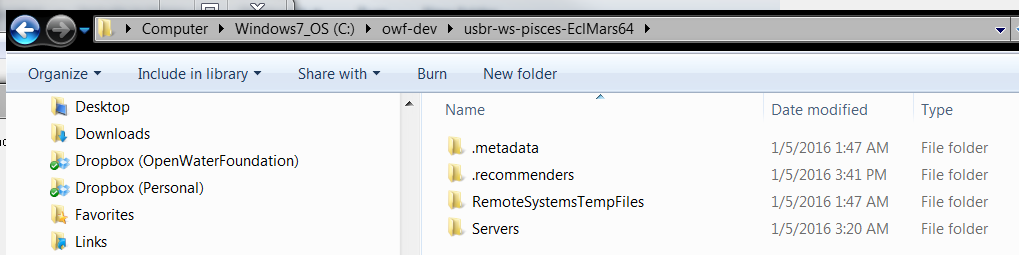
Pressing Finish results in a new project being added to the Eclipse project, as shown below:



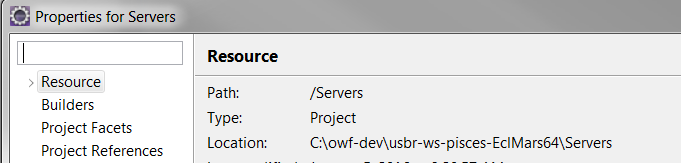
The “Servers” folder was created under the Eclipse Workspace, which means it will not be included in the repository. Tried to move using the file system view and moving from the Eclipse workspace folder to the Git working directory, but the “Servers” project was corrupted and had to be removed. So… it looks like “Servers” will need to live under the project and not be in the repository. As long as the server name is consistent between developers hopefully references from the main project will not need to be changed. Actually, what resulted on the file system is files under the Git working directory:



AND also in the workspace:



Both versions seem to be getting updated by Eclipse. Viewing the properties on “Servers” in Eclipse shows that it things the files are in the workspace folder and not under the Git working folder:



Refer to the main document, which describes how to clone the repository and do development – this issue may require more time to understand.

## Add Java Code to Project for Web Services

Based on the initial draft add code to create a simple web service application. Start by defining a class under the src / main / java folder under the PiscesRest.java class and then copy in other classes. Edit the code to change the package name to that discussed with Karl Tarbet. The content of the code files was determined by experimenting with Hibernate, Jersey, and Jackson APIs to implement the desired web service functionality.

## Configure Hibernate

To configure Hibernate, add the hibernate.cfg.xml file under the src/main/resources folder. The main information is to set the database connection information and the classes that implement persistence (in this case read-only classes).

## Configure Tomcat

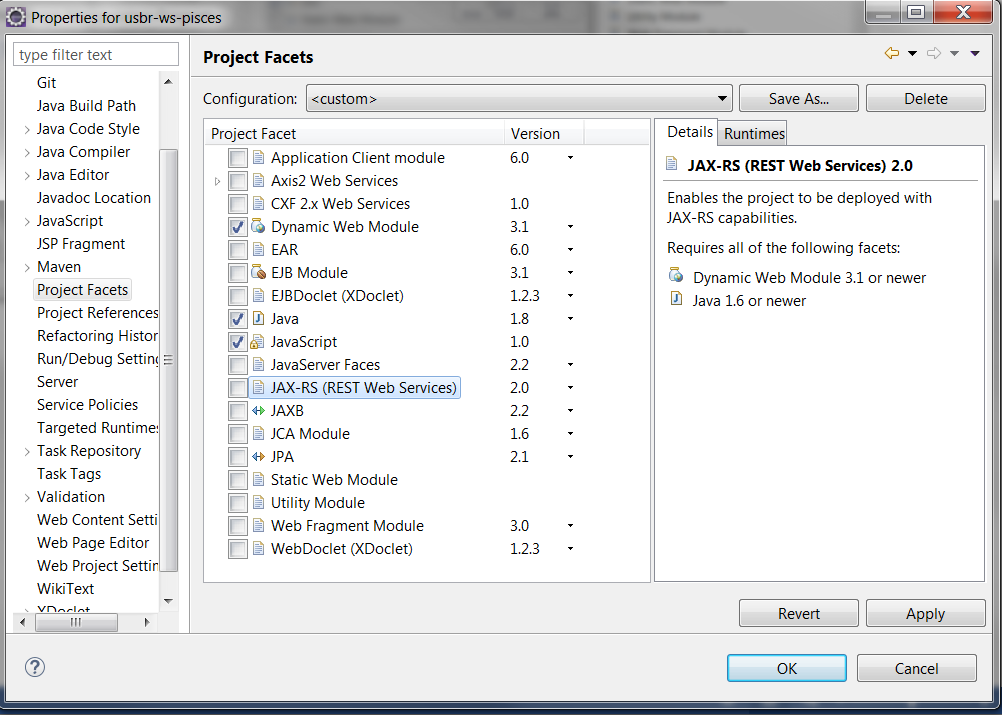
To configure Tomcat, add web.xml and other files under the src/main/webapp/WEB-INF folder.

Also, right-click on the server and use Add and Remove… menu to move the usbr-ws-pisces project into the Configured area.

## Configure Eclipse

Out of the box there are some configuration issues that need to be resolved:

1. Use the Window / Show View / Other / General / Problems view to show issues.
2. A warning may be shown “Build path specifies execution environment J2SE-1.5. There are no JREs installed in the workspace that are strictly compatible with this environment”. Fix with:
   1. Edit the Java Build Path and edit the Libraries to use JavaSE-1.8.
3. Error “Java compiler level does not match the version of the installed project facet”
   1. Facets are a way to configure how the IDE interfaces to the project but don’t change the underlying code: <http://stackoverflow.com/questions/1809918/what-is-facet-in-javaee>
   2. Edit the project properties Project Facets and change to look like:



1. Error “Cannot change version of project facet Dynamic Web Module to 3.0”:

Facet is set to 3.1 and if go into the Project Facets and try to change to anything but 3.1 it won’t let me!

See: <http://stackoverflow.com/questions/15115009/maven-java-ee-configuration-marker-with-java-server-faces-1-2>

Problem appears to be that in web.xml there is version=3.0 and the Dynamic Web Module version cannot be greater than this. Had to manually edit under the project folder the .settings/ org.eclipse.wst.common.project.facet.core.xml file to have:

<?xml version="1.0" encoding="UTF-8"?>

<faceted-project>

<fixed facet="wst.jsdt.web"/>

<installed facet="wst.jsdt.web" version="1.0"/>

<installed facet="java" version="1.8"/>

<installed facet="jst.web" version="3.0"/>

<installed facet="jst.jaxrs" version="2.0"/>

</faceted-project>

Then did an update on the Maven project.

## Configure Logging

Tomcat by default uses the java.util.logging and JULI for logging, see: <https://tomcat.apache.org/tomcat-8.0-doc/logging.html>

The SLF4J logging package is also enabled in an attempt to unify and control logging of the web service code, mainly to start adding diagnostic messages to code for development.

The standard location for Tomcat logging configuration is %CATALINA\_BASE%\conf\logging.properties

When run through Eclipse, the following is printed to the console:

INFO: CATALINA\_BASE: C:\owf-dev\usbr-ws-pisces-EclMars64\.metadata\.plugins\org.eclipse.wst.server.core\tmp0

Jan 06, 2016 1:01:15 AM org.apache.catalina.startup.VersionLoggerListener log

INFO: CATALINA\_HOME: C:\Program Files\tomcat8

Jan 06, 2016 1:01:15 AM org.apache.catalina.startup.VersionLoggerListener log

Therefore the logging properties used should be in tmp0\conf\logging.properties. The following is in the catalina.bat file in the Tomcat bin folder:

rem LOGGING\_CONFIG (Optional) Override Tomcat's logging config file

rem Example (all one line)

rem set LOGGING\_CONFIG="-Djava.util.logging.config.file=%CATALINA\_BASE%\conf\logging.properties"

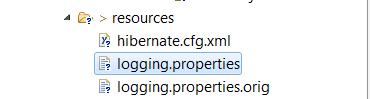
rem

rem LOGGING\_MANAGER (Optional) Override Tomcat's logging manager

rem Example (all one line)

rem set LOGGING\_MANAGER="-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager"

Consequently, to change the global logging settings for Tomcat, the tmp0\conf\logging.properties can be set. However, a similar file also exists under the resources and is copied to WEB-INF/classes for deployment. This is the file that should be edited because it is specific to the application.



Sometimes a serious error occurs, and the server won’t return the content, for example:

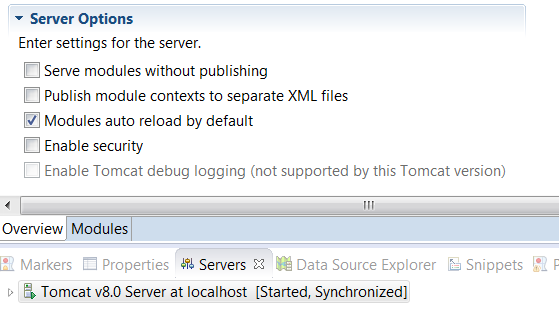


The content of the access log tmp0/logs/localhost\_access\_log.YYYY-MM-DD.txt only shows entries like the following:

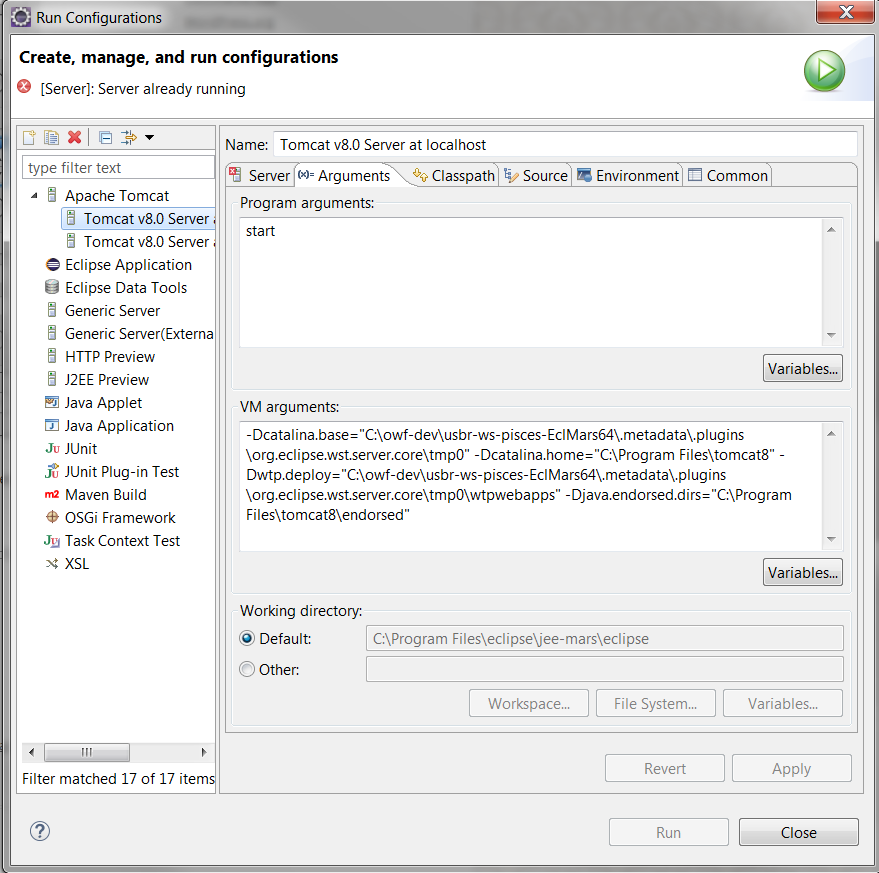
127.0.0.1 - - [06/Jan/2016:01:01:18 -0700] "GET / HTTP/1.1" 404 994

0:0:0:0:0:0:0:1 - - [06/Jan/2016:01:04:16 -0700] "GET /usbr-ws-pisces/v0 HTTP/1.1" 500 2582

The log messages for exceptions is a different file but by default does not seem to get created under the tmp0/logs folder. Double-clicking on the specific server in the Eclipse interface to edit the properties shows the following (debug disabled for some reason):



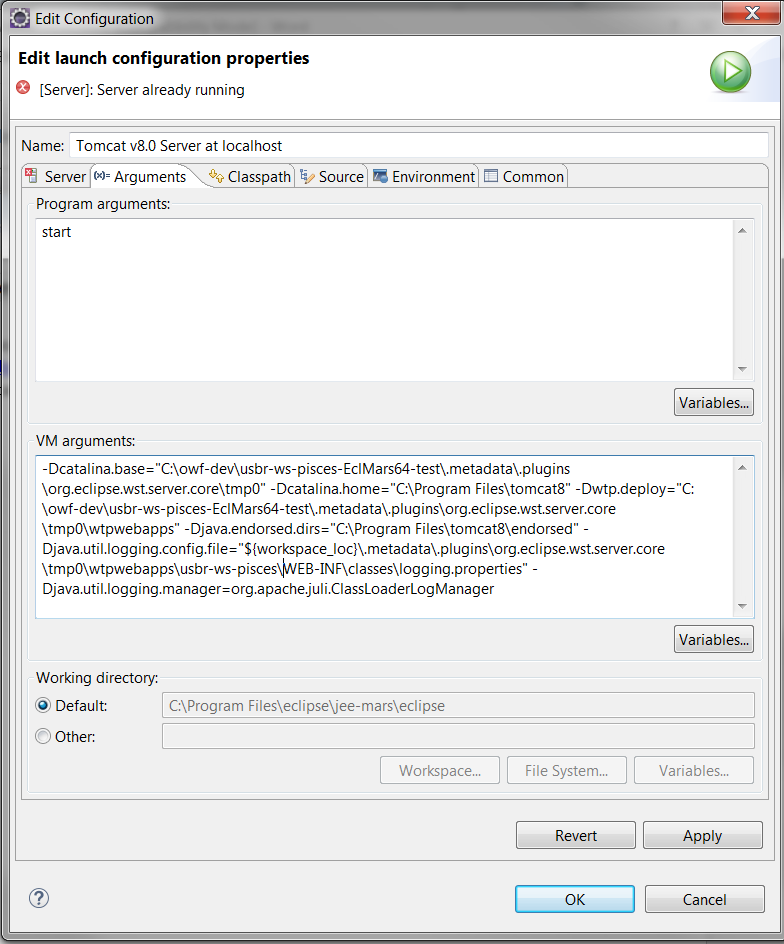
The core issue is that Tomcat when run from Eclipse does not run the standard startup script (see: <http://happygiraffe.net/blog/2008/08/01/tomcat-logging-in-wtp/>). Editing the run configuration for the “Tomcat v8.0 Server at localhost” (see the green right-arrow in the toolbar) shows the following:



This illustrates a few issues that need to be resolved:

* The paths to the workspace are hard-coded and could be an issue if sharing files. However, trying to use ${workspace\_loc} or other variable for some of the basic Tomcat parameters gets reset by Eclipse!
* There is no specification of logging configuration

The following shows the new settings (add the last two properties):



Restarting Tomcat outputs a lot of information to the console in Eclipse, too much actually if the logging level is set to ALL. More importantly, there are now “catalina.YYYY-MM-DD.log” and similar files in the tmp0/logs folder.

Dial back the levels to INFO and see what happens… get more reasonable level of output but still not seeing exception corresponding to HTTP 500 error. Compared with the original work and realized that I missed one Maven dependency: javax-transaction and had an error in the web.xml. Fix that and the web services work.

Hibernate prints some debug information to the Eclipse console but this is not logged in the Tomcat logs (Hibernate uses SLF4J and not the JULI logging). To control, it is necessary to set properties for logback, which is the run-time implementation of SLF4J (see: http://logback.qos.ch/manual/configuration.html). Therefore, add a file logback.xml in the “resources” folder. This gets copied to the WEB-INF/classes folder at runtime.

## Run Web Services

To restart Tomcat and make the web services available, right-click on the “Servers” tab and select “Restart”. Then view the web service pages in a browser for example with top-level URL:

<http://localhost:8080/v0>

If any errors result, look at the console messages in Eclipse. If web services are working, there should not be any exceptions and queries are echoed to the console because the hibernate.cfg.xml file indicates to do so (show\_sql=true).

See the logging configuration section above for additional troubleshooting.

## Test Configuration

Tests were not implemented for the web services but if they were would exist in the src/test folder according to normal Eclipse/Java test standards.

## Git Configuration and Best Practices

The following actions were performed to ensure that the Git repositories are properly configured to allow check-out and development by other developers:

* See best practices for handling end of line issues when developers are a mix of Windows and Linux:
  + <http://adaptivepatchwork.com/2012/03/01/mind-the-end-of-your-line/>
  + Consequently, use a .gitattributes file for Java from here: <https://github.com/Danimoth/gitattributes>
  + Eclipse JGit plugin DOES NOT recognize .gitattributes and appears to ALWAYS commit the native newline representation (CRLF on Windows, LF on Linux), unless global Git settings are defined.
  + Bottom line:
    - Use .gitattributes as per above template – works for Windows and Linux
    - Do not use Eclipse Git for check-out or commit of files (only for comparison) – use tools that recognize .gitattributes like Git GUI/Git Bash
* Added a README.md to explain the repository contents
* Added a LICENSE.md consistent with other federal repositories
* Added a .gitignore file to avoid committing class files to the repository – Jar files are not committed because Maven is used to manage dependencies
* Based on experience cloning the repository, the usbr-ws-pisces/src/test/resources folder is needed but was empty in the development environment – add a .gitignore file in this folder to ensure that