|  |
| --- |
| Connecting to git, Using Maven, and Building an Installer for SimWebServer 2.2 |
| [Type the document subtitle] |
|  |
| This document describes how to Check out, Update, and and Create an Installer fro SimWebServer 2.2 |
| This document contains data that is delivered to the Government with Unlimited Rights in accordance with contract HSCG44-16-F-001007 and FAR 52.227-17. |
| Thomas M Kratzke, Strategic Data Systems, 20 July 2017 |

Table of Contents

[I. Introduction 0](#_Toc488309067)

[A. Tracked Code for Java Part 0](#_Toc488309068)

[B. Tracked Code for C++ Part 1](#_Toc488309069)

[C. Tracked Code for Advanced Installer Part 1](#_Toc488309070)

[D. git and BitBucket 1](#_Toc488309071)

[II. Preliminary Setup for eclipse and Maven 1](#_Toc488309072)

[III. Configure SSH 2](#_Toc488309073)

[IV. Configure git 4](#_Toc488309074)

[V. First Download; Cloning the Remote Repository. 6](#_Toc488309075)

[VI. Subsequent Times 8](#_Toc488309076)

[A. Merging Changes 8](#_Toc488309077)

[B. Checking in your own Changes 8](#_Toc488309078)

[VII. Building the Eclipse Projects and Launch Configurations 8](#_Toc488309079)

[A. Loading Preferences 8](#_Toc488309080)

[B. Adjusting the User Variables 8](#_Toc488309081)

[VIII. Building the Java Part 8](#_Toc488309082)

[IX. Building the C++ Part 8](#_Toc488309083)

[A. Adjusting the DirSpecs File 8](#_Toc488309084)

[X. Building the Installer 8](#_Toc488309085)

[XI. Testing the C++ Part 8](#_Toc488309086)

List of Tables

[Table 1: “Source Code” for Java Part 1](#_Toc488309087)

[Table 2: Very Preliminary Steps 1](#_Toc488309088)

List of Figures

**No table of figures entries found.**

Connecting to git, Using Maven, and Building an Installer for SimWebServer 2.2

[Type the document subtitle]

# Introduction

The installer for SimWebServer22 has Java, C++, and Advanced Installer source code. The Integrated Development Environment (*IDE*) eclipse deals with the Java code, creating a .jar file and gathering 3rd-party .jar files. The C++ *IDE* Visual C++ 2015 (or simply Visual C++) deals with the C++ code, ultimately creating dll’s. Advanced Installer takes the two parts and creates the final product, an .msi file.

We are using git as our version control system (VCS), and we refer to a file as being *tracked* when we mean that it is checked into the VCS. Files that are generated during the build process are not tracked. For example, the .class files generated by eclipse, and the .obj files generated by Visual C++ are not tracked.

## Tracked Code for Java Part

The Java code is more than simply .java files. Because of SimWebServer’s heavy reliance on 3rd-party libraries for logging, running its own web server, reading and writing shape files, reading and writing NetCdf files, and other tasks, eclipse must coordinate 3rd-party libraries as well as compile the SimWebServer code. eclipse does this by using a built-in version of Maven, which we will call m2e. m2e reads project object model files, or pom.xml’s. There is a pom.xml for each of SimWebServer’s sub-modules. These sub-modules are util, sarops, and sws22. There are three more sub-modules buildSimLand, sws22X, and landData, but these do not need to be built for SimWebServer to work. Although buildSimLand does not need to be built for SimWebServer, but it is a required delivery, and is used when building the SimLandData .jar file. The other two are used only during development.

eclipse not only coordinates the downloading of the 3rd-party libraries, but it collects them and creates a .jar file from the SimWebServer Java code. It also places these in a directory from which Advanced Installer can load them into the .msi. To do this, eclipse requires a *launch configuration*, which is a file that is tracked by git. Finally, the project files for the sub-modules are also tracked. Summarizing, the main files corresponding to the Java part, that must be tracked by a version control system are:

1. Java code
2. pom.xml’s
3. launch configuration(s)
4. Project files for Individual projects

Table 1: “Source Code” for Java Part

eclipse requires more files than this, and we track more, but a complete discussion is beyond the scope of this document.

## Tracked Code for C++ Part

The C++ of SimWebServer22 part uses *JNI*, or Java Native Interface code. This means that Eclipse must write some of the include files, and generate files that are useful for reference within the C++ code itself. This latter includes some “magic-looking” strings that must be used within the C++ code. These strings are generated by programs that are managed by Eclipse.

Visual C++ serves the role of Eclipse, and the project and solution files are required to work with this part. Although the C++ part is much simpler than the Java part, we still must track the code and the Visual C++ project and solution files.

## Tracked Code for Advanced Installer Part

After Eclipse and Visual C++ have compiled the code, a 3rd program, Advanced Installer, assembles the parts into an .msi file. It too requires a script, called an .aip file. We track this as well.

## git and BitBucket

These files are kept in a single directory, and this directory constitutes a git repository. The program git maintains the version control of this repository, and it uploads the “tracked” files to the BitBucket repository within our di2e account. Because of the large number of files that we do not wish to track, we make heavy use of .gitignore files to reduce the number and size of the files we track, and these files are also tracked.

# Preliminary Setup for eclipse and Maven

1. Delete your ~home/.m2 directory (for Maven)
2. Delete any workspace you've tried for this project
   1. A workspace is just a directory, so just delete these directories.
3. Create a directory called sws22Workspace
   1. If you usually work in your home directory, create it there
4. Open eclipse, and when it asks for a workspace, navigate to sws22Workspace and select that directory as your workspace.

Table 2: eclipse and Maven Setup

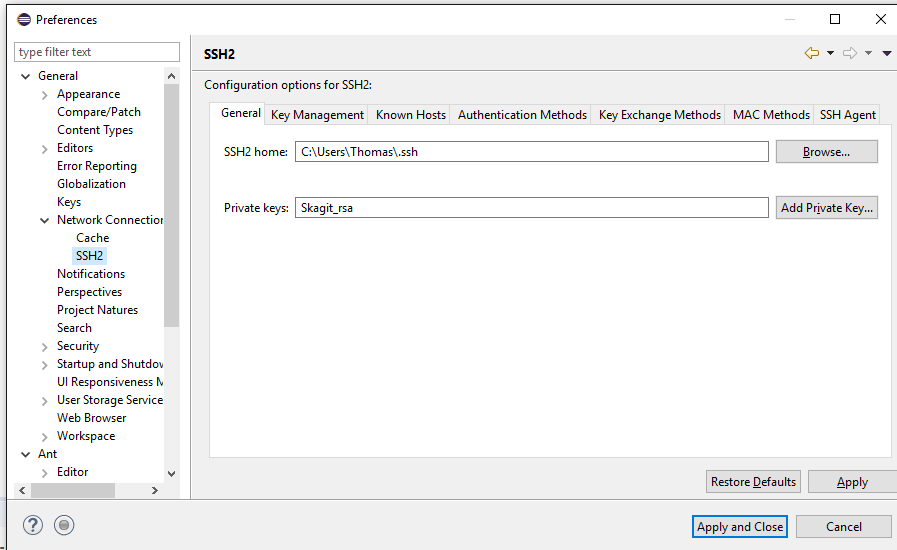
Close the welcome screen if you have one, and select the tab . A dialog window will appear and a list of choices on the left. Select Maven from that list. Check the box Download repository index updates on startup. Restart eclipse and open the workspace sws22Workspace again (or from the file menu, you can simply select Restart). This takes a long time, but you only need to do this periodically (e.g. weekly). While you’re waiting, you can open the same dialog and *un*check that box so that eclipse will *not* update the indexes the next time.

Let the indexes finish updating before going on to the next step.

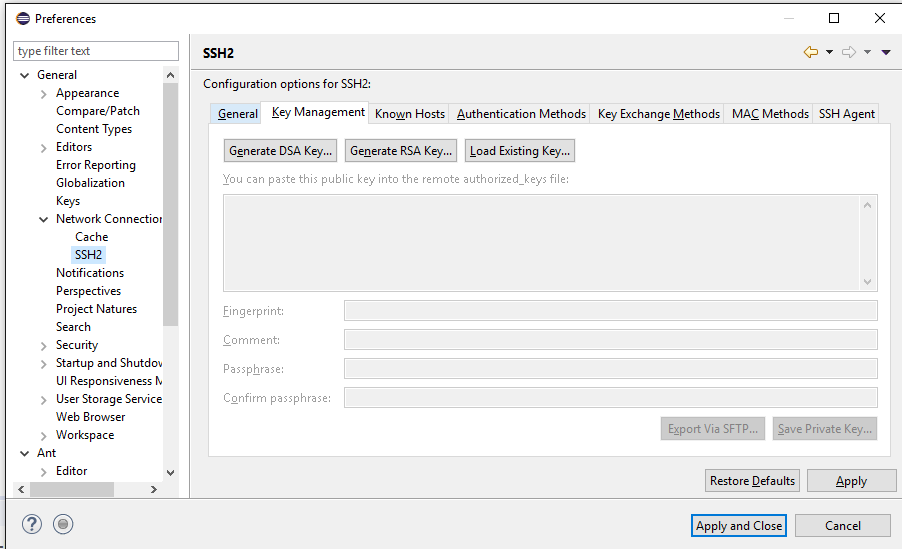
# Configure SSH

Let the work in §II finish before going on to thSelect to open the Preferences dialog. From the list of Preferences on the left, select

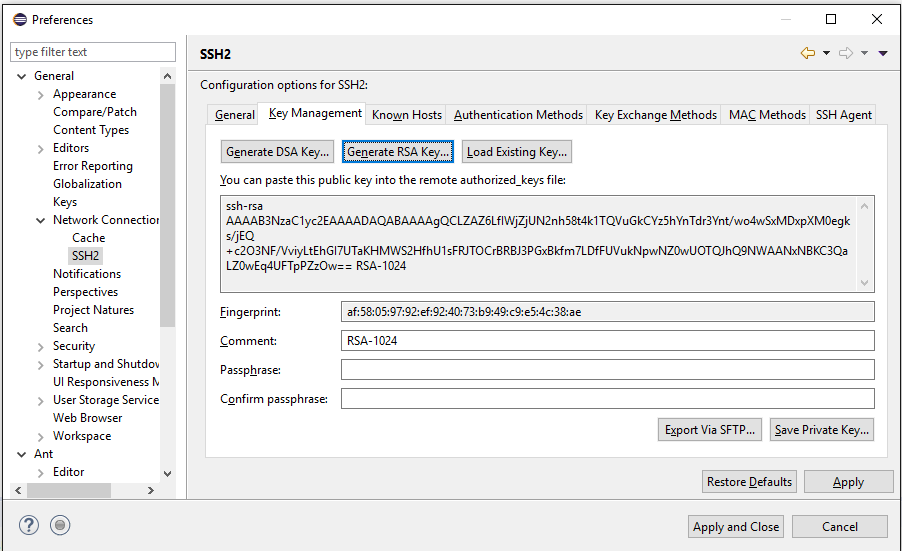
You should get something that looks like:



Click the Key Management tab.



Click Generate RSA Key…. You’ll get a key generated for you. I recommend entering a passphrase such as GreatFalls and a short comment in such as Ng-SWS22. This makes later steps more readable.



Copy the public key onto your clipboard.

Leave eclipse open, and use a browser to sign in to your di2e account. Go to your own personal settings and you should see SSH keys. Add the key that’s on the clipboard.

Go back to eclipse. Click Save Private Key and save it to something like Ng-SWS22\_rsa within ~/.ssh; the save dialog should start there. Hit Apply, (not Apply and Close)

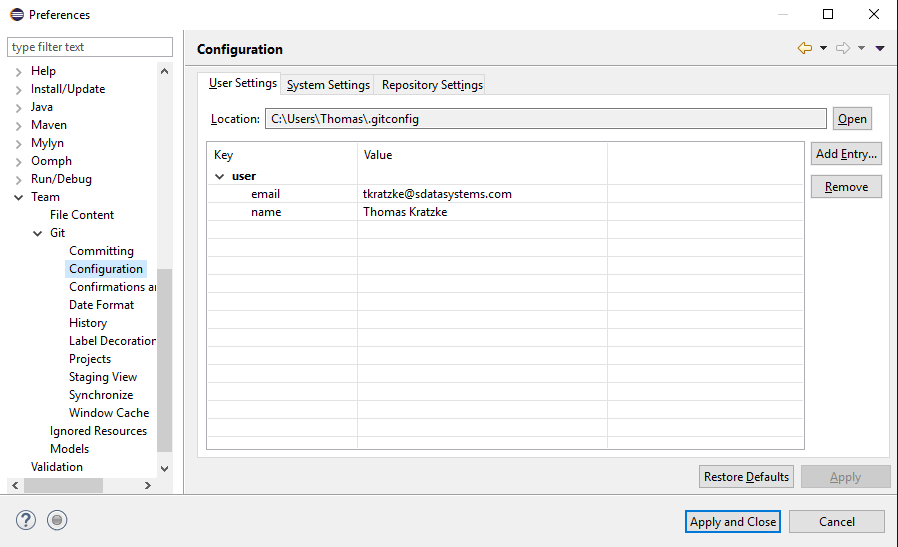
Click the tab.

Click the button and select the Ng-SWS22\_rsa key you created. It will then show up in the list of Private Keys that eclipse recognizes. That list is an editable string and we recommend that you erase everything from that string except the key that you created, so you would be left with just Ng-SWS22\_rsa in that text box.

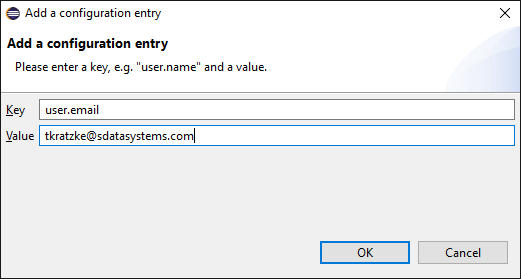
# Configure git

Now create a directory to store your git repositories in. You will only have one repository for this project (so far), but it’s still nice to have such a directory. Call this directory GitRepos. You will now tell eclipse/git about this directory and a little more about yourself.

Again, open the dialog, but this time select from the left. You should get a dialog like:



You need to register your name and email address as entries in this dialog. When you Add Entry…, the names of the keys that you want to enter are user.name and user.email. For example:



When you finish entering these values, go back to the dialog and enter the Location of your GitRepos directory.

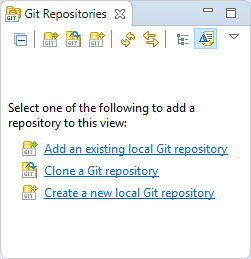
# First Download; Cloning the Remote Repository.

We assume that you are in eclipse and the sws22Workspace, the maven indexes have been updated, and the maven checkbox about updating indexes on startup is unchecked. Dismiss the welcome screen if it shows up.

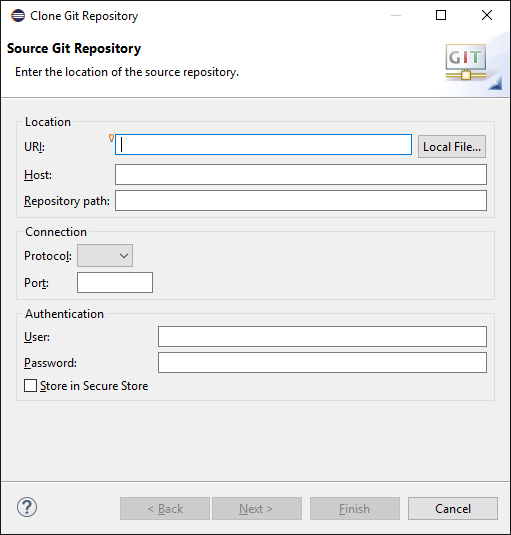
Go to the Sws22-repo repository within di2e, currently the only repository under the Search and Rescue Optimal Planning System project.

On the lower left, expand the icon list so you know what you are clicking. Click the “clone” button on the left, and copy the ssh url from the dialog box. It should already be highlighted for you. That window might have shown up with the http selected. If it did, switch back to ssh.

Your eclipse workspace should be blank. We are now going to use , but *not .* Click , and select . Part of your workspace should now have the Git Repositories view:



Click on Clone a Git repository and you should get:



Paste the contents of your clipboard into the first line and the rest should fill in automatically.

Click next, and make sure that the clone of Sws22-Remote is saved in a subdirectory of your GitRepos directory.

# Building the Workspace from the Clone: First Time

The repository has already been downloaded, and we need to load preferences to get some user variables set. Unfortunately, this will overwrite some preferences. So we load the preferences only once.

We do this by getting the Import dialog up: . Then , and navigate to the cloned repository’s Preferences file, which is in the repository directory under parentProject. Load that, but then you’ll have to reset your git preferences.

You will need to reset your default git Repositories folder and your user name and email, and reload (but not re-create) your SSH key.

After doing this the first time, do not reload the preferences file. The preferences file will be marked as dirty, but there’s no need to commit it. On the other hand, it does no harm either.

# Subsequent Times

## Merging Changes

## Checking in your own Changes

# Building the Eclipse Projects and Launch Configurations

## Loading Preferences

## Adjusting the User Variables

# Building the Java Part

# Building the C++ Part

## Adjusting the DirSpecs File

# Building the Installer

# Testing the C++ Part