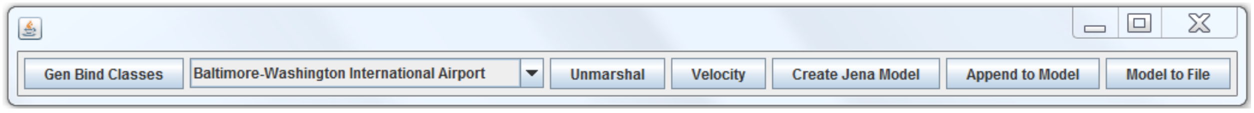
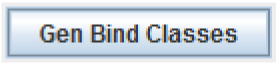
Converting XML to RDF Using JAXB, Velocity, and Jena

Authors: Seth Moore and Jordan Hoffman

This report covers a Java application (WeatherConverter) that demonstrates generating Java bindings from an XML Schema, unmarshalling XML files into instances of the generated classes, using the Velocity Template Engine to translate the instances into RDF/XML, and using Jena to create an RDF model and save it to a file in Turtle format. The data being converted is current observation data from http://www.weather.gov. The example outlined on pages 328-337 of *Semantic Web Programming* by John Hebeler, Matthew Fisher, Ryan Blace, and Andrew Perez-Lopez was used as a starting point for developing WeatherConverter.

When the application is started, the user is presented with a simple graphical user interface (GUI) with six buttons and a drop down list of locations to choose from.

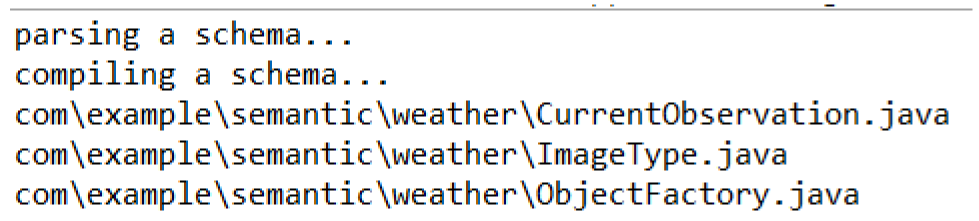


When run for the first time after being cloned from https://github.com/SethJMoore/semantic-web-weather-example.git (see Appendix A for instructions on cloning and running), the user must click the 

button in order to generate the Java classes that the unmarshalling step will read the XML data into. This binding step uses XJCFacade to parse **current_observation.xsd**, which is the schema used for the observation XML files, and create **CurrentObservation.java**, **ImageType.java**, and **ObjectFactory.java** in the **gen** source folder. The code consists of preparing a String array to send to XJCFacad's **main** method, then calling it within a **try** block.

```
String targetPackageName = "com.example.semantic.weather";
String XMLSchema = "Resources/current_observation.xsd";
String targetDirectoryForGeneratedClasses = "gen";
final String[] arguments = {"-p",
           targetPackageName,
           XMLSchema,
           "-d",
           targetDirectoryForGeneratedClasses};
try {
     XJCFacade.main(arguments);
} catch (Throwable e) {
     // TODO Auto-generated catch block
     e.printStackTrace();
}
```

There will be console output generated by this process similar to the following:



```
parsing a schema...
compiling a schema...
com\example\semantic\weather\CurrentObservation.java
com\example\semantic\weather\ImageType.java
com\example\semantic\weather\ObjectFactory.java
```

The drop down list,

which is the next item on the GUI, has six location choices, but could be expanded upon by adding observation locations from http://w1.weather.gov/xml/current_obs/index.xml. This is as simple as adding new station name and XML URL pairs to the **locations** array.

```
LocationWithURL[] locations ={
new LocationWithURL("Baltimore-Washington International Airport",
"http://w1.weather.gov/xml/current_obs/KBWI.xml"),

new LocationWithURL("Boca Raton Airport",
"http://weather.gov/xml/current_obs/KBCT.xml"),

new LocationWithURL("Bozeman/Gallatin",
"http://w1.weather.gov/xml/current_obs/KBZN.xml"),

new LocationWithURL("Molokai Airport",
"http://weather.gov/xml/current_obs/PHMK.xml"),

new LocationWithURL("Green Bay, Austin Straubel International
Airport", "http://weather.gov/xml/current_obs/KGRB.xml"),

new LocationWithURL("Anchorage International Airport",
"http://weather.gov/xml/current_obs/PANC.xml")
      };
```

Another option would be to further develop WeatherConverter to be able to download the locations index, and extract all the stations.

Next on the GUI is the

button. We don't generate any console output during this process, so you'll just have to assume that if no errors show up, it worked. An InputStream is opened to the XML URL of the chosen weather station, and JAXB is used to unmarshal the XML data into an instance of the CurrentObservation class.

```
// get the xml file input
URL xmlFileUrl = new URL(xmlFile);
InputStream xmlFileInputStream = xmlFileUrl.openStream();
// unmarshal the information from xml
JAXBContext jaxbContext = JAXBContext
     .newInstance("com.example.semantic.weather");
Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
     CurrentObservation currentObservation = (CurrentObservation)
unmarshaller.unmarshal(xmlFileInputStream);
```

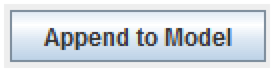
Clicking the

button will use the **Velocity Template Engine** to write RDF/XMS from the CurrentObservation instance, based on the Velocity Template file **weather-rdf.vm** in the **Resources** directory. This template uses eleven of the sixty-one potential fields available, with each one surrounded by an **if end** pair to make sure they actually have meaningful content. For example:

```
<w:WeatherObservation>
     #if ( $observation.CreditURL )
          <w:source rdf:resource="$observation.CreditURL"/>
     #end
     #if ( $observation.ObservationTimeRfc822 )
          <w:time>$observation.ObservationTimeRfc822</w:time>
     #end
     #if ( $observation.Location )
          <w:location>$observation.Location</w:location>
     #end
     #if ( $observation.Latitude )
          <w:latitude>$observation.Latitude</w:latitude>
     #end
     ...
```

The resulting RDF/XML is save to a ByteArrayOutputStream, as well as being printed to the console.

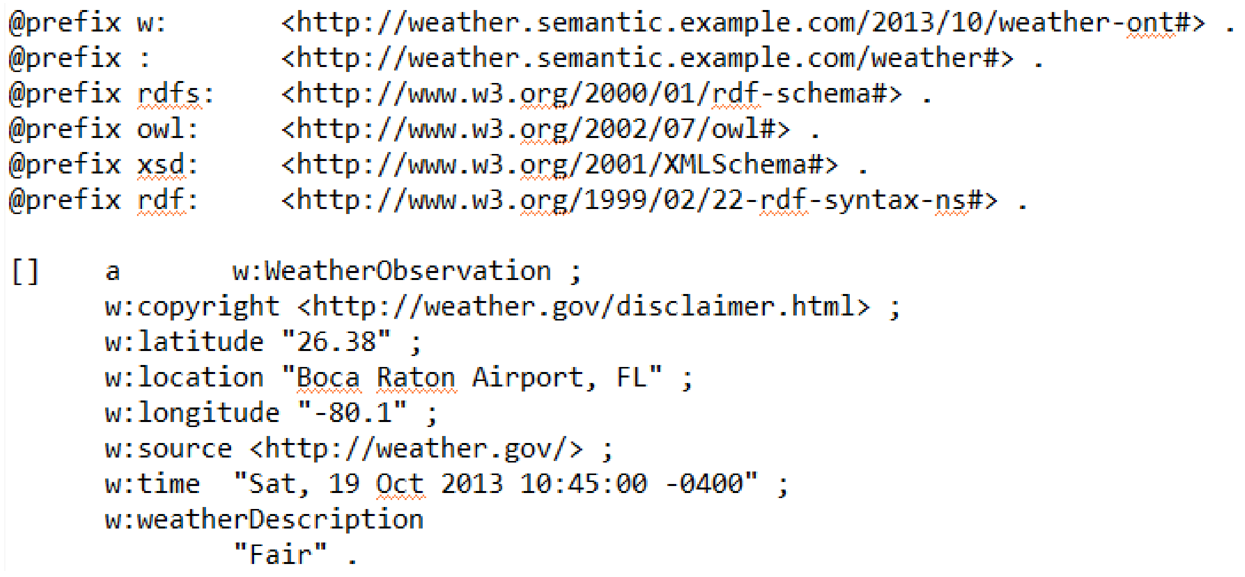
The

and 

buttons both read from the ByteArrayOutputStream that was generated by **Velocity** into an RDF model, but the **Create** version removes all the RDF statements from the model before reading the data, while the **Append** version skips the removal, so the new statements are simply added to the model. They both also output the model as RDF/XML to the console, for the user's entertainment. The effective code here is really just the two lines:

```
ByteArrayInputStream modelInputStream = new ByteArrayInputStream(
                                              vmOutputStream.toByteArray());
rdfModel.read(modelInputStream, null, "RDF/XML");
```

Finally Model to File writes the model to **output.ttl** in Turtle format. If it's the first time writing this file, you may have to **Refresh** the project in order for the file to show up in Eclipse's Package Explorer. Here is an example of the resulting output.ttl file that I just generated (2 Nov 2013) from the current observation from the Boca Raton Airport:



```
@prefix w:       <http://weather.semantic.example.com/2013/10/weather-ont#> .
@prefix :        <http://weather.semantic.example.com/weather#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

[]    a       w:WeatherObservation ;
      w:copyright <http://weather.gov/disclaimer.html> ;
      w:latitude "26.38" ;
      w:location "Boca Raton Airport, FL" ;
      w:longitude "-80.1" ;
      w:source <http://weather.gov/> ;
      w:time  "Sat, 19 Oct 2013 10:45:00 -0400" ;
      w:weatherDescription
              "Fair" .
```

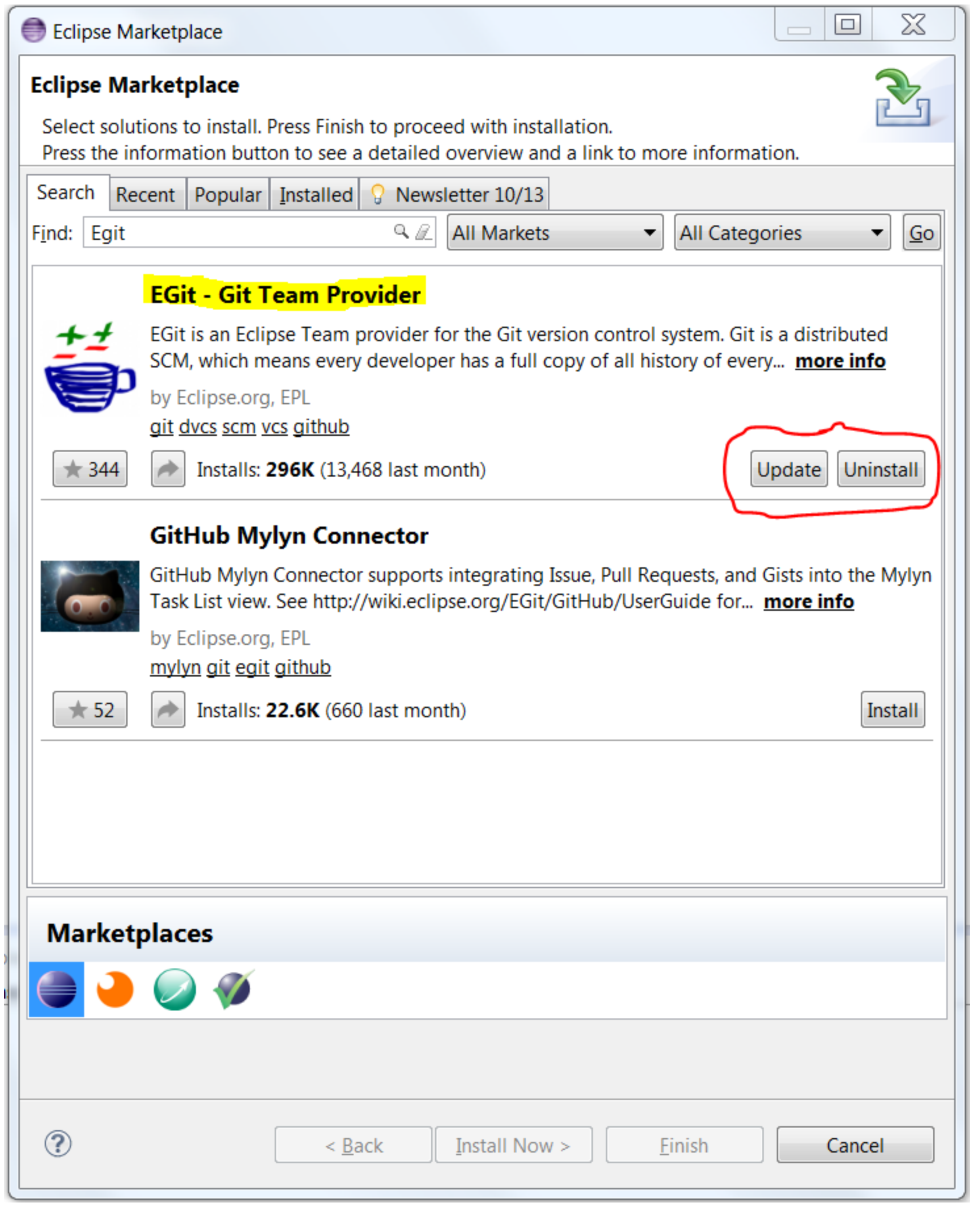
It doesn't look like Boca Raton Airport updates its observations very frequently.

With the instructions in Appendix A, this application should be easy to explore and experiment with. We used **Ivy** to deal with dependencies, so a bunch of .jar files don't need to be included with the source or sought out and downloaded by new users. Although it does a good job of demonstrating how to use XML bindings to convert XML into RDF (as well as some Java's Swing library), there are a number of areas that could be addressed to make WeatherConverter potentially more useful:

- Exceptions could be handled more thoroughly. Currently they're just caught and printed out.
- The process of generating the bindings should probably be done separately.
- Allow the user to choose stations he is interested in from a list of all the available stations.
- Update the Velocity template file to cover all the fields a CurrentObservation might have.
- Perhaps map those fields to an established weather ontology.
- Develop a useful way for the GUI to display the data from the models that are created.

Appendix A: Clone and Run WeatherConverter

Start up Eclipse (these instructions are based on Eclipse Kepler, which can be obtained from http://www.eclipse.org/kepler/). If you don't already have **Egit** installed, click on Eclipse's **Help** menu button, and select **Eclipse Marketplace…**, where you will do a search for **Egit**. This search should return **Egit - Git Team Provider** with an **Install** button available, unless it's already installed, in which case **Update** and **Uninstall** will be available. Install it.


Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Find: Egit

EGit - Git Team Provider

EGit is an Eclipse Team provider for the Git version control system. Git is a distributed SCM, which means every developer has a full copy of all history of every... **more info**

by Eclipse.org, EPL

git dvcs scm vcs github

344 Installs: **296K** (13,468 last month)

GitHub Mylyn Connector

GitHub Mylyn Connector supports integrating Issue, Pull Requests, and Gists into the Mylyn Task List view. See http://wiki.eclipse.org/EGit/GitHub/UserGuide for... **more info**

by Eclipse.org, EPL

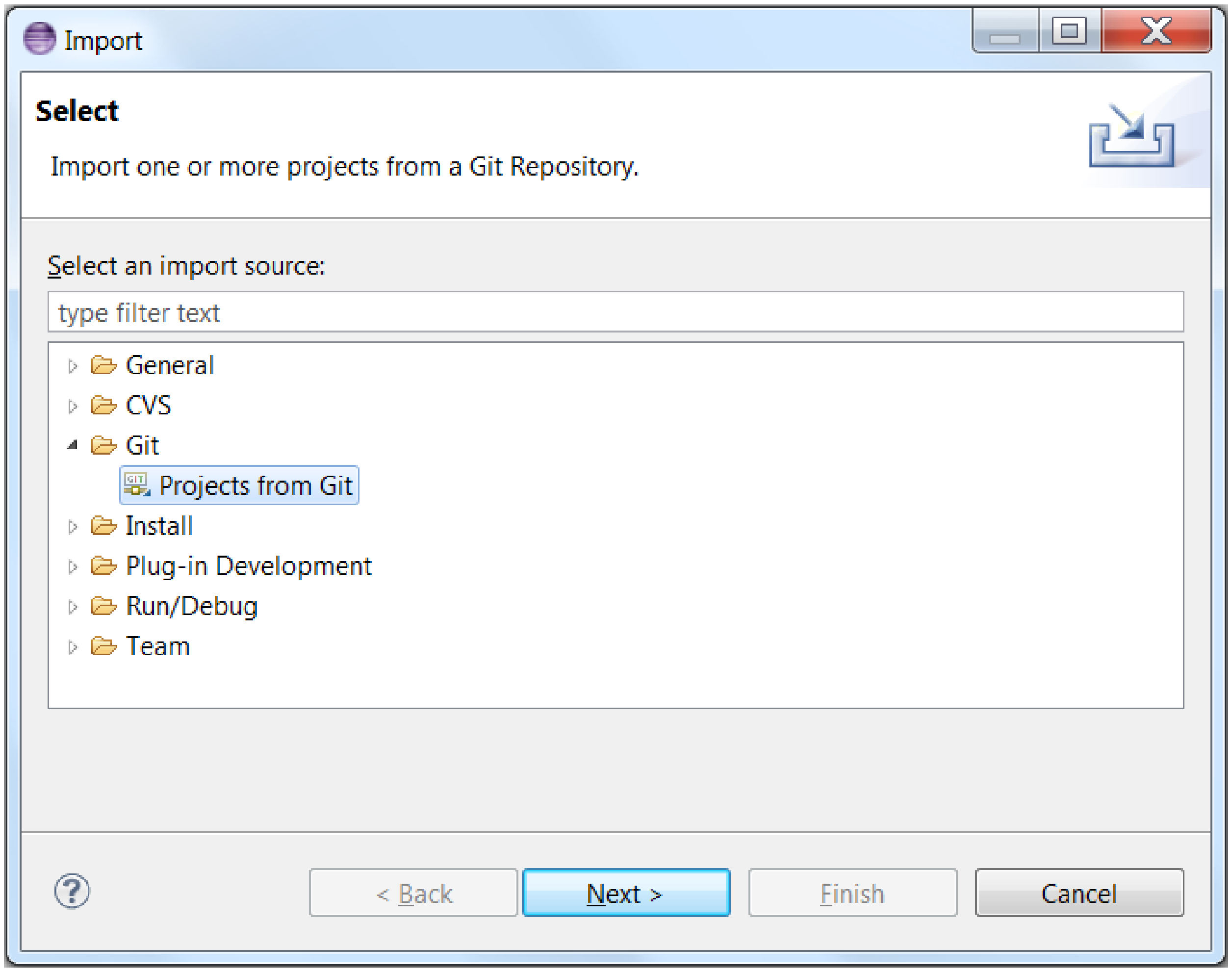
mylyn git egit github

52 Installs: **22.6K** (660 last month)

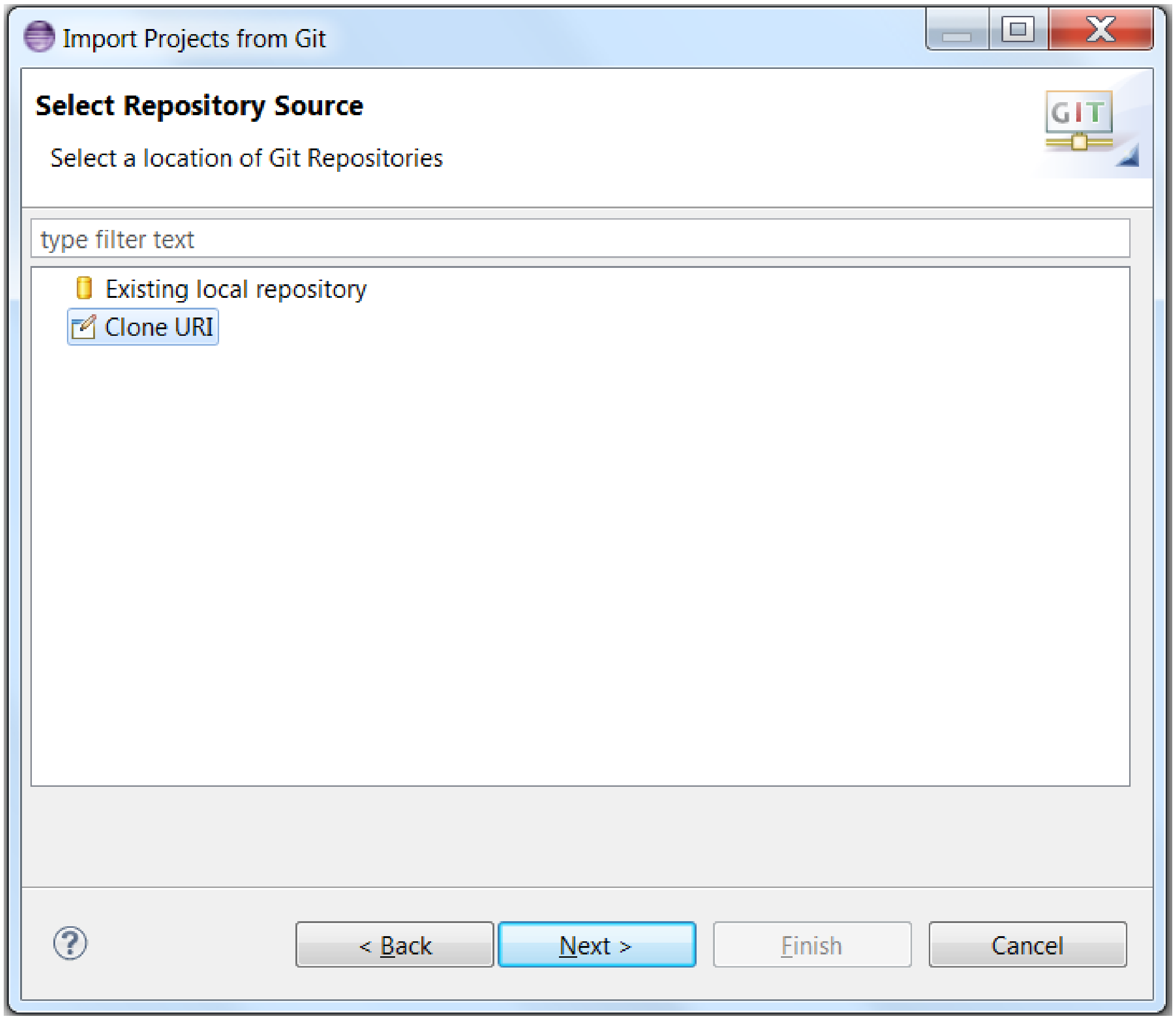
As with **Egit**, if you don't already have **Apache IvyDE** installed, go to the **Eclipse Marketplace**, search for **Ivy**, and install it.

After installing **Egit** and **Ivy**, you may be prompted to restart Eclipse. If so, I recommend that you do so.

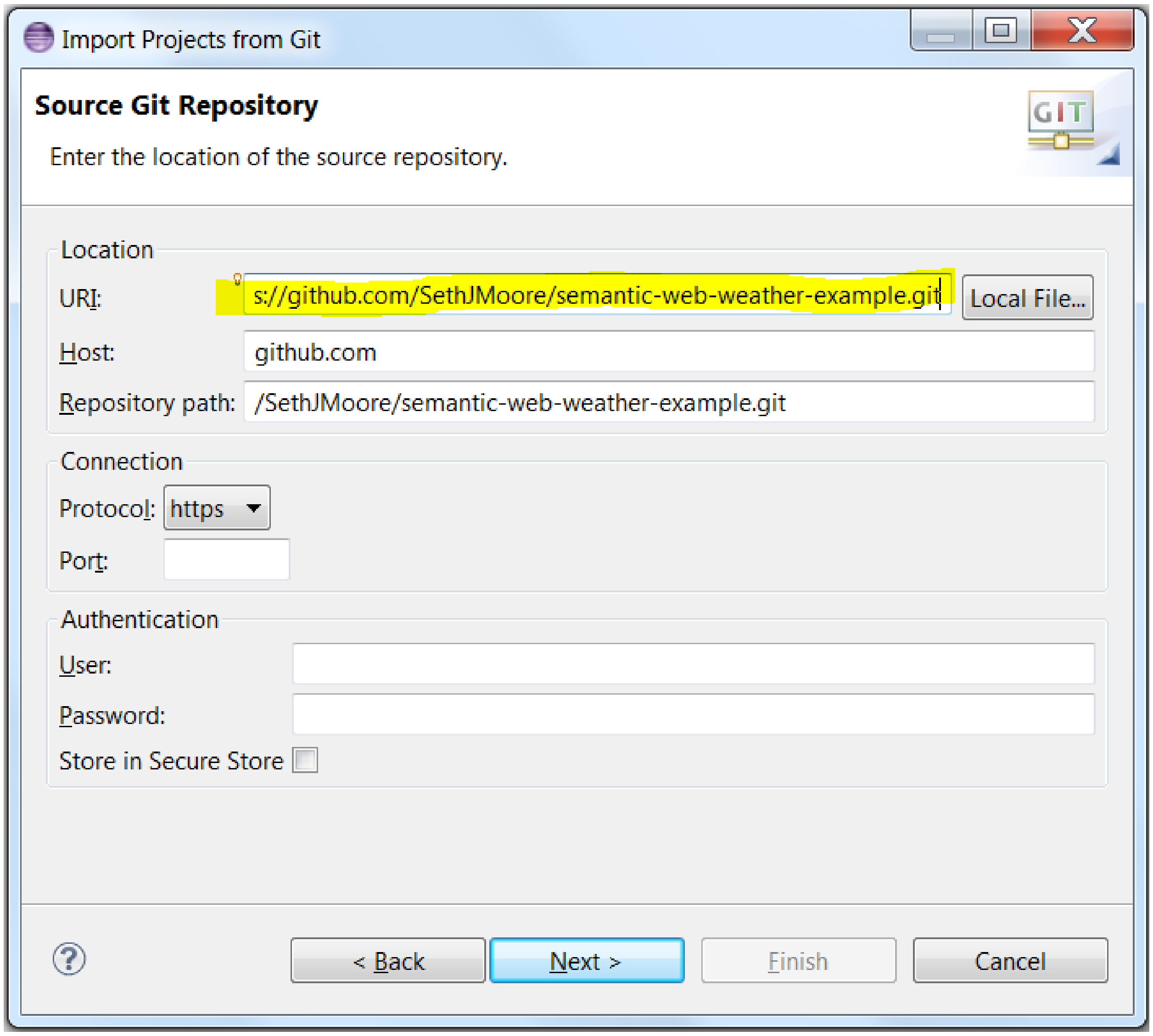
Now you need to import the project, so click **File -> Import…** and choose **Projects from Git**, then click **Next.**



Select **Clone URI**, then click **Next**.



Enter https://github.com/SethJMoore/semantic-web-weather-example.git in the **URI:** field. This should automatically fill in **Host:**, **Repository path:**, and **Protocol:**, so then you just need to click **Next**.



Source Git Repository

Enter the location of the source repository.

URI: s://github.com/SethJMoore/semantic-web-weather-example.git

Host: github.com

Repository path: /SethJMoore/semantic-web-weather-example.git

Protocol: https

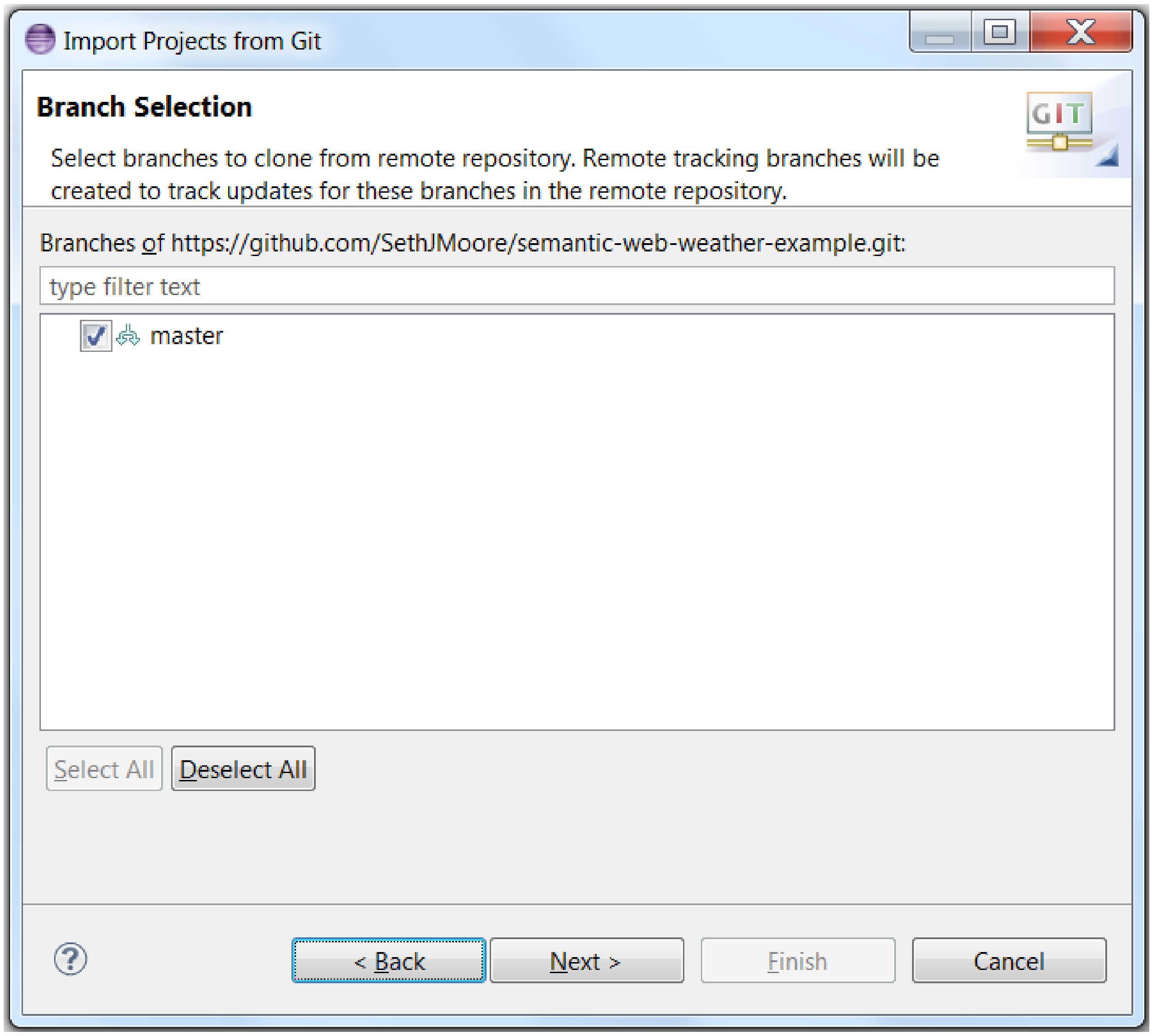
Port:

User:

Password:

Store in Secure Store [ ]

Click **Next** on the **Branch Selection** window.



Import Projects from Git

Branch Selection

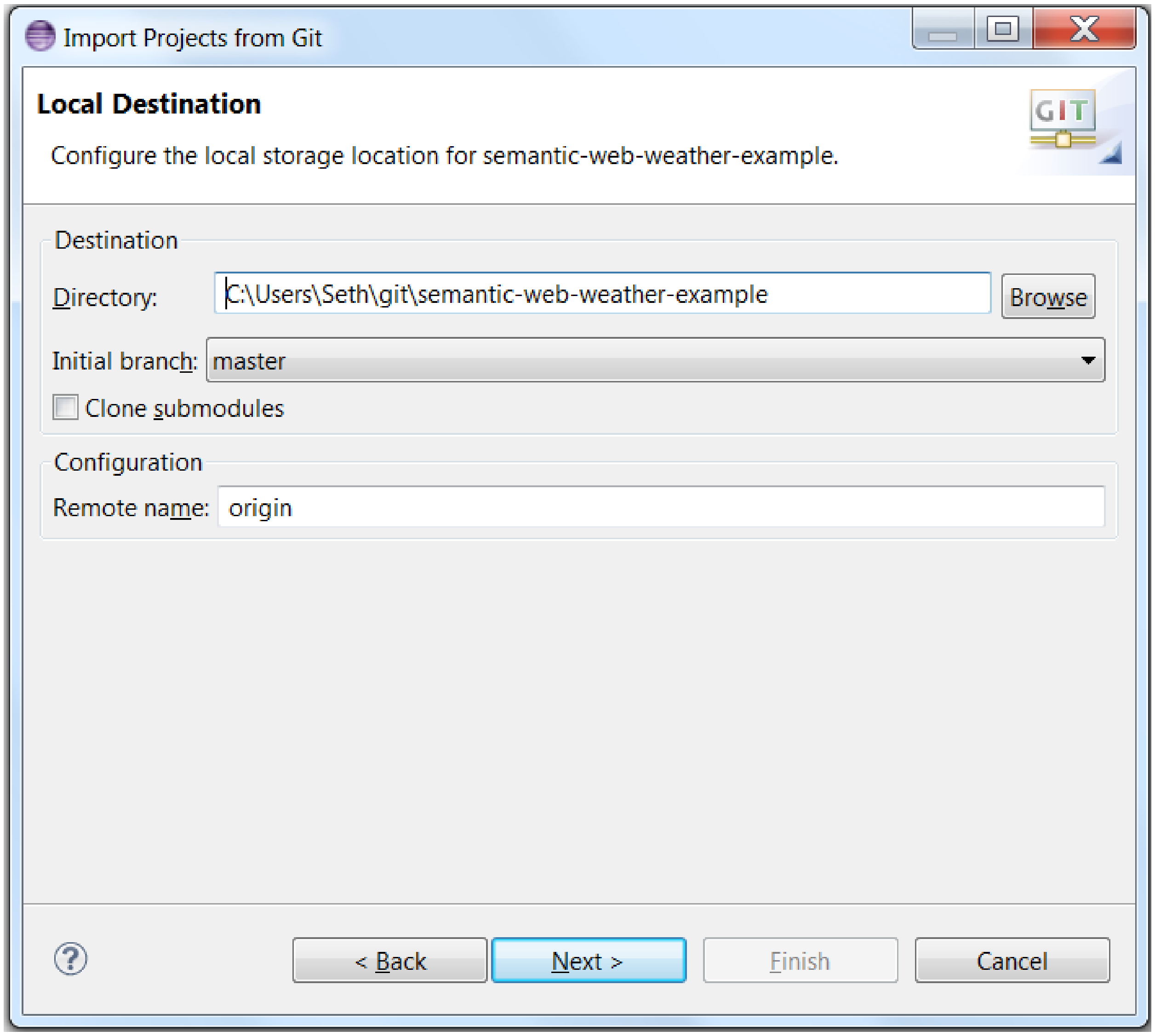
Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository.

Branches of https://github.com/SethJMoore/semantic-web-weather-example.git:

type filter text

[✓] master

On the **Local Destination** window, click **Next** if you are okay with the default destination directory, otherwise specify your preferred directory, then click **Next**.


Import Projects from Git

Local Destination

Configure the local storage location for semantic-web-weather-example.

Destination

Directory: C:\Users\Seth\git\semantic-web-weather-example

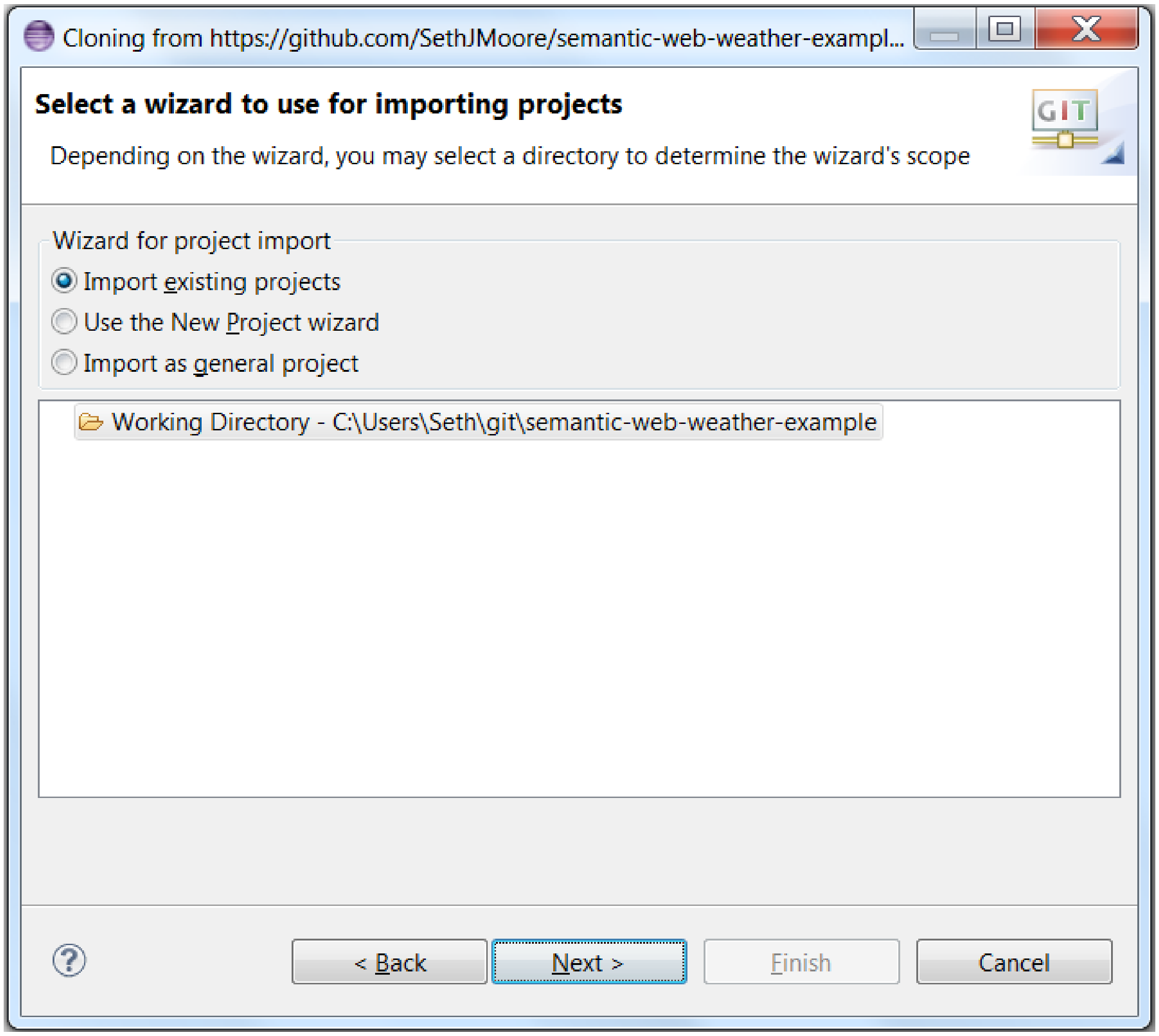
Initial branch: master

Clone submodules

Configuration

Remote name: origin

Leave **Import existing projects** selected, and click **Next**.



Cloning from https://github.com/SethJMoore/semantic-web-weather-exampl...

Select a wizard to use for importing projects

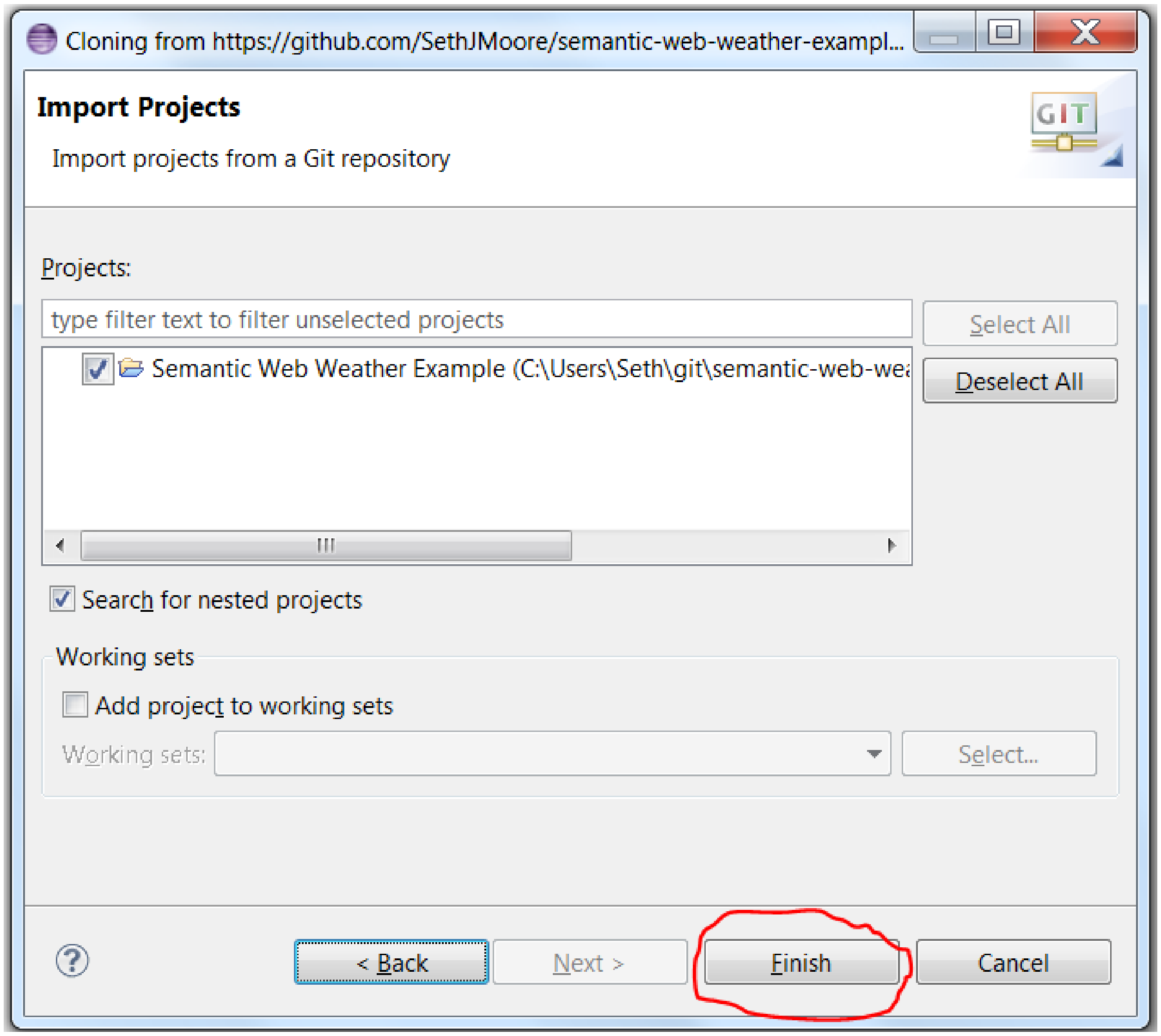
Depending on the wizard, you may select a directory to determine the wizard's scope

Wizard for project import

- [✓ Import existing projects]
- Use the New Project wizard
- Import as general project

Working Directory - C:\Users\Seth\git\semantic-web-weather-example

Make sure **Semantic Web Weather Example** is selected, and click **Finish**.


Cloning from https://github.com/SethJMoore/semantic-web-weather-exampl...

Import Projects

Import projects from a Git repository

Projects:

type filter text to filter unselected projects

☑ Semantic Web Weather Example (C:\Users\Seth\git\semantic-web-wea

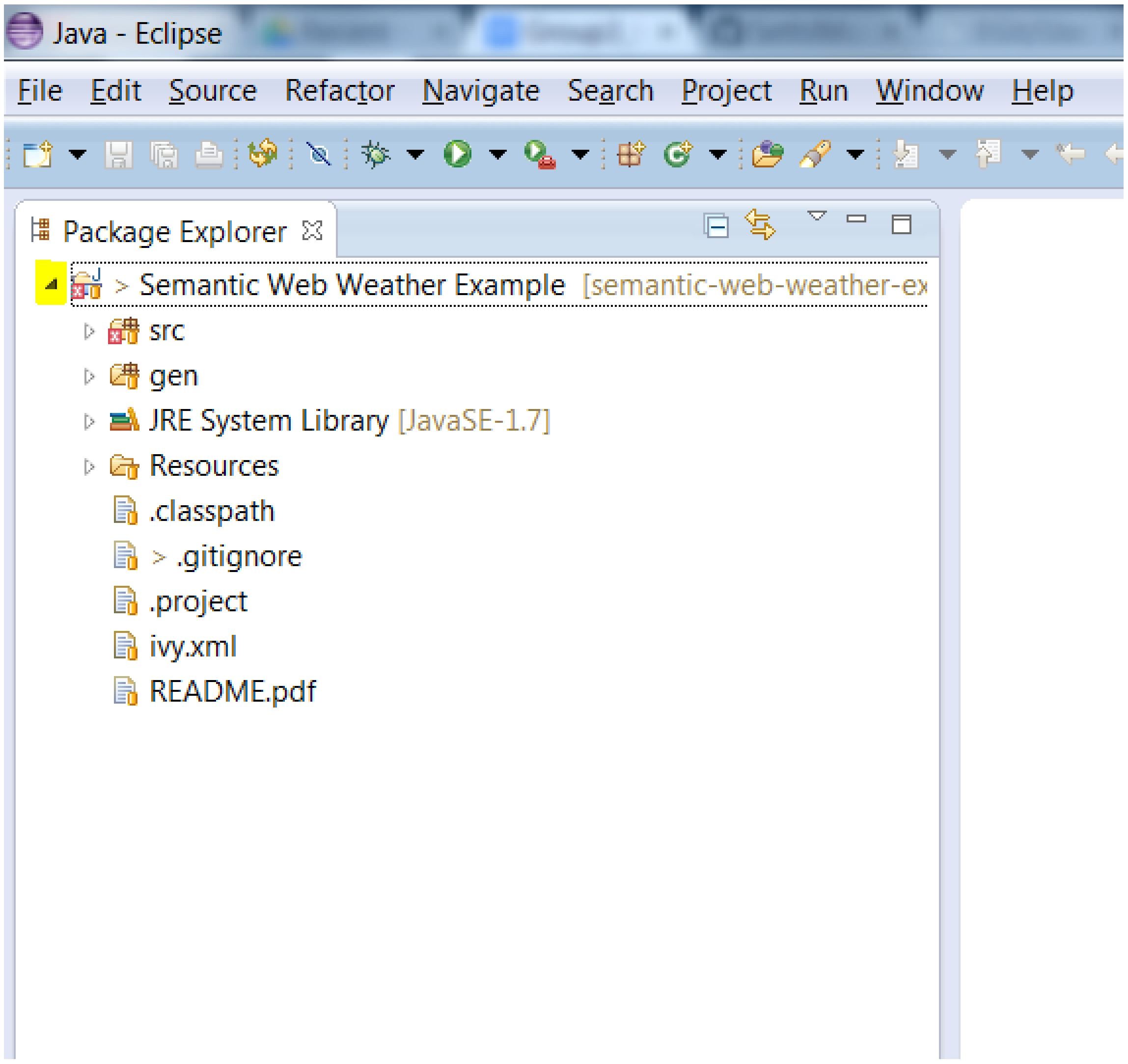
☑ Search for nested projects

Working sets

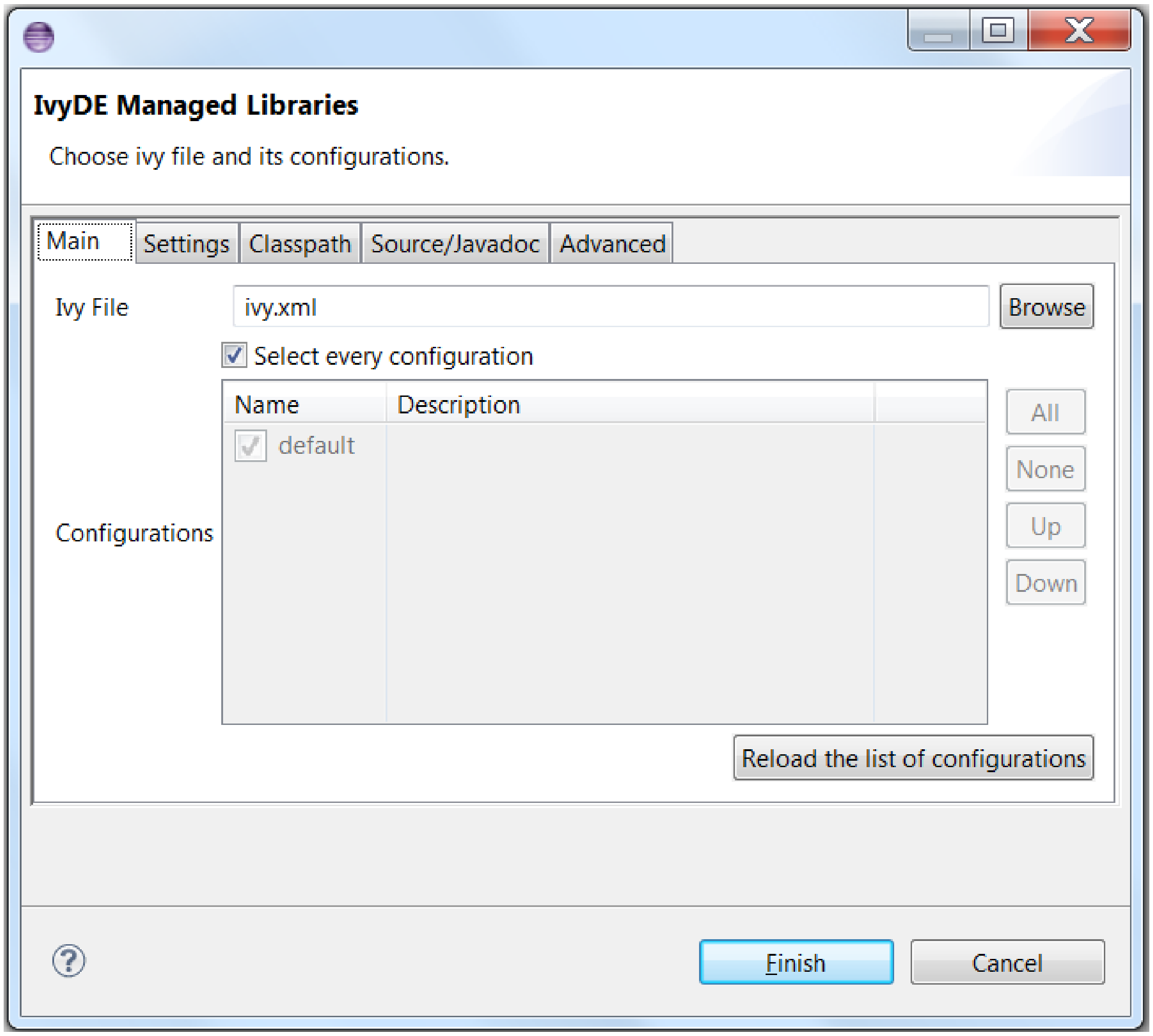
☐ Add project to working sets

Working sets:

Now you should have the **Semantic Web Weather Example** project in your **Package Explorer**. Click the arrow next to in order to reveal its files and directories.



This can't be run until we let **Ivy** take care of the dependencies for us, so right click on the file **ivy.xml**, and click **Add Ivy Library**, then click **Finish** on the window that pops up.



IvyDE Managed Libraries

Choose ivy file and its configurations.

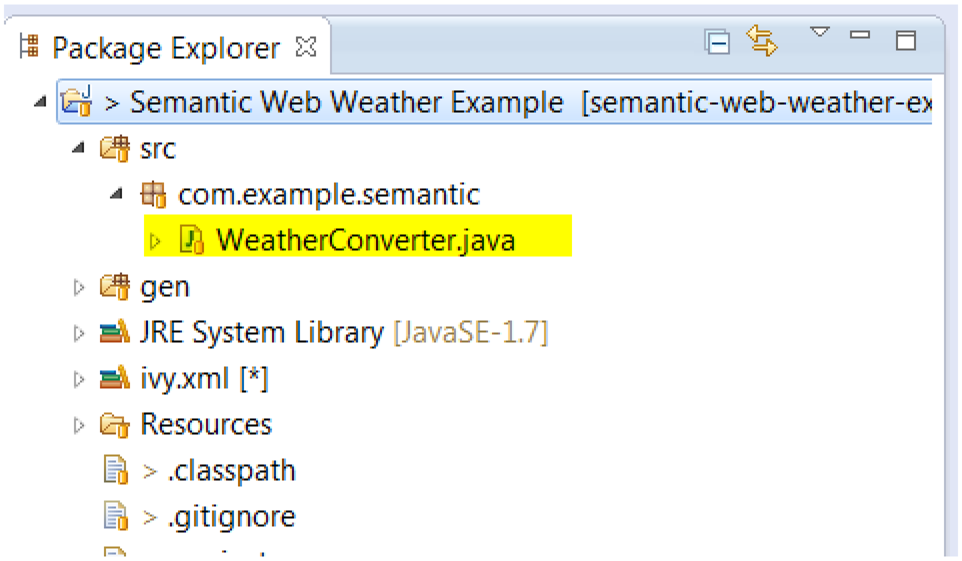
Ivy File: ivy.xml

[✓] Select every configuration

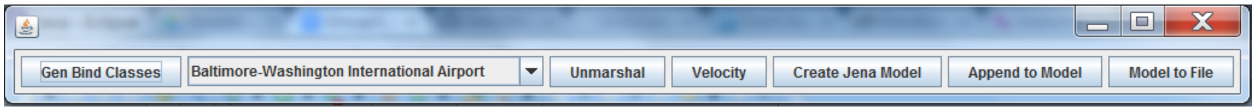
Configurations:

Name	Description
✓ default	

Expand the **src** directory, and the **com.example.semantic** package to reveal **WeatherConverter.java**.



Right click on **WeatherConverter.java**, and select **Run As -> Java Applicaton**. The WeatherConverter user interface should start.



Before clicking any of the other buttons, you need to click **Gen Bind Classes**. This will cause the program to close, after which you need to right click on the **Semantic Web Weather Example** project and click **Refresh** (this allows the project to recognize the newly generated classes in the **gen** folder.) When you start the program again from here on out, the rest of the functionality will work.