Web Applications-Development-Cocoa EO Applications

> This is deprecated information!

Overview

> *Note: these techniques depend on deprecated technologies. Use at your own risk. - Mike Schrag*
> *Hey, they don't even exist on Mac OS X 10.5. It's dead, Jim. - Pascal Robert*

A CocoaEOApplication (formally, in WO 5.2, Cocoa Enterprise Objects Application, but here simply CEO) is a set of objects which allow you to use the Enterprise Objects Framework (EOF) inside Cocoa. Even though the technology is not yet supported by Apple, it had been around since NeXT was independent.

In its previous versions, CEO uses Objectve-C as its primary language (which is now the primary language of Cocoa), but since WO 5.0 the main language to use EOF is Java. In order to use EOF within Cocoa (in particular, in order to use EOModeler in OSX), Apple developed the so-called Java-Bridge (see e.g., http://cocoadevcentral.com/articles/000024.php ---there was a document entitled "Using the Java Bridge", but at the time I am writing this (Jul 24 2003), it was not available any more in developer.apple.com; however you may find a copy in your hard disk under /Developer/Documentation/JavaBridge/JavaBridge.pdf). In order to develope a CEO, you will have to get used to its mechanisms.

*Update (Jun 3, 2005): the JavaBridge.pdf is now in http://developer.apple.com/documentation/Cocoa/Conceptual/Legacy/JavaBridge/JavaBridge.pdf_*

Getting Started

There are some things you have to be aware when creating a CEO project (this note is for WO 5.2 and has been revisted for 5.2.2 ---and now for WO 5.2.4):

When you launch the Project Builder's wizard and ask for a Cocoa/EO application, it will create an unusable MainMenu.nib file. To solve the problem: close your project; delete the "corrupted" file MainMenu.nib (it is inside the English.lproj folder of your project); launch Interface Builder and create an Empty interface; save it as MainMenu.nib (better with the extension) in the same place where the "corrupted" file was; re-launch Project Builder; Done.

- .... in WO 5.2.2 for OSX 10.3 (Panther+Xcode), the previous bug was fixed.
- .... in WO 5.2.4 for OSX 10.4 (Tiger+Xcode), it still works.

The first time you drag an entity from EOModeler into Interface Builder, an EOEditingContext will be created for you (if you drag the entity onto a window, an NSTableView and an EODisplayGroup will also be created; if you do onto the class-viewer, only the EODisplayGroup will be created). The default (not always desirable) is to "fetch on load" all the objects in the entity; if you do not want this, go to the inspector (with the EODisplayGroup selected) and uncheck the corresponding button.

Basically, you are there. You can use all actions of your display groups and your editing context to see and manipulate your database, without writing a single line of code ---in fact, you do not have neither to compile your project; simply connect your buttons and run the "Test Interface" command from the file menu (or cmd+r from the keyboard) and you will see your database showing up.

Of course, if you are an experienced programmer and want to add some "logic" to your app, you can do it. The first thing you have to do, is to choose a language; I prefer Obj-C, but also Java can be used... in fact it will be used even if you decide not to. The EOF had been written in pure Java so, even if you do not write a single line of code in Java (which some times seems unavoidable), you will be using it via the Java Bridge. I'll be back on this later...

A "tutorial"

Following the recomendation of a virtual-friend (Arturo Perez), I decided to add a simple guide to build up a CEO.

It is really simple... almost as simple as D2W, but more beauty.

Have a model? if so, you can manipulate your db with out writing a single line of code.
Try the following:

- create a Cocoa/EO project importing your model (if you are still in 5.2.1, taking care of recreate the nib file)
- open your MainMenu.nib (from PB or Xcode) and drag a new window from the IB's palette

- open your model and drag an entity to the window
- in IB, press cmd+r

Your table will show up!!

If you are familiar with IB, the reset is history, but in case you are not, you can try:

- drag a button from the palette to you window
- connect it (with cntrl+drag) to your display group (it was created in step 3 below) and select the outlet "insert" by double-clicking it
- press cmd+r
- push the button

You will see a new row in your table!!

Do you want to add some custom logic?

- select the Classes tab, select your root class (I prefere NSObject) and press enter
- press cmd+1, add one outlet (displayGroup) and one action (doit
- press cmd+opt+f (answer ok to the panel) and press cmd+opt+i
- connect your outlets, add a button and connect it to your action. Save.
- in PB (or Xcode) you will see your files. Edit them to implement doit:. For example, try something like

```
- (IBAction)doit:(id)sender{
    [[displayGroup displayedObjects] takeValueForKey:aValue :@"aKey"];
 }
```

- in PB (or Xcode) press cmd+r

You will compile and run your new app. After the table shows up, push the (second) button.

That's it!! You are now ready to develope the best apps in the market ;^()

On Memory Management

If you know you are going to require tones of memory, or you get the "java.lang.outOfMemoryError"; message in your console, or your app is simply not doing what it is supposed to, you may need to read this section... otherwise, simply jump it.

Are you still here? well, your problem may be that the VJM did not allocate enough memory for you.

In Project Builder, under targets and with the main target selected, go to Info.plist Entries > Pure-Java Specific (in Xcode, you have to click the little triangle of the targets group, double-click the main target). Add in the "Additional VM Options" text field something like

```
-Xms256m -Xmx256m
```

(This will give to your JVM 256Mb of memory instead of 64Mb, the default.)

Also, it is convenient that you get used to use NSAutoreleasePool properly (see its documentation and related topics; in particular, read Chapter 4 of the book "The Objective-C Programming Language" http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC). The Java Bridge does a good work using "garbage Collection" in the Java side, and "reference count" in the Obj-C side; just do not worry about that.

Morphing Objects

As I said, even if your code uses only Obj-C, you will be dealing with Java objects via the Bridge; therefore you most know how to manipulate those objects "allocated" in the Java side.

The easiest way to allocate an object in the Java side, from the Obj-C one, is, e.g.,

```
id aJavaObject = [NSClassFromString(@"java.lang.Object") new];
```

Do not forget to release such an object; since you allocated it, you own it!

For those objects comming from com.apple.cocoa.foundation.* you really does not have to take care (with the remarkable exception of NSEnumerator); they go forth and back nicely. Even do some simple objects like String (which maps to NSString), Number (which maps to NSNumber) and some basic types like int, char, boolean and so.

So far, so good. However, be carefull when you want to enumarate an array. In the one hand, if the array was created in the Obj-C side and you ask for an objectEnumerator, you will recive an NSEnumerator. This object implements the method nextObject to traverse the elements of the array. On the other hand, if the array was created in the Java side (e.g., as a result of an allObjects() call) and you ask for an objectEnumerator, you will recive an Enumerator object which implements the nextElement() method instead.

Some examples of the two scenarios are the following.

An Obj-C array:

```
NSArray* anObjCArray = [NSArray arrayWithObjects:objA, objB, objC, objD,
nil];
 NSEnumerator* en = [anObjCArray objectEnumerator];
 id o = nil;

 while(o = [en nextObject]){
   // here your code using o,
   // which will traverse all objects in the array
 }
```

A Java array:

```
NSArray* aJavaArray = [someDisplayGroup displayedObjects];
 NSEnumerator* en = [aJavaArray objectEnumerator];

 while([en hasMoreObjects]){
   id o = [en nextElement];
   // here your code using o, which will traverse all objects in the array
 }
```

These simple examples show some things to have in mind when developping a CEO:

- Obj-C's methods may accept arbitrarly large lists of objects as parameters (they usually end with nil); Java's do not. If you know your method will be processed by a Java's object, NEVER use such a construction... it will simply do not be recognized. Instead, Java uses arrays of the form

```
new Object[] {objA,objB,objC,objD}
```

Therefore, you most create FIRST the array and, once allocated, use it as argument. An important example is the class method qualifierWithQualifierFromat(String format, NSArray arguments) implemented by com.webobjects.eocontrol.EOQualifier. I'll be back with this isue later...

- An NSEnumerator ends its "path" with a nil object; Java's Enumerator throws an exception if it is asked for one more element. Therefore, to avoid such an exception, you have to use the hasMoreElements method.
- Objects in the Java side are called from the Obj-C side as if they were Obj-C objects (with the same sintaxis); that is, if the Java object anObject implements the method someMethod(Object someParameter, Object someOther), you have to call it with a line of the form

```
[anObject someMethod:someParameter :someOther];
```

Arrays, NSArrays, and more arrays...

Maybe the worst part of the CEO integration is the managment of arrays; there are 5 types of array you most be aware:

- NSArray; as defined in Foundation.h (Obj-C side).
- NSArray; as defined in com.apple.cocoa.foundation (Java side).
- NSArray; as defined in com.webobjects.foundation (Java side).
- A C array of objects (like id array[]).
- A Java array of objects (like Object[]).

This is too much!!!! But life is life, and beter get used to them...

Usually, when you are not intended to use EOF inside Cocoa, you can forget about the array in com.webobjects.foundation; the bridge translates, by default, that of Foundation.h to com.apple.cocoa.foundation, back and forth, with no problem... well, almost: from Java to Obj-C each array go as an NSCFArray which is documented NOWARE, but it works as an NSArray. On the other side, from Obj-C to Java, they go as com.apple.cocoa.NSMutableArray which inherits from the usual NSArray ---try to print their classes description in both sides after they had traveled the bridge with the following code:

```
[textField setStringValue:[[aJavaArray class] description]];
```

or

```
textField.setStringValue(anObjCArray.getClass().toString());
```

This is not as bad as it may seems. In practice, you do not have to be aware of this.

The real problem start when you want to use EOF. Most of the Enterprise Objects (defined somewhere in com.webobjects.*), when use an array, they are waiting for a com.webobjects.foundation.NSArray and non of the above.

There are many solutions to this problem. The first one (for those who knows Java and Obj-C) is to have a bridgeTool.java object in charge of allocating those arrays as neaded:

```
public com.webobjects.foundation.NSArray
arrayFromArray(com.apple.cocoa.foundation.NSArray cocoaArray)
 {
     com.webobjects.foundation.NSMutableArray woArray =
        new com.webobjects.foundation.NSMutableArray();
     int i;
     for(i=0;i<cocoaArray.count();++i){
        woArray.addObject(cocoaArray.objectAtIndex(i));
     }
     return woArray;
 }
```

And call this method from your Obj-C object with something like

```
NSArray* cocoaArray; // suppose this exists
 id bridgeTool = [[NSClassFromString(@"bridgeTool") new] autorelease];
 id woArray = [bridgeTool arrayFromArray:cocoaArray];
```

Once you have your woArray, you can use it as an argument in Enterprise Objects' calls.

I had not foud the way to call a Java method which has an Object[] array as argument. If someone reading this finds out, please let us know...

Building qualifiers

When working with EOF, there are several points when you will want to build a qualifier to perform a fetch. A qualifier is an instance of com.webobjects.eocontrol.EOQualifier. If your qualifier does not include any date, then it is as easy as

```
NSString* partOfAName; // suppose this exists
 NSString* qualifierFormat = [NSString stringWithFormat:@"name
caseInsensitiveLike '*%@*'", partOfAName];
 id qualifier = [NSClassFromString(@"com.webobjects.eocontrol.EOQualifier")
qualifierWithQualifierFormat:qualifierFormat :nil];
```

and then use it in a fetch specification with

```
id fetchSpecification =
  [[NSClassFromString(@"com.webobjects.eocontrol.EOFetchSpecification")
     new] autorelease];
 [fetchSpecification setEntity:@"myEntity"];
 [fetchSpecification setQualifier:qualifier];
```

On the other hand, if you want to use a date in your qualifier, since NSCalendarDate is not properly translated to NSTimestamp, you have to use again the bridgeTool.java trick:

```
public NSTimestamp nsTimestampFromString(String dateString){
    NSTimestampFormatter formatter = new NSTimestampFormatter("%d %m %Y");
    ParsePosition pp = new ParsePosition(0);
    NSTimestamp myNSTimestamp =
(NSTimestamp)formatter.parseObject(dateString, pp);
    return myNSTimestamp;
 }
```

and call this from Obj-C by

```
NSCalendarDate aCalendarDate; // suppose this exists
 [aCalendarDate setCalendarFormat:@"%d %m %Y"];
 id nsTimestamp = [bridgeTool nsTimestampFromString:[aCalendarDate
description]];
```

(just be consistent with your formats).

It is also possible (at least in WO 5.2.4 running on Tiger) to use com.webobjects.eointerface.cocoa.EOCocoaUtilities as follows:

```
id objPath = @"com.webobjects.eointerface.cocoa.EOCocoaUtilities";
 id aDate = [NSCalendarDate date];
 id nsTimestamp = [NSClassFromString(objPath)
timestampForGregorianDate:aDate];
```

From here, you can use the nsTimestamp as argument in those classes defined in com.webobjects.* (e.g., in your qualifiers).

Putting things together

Let us start this section describing how to put together two of the most powerful frameworks of OS X; namely, EOF and the NSDocument framework (this section was written, in July 2005, while working in WO 5.2.4 running on Tiger) ---later on, we will integrate these with Core Data which I think that, jointly with Bindings, will substitute EOF in the near future.

Let us do it as a "tutorial":

- Start Xcode (I am using 2.0) and select File>New Project. In the next panel choose "Cocoa-Java Document-based Application". This generates a project which is ready to use the Java Bridge and implements the NSDocument architecture ---take a look to the generated files and to the target; in particular, double-click on the main target and inspect all details showed.
- In the project window, select the Frameworks folder. From the Project menu, select Add to Project (cmd+alt+a). In the browser, go to System>Library>Frameworks and select JavaEOCocoa.framework; click OK.
- Again, press cmd+alt+a, browse to System>Library>Frameworks then to JavaEOAccess.framework>;Resources>Java and select javaeoaccess.jar; click OK.
- Repeat the previous step for: javaeocontrol.jar, javaeointerface.jar, javaeointerfacecocoa.jar, javafoundation.jar and javaxml.jar.
- Add your model ---double-check that the folder of it is shown in blue colour (instead of yellow).

Basically you are done. However, if you --~~like me~~-- prefer to work in Objective-C, then do the following:

- Start Interface Builder by double-clicking the MyDocument.nib file.
- Click the File's Owner and then the Classes tab. You will notice that the selected class is MyDocument which inherits from NSDocument. Select NSDocument and press "enter" to create a new subclass; give it a name (say, MyCDocument). Then from the Classes menu, select "Create Files..." and accept.
- Return to the Instances tab, select the File's Owner and in the "Custom class" inspector (cmd+5) select the new created class. Save and hide.
- Double-click the main target of your project. Select "Document types" and edit accordingly to the new class (e.g., changing MyDocument for MyCDocument).
- In the .m file created three steps below, add the following:

```
- (NSString *)windowNibName
 {
   return @"MyDocument";
 }
```

Compile and run (cmd+r).

Finally, if you will, you can get rid of the warnings about deprecated methods in the .java file created by the wizard:

- From the project window, delete the MyDocument.java file.
- Check the button "Needs Java" in the "Cocoa Java Specific" pane of your main target ---in the window opened three steps below.

Compile and run (cmd+r).

By this moment, you had created a multi-document CEO...

- StrauszRicardo