EOF-Using EOF-Overview

Jerry W. Walker

I was tempted to respond very much as Kieren did, but I understand how daunting WebObjects can be at the beginning. Kieren's advice is good, use it. But perhaps the following will also help.

First: I need some serious help. Once I bring in a row of my database using the fetch command as an object, which works fine, how do I access the cells in the object?

In early projects you will probably use EOUtilities with its shortcuts to do your fetching. But let me go back to the basics so you have the class names to look up in the API reference.

At the basic level, you need to define or reference 7 objects to get data from your relational database:

- an EOModel that maps your relational tables, rows and columns to Java classes, instances and attributes. Use EOModeler to create this.
- an Entity in the EOModel whose data you want to fetch (e.g. Customer) Again, this should be created by EOModeler.
- an EOQualifier that defines the conditions for your fetch (e.g. all Customers in New York)
- an EOSortOrdering that sets the criteria to order the results (e.g. by lastName, then firstName, then state)
- an EOFetchSpecification that collects the criteria for the fetch, especially the previous 3 objects.
- an EOEditingContext to manage your database data. For your early projects, use the default provided by your Session class.
- an NSArray to hold the objects fetched (e.g. myCustomers)

In practice, you don't need all of these, but at minimum you need an EOModel, an Entity, an EOEditingContext and the NSArray to hold your data.

Going back to the basic level, I'll presume that you or someone else has created the EOModel to map your database tables to your Java classes. I'll also presume that someone has defined at least one Entity in that EOModel whose data you're interested in obtaining.

You can also avoid the EOSortOrderings, the EOQualifier and the EOFetchSpecification by using shortcuts in EOUtilities. However you do it, though, you should end up defining an NSArray and filling it with your fetch. For instance, the code for a typical fetch in a WebObjects Component might look like this:

```
EOEditingContext myEditingContext = session().defaultEditingContext();
NSArray myCustomers = EOUtilities.objectsMatchingKeyAndValue(myEditingContext, "Customer", "state", "New York");
```

In the row i have an int called userGroup. How to I get the value of that int from the object and put it into a variable? Basically what I am asking is how do I gain access to the data I bring in so I can use it in my program?

The result of my code sample is an NSArray named myCustomers whose elements are instances of the Java class defined for the "Customer" Entity in your EOModel. (Typically, the Java class would also be named "Customer".)

The NSArray, myCustomers, is the array of Java objects corresponding to the rows that you retrieved from your database. Rather than rows, they have been converted by EOF to full fledged Java objects and can be manipulated as such. The first element of the array is the first row retrieved, the second element is the second row, etc.

I have tried EOUtilities.objectMatchingKeyAndValue, but what exactly is the Key? I have also tried EOUtilitlies.objectMatchingValues, but how do i add data to the dictionary?

In the sample code I provided, I used the EOUtilities.objectsMatchingKeyAndValue() method. The signature for that method is:

```
NSArray EOUtilities.objectsMatchingKeyAndValue(EOEditingContext ec, String entityName, String key, Object value)
```

So from my sample, you can see that:

```
ec = myEditingContext
entityName = "Customer"
key = "state"
value = "New York"
```

In particular, to answer your question, the keys are the EOModel attribute names for the Entity you're fetching. If you look closely at an EOModel, you'll notice that each Entity has three names, the Entity name, the name of the table representing that Entity in the database, and the name of the Java class representing that Entity in your program. Usually the latter two are identical, but don't confuse them.

By the same token, each Attribute for an Entity has a corresponding set of three names: the name of the Entities Attribute, the name of the column in the database representing that Entity Attribute, and the Java member (method or instance variable) representing that Entity Attribute in your Java class. Once again, usually the latter two are identical, but don't confuse them.

Rather than saying "Java member (method or instance variable)..." we just say "key". Object encapsulation should allow us to not care whether the value is represented by accessor methods or by an instance variable. An Apple proprietary technology called Key-Value-Coding (or KVC)

allows us to reference the value without caring. Literally, we provide the key name and we can completely ignore whether the object provides the value by accessor method or by instance variable.

Second: BLOBs. I have found almost nothing on blobs. How do I place them into the database, I know people don't like using BLOBs but my mentor want me to so I must, and how do I recall them latter in the program when I need them as input into another program called from my main program?

Your EOModel defines the object-relational mapping so that any column of a given table can be mapped to a Java class method or iVar (key). The EOModel defines that mapping including not only the name correspondence, but also the property correspondence so that EOF can take any of the column values from a given row in the database table and convert it to the appropriate key value in the Java object.

Remember that the column value has a database property, such as integer or BLOB. By the same token, the key value in your Java object has a class or is a primitive value (such as NSData or int). If you've set up the EOModel mapping correctly, then you simply reference the key in your java object after it's been fetched into the array. From the above example, for instance, to obtain the data for a single customer:

```
Customer aCustomer = myCustomers.objectAtIndex(0);
```

You now have a single Customer object referenced by the variable, aCustomer. If that Customer object has an age property represented by an integer, you reference it as you would any Java object variable value:

```
int theAge = myCustomer.age(); // presuming that there is an age() accessor method, a fairly safe assumption for EOs.
```

If the object member you want is a picture of the customer, some people would store that picture in the database (for WO applications, I would drag my heels a good bit before doing so, but presume I'm forced). That is data that would typically be stored as a BLOB in the database and as an NSData object in your Java class. In that case, you could retrieve the picture as follows:

```
NSData customerPicture = aCustomer.picture();
```