Contents

[Install Summary 1](#_Toc369593918)

[SVN Access Information 1](#_Toc369593919)

[Maven Install 1](#_Toc369593920)

[Tortoise SVN Install 4](#_Toc369593921)

[Maven Basics 5](#_Toc369593922)

[Custom Component Build Example 6](#_Toc369593923)

[Checking out artifacts 6](#_Toc369593924)

[Updating Artifacts 8](#_Toc369593925)

[Installing Artifacts 8](#_Toc369593926)

[Packaging Contents 10](#_Toc369593927)

[Tagging an Artifact 13](#_Toc369593928)

[Cleanup Steps 18](#_Toc369593929)

[Maven Artifact Repository 18](#_Toc369593930)

# Install Summary

**Maven:** Install the latest Maven version onto your computer

<http://maven.apache.org/download.html>

**Tortoise SVN:** Install the latest version of Tortoise SVN or any other SVN client onto your computer. **When installing the SVN client be sure to include the command line options.**

http://tortoisesvn.net/downloads

**Java 6+ JDK:** Install JAVA 6 or newer JDK onto your computer. Be sure to install the Java Development Kit (JDK) and not the Java Runtime Environment (JRE)

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

# SVN Access Information

Custom Component Code Management Project: <https://code.google.com/p/custom-components-svn/>

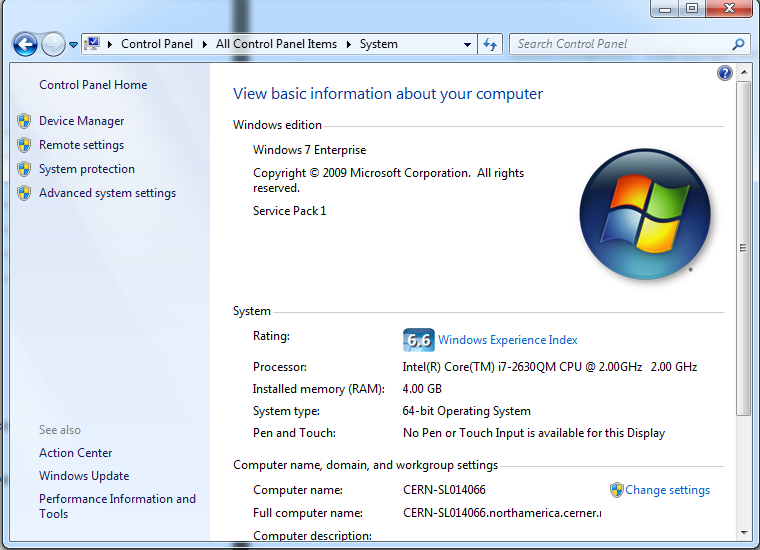
SVN URL: <https://custom-components-svn.googlecode.com/svn/>

Username: [customcomponentsuser@gmail.com](mailto:customcomponentsuser@gmail.com)

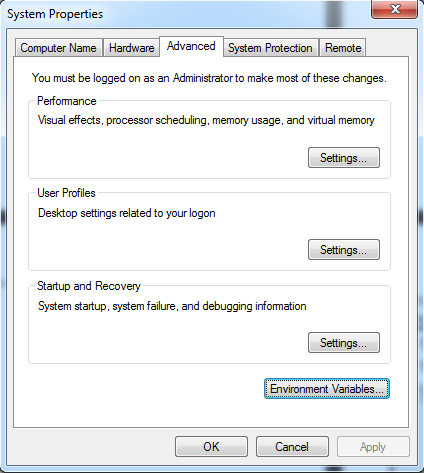
SVN Commit Password: ze7tv5gB8jc7

# Maven Install

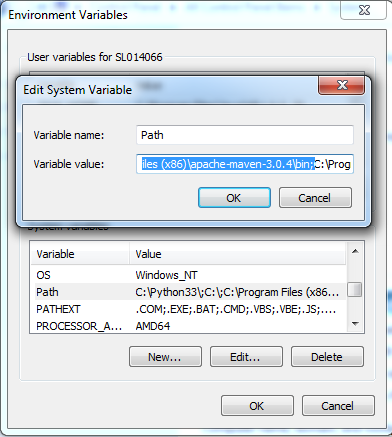
Now that Maven has been installed we will need to add/update some system variables. To do this, right click on My Computer and select Properties. Once that dialog is shown you should see something similar to the following.



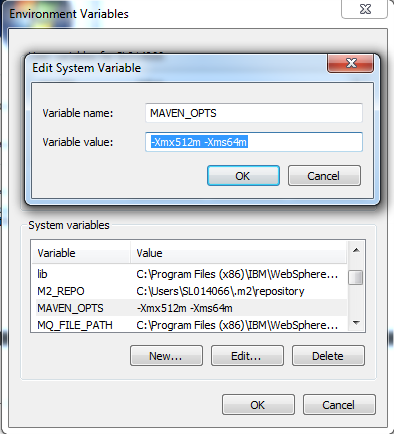
From this screen select Advanced System Settings and in the new dialog select Environment Variables



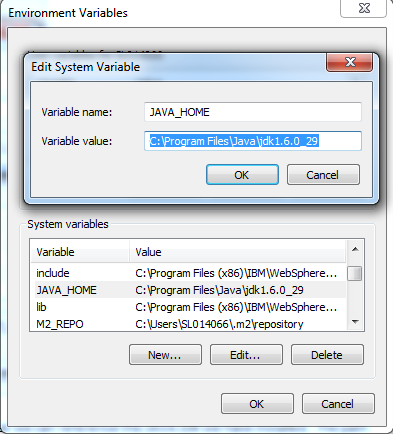
Once Maven has been unzipped to your hard drive we will need to reference its bin directory so that we can utilize the command line functionality. To do this you will need to add the bin directory to your system Path variable.



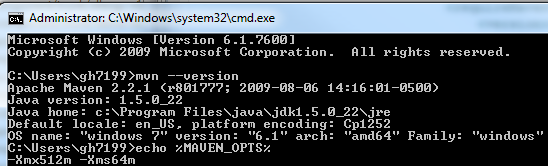
Next we will be adding the MAVEN\_OPTS system variable. This will be utilized to prevent Maven from running out of memory during large build processes.



Finally we will add the JAVA\_HOME system variable so that we can reference the JAVA JDK we have installed. The path will need to point to the root folder of the JAVA JDK install location.



Once you have created all of the necessary Environment Variables validate the setup by opening a command window and typing ‘mvn –version’ and ‘echo %MAVEN\_OPTS%’



Note: you may need to exit cmd prompt and come back in for path variable to be applied.

# Tortoise SVN Install

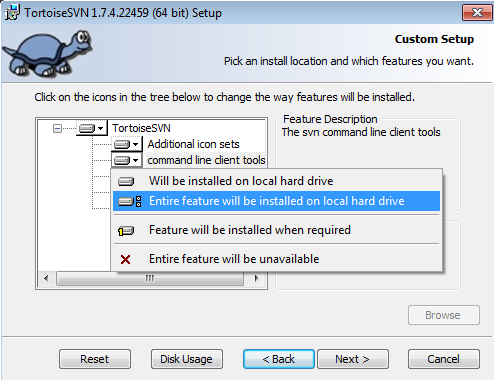
Tortoise SVN:

<http://tortoisesvn.net/downloads>

Our SVN repo is located at: <https://custom-components-svn.googlecode.com/svn/>

NOTE: Two things to be aware of during and after installation.

1. Make sure and check to install the optional command client



# Maven Basics

Maven is command line application and understanding the maven command line is key to understanding what maven is actually doing.

**Maven Commands:**

There are quite a few commands (more properly called “goals”) that are part of the maven core set of functionality, but for now we will focus on only three.

* mvn clean – deletes the target directory to start with a clean build state
* mvn package – (for jars) packages source files into a .jar
* mvn install – moves the packaged endstate (i.e. the jar) to your local maven repository.

**A brief word on how Maven2 works:**

1. You define dependencies in the POM.xml, dependencies are defined by a unique combination of:
   1. Group-id
   2. Artifact-id
   3. Version
2. You define repositories in the POM.xml. Repositories are nothing more than files on a disk that following a naming convention based on the group-id, artifact-id, and version…..repositories that are defined in the POM.xml are files on a disk that available via a web server.
3. When running mvn compile, the first thing maven2 does is search:
   1. Your local repository for the unique group-id,artifact-id, and version
   2. If it finds it in your local repository, then it will use that to compile your code against.
   3. If it does not find it in your local repository then it will search the repositories you have defined in your POM.xml, as well as maven2’s central repository (http://repo1.maven.org/maven2/) for that unique group-id, artifact-id, and version. (This step is only taken if you have a Maven Artifact Repository installed.)
   4. If it cannot find it anywhere, it will error out with a message saying it cannot find that artifact.
   5. If it can find that group-id, artifact-id, version, then it will download the file from the remote repository and place it in your local repository, and then compile against your local repository.
4. By default, your local repository is located at C:\Users\%username%\.m2\repository. This can be thought of as a disk cache of all your dependencies.

**Maven2 lifecyles:**

This is the Maven2 lifecycle: compile->test->package->verify->install->deploy

Each maven2 command implicitly “daisy chains” the previous life cycles. For example, if I run ‘mvn package’, then it also implicitly runs ‘mvn compile’ and ’mvn test’. More appropriate verbage is that if I execute the maven package goal it will implicitly execute the compile and test goals also.

**Multiple commands:**

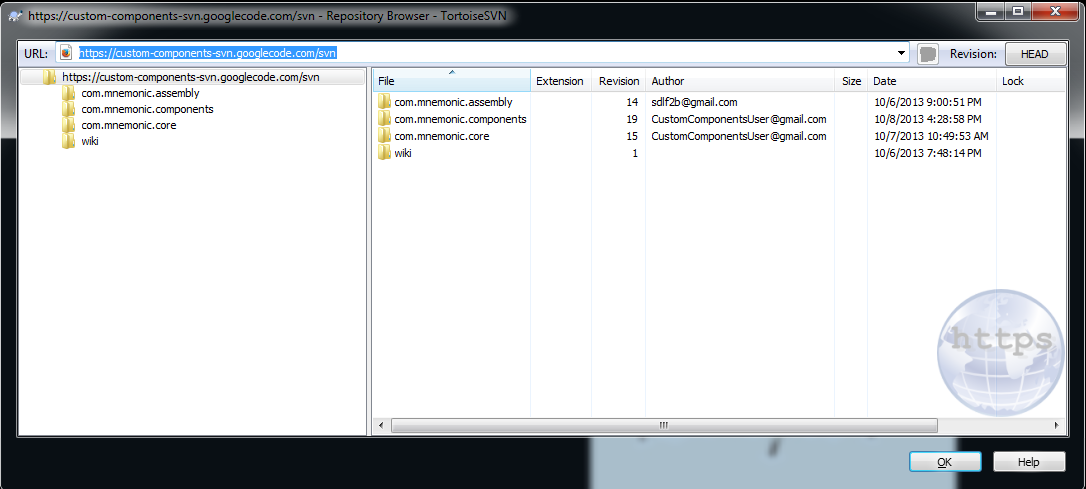
In a single statement you run multiple commands (or goals), for example you can run ‘mvn clean compile’ this will execute the clean and compile goals. ‘mvn clean install’ will execute the clean, test, package, and install goals.

# Custom Component Build Example

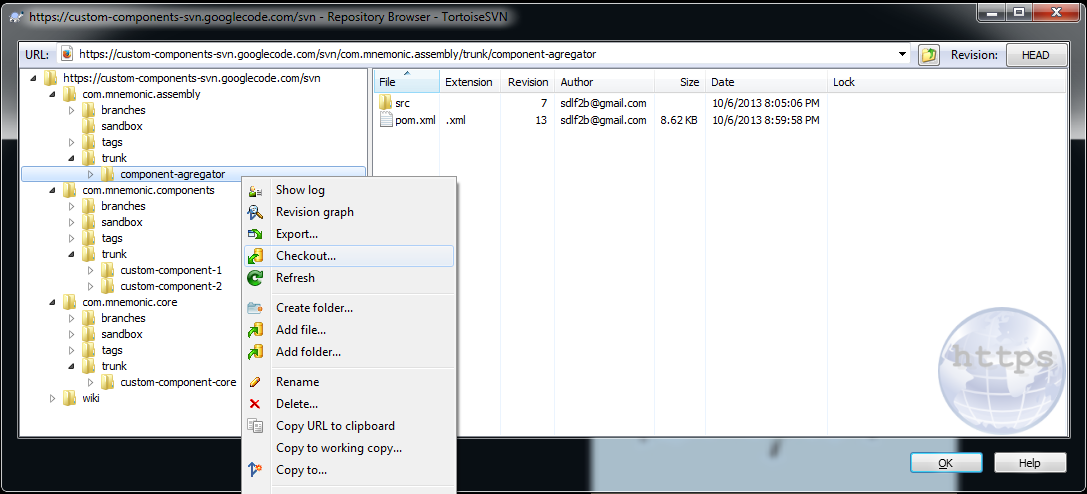
Let’s walk through an example which will aggregate two custom components and a core file into the proper custom-components.js and custom-components.css files.

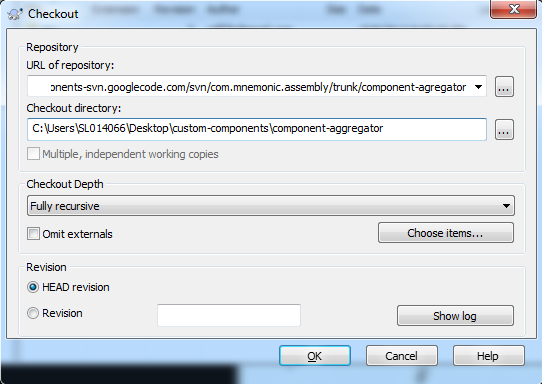
## Checking out artifacts

The first thing we will need to do is check out all of our artifacts. To do this open Tortoise and navigate to the custom components aggregator SVN at <https://custom-components-svn.googlecode.com/svn/>

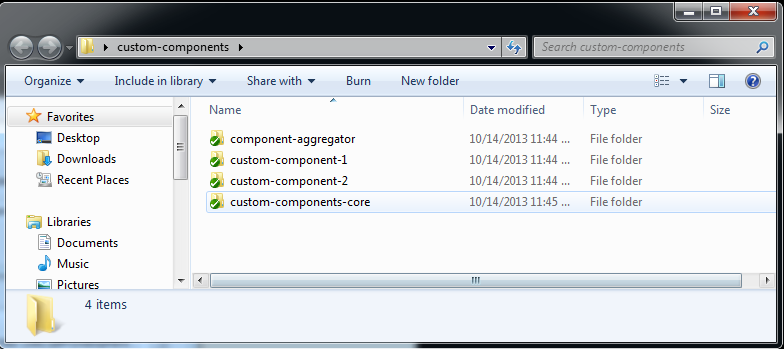


Navigate to the trunk folder for each artifact and check out each one individually. It is not necessary, but makes code management easier if you check out each artifact into the same folder.



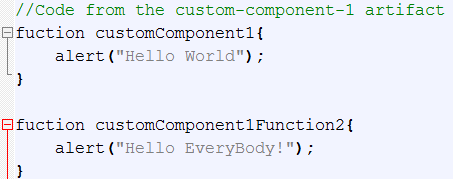


Once you have checked out all dependencies your folder should look like this.



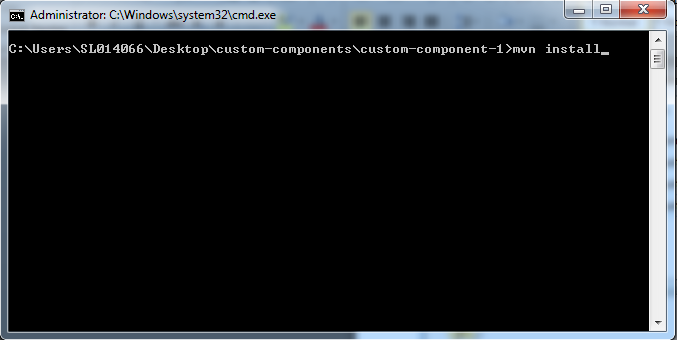
## Updating Artifacts

At this point you would make the updates necessary for your component. For our example we will be adding a new function to custom-component-1 called customComponent1Function2.

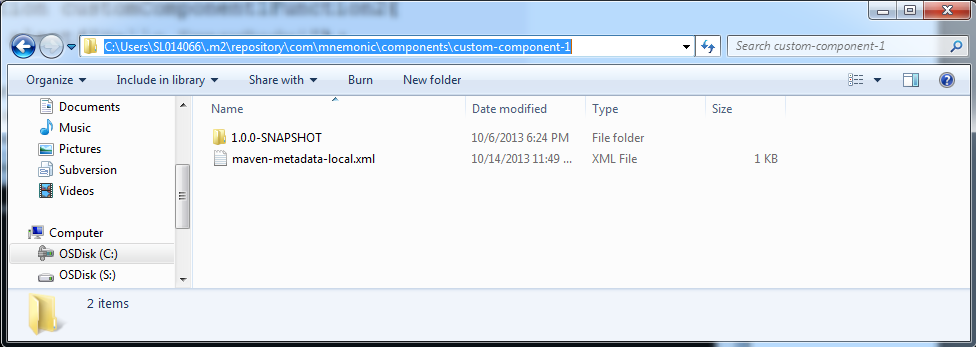


## Installing Artifacts

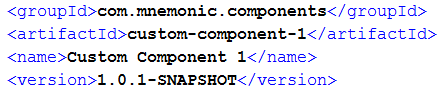
Now that we have made our changes we will need to install the artifact. To do this we will need to open a command prompt at the root location for custom-component-1 (where the pom.xml file is located) and run the ‘mvn install’ command.



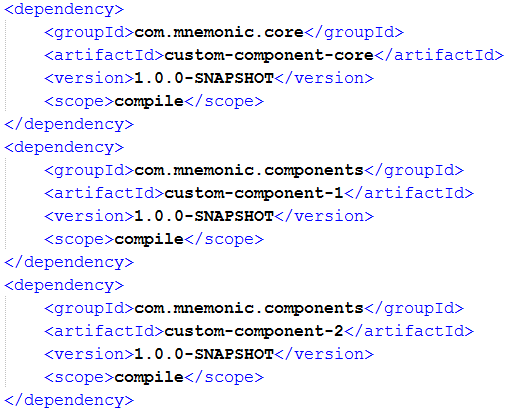
This command will download any maven dependencies that are not currently available in your local repository and also install the custom-component-1 artifact into the local repository.



Notice that the location where your components artifact was installed corresponds to the Maven Group-id, Artifact-id and Version number as defined in the artifact’s pom.xml file. This is essentially how Maven references specific versions of an artifact.



At this point you will also need to install any other dependencies that the component-aggregator is referencing. We can determine these references by examining the dependencies in the pom.xml file of that artifact.



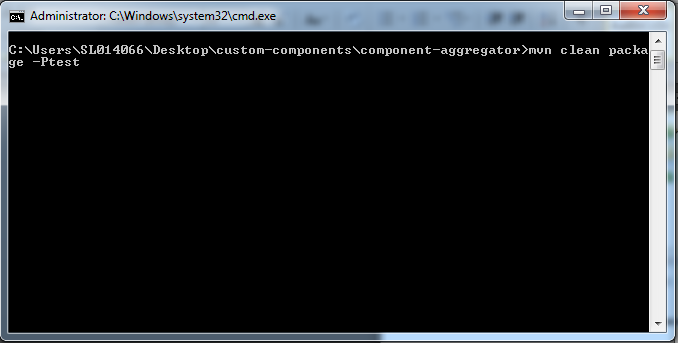
From the pom.xml file we can see that we need to run the ‘mvn install’ command on the custom-component-2 and custom-component-core artifacts so that they will be available within our local maven repository.

**\*Note:** If your site decides to setup a Maven Artifact Repository this step will not be necessary. Maven will automatically retrieve these dependencies and install them. It is strongly encouraged that you setup an artifact repository if you plan on managing multiple dependencies.

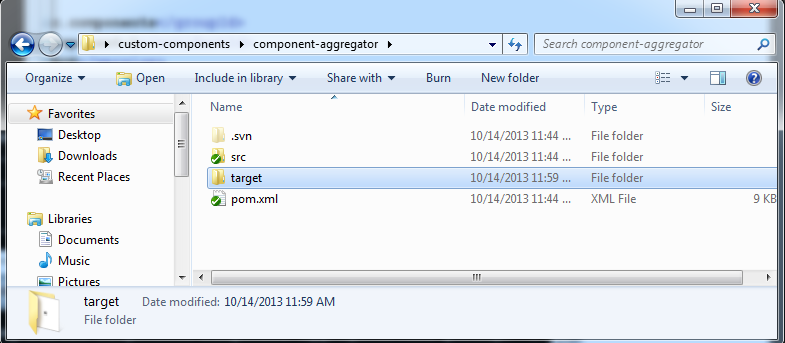
## Packaging Contents

Now that we have installed all of our dependencies we will want to package the contents. To do this we will utilize the ‘mvn clean package –Ptest’ command. The ‘–Ptest’ portion of the command references the testing profile defined in our pom.xml. This profile is used during the development phase which allows us to pull in unreleased versions of our components.

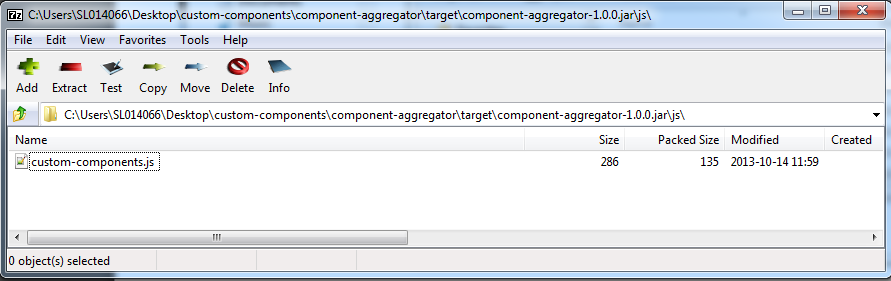


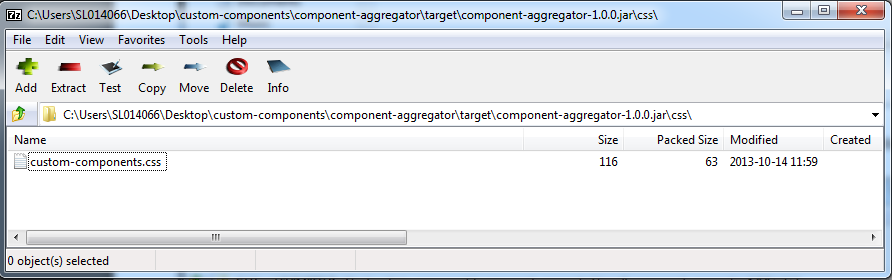


Once this command completes, there will be a new target folder created in the artifact’s folder.

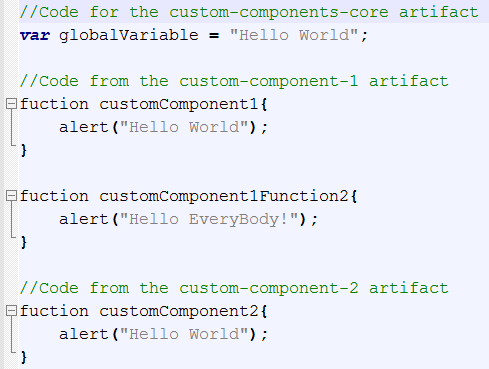


In that folder there will be a component-aggregator-1.0.0.jar file which will contain the contents of the custom-components.js and custom-components.css files.





If we open up our custom-components.js file we can see that our newly added function has now been added to the custom-components.js file.



## Tagging an Artifact

Once you are done making the necessary changes to the component you can create a tag of it. A tag is basically a snapshot of the artifact at a specific point in time. This will allow you to version the components once you are finished testing and the component is stable.

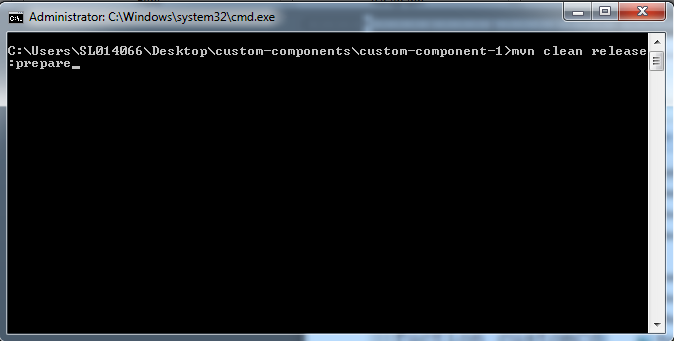
**Note:** Since this is a shared environment created specifically for this walkthrough, please follow the clean-up steps outlined after this section so the next person will be able to walk through the entire guide successfully.

To create a tag you will need to commit all of your changes back to the repository. This will prompt you with a request for username and password. You can utilize the following combination. Be sure to save the credentials, otherwise the next command will fail.

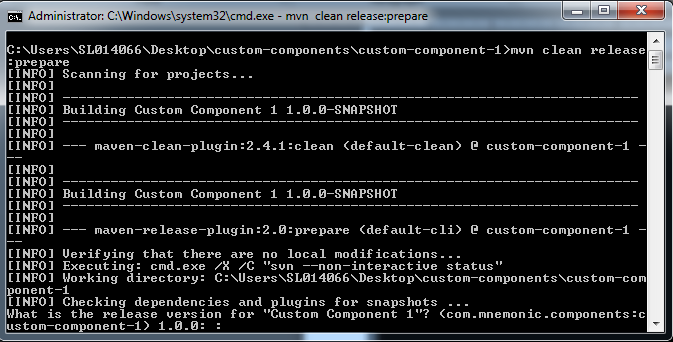
Username: [customcomponentsuser@gmail.com](mailto:customcomponentsuser@gmail.com)

Password: ze7tv5gB8jc7

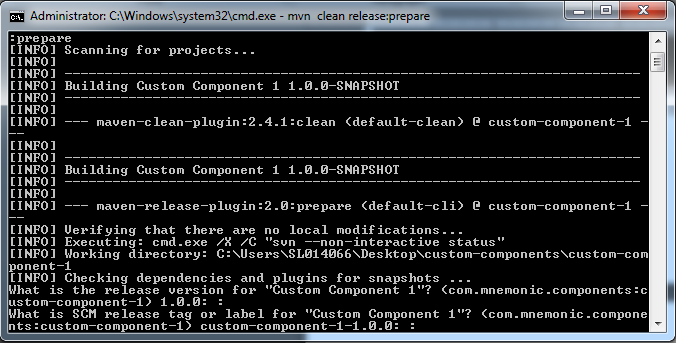
Now that the changes have been checked into the trunk you can use the ‘mvn clean release:prepare’ command to create the Tag. This will result in three prompts as shown below.



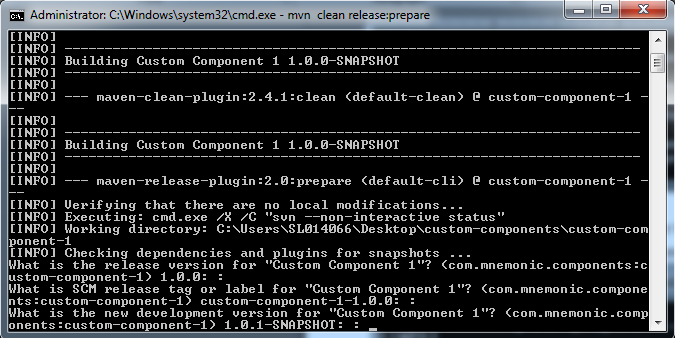
The first prompt asks for you to verify the release version for the artifact. Ensure the version is correct and hit enter if it is.



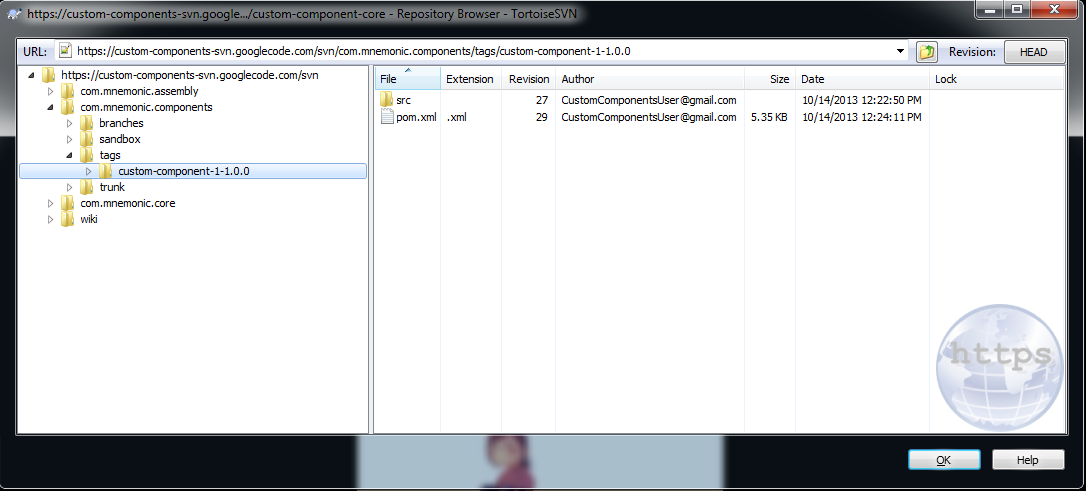
The next prompt wants you to give the tagged artifact a name. This will default to ‘Artifact-id-Version’. If you want to keep that naming convention just hit enter.



The final prompt will ask you what the next version number will be for the artifact. The default behavior will increment the existing version number for the artifact and append the -SNAPSHOT. If the default version number is what you are wanting just hit enter.



Once the command finishes execution a new tag will be created in the tags folder with the name identified in the second prompt.



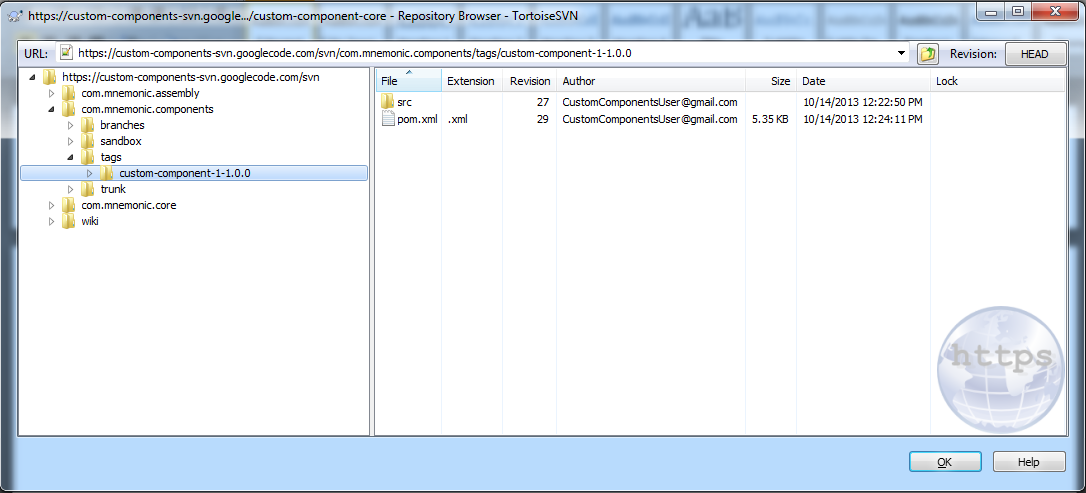
Now that you have tagged the artifact you can update the aggregator pom.xml to reference the new artifact.



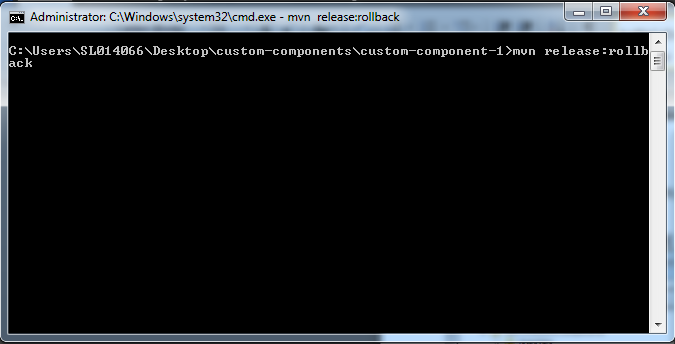
## Cleanup Steps

In order to clean up the example environment for other issue please follow the following steps.

Delete the tag created during the ‘mvn clean release:prepare’ command. This tag will be located in the tags folder.



Run the ‘mvn release:rollback’ command on the custom-component-1 artifact.



## Maven Artifact Repository

As you may have realized at this point, this is a very basic scheme used to manage a handful of artifacts. If you are working with more than just a few artifacts it would be in your best interest to setup a Maven Artifact Repository. This repository will exist solely to store released Maven artifacts. Having an artifact repository will allow developers to check out and update one artifact without having to worry about manually checking out the remaining artifacts. More information about Maven Artifact Repositories can be found [here](http://maven.apache.org/guides/introduction/introduction-to-repositories.html).