EOF-Using EOF-Problems

Problems

This section describes some problems and bugs that sometimes occur when using *Enterprise Objects Framework*.

EOF fails to initialize models when not specifically ordered

Authors: Francis Labrie

Affected products: *WebObjects* 5.2.x, 5.3.x

Bug reference: rdar://4571773

Problem:

Sometimes when having several data models using shared objects in a *WebObjects* application, models load, initialization and connection simply fail with strange and unexpected exceptions. Typical exceptions are:

- `java.lang.IllegalStateException: registeredDatabaseContextForModel() Cannot register the database context for the model<`*Model name*`>`
  `at`
  `com.webobjects.eoaccess.EODatabaseContext.registeredDatabaseContextForModel(EODatabaseContext.java:1145)`

...

- `java.lang.IllegalStateException: addOrderByAttributeOrdering: attempt to generate SQL for com.webobjects.eocontrol.EOSortOrdering <class com.webobjects.eocontrol.EOSortOrdering(`*<Attribute Name>*
  `compareAscending)> failed because attribute identified by key '`*<Attribute Name>*`' was not reachable from from entity '`*<Entity Name>*`'`
  `at com.webobjects.eoaccess.EOSQLExpression.addOrderByAttributeOrdering(EOSQLExpression.java:1803)`
  ...
- *etc.*

I saw several report of this bug in various mailing list, but no one leads to a right solution or a good explanation:

http://lists.apple.com/archives/webobjects-dev/2005/Aug/msg00295.html
http://www.wodeveloper.com/omniLists/webobjects-dev/2004/September/msg00255.html

Solution:

I've finally found that this problem is related to the models load and initialization order. Most of the time, EOF is able to initialize a group of models flawlessly. But in some circumstances, it just fails.

The best fix would be to make EOF dynamically analyse models groups, building a graph of dependancy with all entities for each group and ordering the models and the entities initializations according to this graph. But a such solution is a little bit more complex and should typically be integrated directly into the EOAccess layer.

To work around this bug with a simpler solution, you can add in the `userInfo` dictionary of each model the `priority` key assigned to a value of four digit, example: `0100`, `0500`, `2000`, *etc.* You must use priority values with the same number of digit, because these will be converted to strings, and if you set a priority of `500`, the string value will be considered as higher than a priority of `1000`.

Then create a static method that perform an `assertConnectionDictionaryIsValid()` with each model of the group sorted by priority:

```
private static final NSArray _PriorityDescendingModelSortOrdering = new
NSArray(new Object[] {
 EOSortOrdering.sortOrderingWithKey("userInfo.priority",
EOSortOrdering.CompareDescending),
 EOSortOrdering.sortOrderingWithKey("name",
EOSortOrdering.CompareAscending)
});

...

public static void
assertModelGroupConnectionDictionariesAreValid(EOEditingContext
editingContext, EOModelGroup modelGroup) {
 Enumeration models;
 EOAdaptor adaptor;
 EODatabaseContext databaseContext;
 EOModel model;

 // Get models enumerator from default group
 models = EOSortOrdering.sortedArrayUsingKeyOrderArray(modelGroup.models(),
  _PriorityDescendingModelSortOrdering).objectEnumerator();

 while(models.hasMoreElements()) {
  model = (EOModel)models.nextElement();
  NSLog.debug.appendln("  Connecting " + model.name() + " model, userInfo =
" + model.userInfo() + "...");
  databaseContext =
EODatabaseContext.registeredDatabaseContextForModel(model, editingContext);
  adaptor = databaseContext.adaptorContext().adaptor();

  // Test connection dictionary
  adaptor.assertConnectionDictionaryIsValid();
  NSLog.debug.appendln("  The \"" + model.name() + "\" model is
connected.");
 } // while
} // assertConnectionDictionariesAreValid
```

The sort ordering here sort by descending model priority and ascending model name to ensure constancy on sort result.

You'll have to find the right model order: you can deduce it with logic drawing a dependency graph, or you can find it with tests and errors. Typically, when an exception has above is thrown, it's because the model initialization was done too late. I usually set priority to framework models from `1000` (low) to `9000` (high), and application models from `0100` (low) to `0900` (high).

Exception:

For some cases, this solution still does't help (see this message). I suspect the entities ordering in model to be the problem (see #EOF fails to fetch or save entities when not correctly ordered in model below). If it's still not the case, you can try to use the ERXSharedEOLoader class in Project Wonder.

EOF fails to fetch or save entities when not correctly ordered in model

Authors: Francis Labrie

Affected products: *WebObjects* 5.2.x, 5.3.x

Bug reference:

Problem:

Sometimes when using inherritance with entities in a data model in a *WebObjects* application, fetching or saving data may simply fails with strange and unexpected exceptions. Typical exception is:

- `com.webobjects.eoaccess.EOGeneralAdaptorException: sqlStringForKeyValueQualifier: attempt to generate SQL for` *<Qualifier Class>* `(`*<Qualifier Expression>*`) failed because attribute identified by key 'NeededByEOF0' was not reachable from from entity` *<Entity Name>*
  `at com.webobjects.eoaccess.EODatabaseContext._exceptionWithDatabaseContextInformationAdded(EODatabaseContext.java:4685)`
  ...
- *etc.*

Solution:

Typically, this kind of error occurs when the parent entity is in an another model which was initialized after the current entity model one (see #EOF fails to initialize models when not specifically ordered above) or when the parent entity is coming after the current entity in the alphabetically ordered list of entities of the current model. To illustrate the latter case: if your abstract parent entity name is "Flower" and you are fetching the concrete sub-entity "Anemone" while both entities are part of the same model, you can get this kind of exception on the first "Anemone" fetch.

To avoid this exception, simply edit the "index.eomodeld" file in a text editor and reorder the entities references in the array, putting parent entities above sub-entities. But you should then take care with the *EOModeler*: you'll have to reorder these entities references each time you modify and save the model.

New sub-entity insertion fails because primary key is null

Authors: initially reported by Chuck Hill

Affected products: *WebObjects* 5.2.x, 5.3.x

Bug reference:

Problem:

When a sub-entity have a parent entity with a relationship set to *propagate primary key*, any new sub-entity insertion when saving changes from an editing context will throw an integrity violation exception because the primary key column is set to `null`.

Solution:

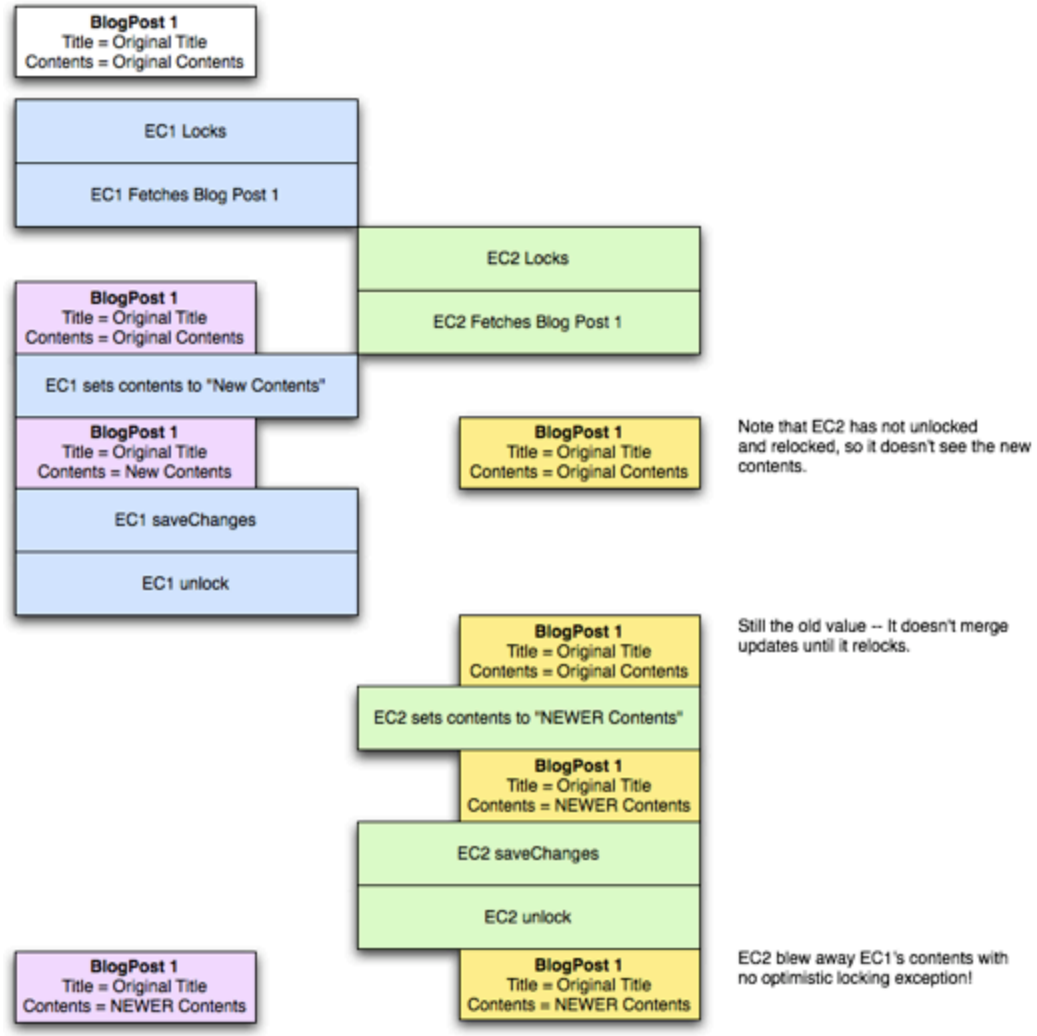
To avoid this bug, uncheck the *Propagate primary key* relationship property of the parent entity. Unfortunately, if the parent entity is not abstract, this solution will not be applicable: you'll probably have to modify the relationship modeling itself.

Strange Locking Behavior

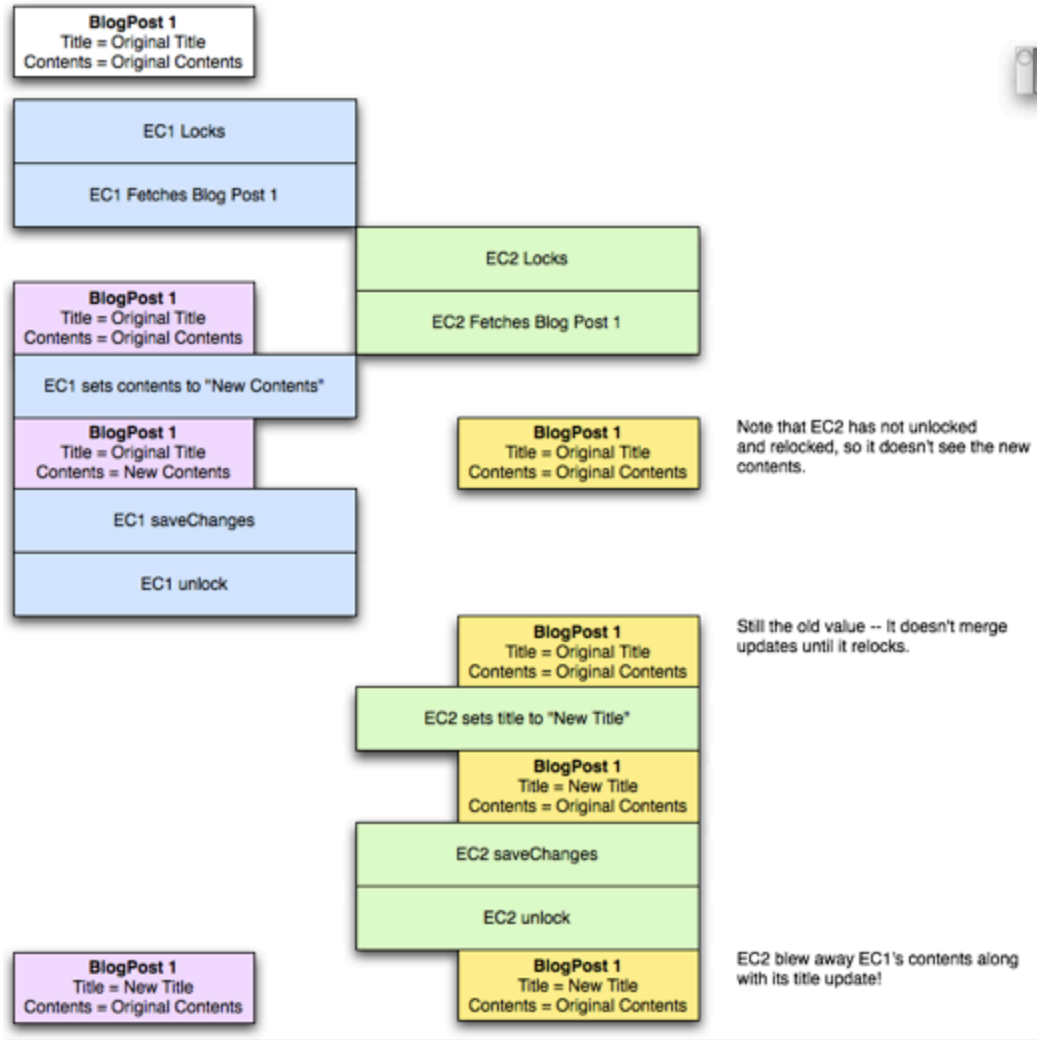
It would appear that there is, in our opinion, some bugs related to optimistic locking within a single EOF stack. Essentially what it boils down to is that it appears that the update database operation that is created as a result of a call to .saveChanges() is backed by the EODatabaseContext snapshot and NOT the "working" snapshot inside in the EO in the editing context it came from. What this means is that while changes are not merged until you .unlock() and .lock() under normal circumstances, because the underlying snapshot that EOF diffs your changes against on save is the DBC snapshot, it's effectively inadvertently "merged" on commit. That is to say that if another EC makes changes and saves, then you make different changes and save, you will blow away their changes with no sign of an optimistic locking exception because your snapshot IS their snapshot now (meaning, it looks like just you are overwriting their changes, versus the reality of the situation that you are actually conflicting with their changes). After discussing this some, we believe that if the update operation used a version of the EO's backing snapshot instead that these weird behaviors would be fixed and it would behave exactly like a normal conflicting update if you were in two EOF stacks. The current behavior smells of bug, but I'm curious if anyone a dissenting opinion on the topic. It's certainly really complicated and nasty down in that code, so it's possible there's some crazy justifiable reason for it.

Here are two diagrams of examples. Left side is EC 1, right side is EC 2, purple is the state of the EO in EC1 at various points, yellow is the state of the same EO in EC2 at various points.

Here's the "blow-away-other-changes-in-the-EO" workflow:



Here's the "no optimistic lock exception" example:


BlogPost 1
Title = Original Title
Contents = Original Contents
EC1 Locks
EC1 Fetches Blog Post 1
EC2 Locks
EC2 Fetches Blog Post 1
BlogPost 1
Title = Original Title
Contents = Original Contents
EC1 sets contents to "New Contents"
BlogPost 1
Title = Original Title
Contents = Original Contents
Note that EC2 has not unlocked and relocked, so it doesn't see the new contents.
BlogPost 1
Title = New Title
Contents = Original Contents
EC1 saveChanges
EC1 unlock
BlogPost 1
Title = Original Title
Contents = Original Contents
Still the old value -- It doesn't merge updates until it relocks.
EC2 sets title to "New Title"
BlogPost 1
Title = New Title
Contents = Original Contents
EC2 saveChanges
EC2 unlock
BlogPost 1
Title = New Title
Contents = Original Contents
BlogPost 1
Title = New Title
Contents = Original Contents
EC2 blew away EC1's contents along with its title update!