WebObjects Database Connectivity: Part I

by Theresa Ray of Tensor Information Systems

Sponsored by Apple Computer, Inc.
Developer Relations Group
for the Apple Media Program

WebObjects Database Connectivity: Part I

by Theresa Ray of Tensor Information Systems

Creating powerful websites for your users often involves integration with a database for storage and retrieval of information. WebObjects provides efficient access to nearly every database on the market through it's Enterprise Object Frameworks (EOF). EOF allows you to map the information stored in your database to business objects used by your applications - whether web based or windowed. And by putting your business logic into one set of objects used by all your applications, behavior such as validation, data conversion, or security restricted access is guaranteed to be the same no matter which application is accessing the data.

The Database Wizard and Enterprise Object Frameworks

One of the best resources for all WebObjects programmers, regardless of experience level, is Apple's on-line technical documentation. One of the tutorials provided at their site is a database access example using the database wizard. The documentation provided there is an excellent introduction to the Enterprise Object Frameworks, creating an eomodel (the mapping of data from your database to your business objects), and the terminology necessary for continued database connectivity discussions. This survival guide assumes that the reader has done the on-line tutorial, and is familiar with the terminology introduced there. Apple's database application tutorial can be found at http://gemma.apple.com/techinfo/techdocs/enterprise/WebObjects/GettingStarted/Movies/MoviesTOC.html.

Display Group versus Simple Array or Vector

When using the database wizard to fetch records from a database, the results are automatically put into a WODisplayGroup class. This is very useful when you are fetching records that you intend to display to your user - as in the Movies example provided in the tutorial. The WODisplayGroup manages many of the details related to fetching and displaying records for you, such as keeping track of all the matching records fetched, which of those records are actually displayed, which record the user is

currently editing, insertion of new records and deletion of old records, and providing navigation between multiple pages required to display all the fetched results. Even when you don't want to use the layout options exactly as provided by the database wizard, using the WODisplayGroup class to help manage your fetch results is usually a good idea.

However, consider the case of a security-restricted application. Maybe you have a web-based timecard entry application for your employees. You would want to have each user provide a login name and password in order to access the data entry part of the web application. You've stored the valid usernames and passwords in your database, and you just want to check to see that what the user typed in was valid. In this case, the fetched results are not to be displayed to the user. You are only interested in the result of the query. In this case, using the WODisplayGroup class is not wrong, but it may be more than you really need. In this case, you might like to have more manual control over the query.

When to Fetch

When you wish to manually code a fetch from the database, you must still construct an eomodel so that EOF can map the information from the database to your business objects. This step is no different than presented in the tutorial or in the Enterprise Object Frameworks documentation shipped with WebObjects Enterprise.

Next you must decide when you want to fetch. The wizard gave you two choices - 1) when the page was loaded or 2) when a query was submitted. You still have the same two choices, but instead of having the code automatically put in the right method for you, you must decide where it should go.

If you want a one-time fetch when the page is first loaded (or when the application is first loaded if you wish to fetch and store application level look-up tables), then the best place is probably in the constructor method for the page, session or application depending on the scope of the data you're fetching (for Objective-C users, the constructor method corresponds to the init method).

If you want to wait to fetch until after you have received some information from the user, such as this username and password example, then the appropriate time to fetch the data would probably be in the submit method for the form.

Construct a Qualifier

Now that you've decided when to fetch, you need to know how to fetch. When using the WODisplayGroup class, WebObjects Builder provided a WODisplayGroup inspector which

allowed you to specify the qualification mode (prefix, contains, suffix, etc) while the wizard constructed the qualifier for you. EOF has several qualifier classes which you should use when constructing your fetch. The most commonly used of these are the EOKeyValueQualifier, EOQualifier, EOAndQualifier, and EOOrQualifier. These classes are documented in the EOControl section of the Enterprise Object Framework Reference. A brief summary of each class is provided with example code in the appendix of this document as a convenience.

Construct a Sort Order

You may wish to sort your results. For this login validation example, sorting the results would probably not be necessary, since you are interested only in whether you have received an exact match. However, in other cases, sorting may be useful. When you used the WODisplayGroup, WebObjects Builder provided an inspection tool for the WODisplayGroup which allowed you to specify the sort ordering attribute and sort type (ascending, descending). When manually building your fetch, you must specify the sort yourself. You can either specify the sort in the fetch specification, which is the example shown here, or you can fetch the results into an array (or vector) and sort the results yourself in memory. The fetch specification's sortOrderings argument is an NSArray (or vector in Java) whose elements are instances of the EOSortOrdering class specifying the attributes to sort by and the ordering technique to use. The number of elements in the array is equal to the number of attributes you are sorting by. The first element is used as the primary sort attribute, etc.

EOF provides an EOSortOrdering class for specifying the sort ordering of a fetch. This class is also documented in the EOControl section of the Enterprise Object Framework Reference. A brief summary is provided with example code in the appendix of this document as a convenience.

Construct a Fetch Specification

Now that you have constructed your qualifier and sort ordering, you must construct a fetch specification. The fetch specification correlates the entity which you wish to fetch from (mapped to a table in the database by your eomodel), the qualifier you have constructed and the sort ordering you have constructed. It provides all the information necessary to construct the SQL for this fetch.

EOF provides an EOFetchSpecification class for correlating this information. This class is also documented in the EOControl section of the Enterprise Object Framework Reference. A brief summary is provided with example code in the appendix of this document as a convenience.

Fetch!

Now that you have related the entity to fetch from, the qualifier to use, and the sort ordering to use in the fetch specification, you can fetch the data. As explained in Apple's database application tutorial, a WebObjects application automatically creates an instance of the EOEditingContext class for each session. This editing context is responsible for managing all fetches, inserts, updates and deletes from the database. Therefore, we need to fetch the data using this editing context object. The session variable defaultEditingContext is a pointer to this session's instance of the EOEditingContext class. By sending this defaultEditingContext object the message objectsWithFetchSpecification:, you are asking it to perform the query using the fetch specification you pass in as the argument. The return value is an array of enterprise objects matching the results of the query. Below is sample code in objective-C syntax for a submit method which validates a user's login and password.

```
EOQualifier *qualifier;
EOFetchSpecification *fetchSpec;
NSArray *results;
EOEditingContext *ec;

// Get a handle to the session editing context
ec = [[self session] defaultEditingContext];

// Build the qualifier - assume login and password are variables storing user entered data
qualifier = [[EOQualifier alloc] initWithQualifierFormat:@'userName = %@ and
password = %@", login, password];

// Build the fetchSpecification - don't care about sortOrdering so we pass in nil
// Can pass in nil for qualifier if you want to fetch the entire table
fetchSpec= [EOFetchSpecification fetchSpecificationWithEntityNamed:@"Users"
qualifier:qualifier sortOrderings:nil];

// Ask the EOEditingContext to fetch the data and store the returned values in an array
results = [ec objectsWithFetchSpecification:fetchSpec];

// See if we have a match
if ([results count]) // Possibly store the user in a session variable, and go on to next page
else //Return some error/retry page
```

Conclusions

The code example to query the database for matching username/password combinations

isn't very complex. One you understand this example, nearly all of your manual fetching will be the same. Just follow the procedure - qualifier, sortOrder, fetchSpec, fetch and you'll be able to generate quick queries in no time.

If you're curious about the exact format of the return results, add in the following line:

[self logWithFormat:@"results = %@", results];

after you fetch and watch your output window. As mentioned previously, the returned information is stored in enterprise objects. Basically, the database information has been converted into an NSDictionary for each element in the array. The keys of the NSDictionary are the attribute names that you have specified in your eomodel and chosen to be class properties. The log results might look something like:

```
(
  { "userName" = "barney";
    "password" = "bedrock";
  }
)
```

This result shows an array with one element. That element is a NSDictionary whose keys are "userName" and "password" and whose values are "barney" and "bedrock". Experiment with trying to print out just the userName or password.

Appendix

EOKeyValueQualifier

An instance of the EOKeyValueQualifier creates a qualifier for a single attribute. It's initialization arguments include the attribute (mapped to the database column through the eomodel), the selector for comparison (EOQualifierOperatorEqual for =, EOQualifierOperatorLike for like, EOQualifierOperatorGreaterThan for >, etc) and the value for comparison (usually some object's value). For example, in objective-C syntax, construction of an EOKeyValueQualifier might be done as follows:

EOKeyValueQualifier *qualifier=[EOKeyValueQualifier qualifierWithKey:@"userName" selector:EOQualifierOperatorEqual value:login];

where userName is the attribute name in your eomodel that refers to the database column storing valid usernames, and login is a page level variable storing the username entered by the user. This statement would roughly translate to the SQL "where userName = value" where value is whatever the user typed in.

EOQualifier

An instance of the EOQualifier creates a qualifier for as many attributes as you specify. It's initialization argument is a string with the pseudo-SQL qualifier. For example, in objective-C syntax, construction of an EOQualifier might be done as follows:

```
EOQualifier *qualifier = [[EOQualifier alloc] initWithQualifierFormat:@"userName = %@ and password = %@", login, password];
```

where login is a page level variable storing the username entered by the user and password is a page level variable storing the password entered by the user.

EOAndQualifier

An instance of the EOAndQualifier class creates a qualifier and-ing as many attributes as you specify. It's initialization argument is an array (or vector in Java) whose elements are instances of the EOKeyValueQualifier or EOQualifier classes. For example, in objective-C syntax, construction of an EOAndQualifier might be done as follows:

```
NSMutableArray *array = [NSMutableArray arrayWithCapacity:2];
[array addObject:[EOKeyValueQualifier qualifierWithKey:@"userName" selector:EOQualifierOperatorEqual value:login]];
[array addObject:[EOKeyValueQualifier qualifierWithKey:@"password" selector:EOQualifierOperatorEqual value:password]];
EOAndQualifier *qualifier=[[EOAndQualifier alloc] initWithQualifierArray:array];
```

This statement would roughly translate to the SQL "where userName = login and password = password".

EOOrQualifier

An instance of the EOOrQualifier class creates a qualifier or-ing as many attributes as you specify. It's initialization argument is an array (or vector in Java) whose elements are instances of the EOKeyValueQualifier or EOQualifier classes. The example and result is exactly the same as for the EOAndQualifier, substituting Or for And.

**NOTE:

Note that you can nest EOAndQualifiers and EOOrQualifiers, making an EOOrQualifier one of the array elements for an EOAndQualifier. There is no restriction on how deep the nesting can go.

EOSortOrdering

An instance of the EOSortOrdering creates a sort order for as many attributes as you specify. It's initialization arguments are the attribute to sort by, and the selector specifying the sort technique (EOCompareAscending for ascending sort order, EOCompareDescending for descending sort order, etc). For example, in objective-C syntax, construction of an EOSortOrdering might be done as follows:

```
EOSortOrdering*sortOrder= [[EOSortOrdering alloc] initWithKey:@"userName "
operatorSelector:EOCompareAscending];
```

EOFetchSpecification

An instance of the EOFetchSpecification creates a fetch specification specifying the details of what to fetch. It's initialization arguments are a string containing the entity name to fetch from, a pointer to the qualifier previously constructed (this can be any of the EOQualifier classes or nil), and a pointer to an NSArray or NSMutableArray (or nil) whose elements are instances of the EOSortOrdering class. If the array contains one argument, the key specified in that sort ordering object will be the only attribute that the fetch is sorted by. If the array contains two elements, the first element's key is the primary sort attribute, and the second element's key is the secondary sort attribute. For example, in objective-C syntax, construction of an EOFetchSpecification might be done as follows:

```
EOFetchSpecification *fetchSpec = [EOFetchSpecification
fetchSpecificationWithEntityNamed:@"Users" qualifier:qualifier
sortOrderings:sortOrderings];
```

References

http://gemma.apple.com/techinfo/techdocs/enterprise/WebObjects
WebObjects Developer's Guide
Enterprise Objects Framework Developer's Guide
http://www.omnigroup.com/MailArchive/WebObjects
http://www.omnigroup.com/MailArchive/eof
http://www2.stepwise.com/cgi-bin/WebObjects/Stepwise/Sites

About the Author

Theresa Ray is a Senior Software Consultant for Tensor Information Systems in Fort Worth, TX. She has worked as a consultant on WebObjects projects for a wide variety of clients including the U.S. Navy and the United States Postal Service. Her experience spans all versions of WebObjects, from 1.0 to 3.5, several versions of EOF, from 1.1 to 2.1, and NEXTSTEP 3.1 to OPENSTEP 4.2. In addition, she is an Apple-certified instructor for WebObjects courses.

Tensor Information Systems is a systems integrator providing enterprise solutions to its customers. Tensor's employees are experienced in all NeXT/Apple technologies including OPENSTEP, NEXTSTEP, EOF and WebObjects. Tensor also provides Apple-certified training in WebObjects, Oracle consulting and training, as well as systems integration consulting on HP-UX.

You may reach Theresa by e-mail: *theresa@tensor.com*

http://www.tensor.com