Best Practices-Model

In construction

This chapter is under construction. Do not expect it to be useful... yet! 🙂

Model

One of the greatest things about WebObjects is the Enterprise Objects Framework (EOF) that handles all your data management. More than 10 years old, it's still one of the most powerful OR mapping tools available, if not the most powerful. It even inspired Code Data, an EOF clone used in the Mac desktop for all the data handling an application needs, introduced in Mac OS X Tiger (10.4).

One great feature of EOF is data modeling. EOF allows you to define your data model in one or more Model files that are contain all the information needed by the application to, in run time, access the database to create, search and delete your data. Here we'll see some useful advices about data modeling and Model file creation.

Prototypes

EOF is a great tool, and one good thing about it is that it doesn't try to hide or make decisions for you. Enterprise applications is serious stuff, and many factors, like backward compatibility with existing databases must be taken care. One place where EOF leaves the decision to you is how it should map your Java objects to the database internal data types.

There are two main reasons for this:

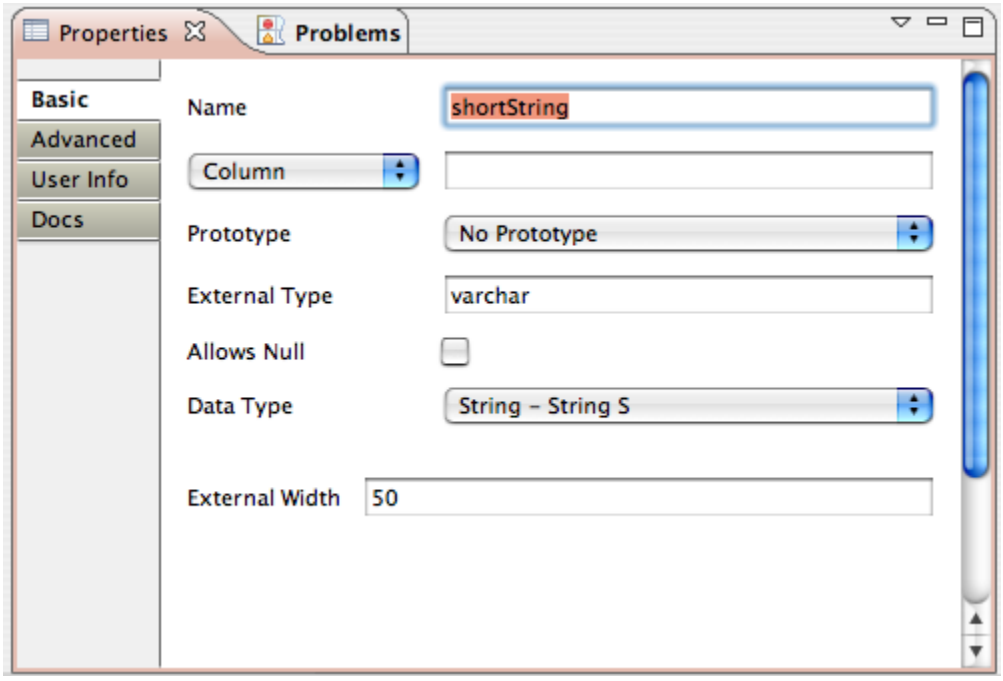
- There are some things EOF simply can't guess. If you have a String, you need to decide how much space you want to use on the database for it. If you know the String will contain a color, like "Red" or "Yellow", something like a varchar 50 should be enough. On the other end, if the String will be used to store a full name of a user, you probably want to make it a varchar 1000 or so (European names can be very long!). Of course, most databases allow you to use unlimited text data types, but that has serious implications in the database speed. So, it's up to you.
- YMMV however, Postgresql recommends that you use a column of type text. Their documentation states there is no speed penalty for choosing this over varchar(some number). The Entity Modeler prototype is 'longText'
- You may be constrained to existing data types. If you are building an WO application on top of an existing database with millions of records, you cannot choose the data types you want to use - you must use the existing ones used by the database. So, you have to tell EOF what types are those, so that it can read and write information to your existing database.

If you already tried to create a model, by following Janine's tutorial or so, you know how flexible it can be - and how tedious it is. It shouldn't be needed to choose, for every integer in your model, the data type for it. Or the data type for every primary key. Or the data type for every string. Worse, if you decide you want to use higher precision floats on all your application, or if you want to increase the size of some of the strings you use in your data model, you should be able to change that information on only one place. After all, duplicating code is bad, right? So why can't you do it?

Well, you can! That's what prototypes are for. A prototype allows you to:

- Define the data type mappings between your model and the database internal types all in one place.
- Use the same mappings for all your applications and frameworks.
- Change information on one place, everything gets updated.
- Create slight variations to the defined mapping in a given attribute.
- Change DB easily without touching your models.

Let's look at an example for a prototype. This is the definition of "shortString", a prototype defined in the WONDER's ERPrototypes framework (we'll talk more about this framework later). The screenshot was taken from the Entity Modeler plugin in Eclipse, part of WOLips.



We can see here that:

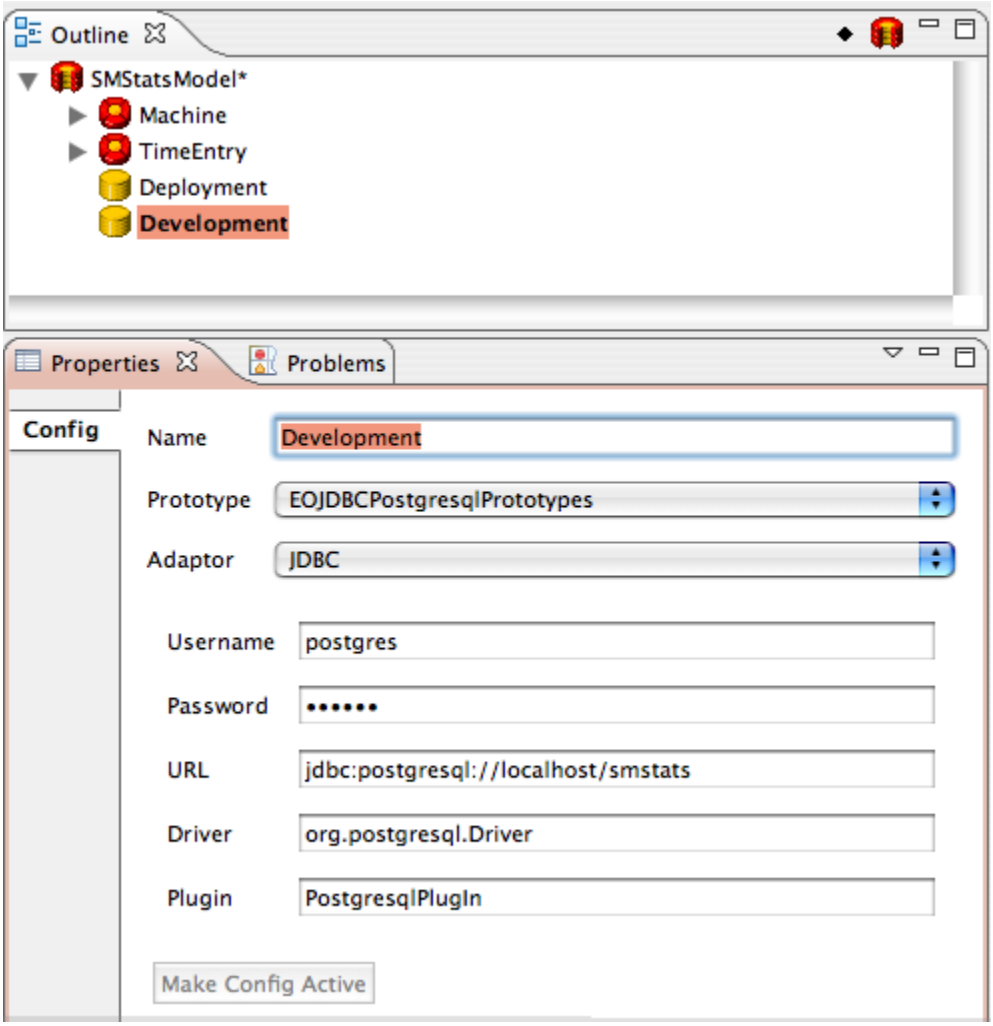
- The prototype name is "shortString".
- The external type (i.e., the database type) is varchar.
- The varchar width is 50, which means you can save up to 50 characters on each record.
- The Java class that will map the database value in memory is String.

(The other fields are not relevant for the prototype definition)

You can create your own prototype definitions, or use WONDER's.

To create your own prototypes, you need to create a special entity in a model file called `EOJDBCPrototypes`. It's recommended that you create a new model file with just the prototype entity, and place that file in a new, empty framework. This way you can share the same prototypes with all of your projects, and you may switch between prototype frameworks to use different databases without making any modification on your application.

WONDER prototype framework, ERPrototypes, has different prototype entities, one for each database. The one for Postgresql framework is called `EOJDBCPostgresqlPrototypes` and so on. To use those prototypes, just link your project with ERPrototypes after installing WONDER. Then, on your model database configuration, choose the right prototype in the appropriate menu:



As state before, one of the advantages of using prototypes is that, if you change a prototype definition, your model will automatically use the new definition for all the attributes that are linked to that prototype. There's also an interesting feature here that is worth mentioning. After setting a prototype in a given attribute of your model, you can decide to change some details of the prototype. So, imagine you have a `shortString` prototype like the one showed above, but you decide that on some specific attribute that links to that prototype that you want the size to me 25 and not 50. You could not use the prototype at all for that attribute, and, instead, defining everything manually. But there's a better way: just set the prototype for smallString, and then, change the size from 50 to 25. When you try that, you'll see the 25 value will show up in black, not brown.

Before:

			0	Prototype	Name	Column Name	External Width	Precision	Scale
				id	machineKey	machineKey			
				shortString	name	name	50		

After:

			0	Prototype	Name	Column Name	External Width	Precision	Scale
				id	machineKey	machineKey			
				shortString	name	name	25		

The brown values are the global values defined by the prototype you chose for that attribute. Those values are directly set by the prototype, which means that, if the prototype changes, the values in that attribute will change too. But when you set the value specifically for that attribute, it will change to black, meaning that your change will override any data from the prototype. So, if you change those values in the prototype, that specific attribute will maintain the values you set, ignoring the prototype.

EOGenerator

To-many relationships