

Johan Henselmans

Eclipse Wonder tutorial
18 jan 2008

ECLIPSE WONDER TUTORIAL: INTRODUCTION¹

Since some time an orphan has been wandering through the world, and its name is WebObjects. WebObjects is a collection of Frameworks that connects databases, via abstract models, to all kinds of output/input devices, be it a website, a webservice or some other View. WebObjects hasn't received the love and attention from Apple it deserved.² This has resulted in the preposterous situation that, since the release of Leopard's Xcode, Apple's developer tools can no longer be used to build new WebObjects applications. This is despite the fact that Apple's iTunes Music store, the Apple store, .Mac and the Apple Developer website run and depend on WebObjects. Since WebObjects 5.4 Apple has also deprecated the frameworks used to create JavaClients.

But all is not lost. An orphanage has been created by some old hands and young bulls that are willing to work irresponsibly long hours to create their own alternative to the developer tools that used to be part of XCode. This replacement environment consists of several tools:

- WOLips, a set of frameworks that work as a plugin for Eclipse creating an environment in which all the files that form a WebObjects application can be edited and in which a project and all its parts can be managed, which are an alternative to WebObjects Builder and XCode (néé ProjectBuilder).
- EntityModeler, which is a standalone version of the EOModeler plugin that is part of WOLips and which can be used to create the abstract models that makes a database-independent model that will be used to connect the controller and the views to a database.
- RuleModeler, which is an alternative of the RuleEditor that came with the Developer tools before Leopard.

These tools are replacements for the tools that Apple supplied. In many cases they are better, as Apple had not updated these tools for a long time, apart from the most glaring bugs. But the orphanage has done more. First of all, some frameworks which include components and business logic that take care of a lot of common situations have been published and open sourced by people from the developer community. The most well known is Wonder, but also the frameworks of Andrew Lindesay (LEWOSTuff), WireHose (WireHose), GVC (GVCSitemaker), WebAppz (PDFKit) and Ravi Mendez (SVGObjects) have been donated as open source or free additions to WebObjects.

There are even two alternative frameworks in development to make it possible to use WebObjects as a backend for Java client from Paolo Sommaruga (<http://www.jpaso.com/name/XMLBindingForJavaClient>) and Florijan Stamekovic (http://web.mac.com/flor385/eSwamp/software/wojc_tutorial.html).

¹ This tutorial was started in 2008, but as there was so much flux in the tools and policies surrounding Wonder, finished in 2011. Things have stabilised since 2008, and it seems the move to github from Wonder creates an new impetus to get Wonder on a new track.

² After it saved the butt of Steve Job's NeXT when things were low, (when a license of WebObjects would set you back for 50.000 US\$ and was considered the serious competitor of the likes of BEA and WebSphere) since the take-over of Apple by NeXT,

Eclipse Wonder tutorial part one: setting up the data

Together these tools and the Wonder framework are available as open source on github, at <https://github.com/projectwonder/>. We call this set of tools and framework Wonder.

In this tutorial we will try to demonstrate some best practices to develop a Wonder application, and to demonstrate how to use some of the tools, with these objectives in mind:

- The application should be database independent
- The application should be built to use as little code as possible
- Code that is general should be reusable without modification in other projects
- The presentation in the web browser should be modifiable for web 'designers' to suit the whims of today's fashion in fonts, layout and navigation
- the latest (fashion's) technologies in authentication (OpenID), web-user interface (Ajax), storage (Amazon S3) and presentation (PDF) are used, and we will explain how to add your own latest and greatest tools.

The application that is built is open source and public domain, use it for your own benefit.

This application will be a shop. Although most people abhor shop applications, it is very useful to demonstrate some basic concepts of a webobjects application. First of all, we will have to work with product categories, products and pictures of these products, with orders and invoices and pdfs of these invoices and with customers and suppliers, and there will be a back-end to update the product info and get reports and excel sheets on all kinds of aggregated information and a front-end for the customers.

Finally, starting with Wonder can be daunting. It is a huge collection of Frameworks, and it is easy to lose track. But as Tom Hanks said to a starting actors: 'Persevere', or as a Dutch artist wrote on a oil-storage tank in a derelict industrial area: 'Houdt moed'. (Courage!)

There is a great community that can help you out at increasing levels of complexity, and the building up of a webobjects community around www.wocommunity.org might help to find the information that you know is somewhere, but where exactly? We hope that this tutorial will give you some feeling where to look for what, and give you a gentle introduction into the current tools and frameworks so that you, in the end, can stand on your own feet and start helping others.

What data do we have to store?

First of all we define the data that we need to get our shop working. Here is a list of information that we must have and store:

1. products: this is an item that we want to sell. It will have a price, a picture, a description and some extra attributes that depend on the type of product.
2. product categories: to make searching easier we have product categories.
3. customers: we sell the products to customers. These can be organizations and individual people. We will need to know their name, addresses, bank accounts, email.

Eclipse Wonder tutorial part one: setting up the data

4. suppliers: these are the suppliers of the products. In case we run out of products, we have to get some more stock.
5. employees: these are the people that have to make sure people get what they want, check the inventory, make sure the payments are on time, and that the whole organization will run without a loss.
6. orders: these are the messages that customers send us to let us know what products they want.
7. order items: an order item will be made for each product
8. invoices: these are the messages that we send the customers to let him know how much that would cost him
9. invoice items: an invoice item will be made for each product
10. payments: this is the information we get from the bank that assures us the customer has paid for his order, and that we have paid our suppliers.

Normally, in a database oriented world, one would look at the data, and say that one needs 10 tables. But that is in a database world. If we look at it from an object world, we see something else.

Customers, Suppliers and Employees are more or less the same. They share common characteristics that would be repeated over and over again: name, address, bank account, login name. On the other hand each of these has some other characteristics.

- A supplier will have a link to products he has supplied and the payments he has made.
- A customer will have a link to payments he has made, the orders he has placed, and the invoices he had gotten.
- An employee will have certain restrictions in what he can do with the data, will have a birth-date, some information on his social status that will lead to some kind of tax-group that his wage will be subjected to, and will have a link to payments he has gotten, to orders he has full-filled and to products he has added to the database.

To reduce the amount of work that we have to do in the database we could make one table that encapsulates all of the characteristics that these three information items need. Then, in our object model we could separate these three information items, and, while making use of their common characteristics, add their own separate characteristics.

The same thing will happen with products and product categories. Suppose our shop will sell tools, and we will have handicraft tools and electric tools. For electric tools it is important to know what voltage they require and the wattage they use. For handicraft tools it is important to describe the usage. Both will have a price, description and weight and size attribute,

Our product table will have attributes for as well the electric as the non electric tools, and depending on the product category it will be displayed in, some of the characteristics will be displayed and will have to be added if one adds a new product of a specific category to the database.

Eclipse Wonder tutorial part one: setting up the data

Why go through this hassle of collecting employees, customers and suppliers in one table, and then separating them again? The answer is laziness. We want to do as little as possible. If we create three tables for these entities, we would have to write three components that would get the name, address, email and login name and password, check if the data is OK and add them to the database. If we collect this information once for all the three entities, we only have to write the verification of this data once.

There is also something else. There has been research about the fallibility of developers. That has lead to the observation that, on average, every 1000 lines of code contains 10 errors³. So if you write less code, that also means that there are less errors. (That also shows that the trumpeting of some software companies about the huge amount of code their software products contain, to emphasize the amount of work that has been done actually says something about the amount of errors their software will contain...)

How will we present the data?

Another question is the presentation of the data. Although everybody has something to do with the products we sell, not everybody will get the same data presented. Also, the presentation will take place in several ways. There will be forms, reports and lists. These reports and lists can be presented via a web-browser, but it can also be an email message, or a pdf file. Also in this case we will have to realize that less is more. The better we can reuse some of the code to present the report as an email message, a web page and a pdf report, the less errors we have made and the less work we will have.

To know how the data has to be presented, the best way is to describe some scenarios, or story boards, and make some mockups of screens and reports that one will get.

An example:

A customer wants to buy a product. He will get to the website and will be presented with today's offers, and a choice of product categories that he can choose from. He will also have the option to search for a specific product if he does not know under which category his product will fall. Depending on his choice he will be presented with a product list. Each product will have the option to add to a shopping cart. When he is ready to order, he will get a page that, depending on his previous visits, will present him with his own information, or will give him the opportunity to give his information: address, name, email, bank account, shipping method etc. After that, a check of his ability to pay will be done. If that is OK, he will be given a confirmation that his order has been received and he will be sent a confirmation via his email address. He will also get the opportunity to print his order, or to download a PDF with his order. In his email there will be a possibility to get back to the shop and to check if his order has been fulfilled and has been sent.

³ The book "Code Complete" by Steve McConnell is quoted on that in <http://amartester.blogspot.com/2007/04/bugs-per-lines-of-code.html>, see also http://en.wikipedia.org/wiki/Source_lines_of_code

Eclipse Wonder tutorial part one: setting up the data

These storyboards will give a feel how the application should work, and will verify that all the steps that have to be taken are included. It will also give you the answer to the question how the data needs to be presented.

Verifying and testing the data

After we have described the scenarios it is time to think about the validity of the data. If developers, the prima donna's of precision and accuracy, make so many mistakes, how bad is it then if we let mere mortals add information to our precious data? How many wrong email addresses, unknown telephone numbers and non-existing addresses and bank accounts will be typed in by these primitive creatures called human beings? Is there a way to prevent the wrong data getting into our database? Fortunately, Wonder and the extra frameworks used gives us ample opportunity to prevent this from happening.

Another thing that we do not want to do, is to repeat ourselves in testing the application. To prevent ourselves from doing that there are tools that will do just that for us.

Let's start!

Now that we have described the environment that we want and what we will need, we can start.

- We will first create a model of the data that we want in our database.
- Then we will start to fill the database with the products.
- Then we will give the customer the opportunity to buy something.
- And finally we will let the employees pick up the orders and send the package to the customer.

For each of these procedures we will give a scenario, after which we will fill in the blanks.

So now it is time to start programming. What do you need?

1. Basic WebObjects from Apple
2. Eclipse
3. WOLips
4. A database: postgresql, frontbase, openbase, oracle, mysql, h2, whatever
5. Assorted frameworks: Wonder, PDFKit, LEWOSTuff

As simple as it sounds, this is the place where a lot of introducedees will experience that they have somehow ended up on the wrong party. Fortunately, there is a new installer on the block: golipse. This will give you all the stuff that you need in one big installer file.

I would advise you to first look at the podcast of Pascal Robert, and proceed from there: it's on http://www.wocommunity.org/podcasts/WOLips_install_in_8_minutes-stream-wifi.mp4 After that, goto <http://wiki.objectstyle.org/confluence/display/WOL/WOLips>, where you can find information on

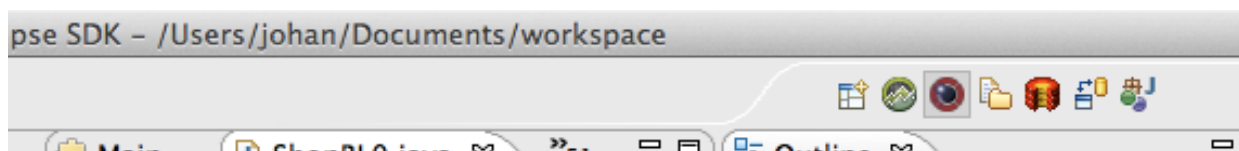
Eclipse Wonder tutorial part one: setting up the data

installing on several platforms. Or you can go to <http://wiki.objectstyle.org/confluence/display/WOL/Tutorials> where a complete section is devoted to installation on OS X or Windows. We will have a look at deployment later; rest assured that WebObjects applications can be deployed in any platform that runs java. If you have everything installed and running, it is time to get going.

Creating projects in Eclipse

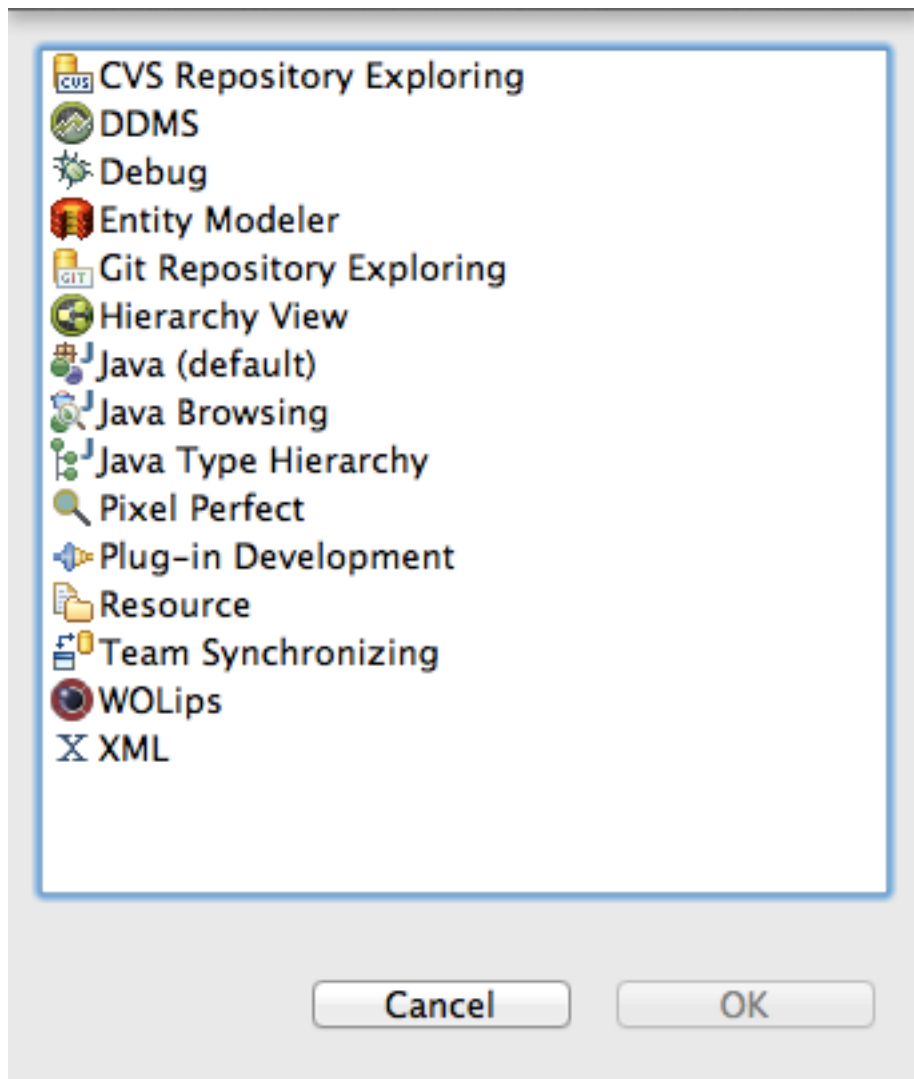
As you have seen before, one can describe the data and the presentation of the data as separate items. This way of looking at an information system is also known as the *Model View Controller* way of building an information system. the Model is the data, via the Controller the data can be presented in a variety of ways, the View. Most of the times you will discover that the data structures will not change a lot (if there has been enough thinking power spent in being as lazy as possible). The presentation, however tends to change a lot. And sometimes there are various applications that make use of the same model. To make it possible that multiple applications make use of the same data model, we will separate the model into its own framework, which different applications can use.

To do that, we start Eclipse. After starting Eclipse, you will be presented with a window in which everything takes place. What exactly takes place in the window depends on the so called *perspective* that you use for Eclipse. The perspectives are chosen in different ways. The most used perspectives will show up in the top toolbar of Eclipse, on the right hand side:



The WOLips perspective is the one we are using now, so make sure you have selected that one.

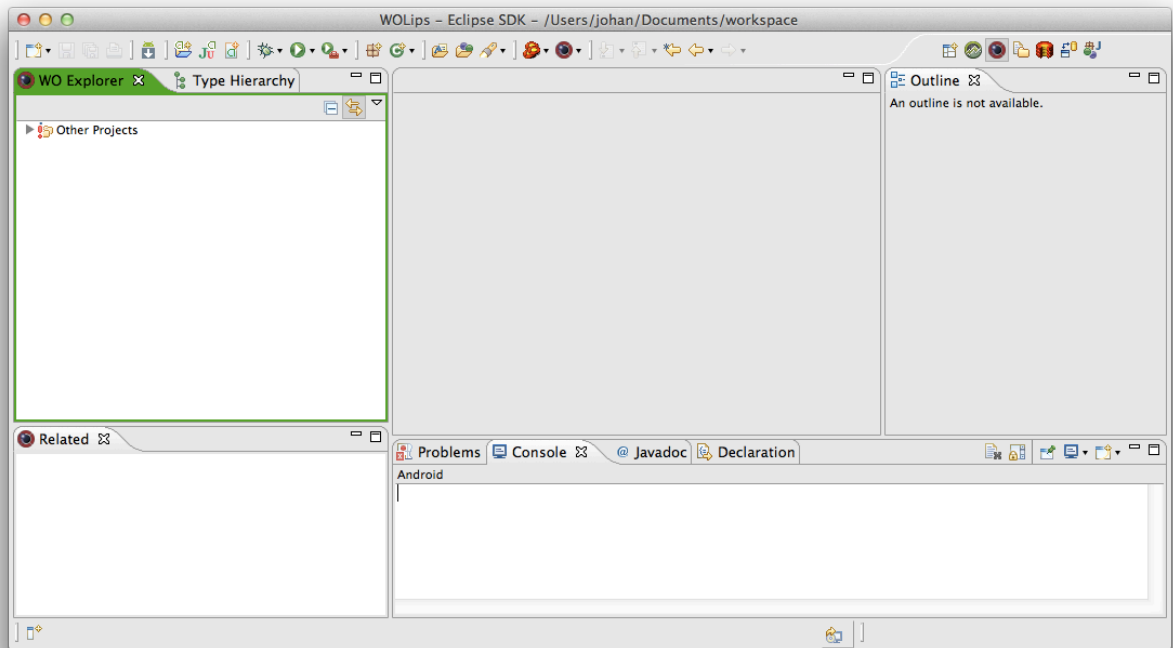
Another way to choose the WOLips perspective is via the top Menu Windows>Open Perspective, which will show you the currently available perspectives:



Choose the WOLips perspective.

The perspective should look like this, more or less:

Eclipse Wonder tutorial part one: setting up the data

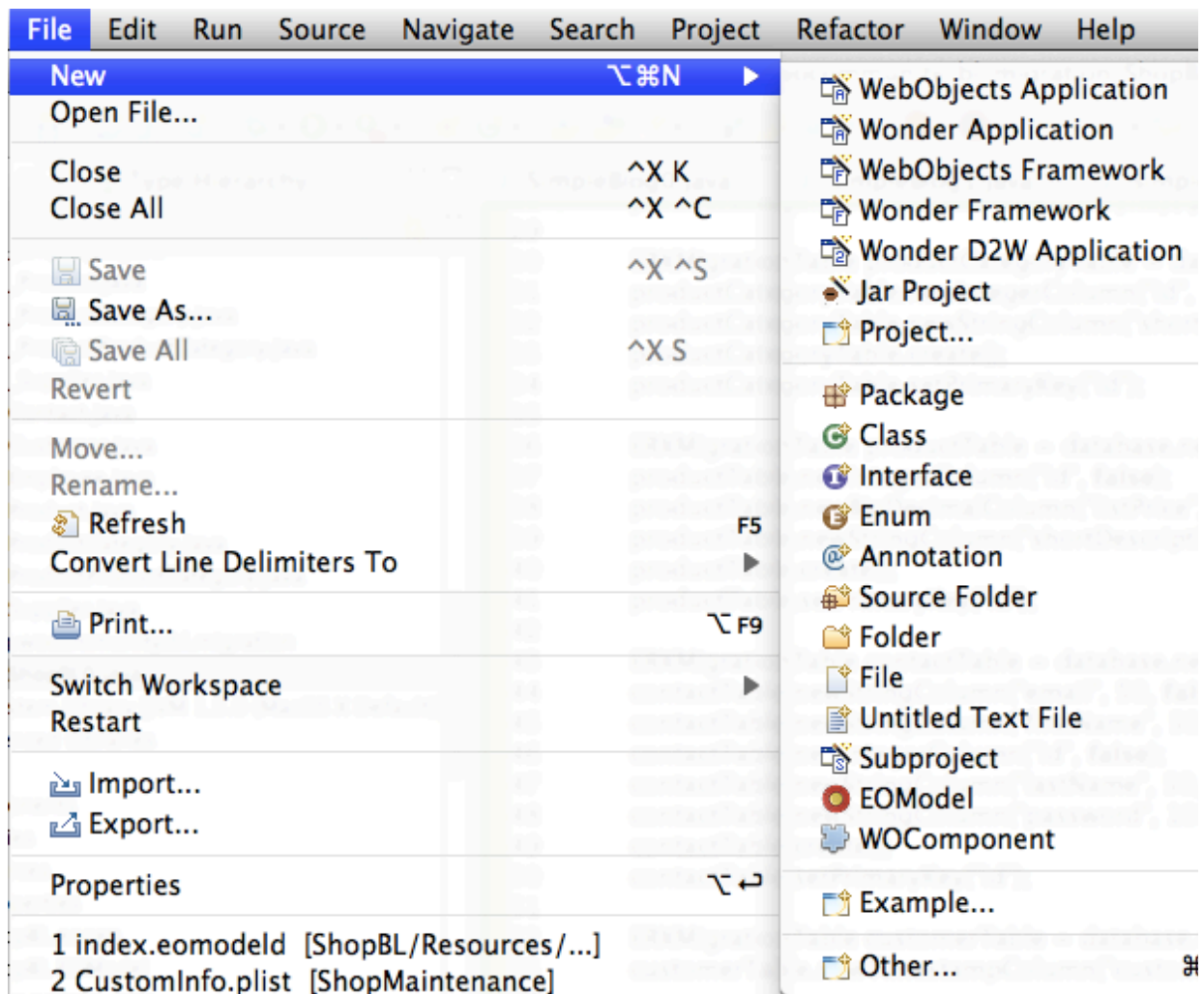


Notice that the way the toolbar on the top can look depends on the amount of plugins that you have installed.

Creating a Framework for our BusinessLogic

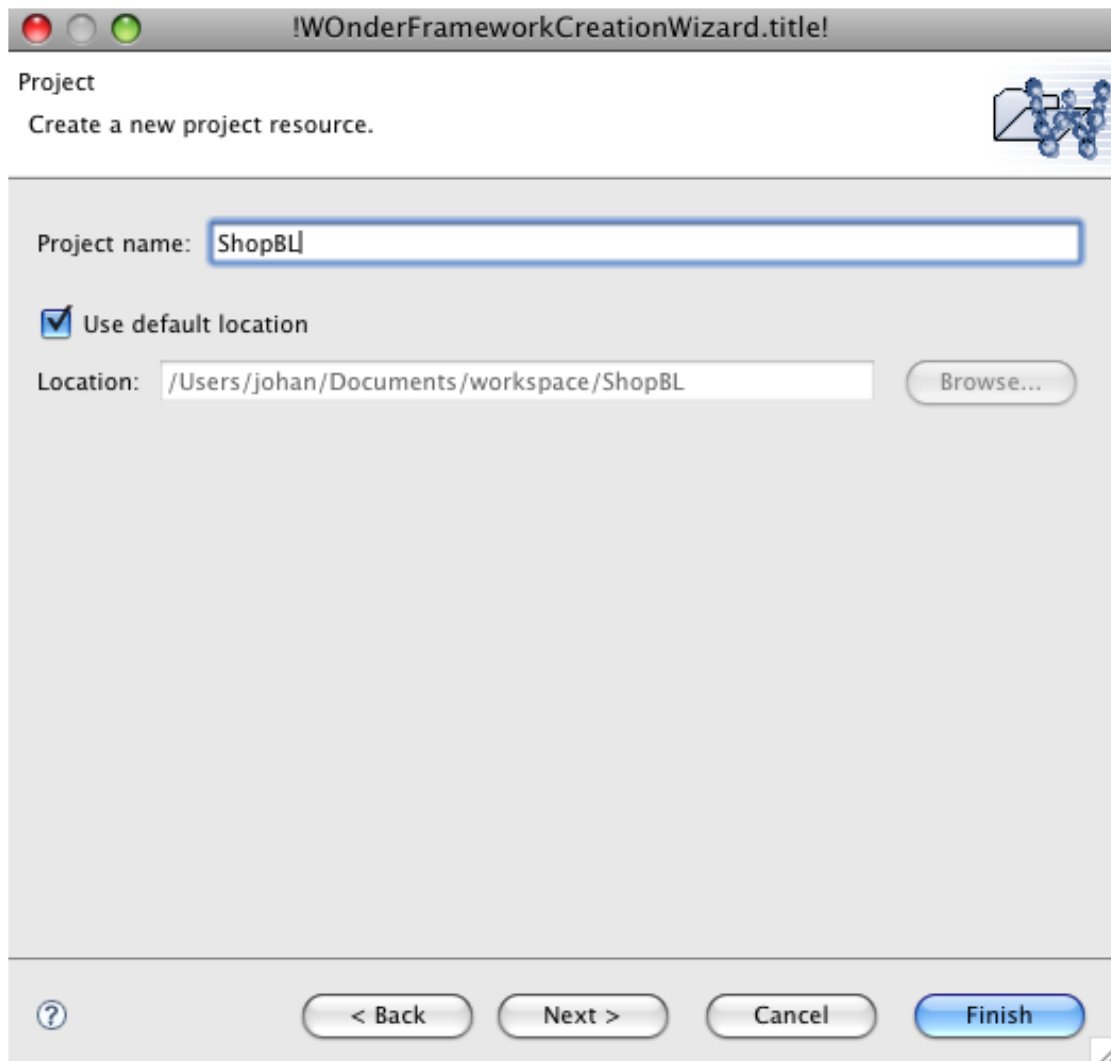
We have already said that we want to combine the three roles: customer, employee and supplier into one table. We will also make the model in such a way that it is easy to swap databases. To see how that is possible we will start with creating a framework from the New Menu Item. We will make a Wonder Framework.

You can create a Wonder Framework under New >:



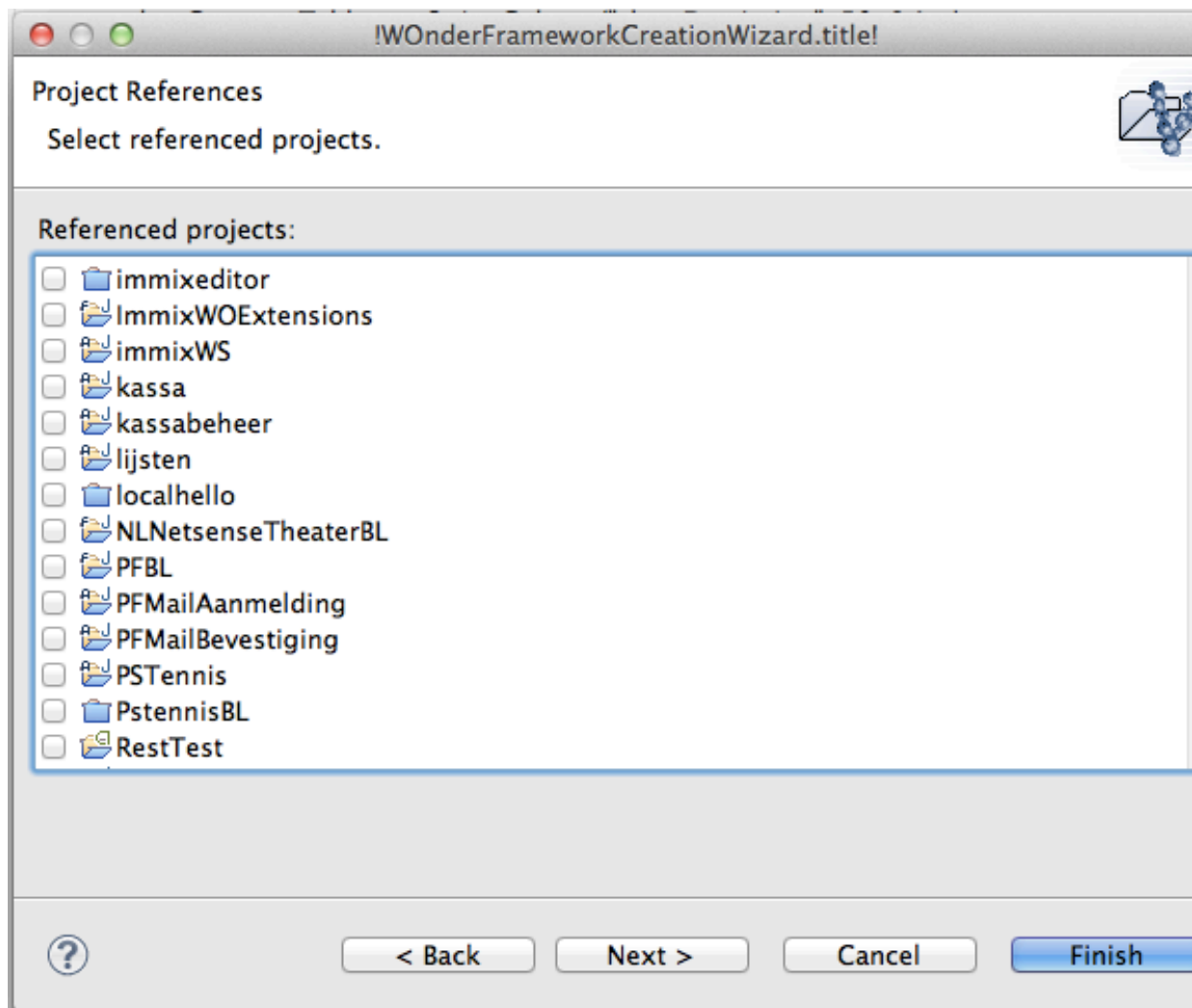
After you have selected the Wonder Framework option, you will get the question to create a project.

Eclipse Wonder tutorial part one: setting up the data

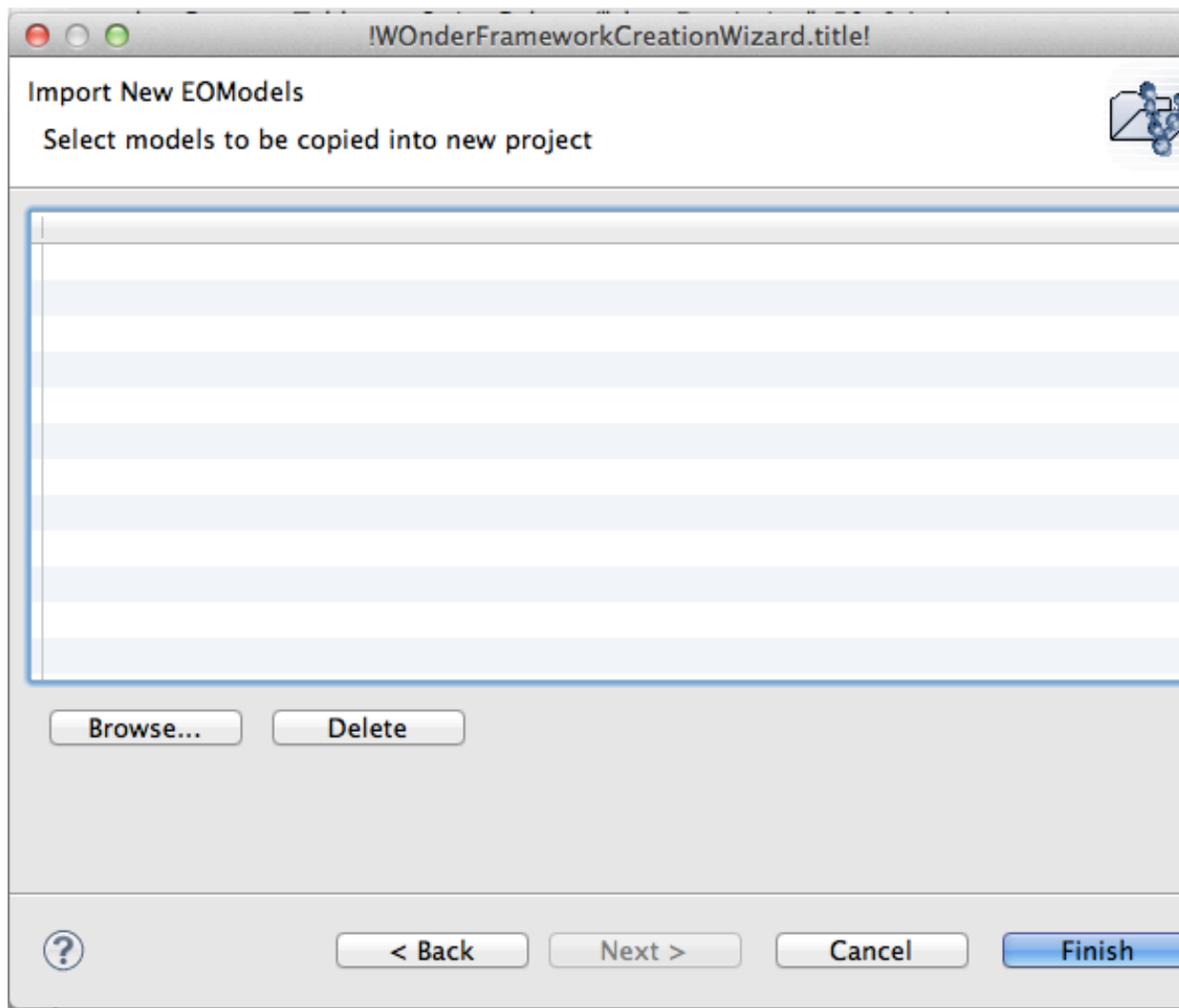


Name the Wonder Framework ShopBL. Click Next.

The next question will be if you want to link it to other Projects. In our tutorial, we have no existing projects yet, so don't select anything, just click next.



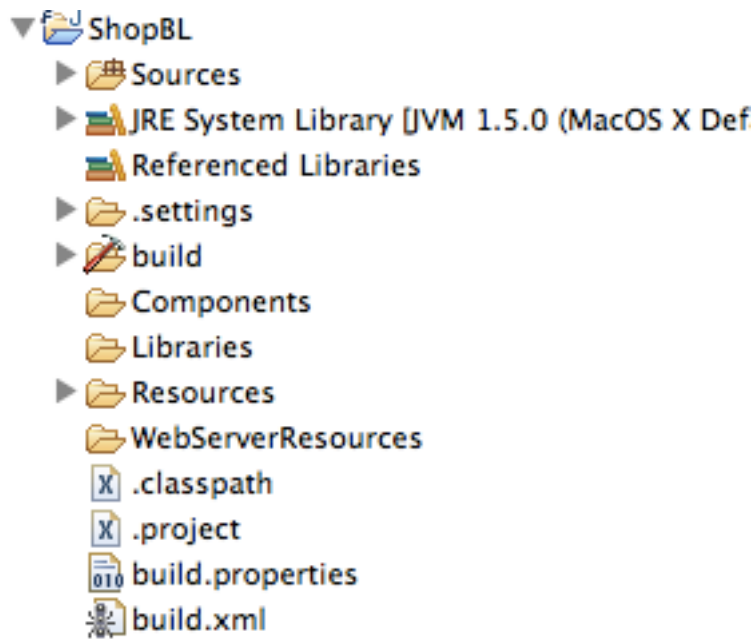
The next question will be if we want to copy EOModels into our project. We do not have a EOModel either (yet), as we still have to construct the model. An EOModel is the Model that will make the link between your classes in Java and the Database. EO stands for Enterprise Objects.



So do not bother about it for now. Click Finish.

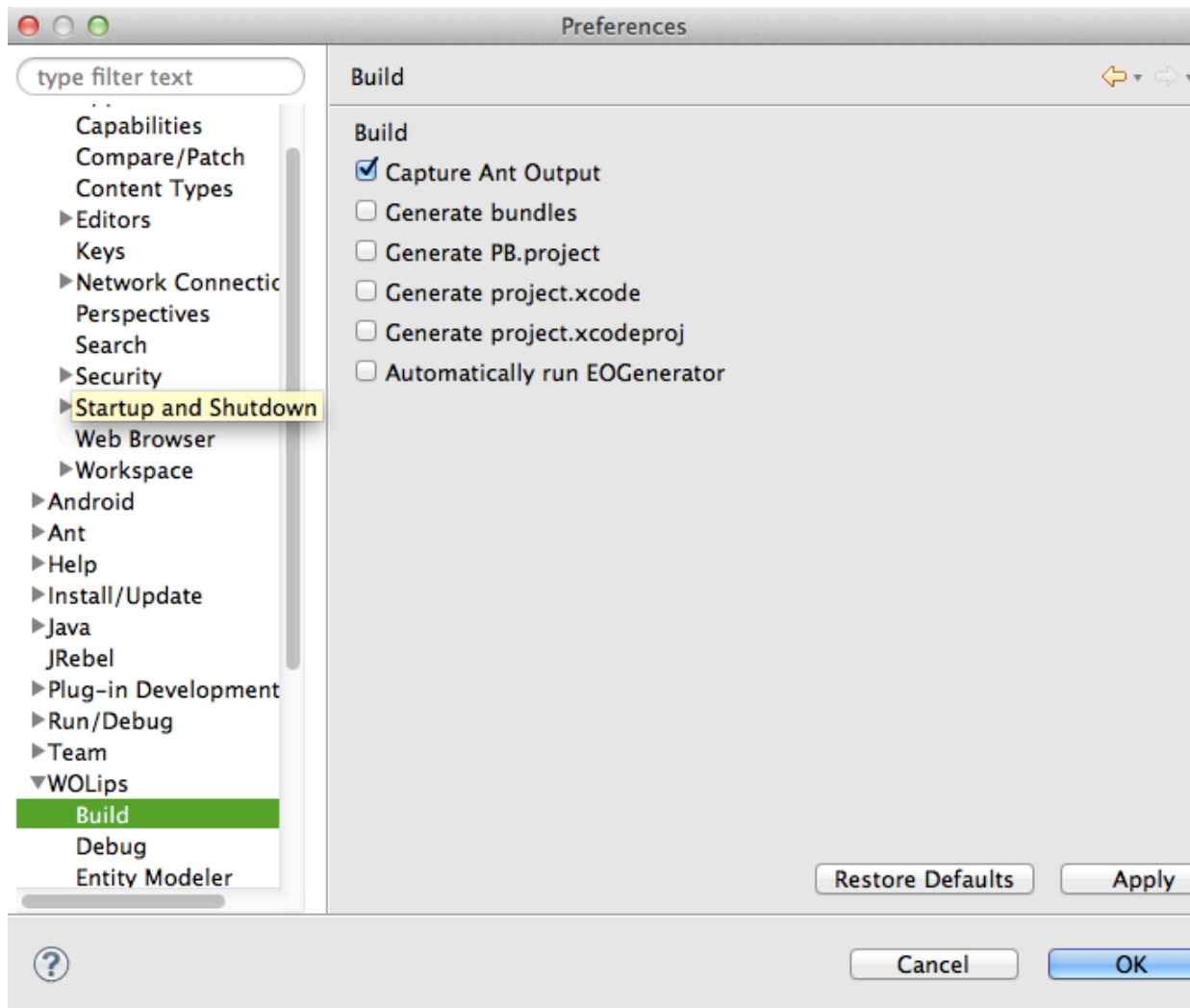
Click on Finish, and then in the WOExplorer there will be a new project, called ShopBL,

Eclipse Wonder tutorial part one: setting up the data



Inside this project you will see a collection of folders.

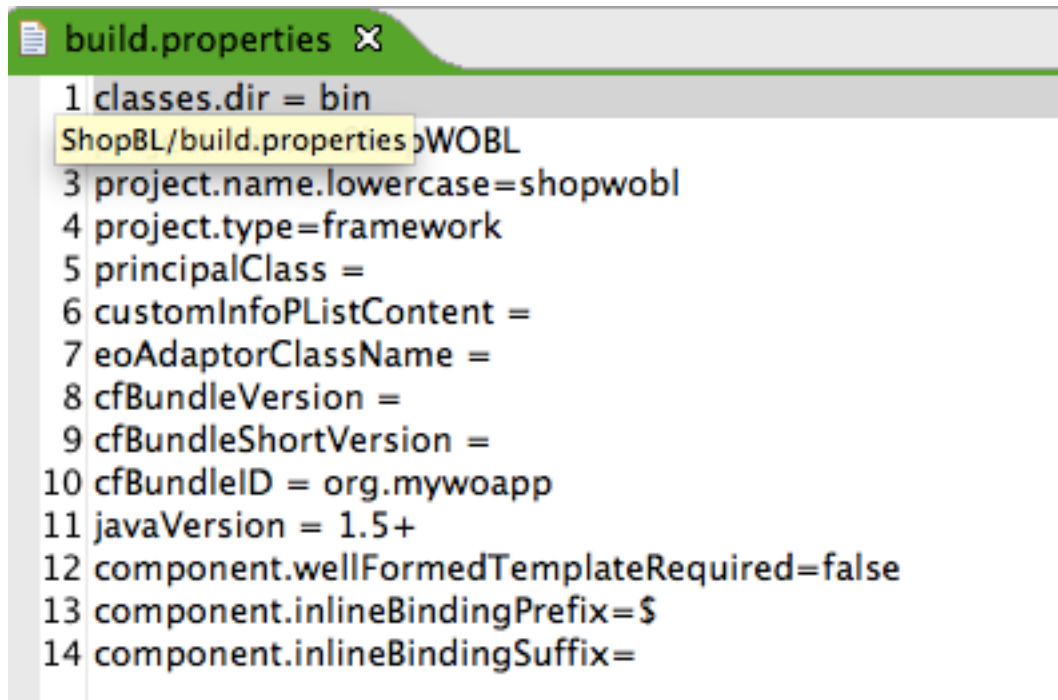
- In Sources, you will find the java files. There aren't any yet
- The WebObjects Frameworks and JRE System Library constitute the Frameworks against which the application will run. In build you will find the compiled product.
- In Libraries you can add extra frameworks to be included in the application.
- Components is the folder in which you will find the files that will constitute the presentation layer of the application
- WebServerResources will contain the files that will be presented to the client via a webserver;
- in Resources you will find the Properties and the EOModel that will define our data., the database that we will connect to, and the way we will handle the communication with the database.
- There might be a folder build. If so, the settings of your WOLips configuration predates some of the changes in 2010, in which so called bundleless builds were introduced. It is important that you will use Bundleless builds (this will become clear in the next chapter). To make bundleless builds, go to Eclipse>Preferences, and make sure that Generates Bundles is turned off



- Finally, you will see some files (.classpath, .project, build.properties, build.xml), that constitute (together with the files in the hidden folder .woproject) the settings of the Project. These are important in case you want to rename the project.

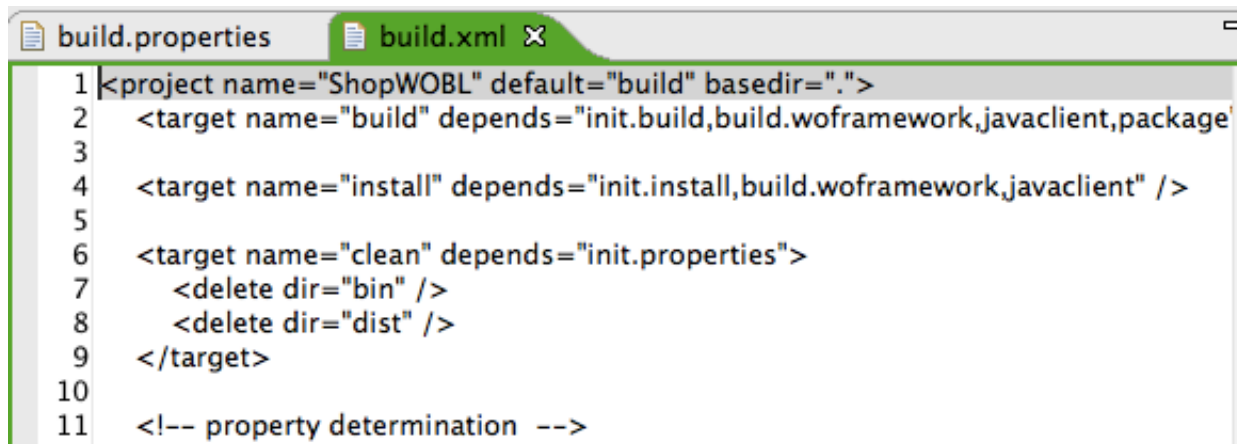
Renaming a project

One would assume that Eclipse would be wise enough to rename all the references to the project in such a case. That is not true. You have to change the projectname yourself in build.xml and build.properties.



```
1 classes.dir = bin
2 ShopBL/build.properties
3 project.name.lowercase=shopwobl
4 project.type=framework
5 principalClass =
6 customInfoPListContent =
7 eoAdaptorClassName =
8 cfBundleVersion =
9 cfBundleShortVersion =
10 cfBundleID = org.mywoapp
11 javaVersion = 1.5+
12 component.wellFormedTemplateRequired=false
13 component.inlineBindingPrefix=$
14 component.inlineBindingSuffix=
```

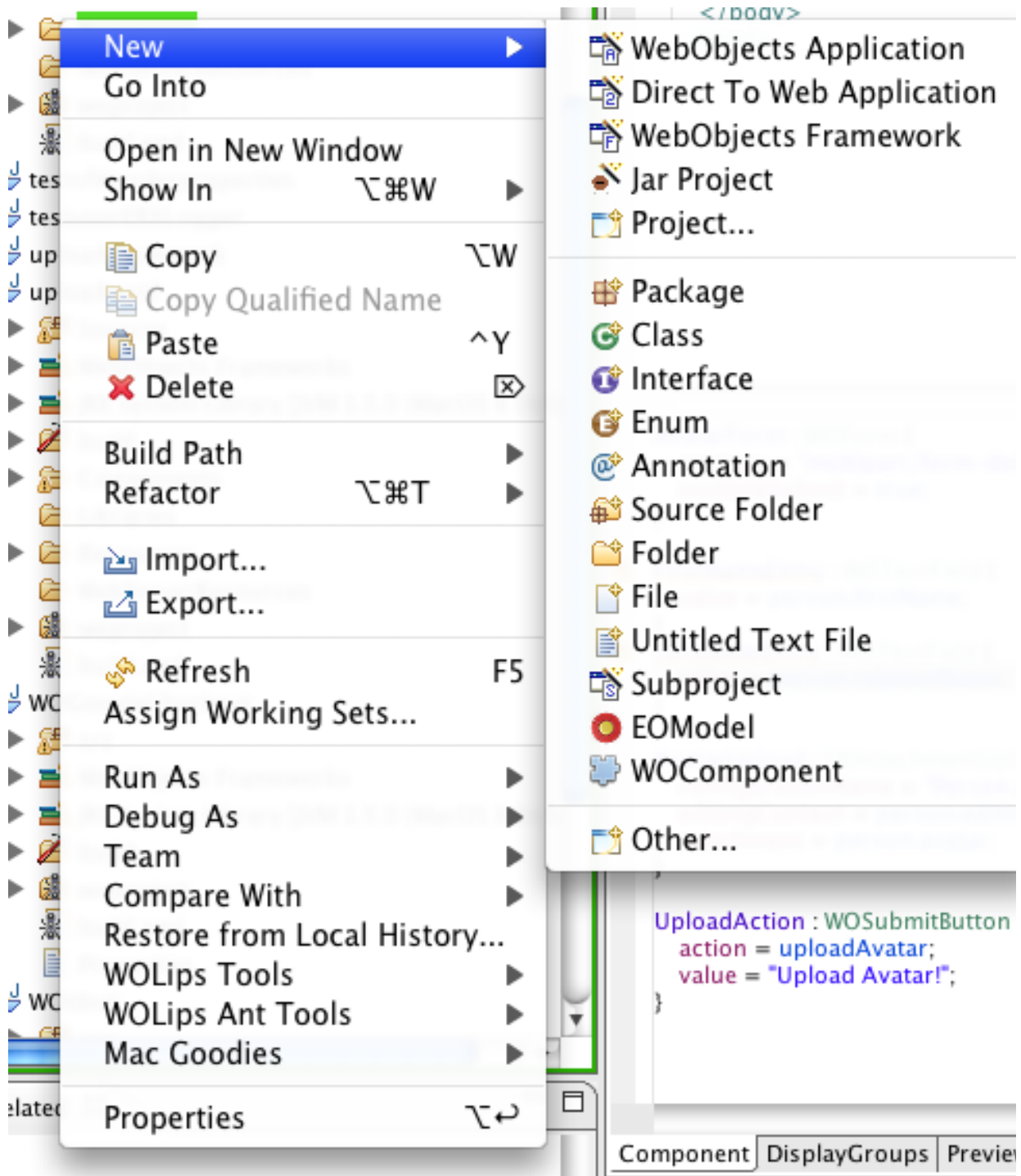
This Framework was called ShopWOBL originally, and then renamed to ShopBL. In the build.properties file and the build.xml file still the old names linger on. Notice that you also have to change the lowercase name in build.properties.



```
1 <project name="ShopWOBL" default="build" basedir=".">
2   <target name="build" depends="init.build,build.woframework,javaclient,package" />
3
4   <target name="install" depends="init.install,build.woframework,javaclient" />
5
6   <target name="clean" depends="init.properties">
7     <delete dir="bin" />
8     <delete dir="dist" />
9   </target>
10
11 <!-- property determination -->
```

Creating an EOModel

Now that we have a framework we will create the Model that will constitute our information system. To do that make sure you are in the WO Lips perspective. In the WO Explorer navigation bar click on Resources, and then right click on New > EOModel. (You could also click on the EOModel icon in the top toolbar of Eclipse).

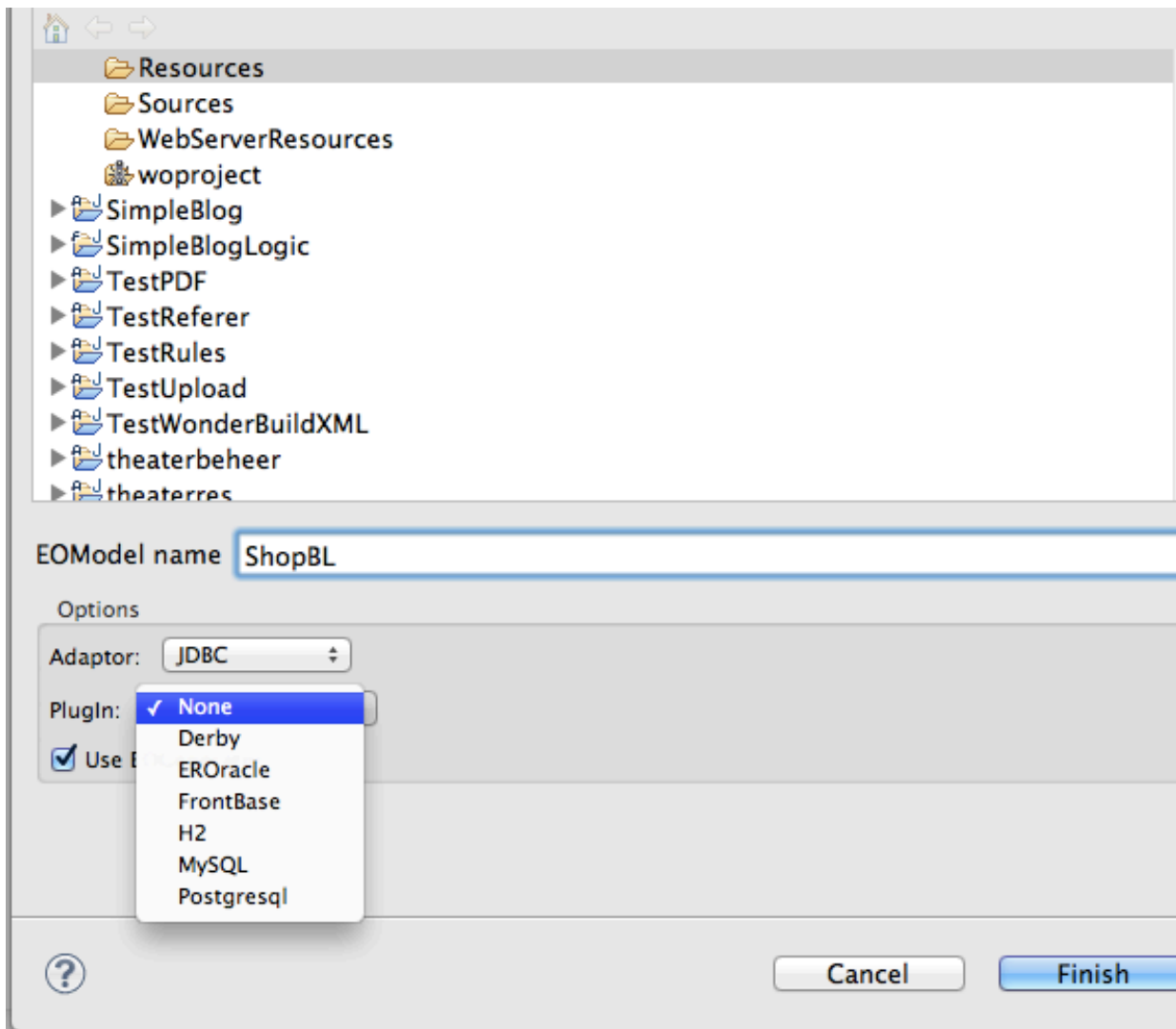


You will be presented with a window that will ask you:

- the location where the EOModel will be added,
- the name of the Model: make that ShopBL
- the Adapter type. Click on JDBC
- the plugin for your database. Use the one you are using.

Eclipse Wonder tutorial part one: setting up the data

- the checkbox “Use EOGenerator”



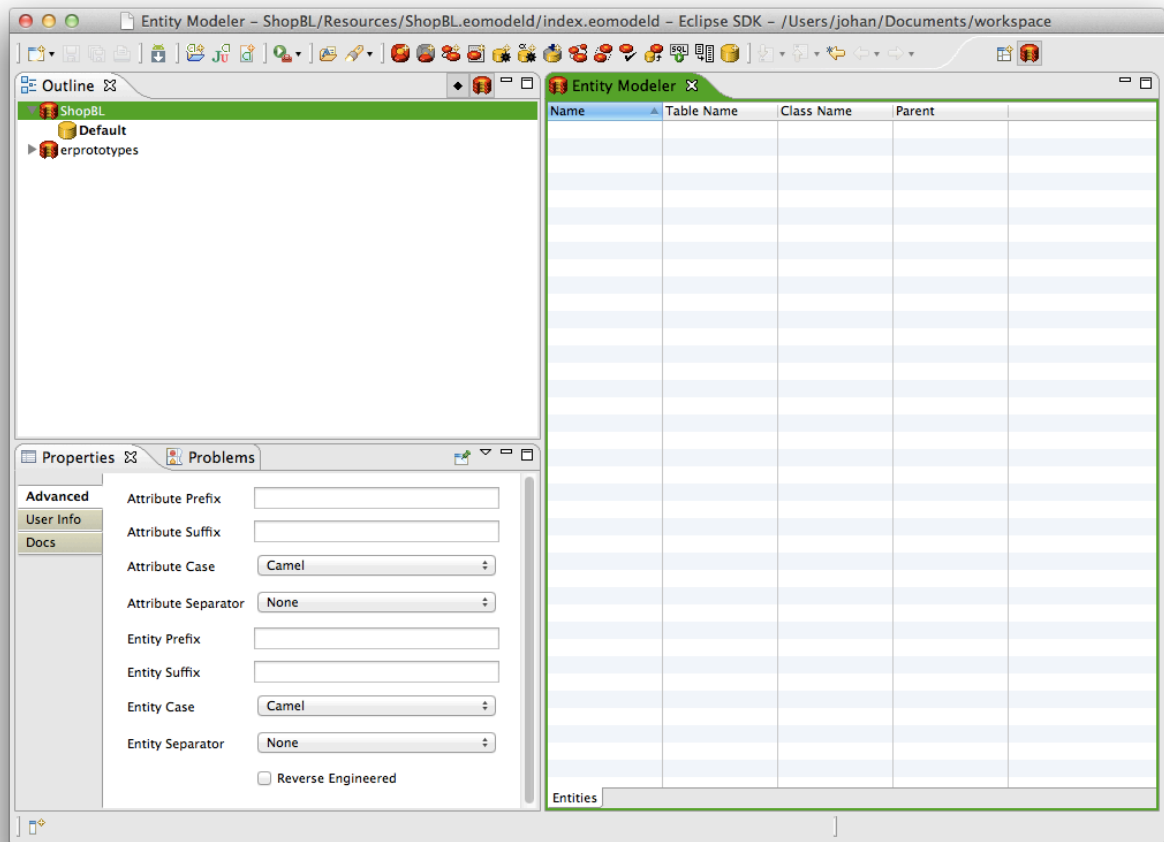
Click on Finish.

Now you will see the EOModel, within a different Perspective: the EOModeller perspective. If that is not the case, goto the EOModel in the Resource folder in the WOExplorer sidebar and double click on the ShopBL icon.

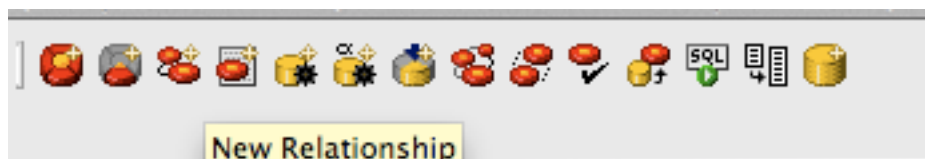
Perspective is the Eclipse lingo for presenting a project or part of a project in a specific way. In this case, we will get a presentation of an EOModel in a specific way. Perspectives can be infuriating, as sometimes the whole layout of your workspace seems to be gone to smithereens, because somehow you clicked on a button that changed the perspective. Don't worry, it is just a presentation: Window > Reset Perspective or Window > Open Perspective > Other... will make your environment sensible again.

Eclipse Wonder tutorial part one: setting up the data

The EOModeler perspective should look something like this:



On the top toolbar of the EOModeler perspective, a whole bunch of icons is added. Scrolling over these icons will show you what kind of action the icon will start.



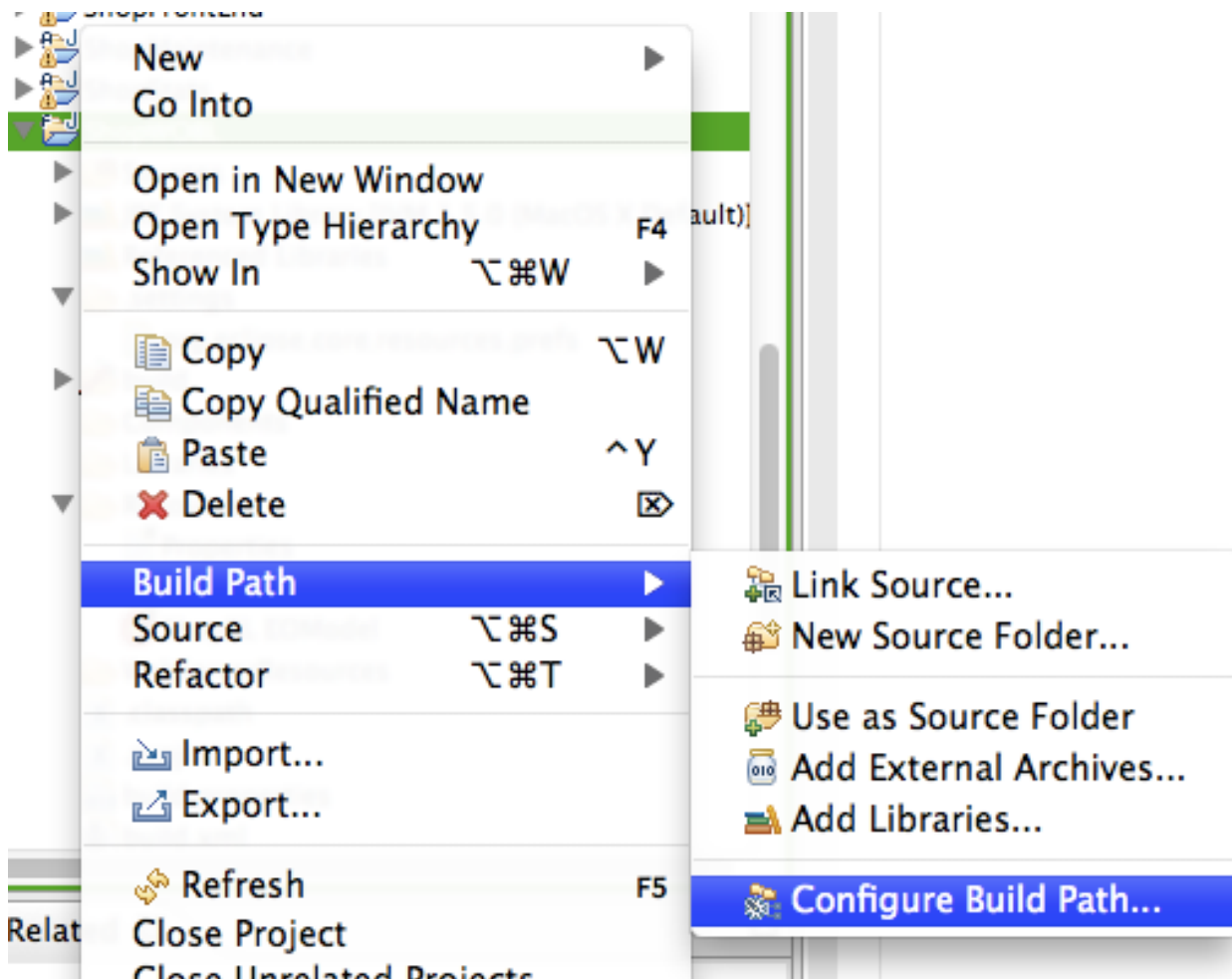
If everything is correct, there will be an Outline sidebar, in which you will see a Default button, a cylindrical object called ShopBL and a cylindrical object called erprototypes. These red-yellowish cylinders are EOModels. Inside an EOModel you will find Entities, that describe the classes and the tables that are linked to these classes, and a connection dictionary, that describes what database this model is connecting to.



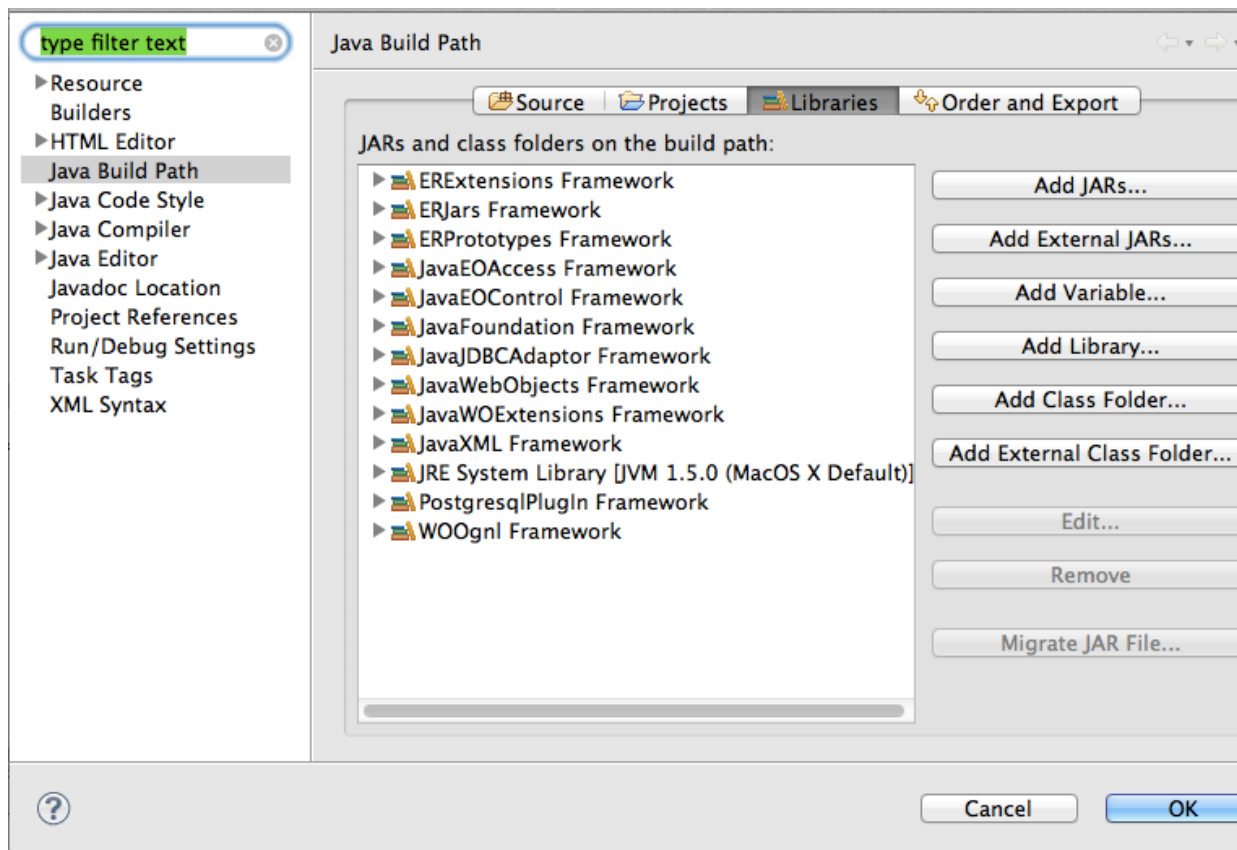
Notice the, although we created a Model called ShopBL, another EOModel, called ERPrototypes also popped up. This EOModel is also present because during the creation of the Wonder Framework, a bunch of standard Frameworks always get included, among which is an ERPrototypes Framework. If you would look inside the ERPrototypes Framework, you would notice that in this Framework there is another EOModel in the Resources folder. EOModeller will show all the EOModels that are available in the Frameworks that have been linked to this Framework.

Which Framework are linked in my project?

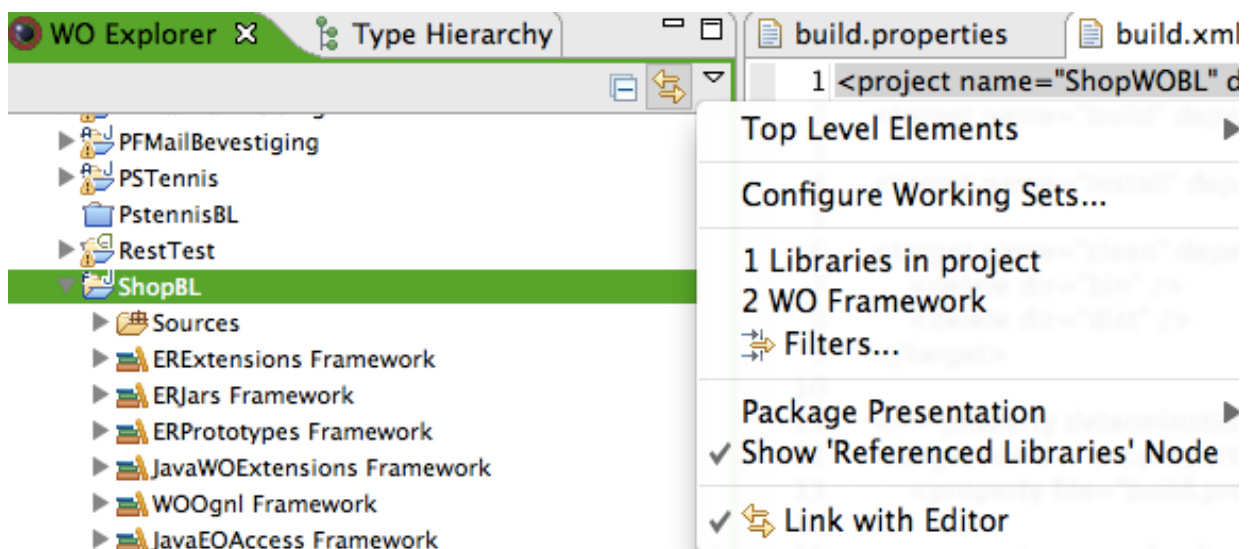
You can find out which Frameworks are also linked in your project by right click on your project name in WOExplorer and selecting the Configure Build Path:



Eclipse Wonder tutorial part one: setting up the data

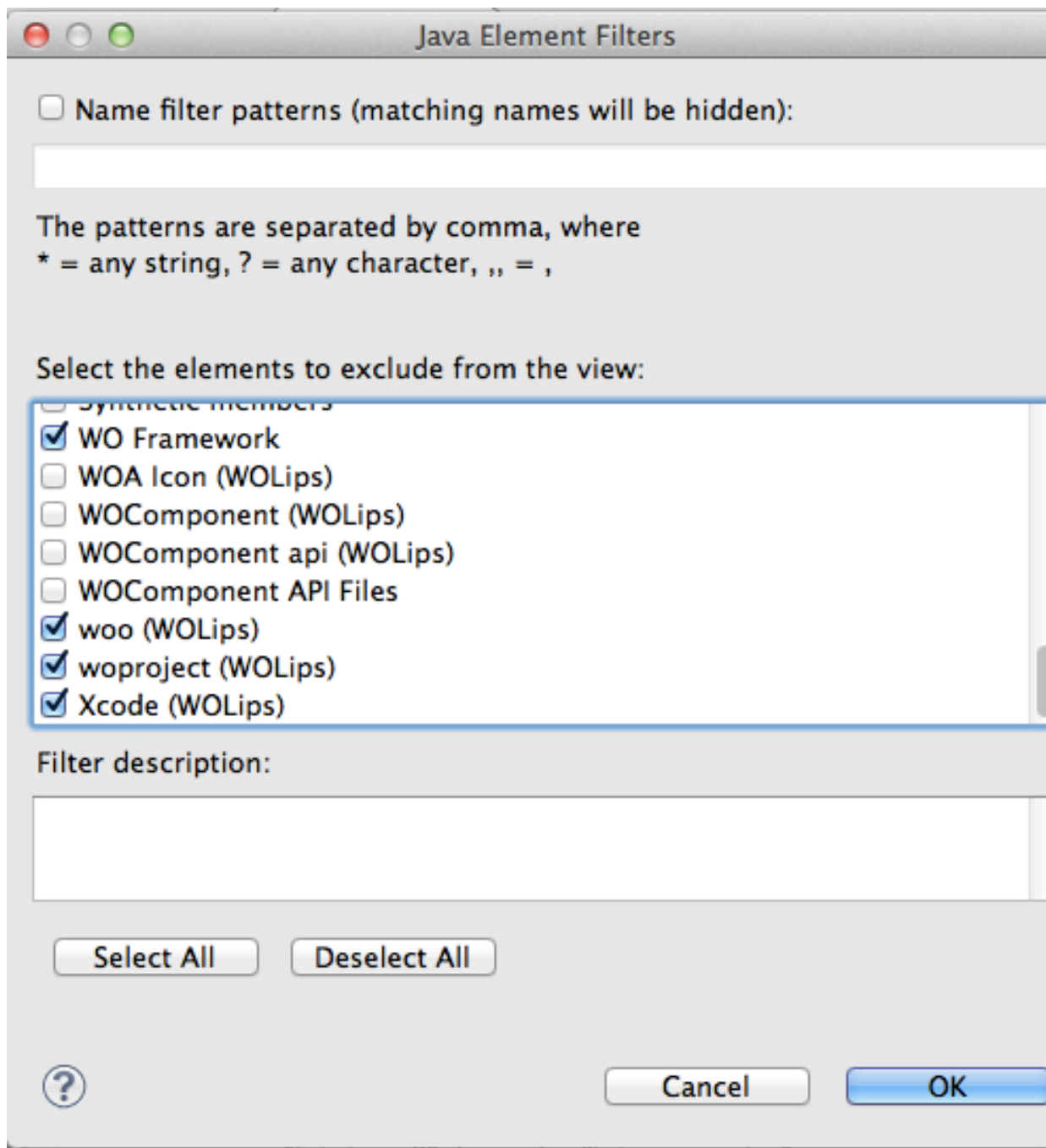


Another way to view your linked Frameworks is in WO Explorer; but then you have to make sure that these files have not been filtered out from the view. To find out which files have been filtered out, click on the small triangle on the right hand side of the WO Explorer navigation bar:



Click on Filters.

You will see which files are excluded from watching for you.



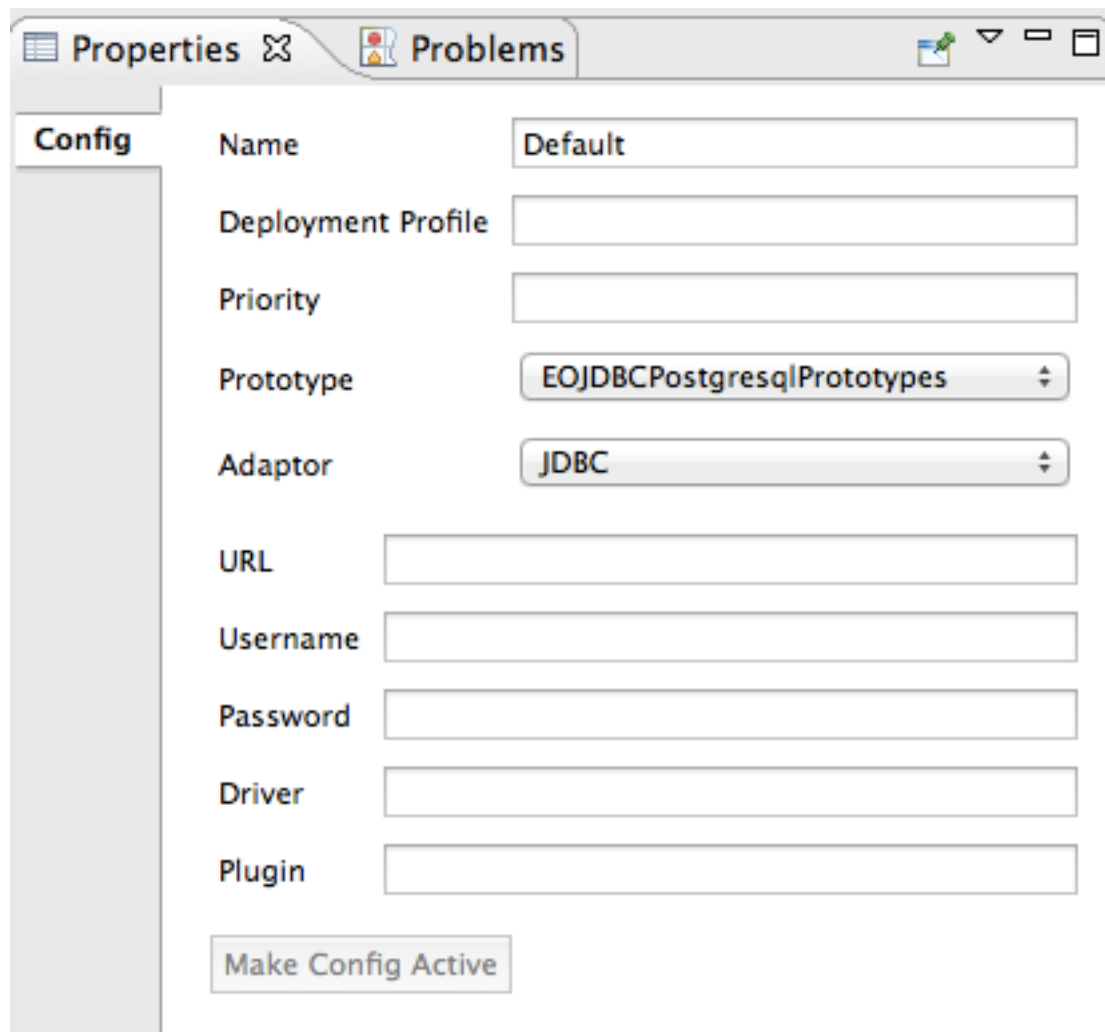
Make sure WO Frameworks is turned off. Now all the WO Frameworks will show up in your sidebar.

The database connection in EOModeler

We go back to the EOModeller perspective. Click on EOModel in the perspective chooser, or double click on the EOModel in WO Explorer; the EOModeler perspective will show up in another window.

The yellow cylinder represents a database connection. Click on it.

On the bottom of the sidebar you will see the properties of the database connection:



The screenshot shows the Eclipse IDE's Properties dialog for a database connection. The 'Config' tab is selected, and the 'Name' field is set to 'Default'. Other fields include 'Deployment Profile', 'Priority', 'Prototype' (set to 'EOJDBCPostgresqlPrototypes'), 'Adaptor' (set to 'JDBC'), 'URL', 'Username', 'Password', 'Driver', and 'Plugin'. A 'Make Config Active' button is at the bottom.

| Field | Value |
|--------------------|----------------------------|
| Name | Default |
| Deployment Profile | |
| Priority | |
| Prototype | EOJDBCPostgresqlPrototypes |
| Adaptor | JDBC |
| URL | |
| Username | |
| Password | |
| Driver | |
| Plugin | |

Make Config Active

What are the database connection fields in EOModeler?

Name: The name of the database connection. You can have several database connections at the same time in your EOModel, and these can be chosen by double clicking on it. Leave it as Default for the moment

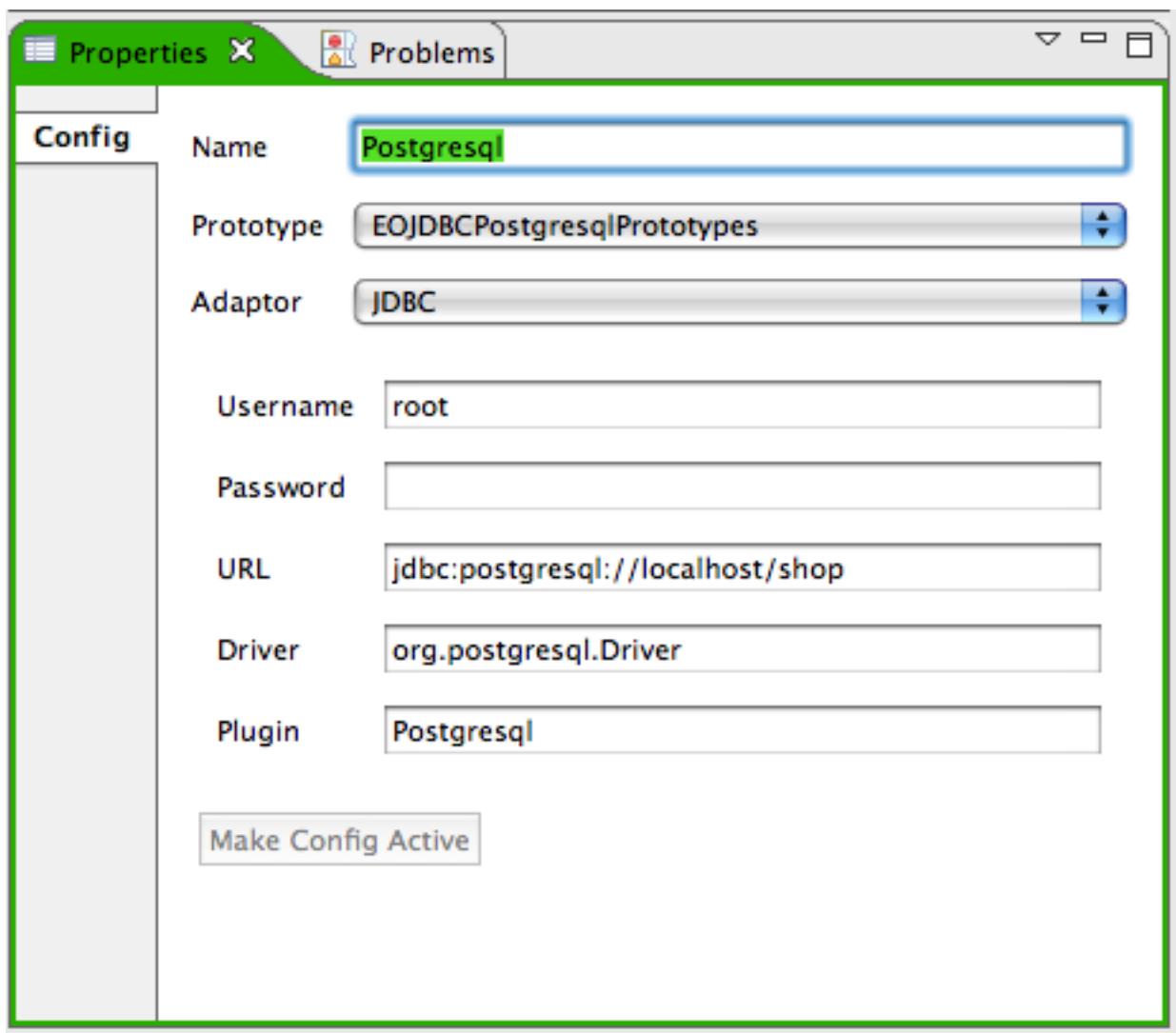
Prototype: Prototypes are abstractions of an attribute. Prototypes are created to define specific attributes that will then be translated to the datatypes that are available in a specific database. For instance, some databases will call a binary large object a blob. Other databases will call this object a bytea, or whatever. These prototypes make sure you use the appropriate translation for this type of database. Because I have chosen to use Postgresql as my database (the default available database in Lion Server), the Prototypes that are used are the EOJDBCPostgresqlPrototypes. Choose the prototype that is appropriate for your database.

- **Adaptor:** JDBC: the way the connection is made to the database. At the moment there are only two officially available: jdbc and jndi (for LDAP connections). At the 2011 WOWODC conference Travis Brit presented a EOAdapter to connect to Solr, an apache NoSQL database.
- **Username:** your username in the database that you use
- **Password:** the password that you use

Eclipse Wonder tutorial part one: setting up the data

- URL: the description JDBC expects to make a connection to the database. eg.
- Driver: Every database has its own quirks that a JDBC driver has to solve. These drivers are normally downloadable from the database's website, in the form of a jar. The Application or Framework has to be able to find this driver. A good place to store this jar file is in /Library/Java/Extensions, or in the Library folder of the project.
- Plugin: Sometimes it is useful to use a database Plugin instead of the Driver to communicate with a database in a more optimal way. So, apart from the JDBC Driver, one can develop a plugin that will translate the EOModel in a more optimal way to the database.

There are plugins for Frontbase, OpenBase, DB2, MySQL, H2, Postgresql etc. There have been rumors of a MS-SQL plugin by GVC. You can ask them politely if you can use it.



After you have filled in the information for your particular database, we can start to create the object abstractions from the tables in the database.

Eclipse Wonder tutorial part one: setting up the data

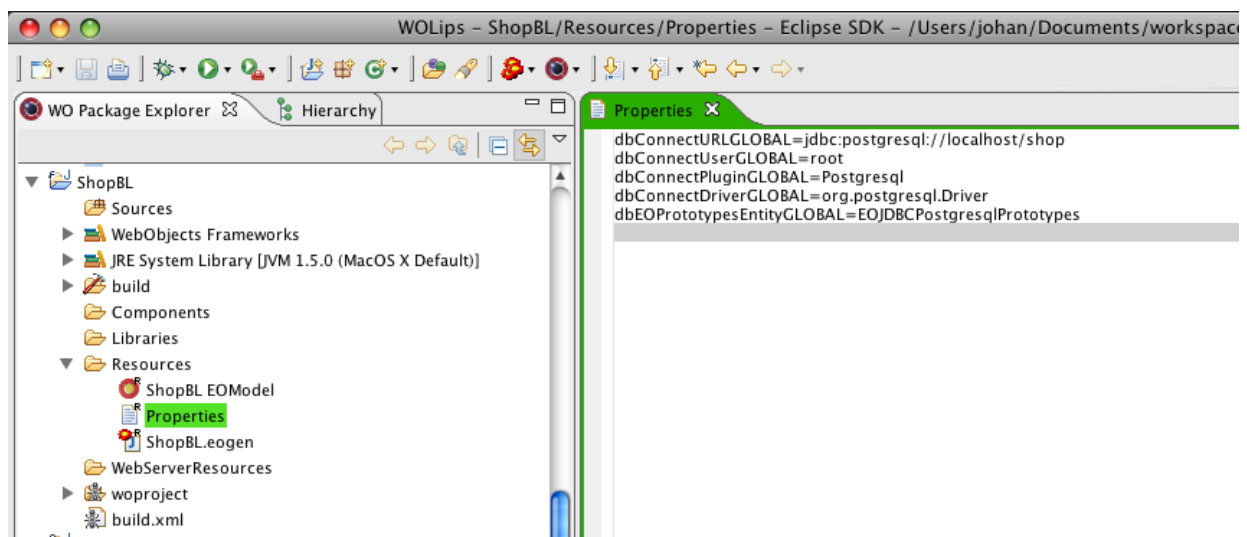
Notice that some databases can make a connection without a Username and Password, because the securitymodel relies on the IP connection from which the client tries to connect.

This is one way to make the database connection. Another way is to fill in all this information in the Properties file that you see under the Resources map.

The database connection via the Properties file

Another way to define the database connection is via the Properties file that is in the Resources folder of a Wonder project. There are some advantages of using the database connection via the Properties file. First, it is possible to have several Properties files next to one another: a Properties file per user, a Properties.dev file, etc. There is also another reason to use Properties files. There is a framework available in Wonder called ERMigrations. What it basically does, is to find the differences between two versions of the models that you use, create a java file with all the changes, and making sure that, the next time you start up, that file is used to migrate the database to reflect the settings of the EOModel. So suppose you started out with an EOModel with Contacts, Customers and Products, and you would like to add some extra Entities and Attributes to the database, then you could do this via the Migration file. We'll discuss this later, when we create the DirectToWeb Managing application.

Because you are using a bunch of Wonder Frameworks, you do not have to set the connection in the EOModel, but you can also set the connection in the Properties file:



These properties are stored in Properties file the Resources folder of your project.

The properties that you have to set, are more or less the same as you can set in your datalink:

User, Password, Driver, Plugin, Prototypes and the ConnectURL.

Here is the example for postgresql:

```
dbConnectURLGLOBAL=jdbc:postgresql://localhost/shop
```

```
dbConnectUserGLOBAL=root
```

```
dbConnectPluginGLOBAL=Postgresql
```


Eclipse Wonder tutorial part one: setting up the data

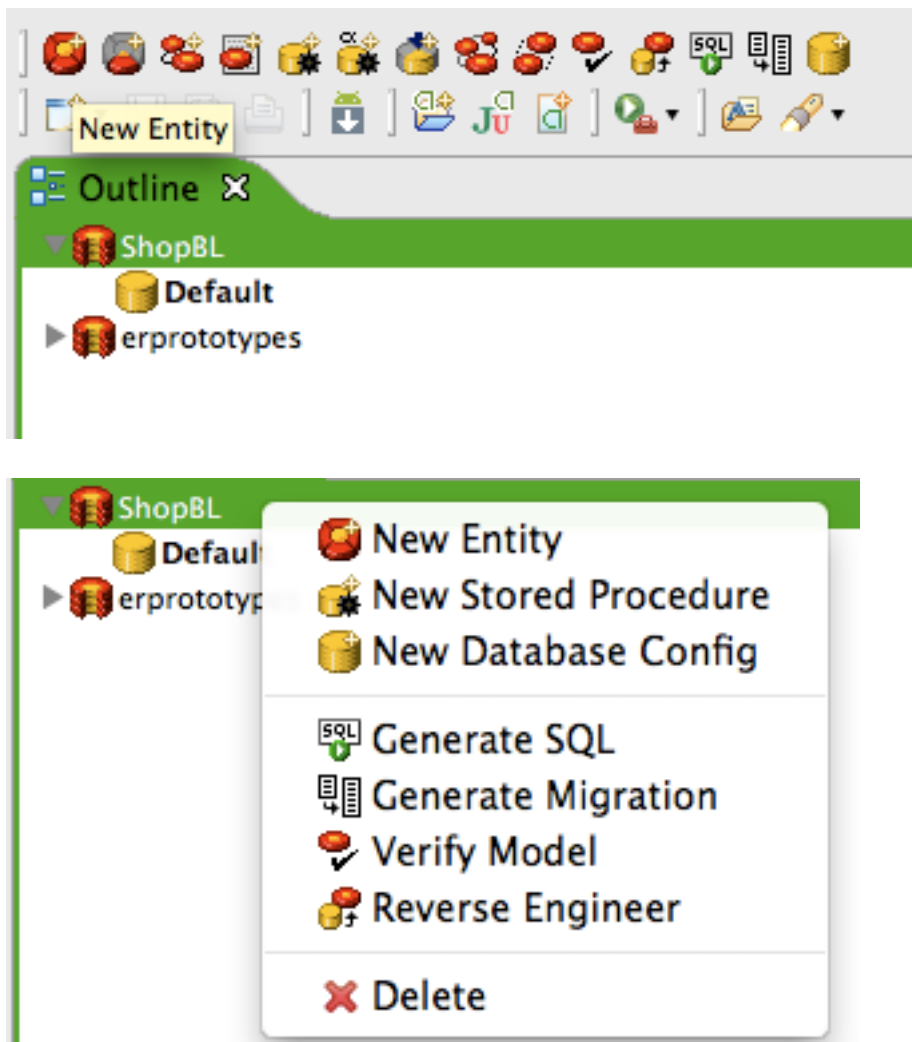
```
dbConnectDriverGLOBAL=org.postgresql.Driver  
dbEOPrototypesEntityGLOBAL=EOJDBCPostgresqlPrototypes
```

If you do not want to use the properties in the Properties file, you can comment them with a hash at the beginning of the line, as we have done with this Frontbase example:

```
#dbConnectURLGLOBAL=jdbc:FrontBase://localhost/shop  
#dbConnectUserGLOBAL=bug  
#dbEOPrototypesEntityGLOBAL=EOJDBCFrontBasePrototypes
```

Creating entities and attributes

If you look to the top of the window, you will see a list of icons. That is one way to create an entity which is the object that we will use in our data representation. The other way to create an entity is via the contextual menu.



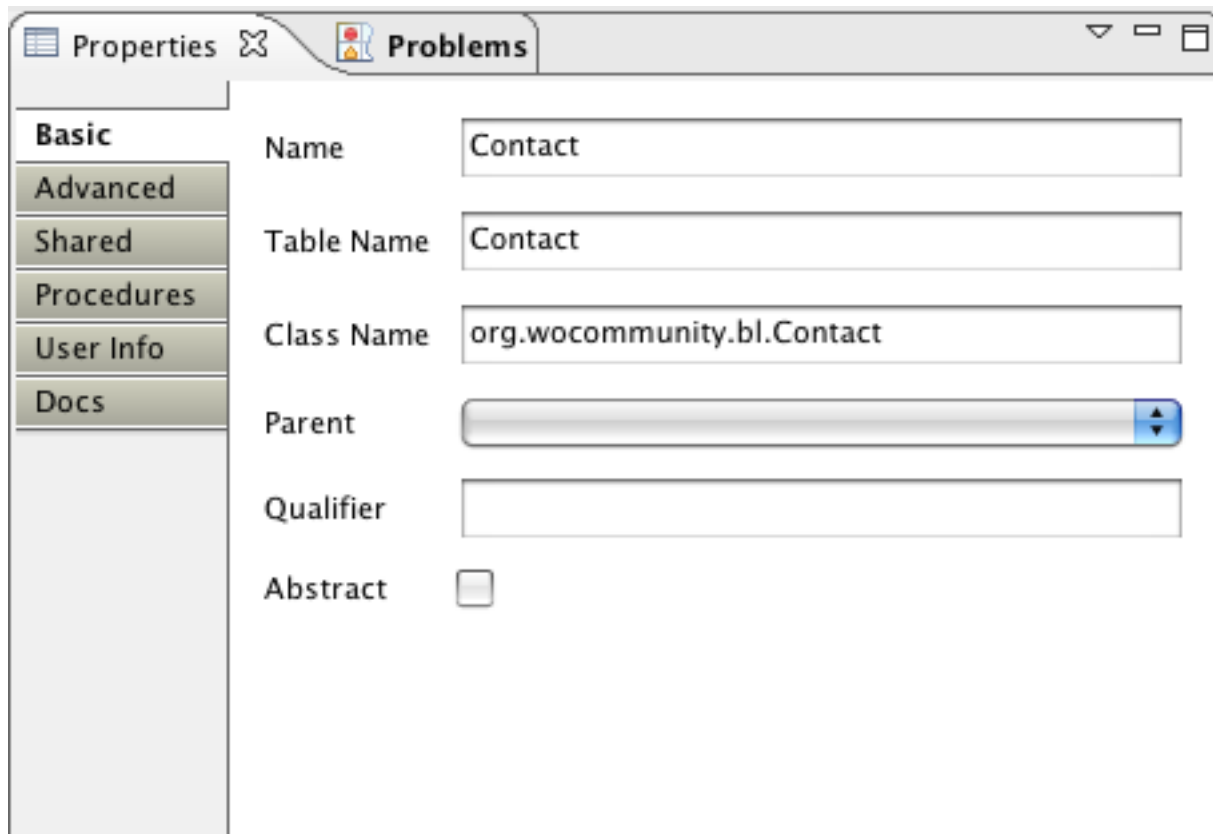
Whatever way you choose, under the Outline an entity will be created with the name NewEntity. On the bottom of the Outline, you will find the properties of this entity: the name, the

Eclipse Wonder tutorial part one: setting up the data

Table name, the Class Name, the Parent, the Qualifier and the an Abstract checkbox. change the name to Contact, for the moment. You will see that all the name (Table, Class) will change at the same time.

We will leave the name of the Table as it is, but we will change the name of the Class. It is best practice to define classes in a package, so to separate the different classes and the role they constitute. In our case, we will name the package `org.wocommunity.bl`.

The moment you create an entity, the Entity Modeler takes some of the work out of your hands: it created an id attribute for you, that would be the primary key in the database that you are connecting to.

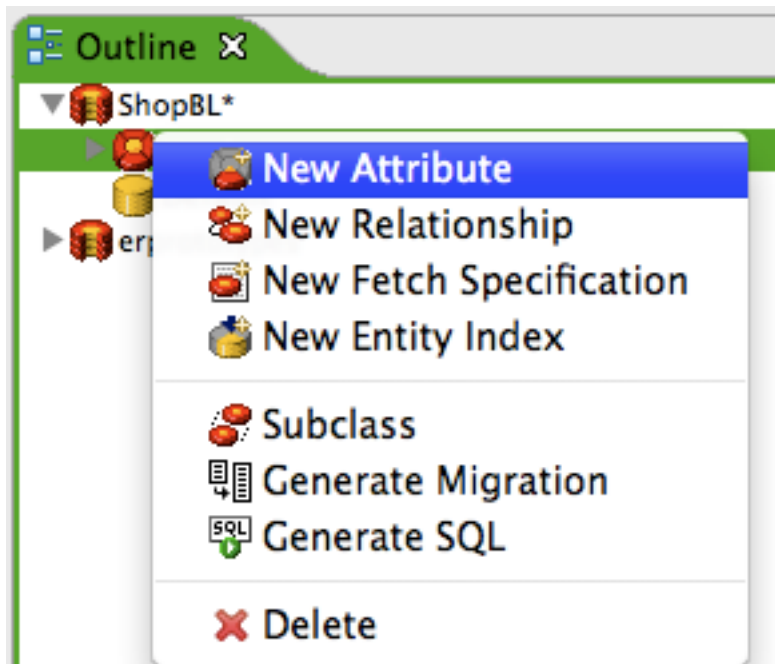


The screenshot shows the Eclipse IDE's Properties window for an entity named 'Contact'. The window has two tabs: 'Properties' and 'Problems'. The 'Properties' tab is active, and the 'Basic' section is selected in the left-hand sidebar. The 'Basic' section contains the following fields:

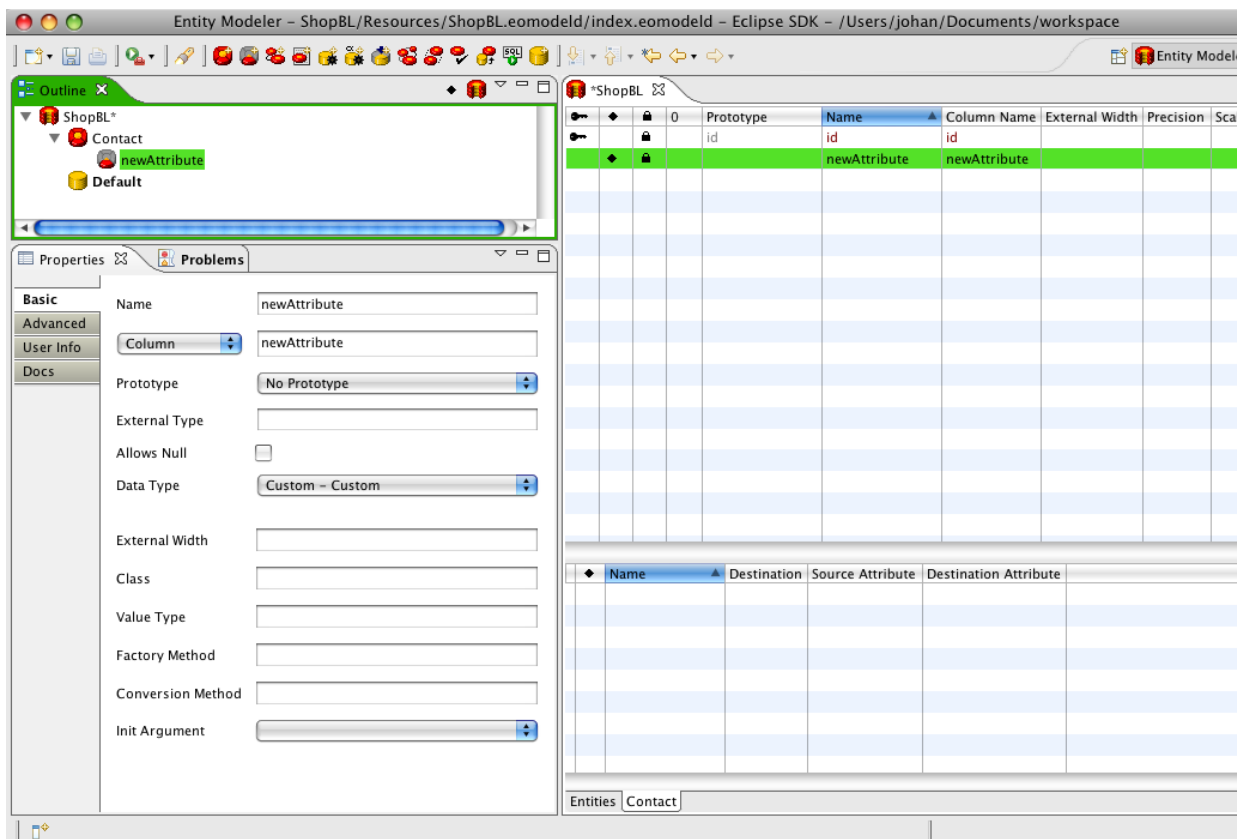
- Name:** Contact
- Table Name:** Contact
- Class Name:** org.wocommunity.bl.Contact
- Parent:** (empty dropdown menu)
- Qualifier:** (empty text field)
- Abstract:** ☐

Now we will add some other attributes to this entity. The easiest way to do this is via the contextual menu: right click on contact in the outline, and choose new attribute.

Eclipse Wonder tutorial part one: setting up the data



Perhaps you will think nothing has happened. But there is, if you open the triangle next to Contact:



As you can see, an attribute with the name newAttribute has been added. (note that entities and classes begin with Capitals, while attributes don't, but if the name of the attribute actually consists of

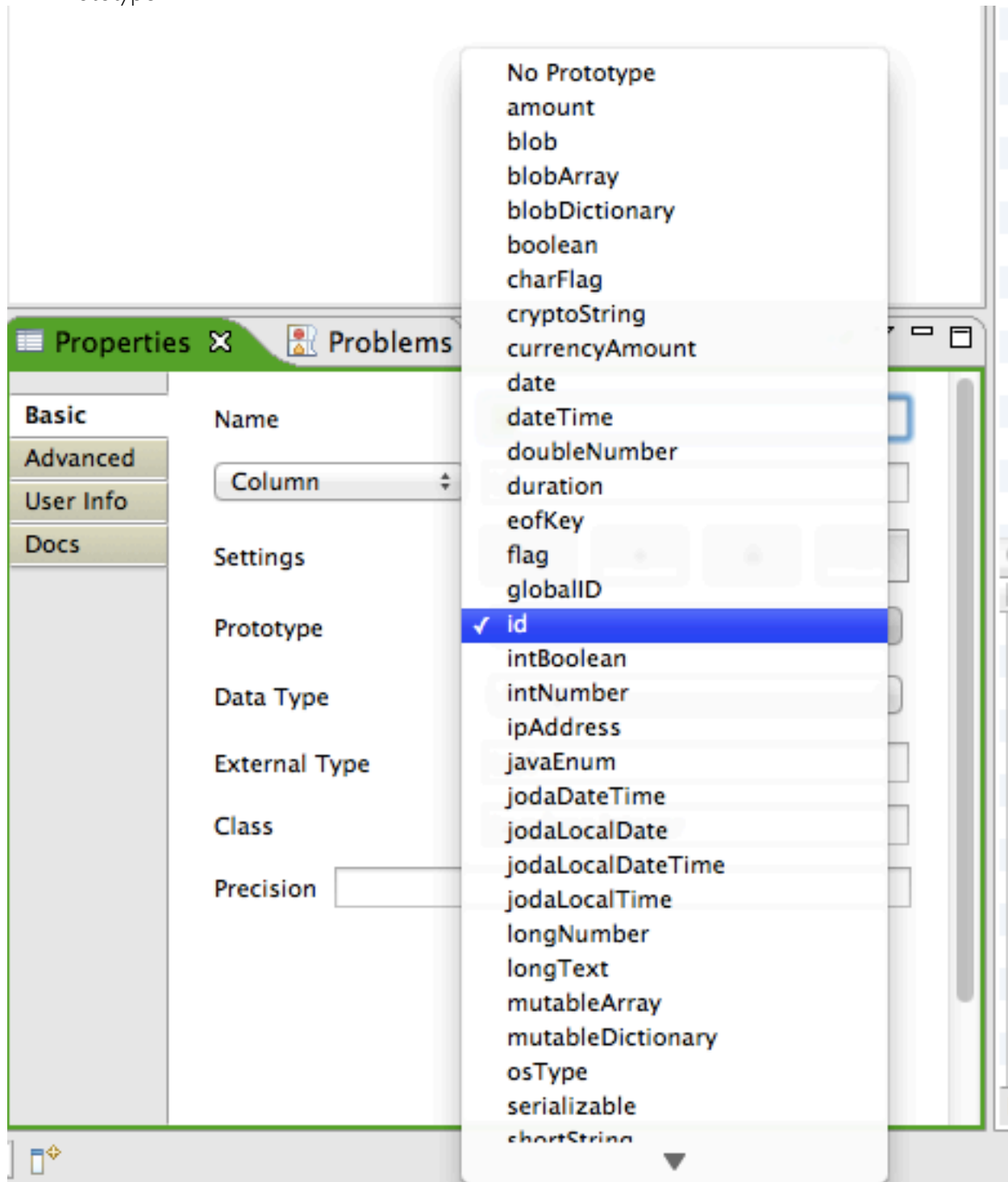
Eclipse Wonder tutorial part one: setting up the data

more than one word, the subsequent words are capitalized. My new and shiny attribute would be myNewAndShinyAttribute. This is often called CamelCase convention.)

The Basic tab of definitions of an attribute contains:

- Name
- Column
- Settings

- Prototype



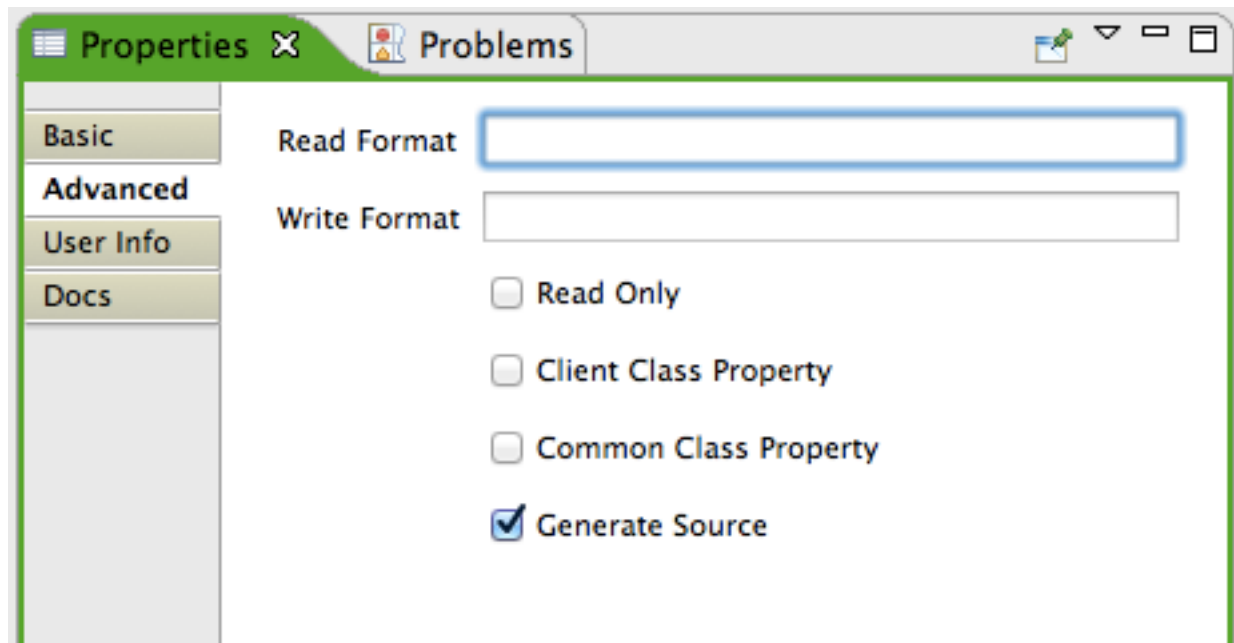
Prototypes allow you to create your own datatypes: in the screenshot above you can see which datatypes are available if you use the Wonder frameworks (no prototypes are created in the default webobjects application).

- External Type
- Data Type
- External Type

Eclipse Wonder tutorial part one: setting up the data

- Class
- Precision

The Advanced tab contains



- Read Format
- Write Format
- Read Only
- Client Class Property
- Common Class Property
- Generate Source

The User Info tab allows you to add extra Keys and Values. These are used for things like localization and auditing of attributes. More on that later.

Finally the Docs tab lets you add information concerning this attribute.

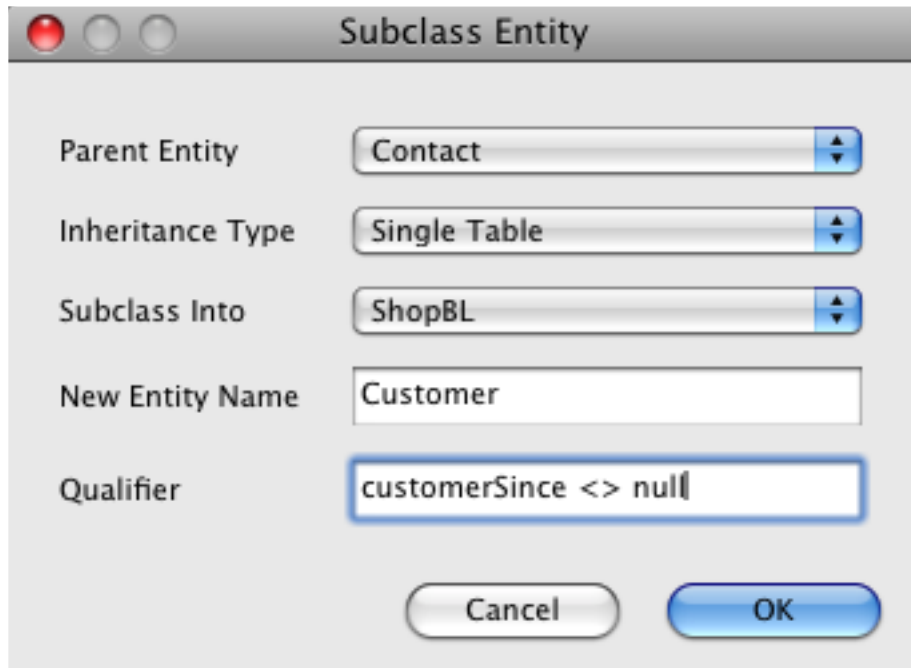
For the time being we will add only

- firstName varchar50,
- lastName varchar50
- email varchar50
- password cryptoString;

Subclassing contact to employee, customer and supplier

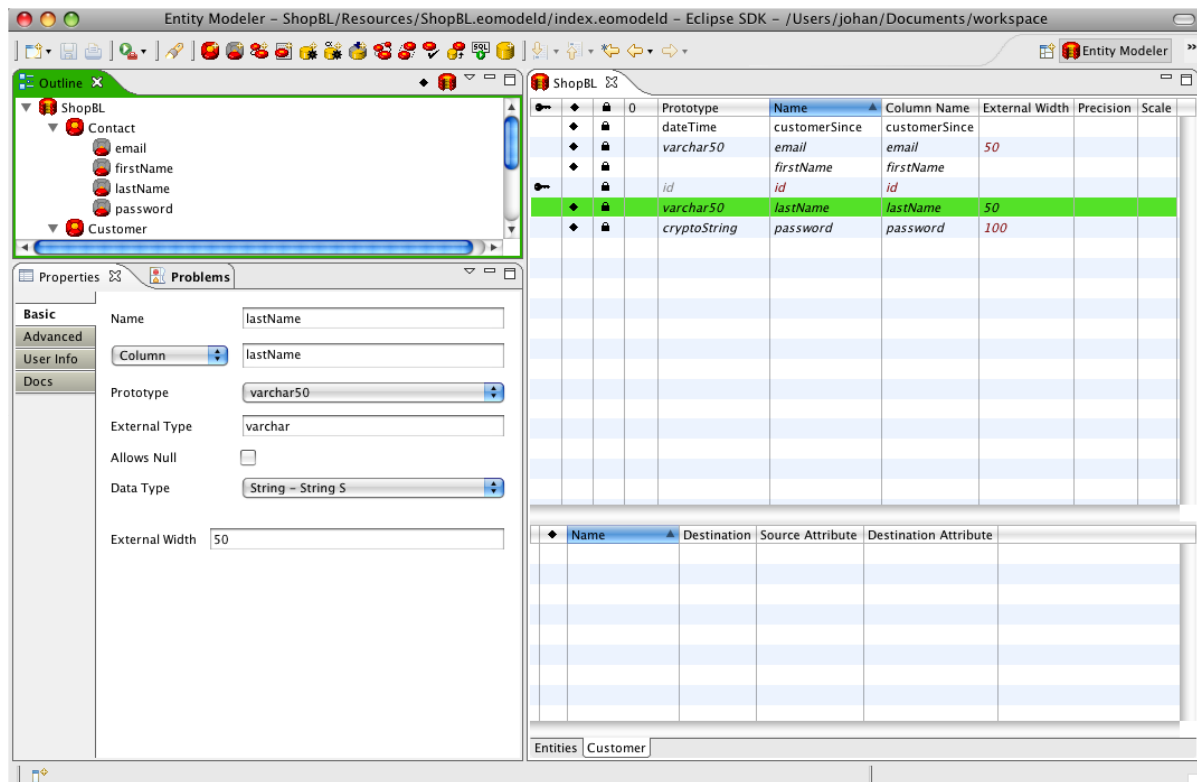
Now we will subclass Contact to create the Employee, Customer and Supplier classes. Right click on Contact, and choose SubClass.

A window will appear that will give you the chance to define the new subclass. As you can see the Parent Entity is Contact.

A screenshot of a 'Subclass Entity' dialog box. The dialog has a title bar with standard window controls (red, yellow, green buttons) and the title 'Subclass Entity'. Inside, there are five labeled fields: 'Parent Entity' with a dropdown menu showing 'Contact'; 'Inheritance Type' with a dropdown menu showing 'Single Table'; 'Subclass Into' with a dropdown menu showing 'ShopBL'; 'New Entity Name' with a text input field containing 'Customer'; and 'Qualifier' with a text input field containing 'customerSince <> null'. At the bottom, there are two buttons: 'Cancel' and 'OK'.

- The Inheritance Type defines how the information will be stored in the database,
- the Subclass Into option gives you the opportunity to define which framework with EOModels will provide the class.
- New Entity Name is the name of the entity,
- the Qualifier determines how it will know that this is a customer.

Eclipse Wonder tutorial part one: setting up the data

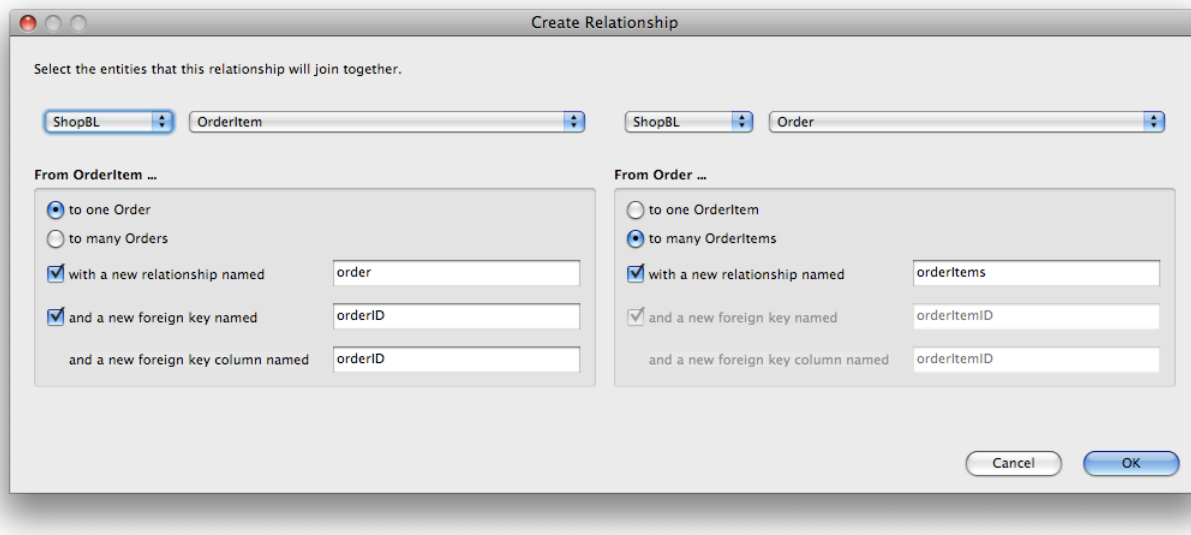


On the right side where you can see the attributes of the customer class, the italics show the attributes that the Customer class has inherited: *id*, *firstName*, *lastName*, *email* and *password*. Now we add the *customerSince* attribute to the Customer entity, and give it the Prototype *dateTime*, with Data Type *TimeStamp* and a *ServerTimezone* of *GMT*. (In the appendix, we'll talk more about the use of *Timezones*).

In the same way, subclass *Contact* for *Employee*, add just the *startAsEmployeeDate*, with a *DateTime* prototype to it and subclass *Contact* for *Supplier*, with the extra Attribute of *supplierSince*, also a *dateTime* prototype. The qualifiers for these classes will be *startAsEmployeeDate* *<> null* and *supplierSince* *<> null*.

Setting Relations

Eclipse Wonder tutorial part one: setting up the data



1:N relation order, orderitem, invoice, invoiceitem

As you probably know, relations can exist in many ways. One can have a relation to one item, or have a relation to many items. These are defined in separate ways. We will first create the entities that will have their relation firmly fixed on one item: the 1:N relations. In this case an orderitem will always be part of just one order; but an order can have many orderitems.

First we create the order: give the order a orderNumber, an orderDate, and create an orderitem, which contains the attributes quantityOrdered, pricePerProduct, and quantityPicked (which we will use in the warehouse). In the same way we can create an invoice with invoiceNumber and invoiceDate and an invoiceitem with quantityInvoiced, pricePerProduct.

Now we can create the relationship between the orderitem and the order: click on orderitem and select New Relationship from the right-click menu. A Create Relationship window will show up, that gives you all the possibilities of the different type of relationships that are possible. For the moment we will choose that an OrderItem can have a relationship to One Order, and that an Order can have many relations to OrderItems.⁴

N:M relation: product and productcategory

N:M relations have a different way of looking at one another: here, both items can have a relation to many items of the opposite side. The way that is taken care of in a relational database is by creating an in-between table that consists of the keys of the items that will form a relationship. These keys form a unique compound key.

So we create the products and product categories. Add a Product entity, with the attributes:

listPrice: currencyAmount

⁴ In a next tutorial on we will see that for the sake of speed, sometimes it is better not to define the reciprocal relationship.

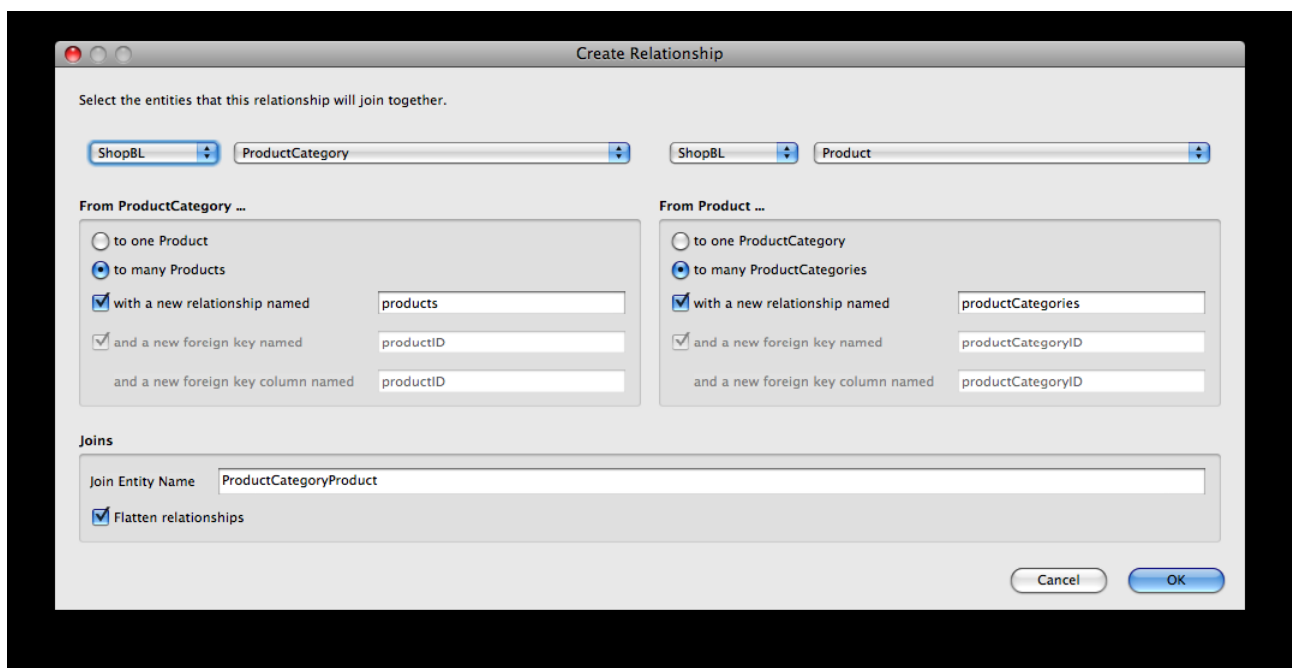
Eclipse Wonder tutorial part one: setting up the data

shortDescription varchar50

And a ProductCategory entity with the attributes:

shortDescription varchar50

Now we will create an n:m relation between these two: right-click on Product or ProductCategory, and Choose New Relationship. You can also choose both entities, and right-click: you will get the option to choose a type of relation between the entities. Otherwise you will have the option to choose the Model, and the entity in the model.



As you will see, the connecting entity will be created in ProductCategoryProduct (or ProductProductCategory depending on which table you choose as the first one) with a compound primary key productCategoryID and productID.

The entity and table definition will be created automatically, and comprise of the primary keys of both tables.

Creating the database.

We have more or less defined the data that we want to have, and now we will create the database. Notice that up until now, we have not done anything about the database. Now is the time to define which database we will use.

If you would have looked carefully, you would have noticed that, apart from the ShopBL model, there was an eoprototypes model that you could choose in your subclassing and relation setting.

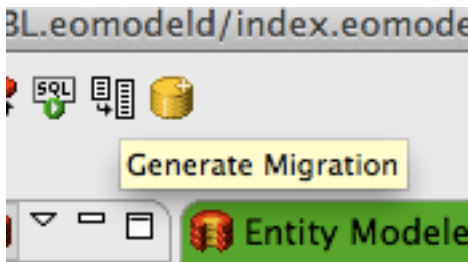
Eclipse Wonder tutorial part one: setting up the data

Now you can create the database: there are two ways to create the database:

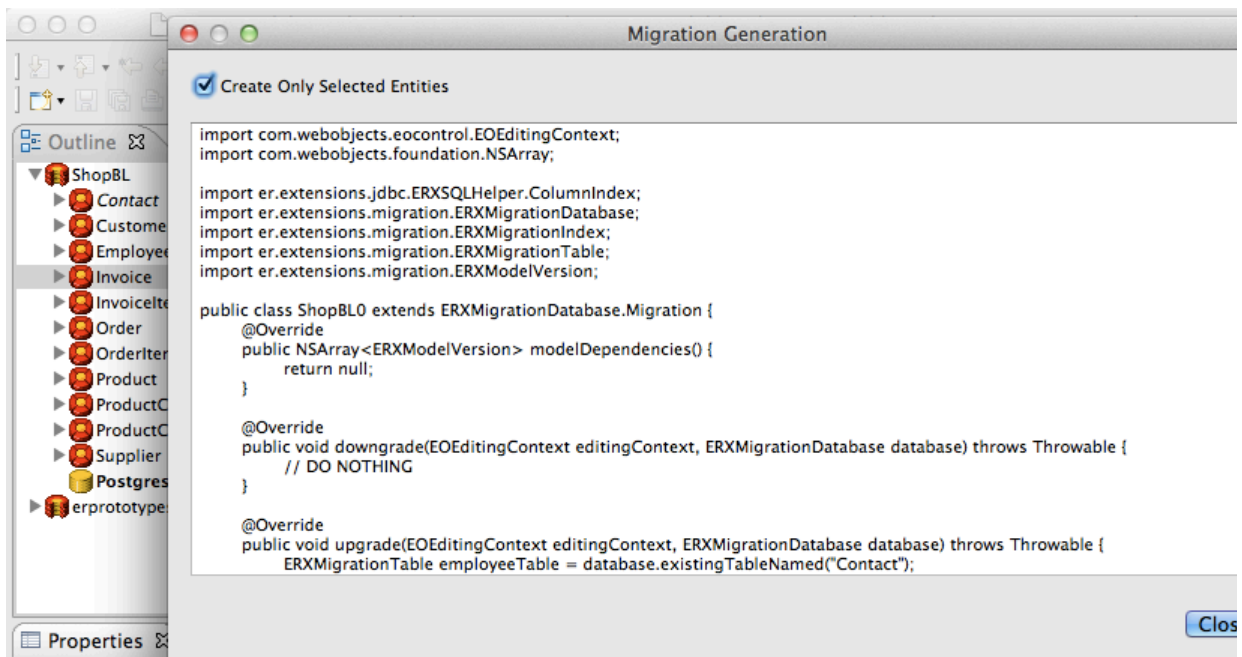
- via the button Generate SQL, or
- via ERMigrations.

Setting up ERMigrations

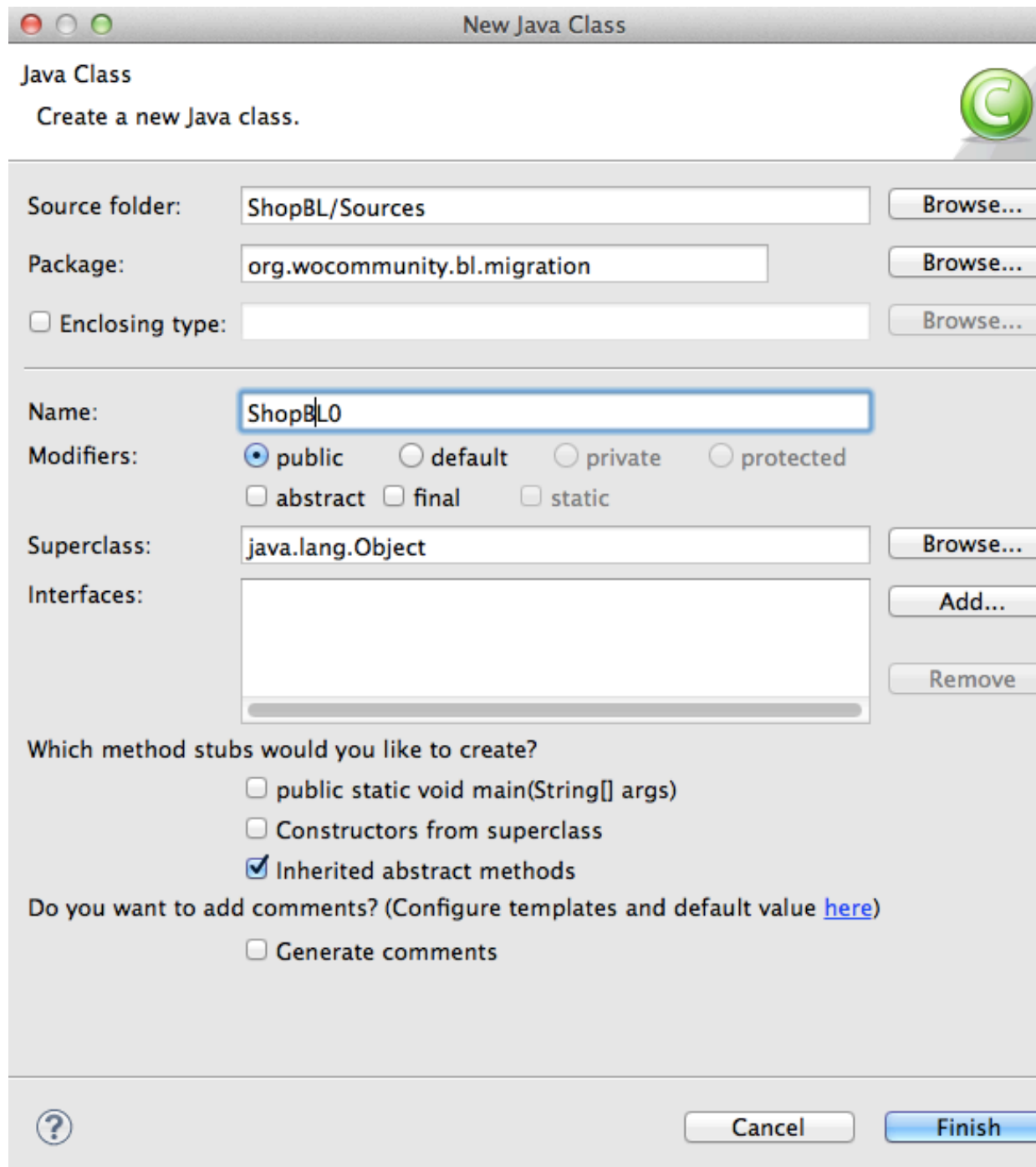
Migrations are done via EntityModeler. Next to the menuitem Generate SQL there is a button to generate a Migration java file:



If you have selected an entity in the Model, it will only create that entity, but you can select to create all entities:



If this java file has to be saved in the package location <tld>.<mainpackage>.migration, under the name of the EOModel. In our situation, the location is org.wocommunity.bl, the java file will be named ShopBL0.java:



The screenshot shows the 'New Java Class' dialog box in the Eclipse IDE. The title bar reads 'New Java Class'. Below the title bar, it says 'Java Class' and 'Create a new Java class.' with a green 'C' icon. The dialog is divided into several sections. The first section contains 'Source folder:' with the text 'ShopBL/Sources' and a 'Browse...' button; 'Package:' with the text 'org.wocommunity.bl.migration' and a 'Browse...' button; and an unchecked checkbox for 'Enclosing type:' with a 'Browse...' button. The second section contains 'Name:' with the text 'ShopBL0' in a text box; 'Modifiers:' with radio buttons for 'public' (selected), 'default', 'private', and 'protected', and checkboxes for 'abstract', 'final', and 'static'; 'Superclass:' with the text 'java.lang.Object' and a 'Browse...' button; and 'Interfaces:' with an empty text area, an 'Add...' button, and a 'Remove' button. The third section is titled 'Which method stubs would you like to create?' and contains three checkboxes: 'public static void main(String[] args)' (unchecked), 'Constructors from superclass' (unchecked), and 'Inherited abstract methods' (checked). Below this is the question 'Do you want to add comments? (Configure templates and default value [here](#))' with an unchecked checkbox for 'Generate comments'. At the bottom, there is a help icon (question mark in a circle), a 'Cancel' button, and a 'Finish' button.

New Java Class

Java Class
Create a new Java class.

Source folder: ShopBL/Sources Browse...

Package: org.wocommunity.bl.migration Browse...

☐ Enclosing type: Browse...

Name: ShopBL0

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add... Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

? Cancel Finish

Next you paste the SQL from the Migrations into the java file:

```
ShopBL0.java  ShopBL1.java
1 package org.wocommunity.bl.migration;
2
3 import com.webobjects.eocontrol.EOEditingContext;
4
5
6
7
8
9
10 public class ShopBL0 extends ERXMigrationDatabase.Migration {
11     @Override
12     public NSArray<ERXModelVersion> modelDependencies() {
13         return null;
14     }
15
16     @Override
17     public void downgrade(EOEditingContext editingContext, ERXMigrationDatabase database) throws Throwable {
18         // DO NOTHING
19     }
20
21     @Override
22     public void upgrade(EOEditingContext editingContext, ERXMigrationDatabase database) throws Throwable {
23
24         ERXMigrationTable productCategoryProductTable = database.newTableNamed("ProductCategoryProduct",
25             productCategoryProductTable.newIntegerColumn("productCategoryId", false);
26             productCategoryProductTable.newIntegerColumn("productId", false);
27             productCategoryProductTable.create();
28             productCategoryProductTable.setPrimaryKey("productCategoryId", "productId");
29
30         ERXMigrationTable productCategoryTable = database.newTableNamed("ProductCategory");
31         productCategoryTable.newIntegerColumn("id", false);
32         productCategoryTable.newStringColumn("shortDescription", 50, false);
33         productCategoryTable.create();
34         productCategoryTable.setPrimaryKey("id");
35     }
36 }
```

So how does this stuff start creating tables etc in the database?

That depends on the settings of some properties in an application Property file:

```
# Migrations
er.migration.migrateAtStartup=true
er.migration.createTablesIfNecessary=true
er.migration.modelNames=ShopBL
```

Depending on the modelnames that are available here, starting the application will start creating the necessary tables and foreign key constraints, indexes and sequences. If you add more modelnames over here (later we will add ERCoreBusinessLogic and ERAttachment for instance), you can also add them over here:

```
er.migration.modelNames=ERCoreBusinessLogic,ERAttachment,ShopBL
```

and the necessary database adjustments will be made from these EOModels too.

Note that these Migrations will only run if you have the database connection properties set in your Property file, and have set the er.migration properties.

Another thing to keep in mind is that EOModeler can not add just one attribute to a table, by default. It will completely try to recreate an entity, so it is useful if you have added an extra entity. You can however massage ERMigrations to make an extra attribute:

first create the attribute in EOModeler; then create the ERMigration file for this entity, which normally would be something like this:

```
public void upgrade(EOEditingContext editingContext, ERXMigrationDatabase
database) throws Throwable {
```

```
ERXMigrationTable productTable = database.newTableNamed("Product");
productTable.newIntegerColumn("id", false);
productTable.newBigDecimalColumn("listPrice", 38, 2, false);
productTable.newStringColumn("shortDescription", 50, false);
productTable.create();
productTable.setPrimaryKey("id");

}
```

Now replace the `newTableNamed` method with `existingTableNamed`, and only leave the attribute that has to be added:

```
public void upgrade(EOEditingContext editingContext, ERXMigrationDatabase
database) throws Throwable {
    ERXMigrationTable productTable =
database.existingTableNamed("Product");
    productTable.newBigDecimalColumn("listPrice", 38, 2, false);
}
```

Next time the application is started, this column will be added.

Another trick is that you can use migrations to populate your database with some values. Suppose you must have some users and roles in the database, to start up and to be able to login (after which you immediately change the default password of course!)

Here is an example from David Holts Simple Blog database:

```
public void upgrade(EOEditingContext editingContext, ERXMigrationDatabase database) throws
Throwable {

    ERXJDBCUtilities.executeUpdate(database.adaptorChannel(), "INSERT INTO Role (roleDescription,id)
VALUES ('Admin',1)", true);
    ERXJDBCUtilities.executeUpdate(database.adaptorChannel(), "INSERT INTO Role (roleDescription,id)
VALUES ('Normal',2)", true);

    ERXJDBCUtilities.executeUpdate(database.adaptorChannel(), "insert into Person (firstName, lastName,
password, login, id , email) values ( 'Admin', 'User', 'test', 'test',1,'dummy@test.com')", true);
    ERXJDBCUtilities.executeUpdate(database.adaptorChannel(), "insert into Person (firstName, lastName,
password, login, id , email) values ( 'David', 'Holt', 'david', 'david',2,'dummy@test.com')", true);
    ERXJDBCUtilities.executeUpdate(database.adaptorChannel(), "insert into Person (firstName, lastName,
password, login, id , email) values ( 'Pascal', 'Robert', 'pascal', 'pascal',3,'dummy@test.com')", true);
    ERXJDBCUtilities.executeUpdate(database.adaptorChannel(), "insert into Person (firstName, lastName,
password, login, id , email) values ( 'Public', 'User', 'safds987007gdsfg', 'system',4,'dummy@test.com')",
true);
    ERXJDBCUtilities.executeUpdate(database.adaptorChannel(), "insert into XPersonRole (personId, roleId)
values ( 1, 1)", true);
    ERXJDBCUtilities.executeUpdate(database.adaptorChannel(), "insert into XPersonRole (personId, roleId)
values ( 2, 2)", true);
    ERXJDBCUtilities.executeUpdate(database.adaptorChannel(), "insert into XPersonRole (personId, roleId)
values ( 3, 2)", true);

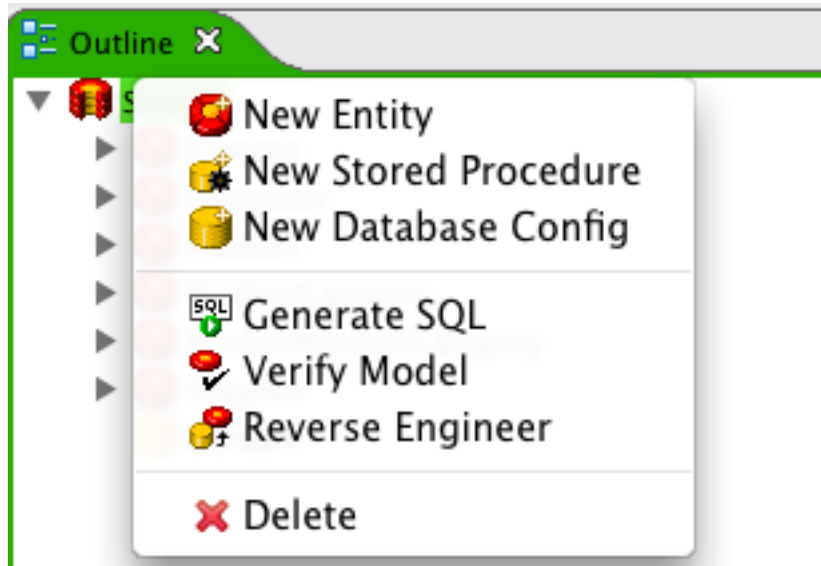
}
```

Creating via Generate SQL

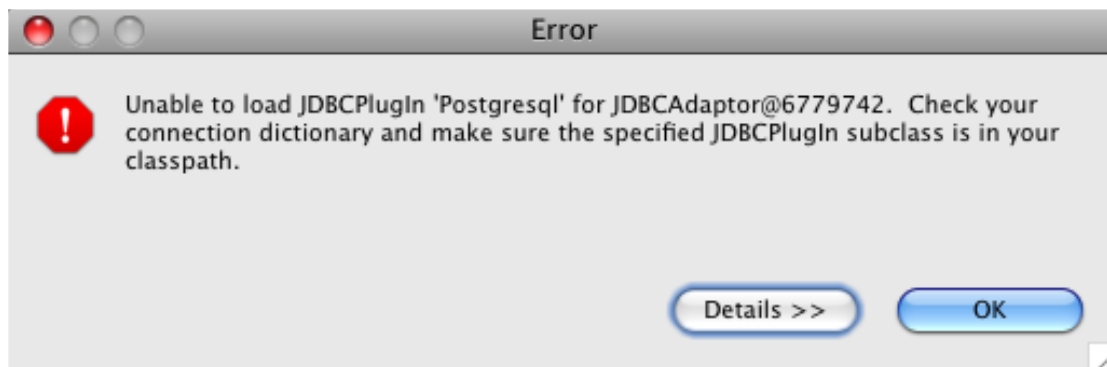
Generate via SQL supposes that you have setup the connection to your database via the Database Config in EOModel. If you use the Properties file to define your database connection, it will not work.

Also, it assumes that the database already exists, so make sure that one is up and running.

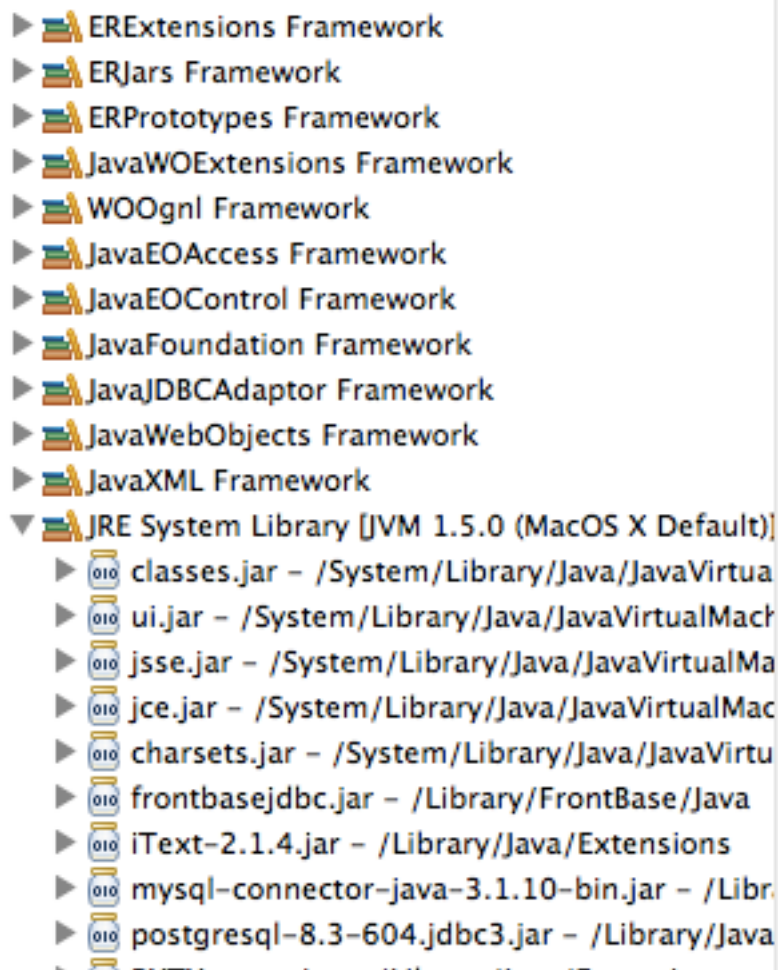
Click on the model and click on Generate SQL.



If everything is fine, a connection is made to the database and the tables and everything related is created. But this is not always the case:

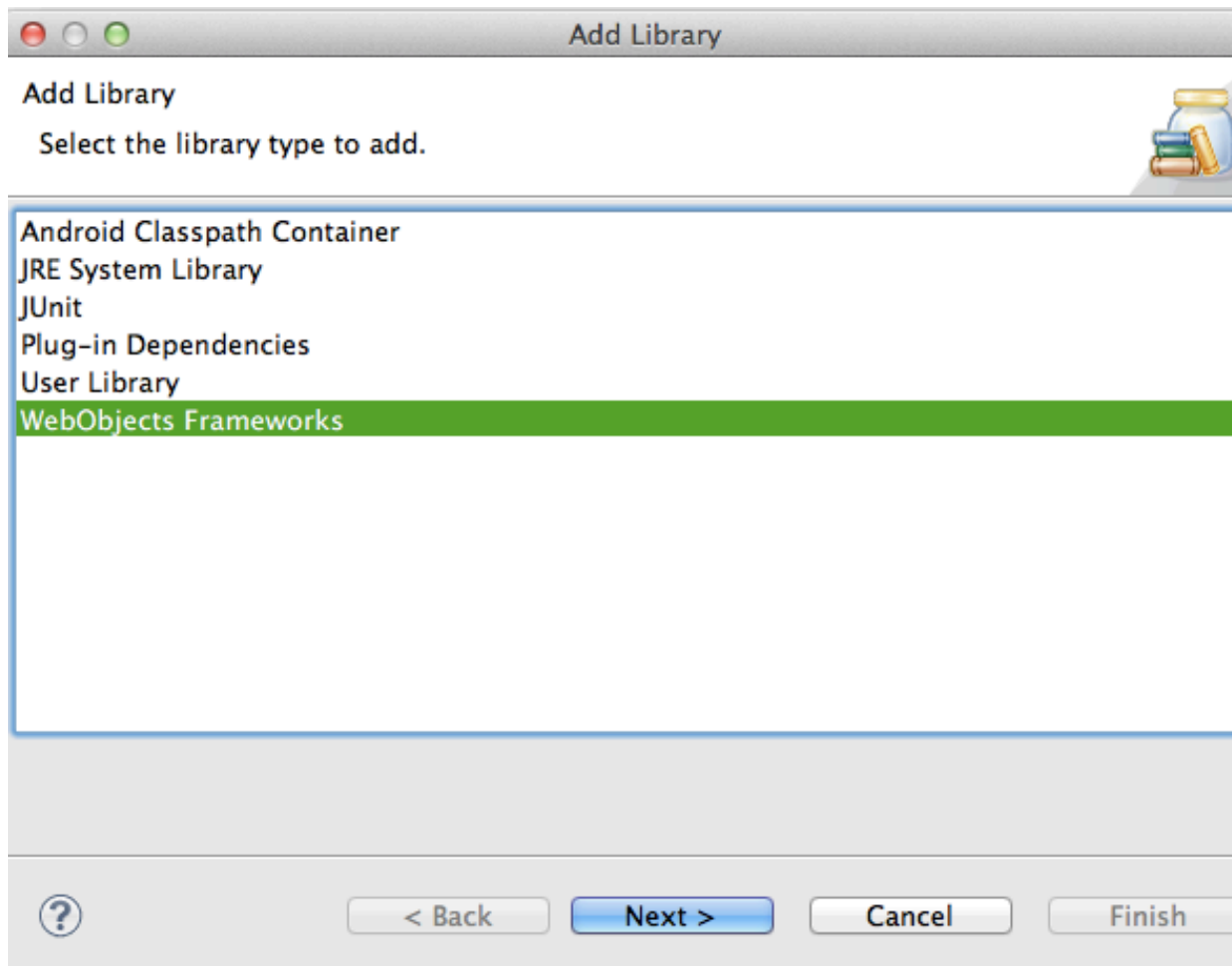


In this case, the plugin that we have defined could not be found in the classpath that Eclipse uses to resolve dependencies. The reason of that can be found if you would look at the WebObjects Framework in your build path.

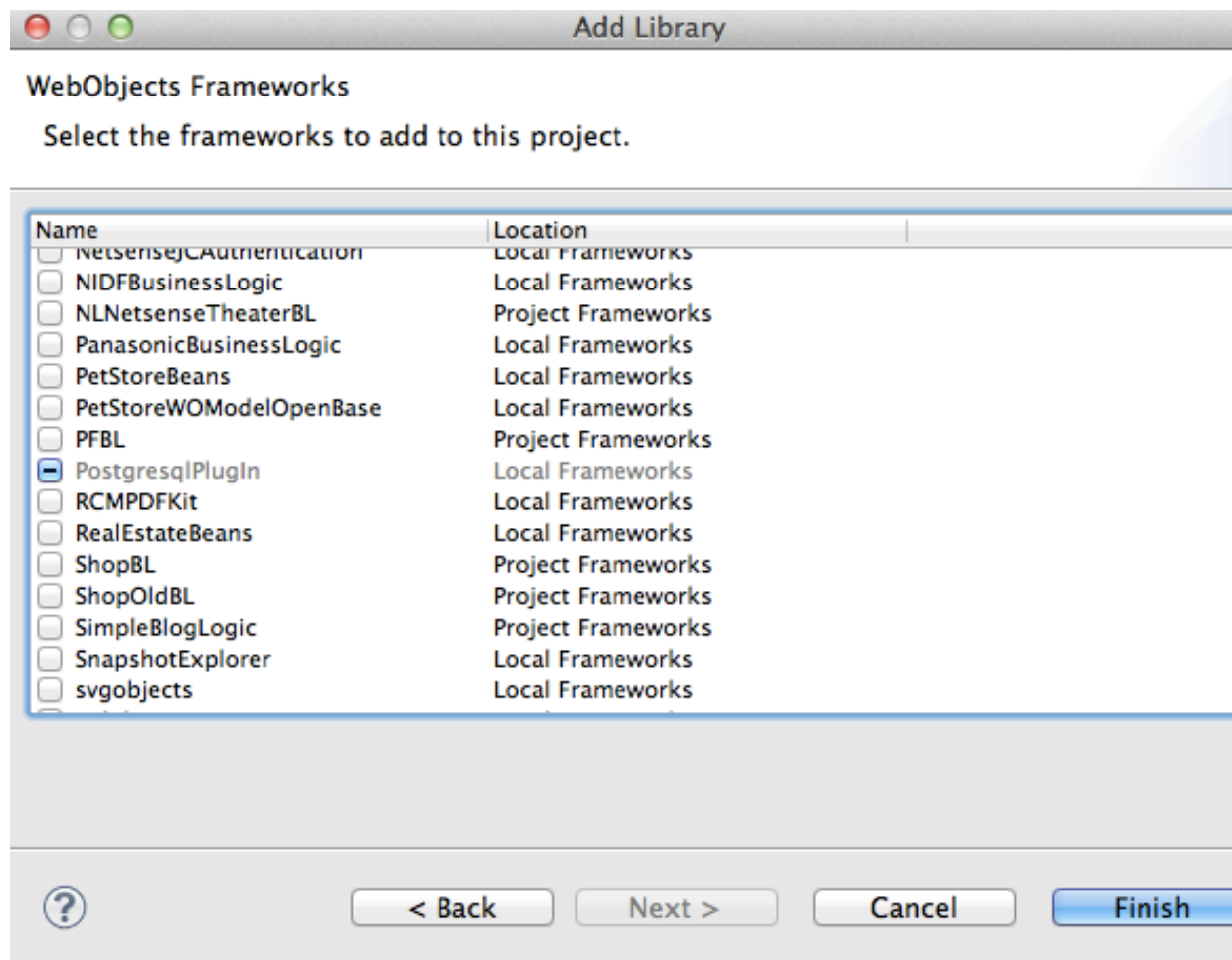


If you look carefully you will see that there is the postgresql.jar that makes the database connection, but not the PostgreSQLPlugin Framework.⁵ To add this, right click in WO Explorer on Build path>Add Libraries, and you will be confronted with the possibility to edit the libraries which need to be included:

⁵ If you do not see the WebObjects Frameworks in your project, go to the filtersettings and make sure you have not excluded the WO Frameworks



If you click open WebObjects Frameworks, you will notice (if you have installed Project Wonder completely) that there is a PostgreSQLPlugin framework. Add this one:



Creating the database

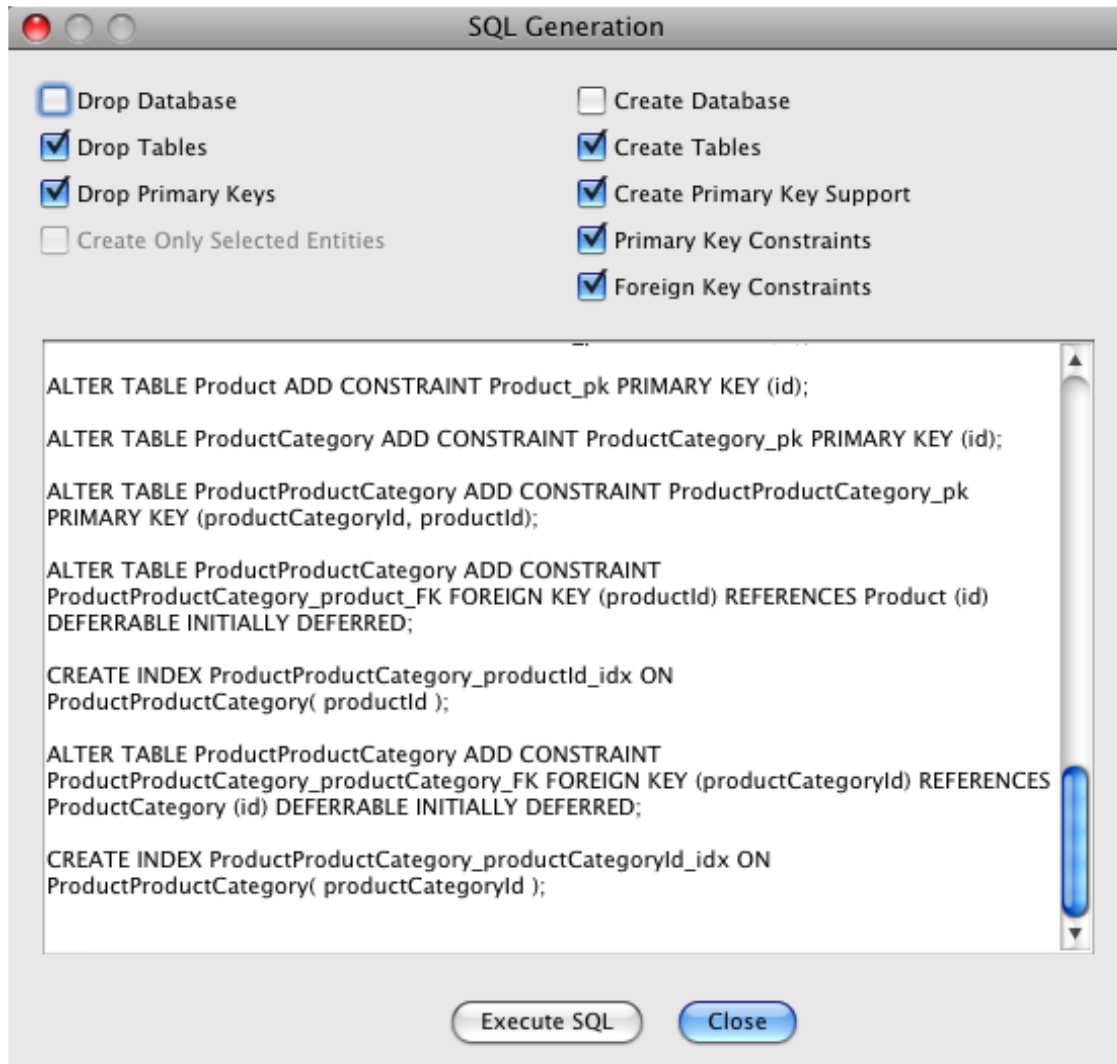
Now you can go back to the EntityModeler, and right click on Generate SQL. But still this may not happen. Although the interface that supplies the SQL provides you with a checkbox to create the database, it will not always succeed, and you have to create the database beforehand. In my case, I had to create the postgresql database. This will also be necessary for Frontbase. The postgresql command is:

```
sh-3.2# /opt/local/lib/postgresql90/bin/createdb shop -E=UTF8
```

which will give the message:

```
CREATE DATABASE
```

Notice I am using UTF8 as encoding: that ensures that I can handle every language available on the planet, while not getting the complete overload of 16 bits per character that UTF16 gives. Note that, if you have to do something with languages that use 16 bits characters, your storage increase doubles. So the lastName field which accommodates 30 ASCII characters can only store 15 Japanese characters.



The exact sql that is generated depends on the plugin: that is the 'translator' of the object model to the database. If there is no plugin you will notice that the sql that will be generated is of a more generic kind. The SQL-generator then does not take any chances, and creates something of which it is sure will work with almost any database.⁶

If everything went fine, you'll have a working database now, set up with some tables and relations between them. You can test if everything worked out fine with your local database management application (I use postgresql from MacPorts on the Mac, which comes with the very appropriate command line tool psql90). We will not do anything with the database for the moment, because we are still focusing on the modeling of our data.

Setting up the java classes for our EOModel: EOGeneration

Now we will build the java classes that have been defined while creating the Object-Relational model (that's another word for the EOModel). We can do that in a number of ways. With the old tools, there used to be a button in the Modeler application, that said: create java classes. The program

⁶ There used to be a EO_PK table that was one of the things that was created if the database did not support sequences. In this table WebObjects would store the primary keys of the various tables. The problem was that other programs than webobjects programs would not know about this convention, so only webobjects applications could change the information in the database. That is why one of the function of the plugin is to get rid of this way of creating primary keys.

Eclipse Wonder tutorial part one: setting up the data

would then create a java class based upon the definitions in EOModel. If you had edited this class already, say you had added some extra methods, the program would find out and give you the opportunity to replace, cancel or merge the new class with the existing class. That was a lot of hassle, caused by the fact that you had changed the java class.

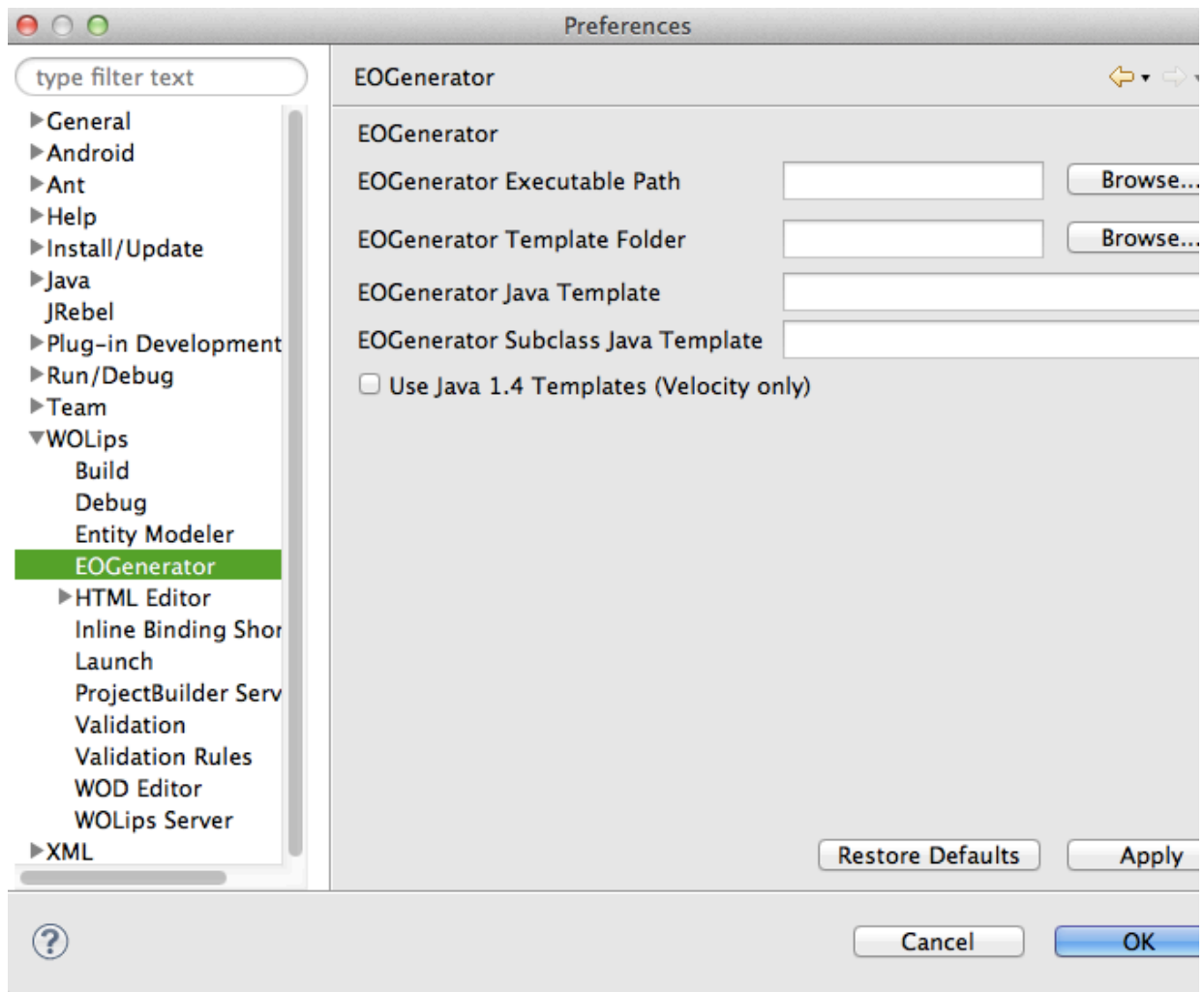
So the people from Rubicode came up with a clever solution (a lot of clever people are part of the Wonder/WebObjects community), that made it possible to generate the defined java classes from the EOModel, but at the same time extend these classes in separate files. The trick is to add your own logic in the extended classes. Now, if the data model changes, the generated java classes will be overwritten, but not the extended java class. No more worries about accidentally overwriting your carefully crafted code: which will be in the class extension. (You do use a versioning system, don't you? And you do make backups of the machine with the versioning system, don't you?).

EOGenerator relied on frameworks that were included in the WO install by Apple prior to Leopard. When the default WO installation post-MacOS X 10.4 removed those private Apple frameworks, EOGenerator no longer worked.. So two groups came up with a solution that would make it possible to continue to build java classes from your EOModel without overwriting your code. One was from the Apple WebObjects group, called JavaEOGenerator. The first installment of that program did not work completely well (it's fixed now). Another one came forward from the hardworking developers of MDimension, in the shape of Mike Schrag. He used a built-in Velocity template Engine from Eclipse to create java classes from EOModels. We'll use that one for the moment.

The java class generation depends on templates: one for the EOModel created class, and one for the java class extension. You can define these templates by yourself (a lot of experienced developers tweak their templates because they have a specific way of working). There are default templates in the WOLips configuration. If you want to see more, have a look at <http://wiki.objectstyle.org/confluence/display/WOL/EOGenerator+Templates+and+Additions>.

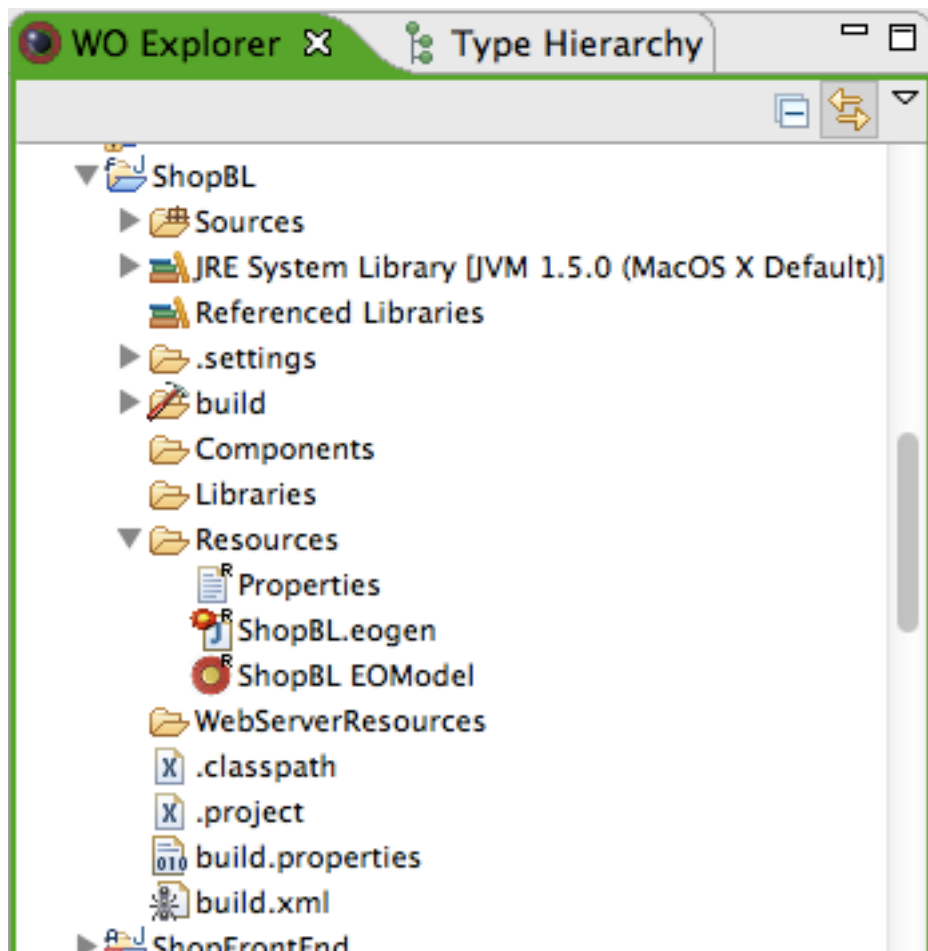
Normally, you do not have to define anything to get the default settings, but if you want to adapt them, have a look at Eclipse > Preferences > WOLips > EOGenerator. You can fill in the path to the Template folder, and which templates to use.

Eclipse Wonder tutorial part one: setting up the data



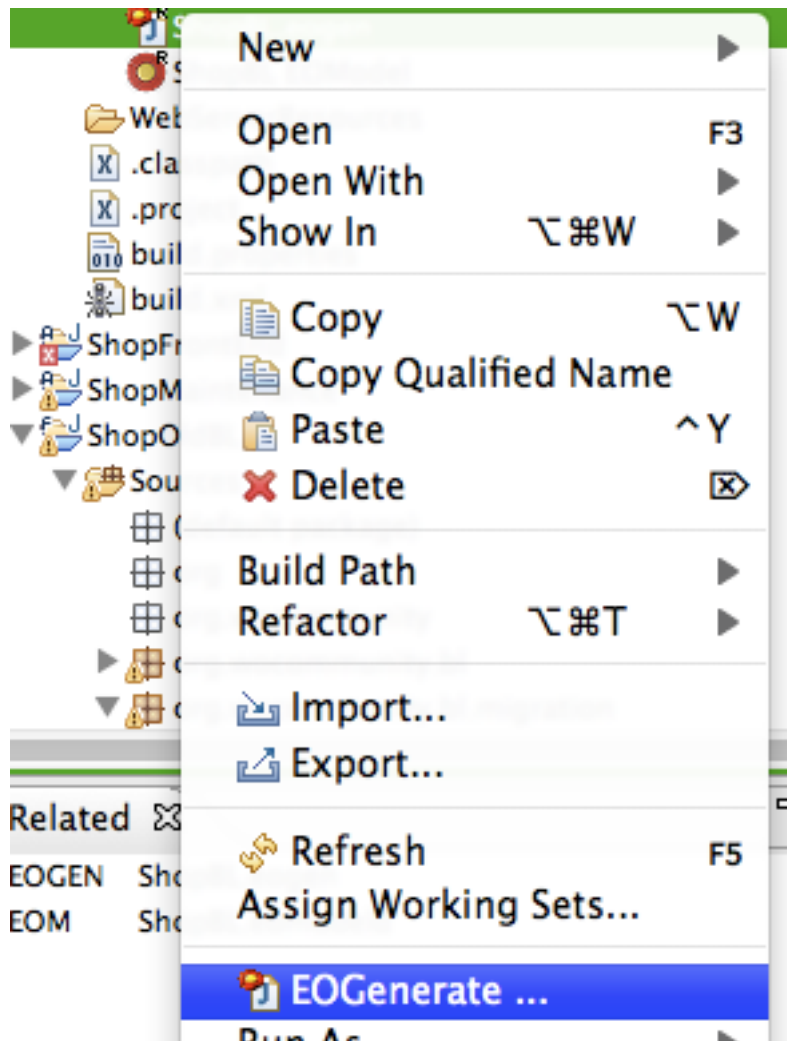
Now we are ready to start creating java classes.

Go to the Resources folder in the ShopBL framework, and have a look above the EOModel we created:



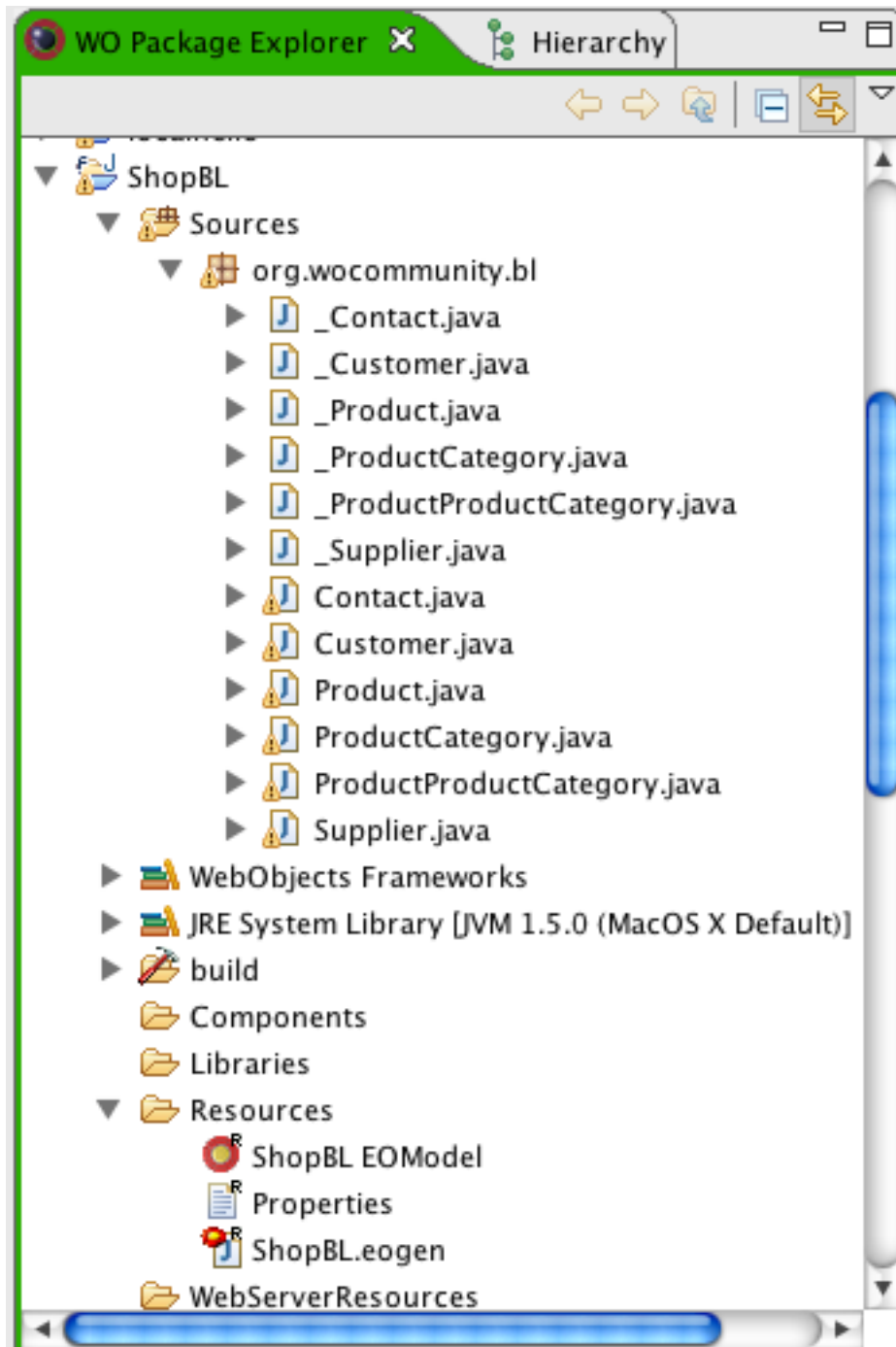
The ShopBL.eogen file is a file created by default when you created the Framework and said you wanted to use EOGenerator.

Now, to actually create the Java files, right click on the ShopBL.eogen file, and click on EOGenerate....

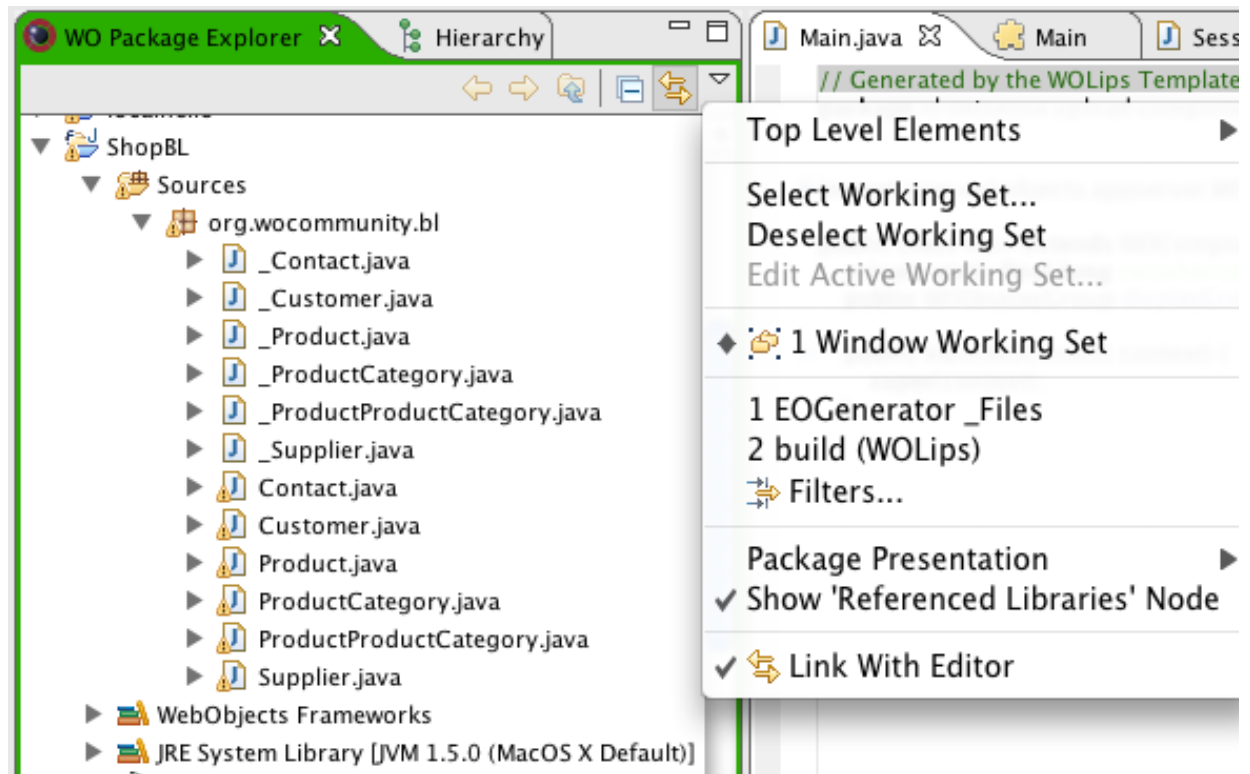


Under WOLips tools, there will a menu option to create a EOGenerator file. Select it.

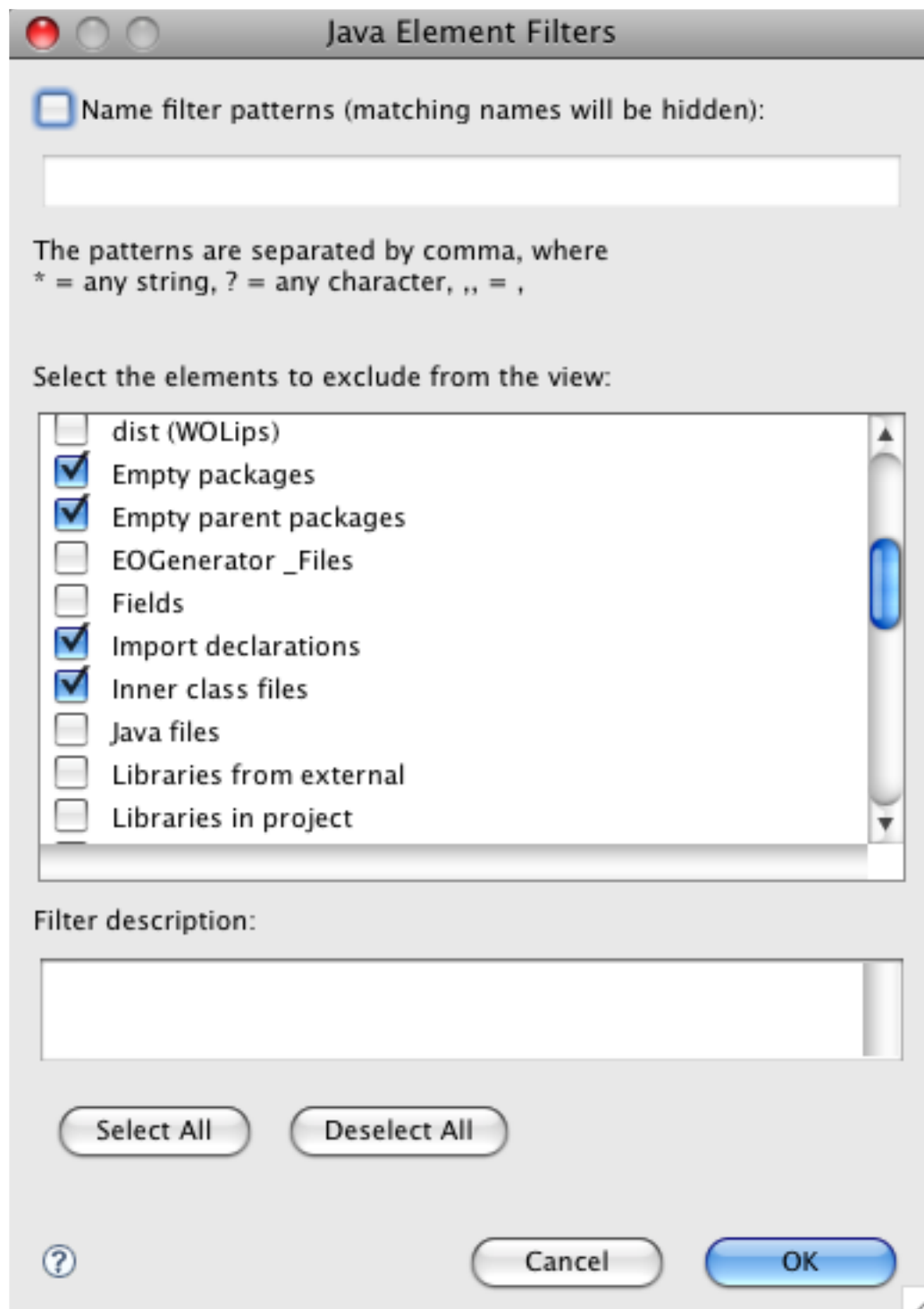
Now the magic will happen, and the java classes will be generated in Sources/
org.wocommunity.bl.



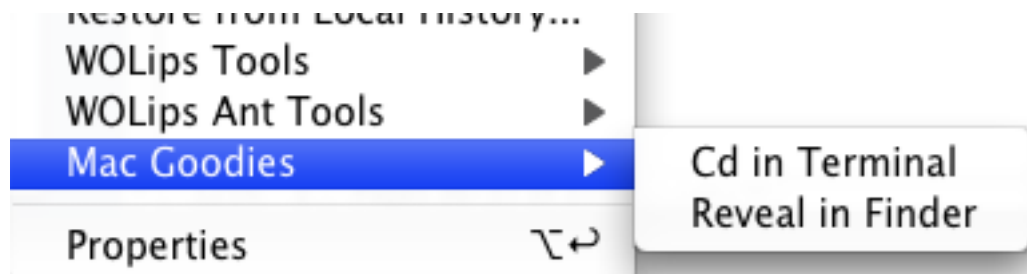
If you do not see these files, there can be several reasons, the most obvious being that the Eclipse Filter Settings are not displaying them. The small Triangle on top of the sidebar will give you the opportunity to change the filter settings.



As you can see, there is a setting for EOGenerator _Files. If this checkbox is turned on, then the files will not show up in the Package Explorer:

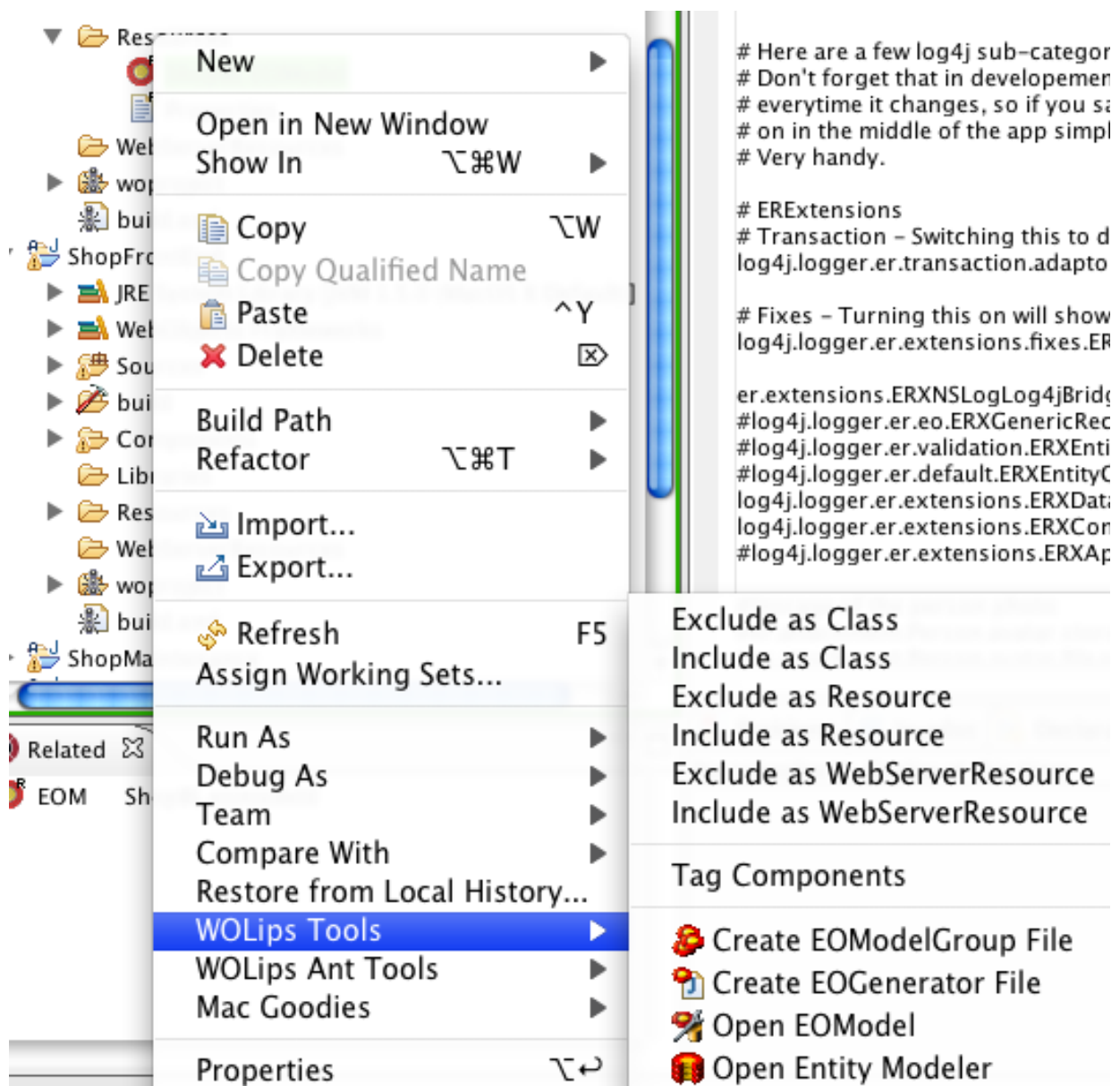


You can always look at your project in the finder or the terminal, to see what is really there (mind you, the Finder also has ways of hiding files, so your best option would be to use the Terminal if you really want to be sure you are seeing everything). WOLips even has a shortcut to look at your project or folders inside Eclipse: right click while you are in the Package Explorer. In the contextual menu you will find Mac Goodies, that give you the option to look at your currently selected file or folder in the Finder or in the Terminal



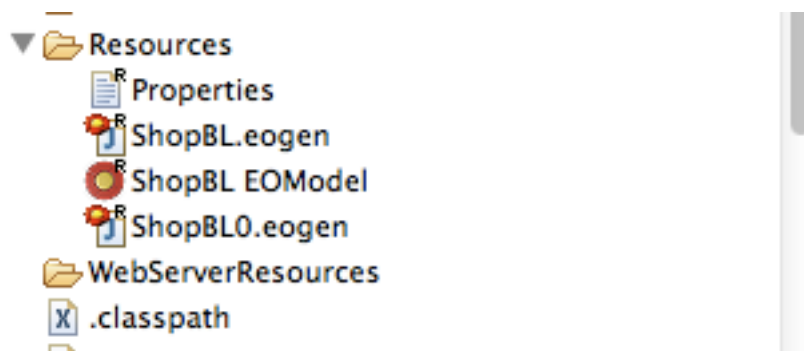
Creating your own EOGenerator file

Sometime you want to change the settings in the EOGenerator file. The way to do that is to right click on the EOModel and select Create EOGenerator file.



Eclipse Wonder tutorial part one: setting up the data

As you already had a ShopBL.eogen file under Resources, it will make another one:



Adding some business logic: default values.

Now that we have our java classes, and separated them into two files (one that will represent the state of the EOModel and one that will contain your own enhancements) we will try to add some business logic. Some of the stuff that people will always complain about if you're building an information system is that they have to fill in things which are obvious.

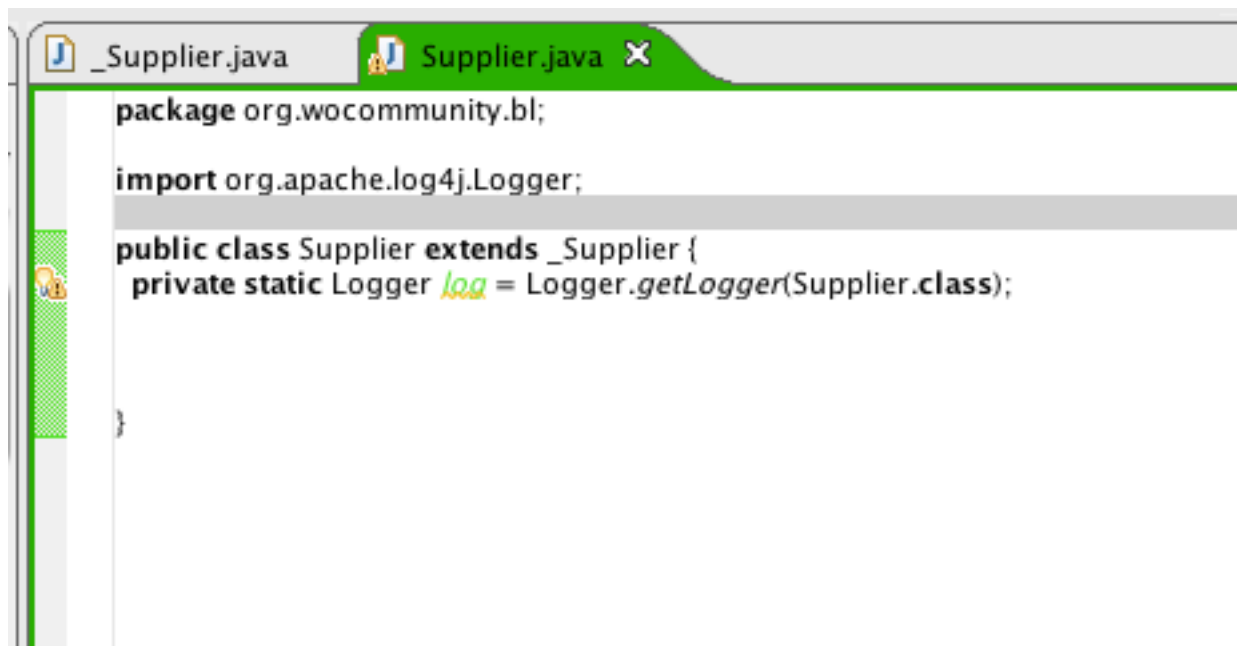
'I am adding a new customer, so why do I have to fill in the date when he becomes a customer? It is pretty obvious that date is today, so why should I have to fill this in? Can't this thing work out what day it is today?'

'99 % of our suppliers are US (France, Germany, Canada) based. Why do I have to choose the country every time again and again when I am adding a supplier? Why is there not a short list of countries, with our country as default, and the rest under it?'

This is something you can solve in your business logic. You can pre-cook some of the information that gets into the database.

As an example we will add a default date and time to the date that a supplier has started to be our supplier. The way to do that is to add the method `awakeFromInsertion` to our class definition. To do that, go to `Supplier.java` in the `Sources/org.wocommunity.bl` folder in WO Package Explorer, and double-click on it. Twice. Rapidly.

The first thing you will notice, that there is not very much in it:

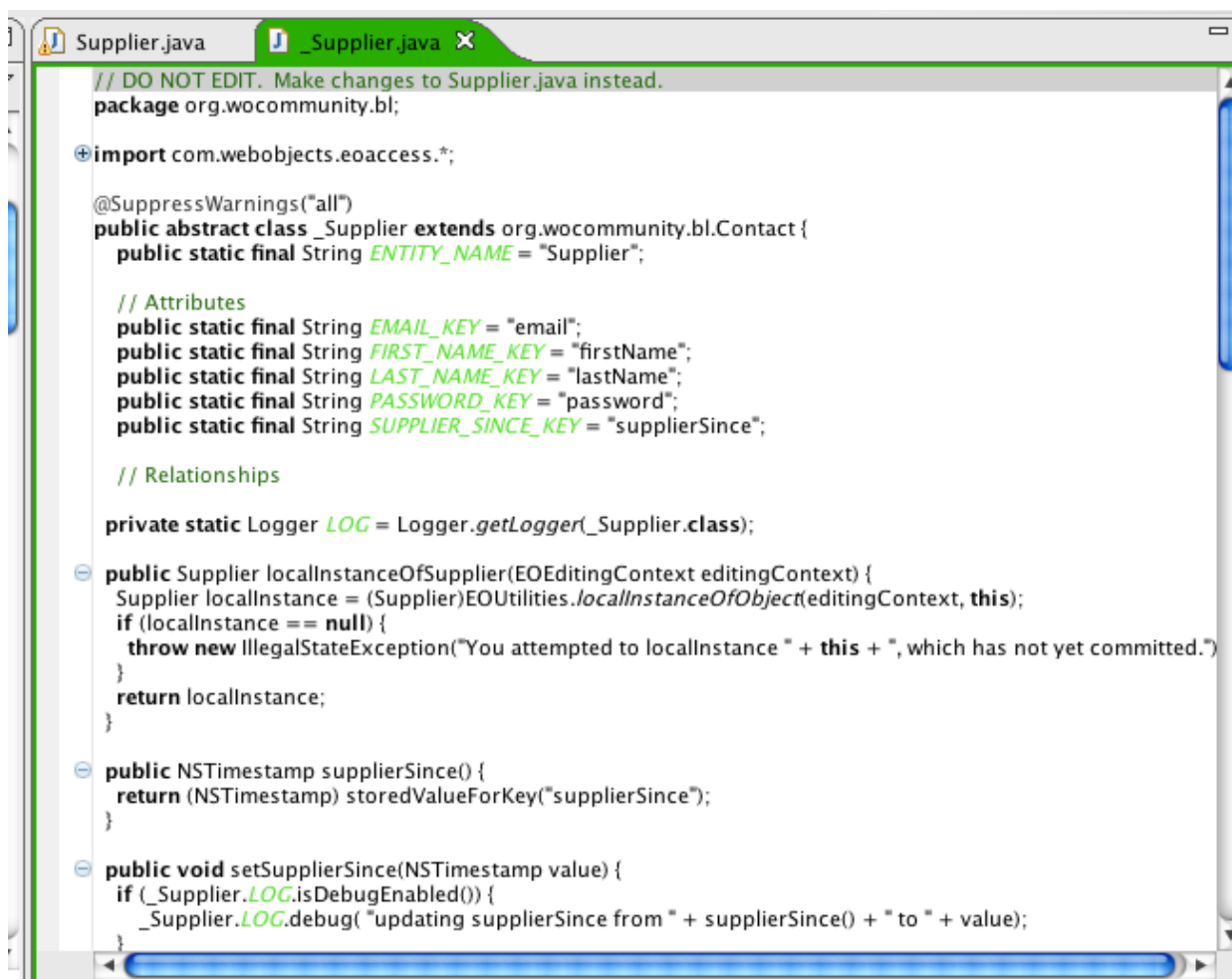


```
package org.wocommunity.bl;

import org.apache.log4j.Logger;

public class Supplier extends _Supplier {
    private static Logger log = Logger.getLogger(Supplier.class);
}
```

The _Supplier.java class, however is filled to the brim:



```
// DO NOT EDIT. Make changes to Supplier.java instead.
package org.wocommunity.bl;

import com.webobjects.eoaccess.*;

@SuppressWarnings("all")
public abstract class _Supplier extends org.wocommunity.bl.Contact {
    public static final String ENTITY_NAME = "Supplier";

    // Attributes
    public static final String EMAIL_KEY = "email";
    public static final String FIRST_NAME_KEY = "firstName";
    public static final String LAST_NAME_KEY = "lastName";
    public static final String PASSWORD_KEY = "password";
    public static final String SUPPLIER_SINCE_KEY = "supplierSince";

    // Relationships

    private static Logger LOG = Logger.getLogger(_Supplier.class);

    public Supplier localInstanceOfSupplier(EOEditingContext editingContext) {
        Supplier localInstance = (Supplier)EOUtilities.localInstanceOfObject(editingContext, this);
        if (localInstance == null) {
            throw new IllegalStateException("You attempted to localInstance " + this + ", which has not yet committed.")
        }
        return localInstance;
    }

    public NSTimestamp supplierSince() {
        return (NSTimestamp) storedValueForKey("supplierSince");
    }

    public void setSupplierSince(NSTimestamp value) {
        if (_Supplier.LOG.isDebugEnabled()) {
            _Supplier.LOG.debug("updating supplierSince from " + supplierSince() + " to " + value);
        }
    }
}
```

You can have a look at the _Supplier.java class later, for the moment we are interested in the extension of that class, Supplier.java. That is where we will put our stuff.

Eclipse Wonder tutorial part one: setting up the data

As I have already said, we will do something with `awakeFromInsertion`. How do we know what the arguments are, and why do we use that? Where did we get the idea to do that? The first place to look if you get such a semi-automated construction of your classes would be at the java doc information of the classes that have been defined. If you look carefully, you'd see that `Supplier` extends `_Supplier`, `_Supplier` extends `Contact`, `Contact` extends `_Contact`, and finally, in `_Contact` we discover that this is built up from :

public abstract class `_Contact` extends `EOGenericRecord` {

So if we want to know what `Supplier` is capable of, we could have a look at `EOGenericRecord`.

You can find documentation on <http://webobjects.mdimension.com/javadoc/WebObjects/5.4/>. Searching for `EOGenericRecord` will show you a whole lot of methods that have been inherited from `EOCustomObject`. So if you want to know what our class is capable of, this is one of the places to look. You will find `awakeFromInsertion` there.

WebObjects 5.4
[All Classes](#)

Packages
[com.webobjects.appserver](#)
[com.webobjects.appserver.i](#)
[com.webobjects.appserver.r](#)
[com.webobjects.appserver.r](#)
[com.webobjects.appserver.r](#)

[EOFormElement](#)
[EOFrame](#)
[EOFrameController](#)
[EOFrameController](#)
[EOGeneralAdaptorExceptio](#)
[EOGenericRecord](#)
[EOGlobalID](#)
[EOHelpWindowAction](#)
[EOHTTPChannel](#)
[EOImageView](#)
[EOImageViewController](#)
[EOImageViewController](#)
[EOInsertAction](#)
[EOInspectorController](#)
[EOInspectorController](#)
[EOInterfaceController](#)
[EOJoin](#)
[EOKeyComparisonQualific](#)
[EOKeyGlobalID](#)
[EOKeyValueArchiver](#)
[EOKeyValueArchiver.Dele](#)
[EOKeyValueArchiving](#)
[EOKeyValueArchiving.Aw](#)
[EOKeyValueArchiving.Fini](#)
[EOKeyValueArchiving.Su](#)

Constructor Summary

[EOGenericRecord](#)()
Default constructor.

[EOGenericRecord](#)([EOClassDescription](#) classDescription)
Create a new instance and assigns it classDescription

[EOGenericRecord](#)([EOEditingContext](#) editingContext, [EOClassDescription](#) classDescription, [EOGlobalID](#) gid)
Deprecated. the default or 0 argument constructor should be used instead

Method Summary

| | |
|------------------------------------|--|
| EOClassDescription | classDescription () Returns the EOClassDescription registered for the receiver's class by invoking the EOClassDescription static method <code>classDescriptionForClass</code> . |
| static boolean | usesDeferredFaultCreation () Returns true, specifying that EOGenericRecords use deferred faulting. |

Methods inherited from class com.webobjects.eocontrol.EOCustomObject

[addObjectToBothSidesOfRelationshipWithKey](#), [addObjectToPropertyWithKey](#), [allPropertyKeys](#), [attributeKeys](#), [awakeFromClientUpdate](#), [awakeFromFetch](#), [awakeFromInsertion](#), [canAccessFieldsDirectly](#), [changesFromSnapshot](#), [classDescriptionForDestinationKey](#), [clearFault](#), [clearProperties](#), [deleteRuleForRelationshipKey](#), [editingContext](#), [entityName](#), [eoDescription](#), [eoShallowDescription](#), [equals](#), [excludeObjectFromPropertyWithKey](#), [faultHandler](#), [handleQueryWithUnboundKey](#), [handleTakeValueForUnboundKey](#), [hashCode](#), [includeObjectIntoPropertyWithKey](#), [inverseForRelationshipKey](#), [invokeRemoteMethod](#), [isFault](#), [isReadOnly](#), [isToManyKey](#), [opaqueState](#), [ownsDestinationObjectsForRelationshipKey](#), [prepareValuesForClient](#), [propagateDeleteWithEditingContext](#), [readResolve](#), [reapplyChangesFromDictionary](#), [removeObjectFromBothSidesOfRelationshipWithKey](#), [removeObjectFromPropertyWithKey](#), [shouldUseStoredAccessors](#), [snapshot](#), [storedValueForKey](#), [takeStoredValueForKey](#), [takeValueForKey](#), [takeValueForKeyPath](#), [takeValuesFromDictionary](#), [takeValuesFromDictionaryWithMapping](#), [toManyRelationshipKeys](#), [toOneRelationshipKeys](#), [toString](#), [turnIntoFault](#), [unableToSetNullForKey](#), [updateFromSnapshot](#), [userPresentableDescription](#), [validateClientUpdate](#), [validateForDelete](#), [validateForInsert](#), [validateForSave](#), [validateForUpdate](#), [validateTakeValueForKeyPath](#), [validateValueForKey](#), [valueForKey](#), [valueForKeyPath](#), [valuesForKeys](#), [valuesForKeysWithMapping](#), [willChange](#), [willRead](#), [willReadRelationship](#)

If you click on the method, you will find the definition of it:

awakeFromInsertion

```
public void awakeFromInsertion(EOEditingContext ec)
```

Overridden by subclasses to perform additional initialization on the receiver upon its being inserted into *ec*. This is commonly used to assign default values or record the time of insertion. *EOCustomObject*'s implementation merely sends an *awakeObjectFromInsertion* to the receiver's *EOClassDescription*. Subclasses should invoke *super*'s implementation before performing their own initialization.

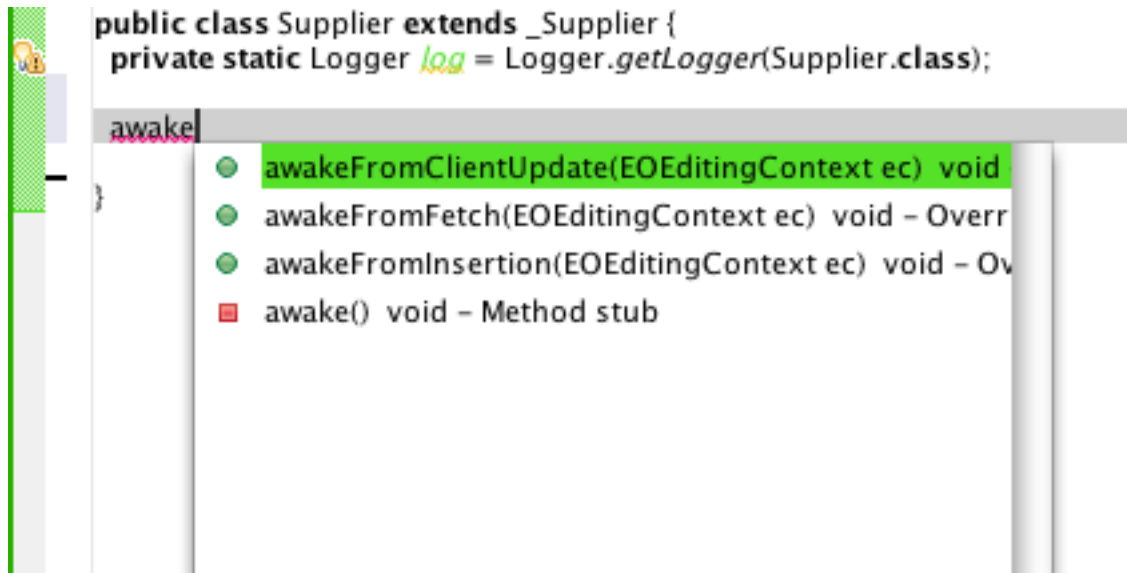
Specified by:

[awakeFromInsertion](#) in interface [EOEnterpriseObject](#)

Parameters:

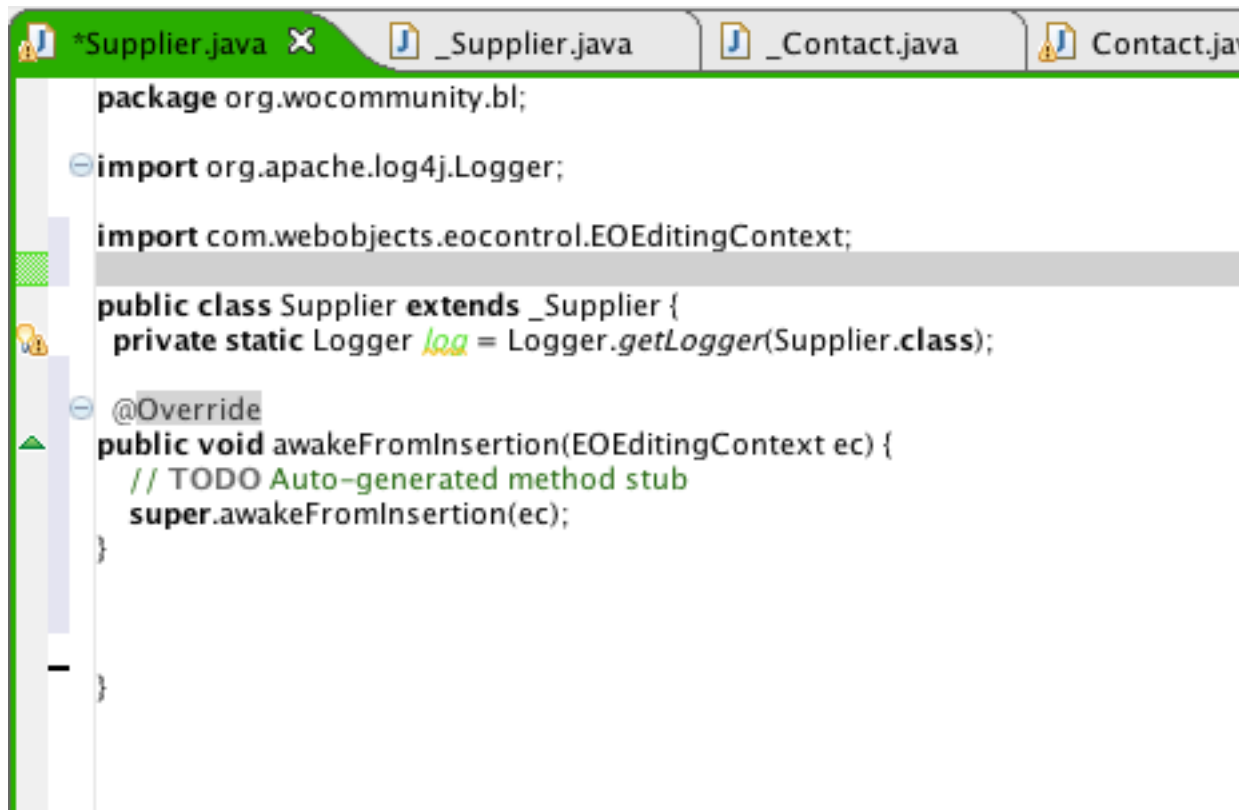
ec - An *EOEditingContext* object.

This looks like a nice place to add some default settings. So how do we add this method to our class. Actually, it is very simple: open our *Supplier.java* file, and start typing *awake* between the curly braces. Be careful! If you do not type between the curly braces, your code will not be part of the class.⁷ Make some space between the braces by pressing Return several times, Now press the Control key and the Space bar at the same time.



You will see some options to autocomplete the method. Choose *awakeFromInsertion*, and press Return. Suddenly; a whole lot of extra code is inserted:

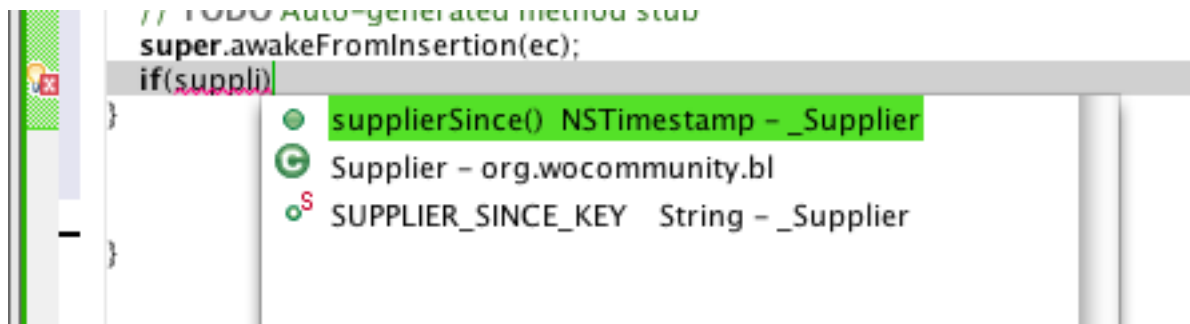
⁷ in Xcode the end curly brace used to be a lot lower, so you always typed between the curly braces, said the grumpy old man



```
package org.wocommunity.bl;  
  
import org.apache.log4j.Logger;  
  
import com.webobjects.eocontrol.EOEditingContext;  
  
public class Supplier extends _Supplier {  
    private static Logger log = Logger.getLogger(Supplier.class);  
  
    @Override  
    public void awakeFromInsertion(EOEditingContext ec) {  
        // TODO Auto-generated method stub  
        super.awakeFromInsertion(ec);  
    }  
}
```

Now we can add our own little addition to this method. What we want, is that if the supplierSince method is not filled with a date, it will be filled automatically with today's date.

On the next line after super.awakeFromInsertion(ec); start typing: 'if (suppli', then once again, press Control Space. You will see the methods and classes that can be auto-completed:



```
// TODO Auto-generated method stub  
super.awakeFromInsertion(ec);  
if(suppli)
```

Choose the **supplierSince()**, press Return.

Now complete the argument:

```
if(supplierSince() == null)
```

Notice that WOlips had also already completed the other ')' after our typing. Now go on, and start a curly brace and press Return. The other curly brace is inserted, and an extra line in between:

```
if(supplierSince() == null){  
    }  
}
```


Eclipse Wonder tutorial part one: setting up the data

Now we will add our default values. Once again we will use WOLips magical autocomplete trick. Place the cursor between the curly braces, and start type 'setSu', then once again type Control-Space. The method will be completed to 'setSupplierSince(Value)', and the cursor will be positioned at the 'Value', with the complete work selected. If you start typing, the word will be replaced. Pretty nifty, eh? Now type 'new NST', use autocomplete.

This is how the complete method should look like now:

```
@Override
public void awakeFromInsertion(EOEditingContext ec) {
    // TODO Auto-generated method stub
    super.awakeFromInsertion(ec);
    if(supplierSince() == null){
        setSupplierSince(new NSTimestamp());
    }
}
```

The TODO can be removed, as we have now done what we had to do (The TODO's can be shown in a list, so that you know what you still have to do). Also, the @Override could be removed, but XXXXX.

Another thing to notice is that the imports which are needed to have the methods defined properly are automatically included. Perhaps you hadn't noticed it yet, because the imports are (by default) not all displayed. This is what you see:

A screenshot of the Eclipse IDE's import list. It shows a single line: `+import org.apache.log4j.Logger;` with a small '+' icon to the left, indicating that the list of imports is collapsed.

However, if you click on the '+' sign, this collapsible list will open and you will see all the imports:

A screenshot of the Eclipse IDE's import list with the '+' icon clicked, showing an expanded list of imports. The visible lines are: `import org.apache.log4j.Logger;`, `import com.webobjects.eocontrol.EOEditingContext;`, and `import com.webobjects.foundation.NSTimestamp;`.

If you click on the minus sign, the imports will fold together again, preventing serious information overload.

Adding some business logic: validation of values

Another thing that you do in your business logic is to prevent information being entered that is obviously false, or not sufficient.

'Why can't I fill in "john" as an email address? What do you mean it is not a real email address? Here it says in his email to me: "john"! And if I reply, all I see is "john"!'

'What does that machine mean that the bank account is not valid? I am sure I have checked it! Do you want to read it to me aloud? OK. 5, 6, 3, oh... I seemed to have misread an 3 for an 8.'

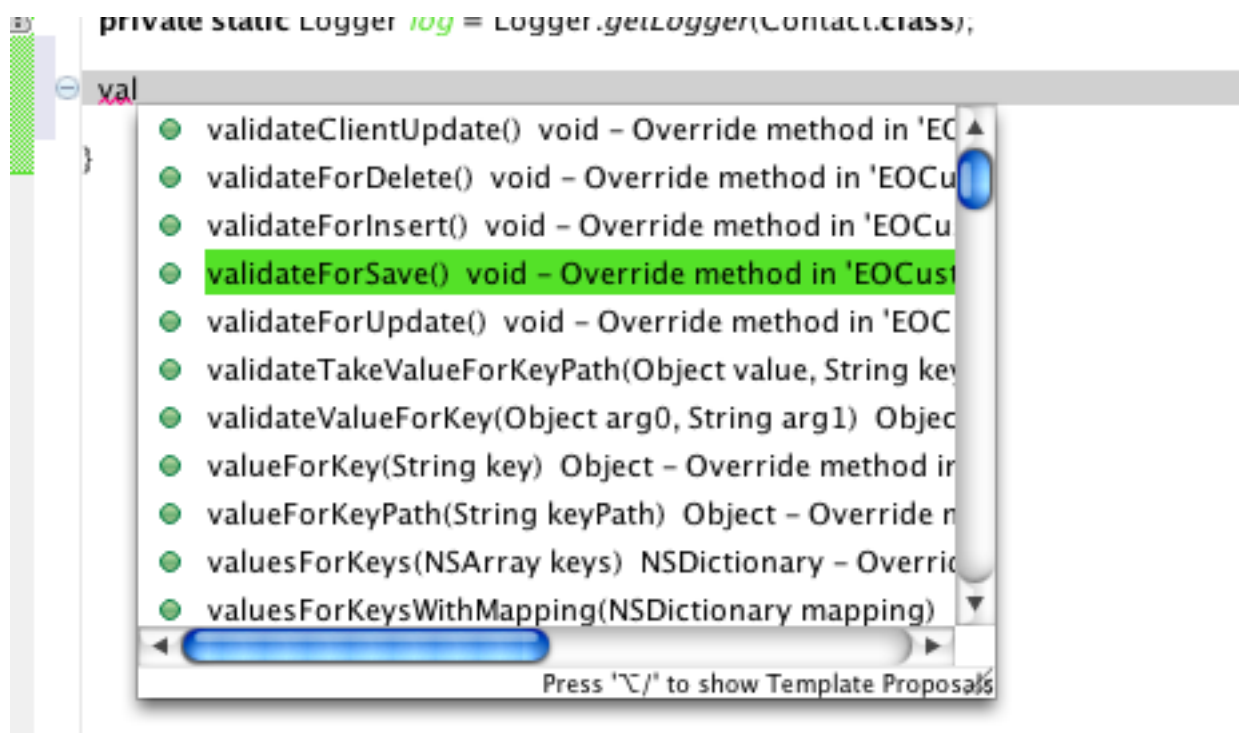
Eclipse Wonder tutorial part one: setting up the data

Most of the time your client (the person that has to fill in the information) will not be as soft in the head, but at least it will give you a feeling what kind of errors you can catch while validating user-entered information.

Just how these validation errors are taken care of, we will see in our next tutorial. For now, we will demonstrate its usefulness, by getting rid of email address errors. As you might have guessed, we will look for a method with something about validation. The one we want now is **validateForSave()**. Look it up in the javadoc, or at the about Enterprise Objects documentation (see the **Some more reading** section find the documentation).

We have already said that a WebObjects developer should be a lazy developer. If he is not, there is a big chance that he is not using the Frameworks properly. If a WebObjects developer writes a lot of code, one should look suspiciously about his WebObjects skills. We are proving now we have The Right Stuff. We did not create three email attributes in employee, supplier and customer, but we only had one: in the super class Contact. So by creating a validation of the email address over there, we validate all the email addresses in all the subclasses. Isn't object-programming great?

We start by opening Contact.java via the WOPackage Explorer. Once again you will see a very barebones class. Now, go into the class definition between the curly braces, again make some space by pressing Return several times, and start typing **val**, followed by Control-Space. Choose **validateForSave** from the suggestions. (Notice there are lotsof validation options).



Now we have a skeleton for filling in our validation requirements for our Contact class. For the moment, all we want to validate is email. Now there are two ways to do that. First one is to do some checking in the **validateForSave** method on the email address. And then return an exception if it is not OK:

```
public void validateForSave() throws ValidationException {
    super.validateForSave();
    if (email() == null) {
```

```
        throw new NSValidation.ValidationException("The email  
address must be filled in!");  
    }  
}
```

Note that, depending where you put the **super.validateForSave()** method in this definition, the validation of the superclass will precede or follow the custom validation that is taking place here.

Another way to validate, is to use the name of our Key/method that we want to validate in this case **email()**, and precede that with **validate**. In this case, we would have a new method, **validateEmail()**, that will take care of the validation:

```
public String validateEmail(String s) throws NSValidation.ValidationException {  
    if (s==null)  
        throw new NSValidation.ValidationException("Email address was not  
provided");  
    if (s.length()==0)  
        throw new NSValidation.ValidationException("Email address was not entered");  
    // now we start seriously, via pattern matching:  
    //Set the email pattern string  
    Pattern p = Pattern.compile(".+@.+\\.\\.[a-z]+");  
    //Match the given string with the pattern  
    Matcher m = p.matcher(s);  
    //check whether match is found  
    boolean matchFound = m.matches();  
    if (!matchFound)  
        throw new NSValidation.ValidationException("Email address must follow  
rules: use domainname and @ sign. Example: user@wocommunity.org");  
    return s;  
}
```

If you look at the **NSValidation** class in the API reference you will see a somewhat cryptic description of this, under **Writing validateKey Methods**.

Of course, you can enhance that with some serious checking. Personally, I would do most of the checking in javascript, as that will prevent a round trip to the webserver and to WebObjects: all the error handling can be taken care of in the web browser. But not every webbrowser supports javascript, or has javascript turned on.

Another tip is to look at the WebObjects PetStore example that Apple provides as part of the WebObjects installation. In there, laziness has been taken to another level by creating a Validate class that concentrates all the possible Validation methods. We will come back to this when we will start talking about refactoring your code in Eclipse, to make sure you are not typing too much.

This is the end of the first part of the tutorial. Notice that you have done nothing that you can show off to your superiors or colleagues. I would hide all this for the moment, until you have concluded the second and the third part of this tutorial. Then you can start showing off. If you want to, you can download the completed ShopBL project from www.wocommunity.org.

Some more reading

Eclipse Wonder tutorial part one: setting up the data

If you want to know more about modeling your object models and about database connections, setups etc. the best place to go first is:

http://developer.apple.com/documentation/WebObjects/Enterprise_Objects

There is the PDF and HTML version of the **Introduction to WebObjects Enterprise Objects Programming Guide**, which is the authoritative guide on Enterprise Objects.

After that there are the classes and methods itself, which are in:

<http://webobjects.mdimension.com/javadoc/WebObjects/5.4/>

specifically in the packages `com.webobjects.eoaccess` and `com.webobjects.eocontrol`.

Next I would advice you to go to the wiki that has been set up by the WebObjects community at:

<http://wiki.objectstyle.org/confluence/display/WO/Home>

where you will find information on all of WebObjects. Look for Enterprise Object Framework.

After that, it is time to look at the Wonder extensions of WebObjects, that we used in this tutorial for getting ERPrototypes, which is just another click away, fortunately. Go to:

<http://wiki.objectstyle.org/confluence/display/WONDER/Home>

You can find the Wonder Javadoc API at

<http://webobjects.mdimension.com/wonder/api>

If you are still completely confused about why and how, consult the WebObjects mailing lists. For details of these lists go to http://www.wocommunity.org/webobjects_related.html