EOF-Modeling-Prototypes

Overview

Prototypes are EOF's mechanism for database-agnostic modeling. By including an `EOModel` that has an entity with the special name `EOJDBCDatabaseVendorPrototypes` (like `EOJDBCFrontBasePrototypes`, or `EOJDBCOraclePrototypes`), or the fallback general prototype `EOJDBCPrototypes`, you can define reusable type definitions. For instance, you might declare a type definition `money` that maps to a floating point number with precision 32 and scale of 2, but different database have different names and syntaxes for this. With prototypes, your model simple refers to its prototype `money` for its type, and the prototype define the rest, and can we switched according to the database you are using.

Because prototypes centralize your type information, it also makes it much easier to make bulk changes to your types. For instance, if you declare an `address` type to be a `VARCHAR(50)` and realize it's not enough, you can simply change your prototype to `VARCHAR(100)` and all of your types inherit the change.

WebObjects 5.4

Under WebObjects 5.4, the prototypes are loaded as the model is loaded, based on the connection dictionary of the model. The following order is observed for prototype loading:

- `EOPrototypes`
- `EOJDBCPrototypes`
- `EOJDBC<PlugInName>Prototypes` (*e. g.* `EOJDBCFrontbasePrototypes`)

How the Prototype is Applied

- The `valueType` is always overriden from the prototype. This covers things like `S` for Strings and `T` for timestamps etc...
- The `externalType` is overridden by the model attribute. This covers things like `VARCHAR` and `FLOAT`.
- The `width` is overridden by the model attribute. This is used where something like the width of a string is enforced.

Chuck Hill

A model without prototypes is, um, dumb. A reusable framework without prototypes is not reusable. See *Practical WebObjects* pages 45 - 55 for a discussion of this and code. There is some very superficial discussion in the EOModeler docs.

Anjo Krank

Project Wonder has neat code that allows you to switch then based on properties and not on the mumbo-jumbo that normally gets used. And we allow you to actually support multiple types of DBs, which is not possible with the `EOJDBCPrototypes`. However, I'd advice prototypes only when you have either only one model or are very disciplined.

Gotchas

EOModeler is not without its faults. It is, in particular, quite finicky about prototypes. The minor pains are, however, worth it.

Framework? What Framework.

EOModeler must have a `.pbproj` or a `.xcode` build folder in the path that you open the model from, or it will not be able to load other frameworks (in particular, prototype frameworks).

The Brown Column of Death

If you ever change the prototype of a previously defined column, you might notice the column name in your EOModel turns brown. Any column that is brown in your model is going to be defined by the prototype, but you may have noticed you typically don't define column names in your prototype. As a result, when you save your model, it will **blank** your column name.

If a column **does** get blanked, it results in a **terribly** annoying and useless error message from inside of EOF. I believe you get a `ClassCastException` from SQL generation from a method that is named something about creating a comma separated list from an `NSArray` (I don't have the exception around offhand). So if you ever see a `ClassCastException` from way down in EOF, log this note away because it will save you much pain trying to debug it.

It is best to always select the prototype first, then fill in the rest of the column. It has the least potential for annoying EOModeler bugs in exchange for a small behavioral change. If the column turns brown, you will need to rename the column and then change it back to "unbrown" it.