EOF-Using EOF-Undo and Redo

Overview

Can anyone tell me if the NSUndoManager is only used with EOF or can I use it in my own application (not using EOF) ?

Undo/Redo can be used with any kind of app (that's why you find NSUndoManager in Foundation and not some higher policy level framework like WOF or EOF). Of course WOF and EOF come preconfigured to use an NSUndoManager WOF automatically creates Undo Groups around the request/response loop (although you can create subgroups to get finer grain control if you like), and EOF's EOEditingContext comes prewired to use an NSUndoManager.

I don't know why people don't use this more... I guess it's inertia. If you only ever use one EOEditingContext in your app and the user clicks Undo, they may get confused about what happens unless you are careful to warn users that they are making an Undo request in a page that isn't active (they backtracked). If users backtrack to an old page and push undo there, the change that gets undone may have happened more recently. To the user nothing would appear to change when in fact something has changed, but they just don't know what. Another reason is that often EC changes are accompanied with UI state changes and undo only applies to changes in the EC. For example, you might want to undo shipping settings on order. Just calling undo would do that, but it wouldn't shift state representing your step in the check out process.

Another issue is that users and developers aren't aware that undoing some things may not be reversible without additional intervention... Like a update to a counter stored in the database. Or they may be reversible, but the user or developer doesn't realize another call to saveChanges is needed to push the undone object graph in the EC back to the DB.

Instead of using Undo, people sometimes use task-based editing contexts combined with revert() and/or EC replacement... that is, you dedicate an EC to the checkout process, and another EC to data browsed in a Store, and another EC for editing account info. If a user doesn't wish to commit changes, you can call revert on the EC discarding all changes since the last commit. However, once you've broken the EC into tasks like this, it may be more clear to you and your users about what happens with Undo/Redo. For example, an Undo in Customer Edit, won't undo your last change in to your Product Management edit EC.

One desirable way to undo/redo in a web app would involve using the NSUndoManager notifications messages (that get called as undo groups are formed). This may allow the Object Graph changes (state that the EOEditingContext & NSUndoManager manage) to be kept in sync with your app specific session state (that you directly manage). It also might give you the leverage to associate the Page where a given change was undone. For example... imagine a user edits some Customer info. The the user moves on to some other customer or some other type of information and edits there two. The user hits a few times. Now suppose any action that calls undo, correlates the Undo group with state stored about that group at the time the undo group was formed? You might be able to recover info about the PK and entity name associated with the page that made the change; this in turn would allow you to take the user back to an appropriate editor page pre-populated with the EOEnterpriseObject object(s) that were chiefly affected by the undo. I've never tried doing something like this, but it might be possible.

None of this would be actually that interesting or needed if a user typically views in just a few areas all the data that could be changed. Undo/Redo impact would be immediately evident. If there are uncommitted changes in an EC, you might want to still flag the user about that so they would know to try to save their changes if needed.

If users can't easily see the impact of undos and redos, another approach you might use would involve summarizing the current content of EOs in the EC's updated, inserted, and deleted lists. This might be a little region at the bottom of the screen that listed counts of these arrays or that listed the Entity names of uncommitted EOs along with the their count. You could even use the changes-from-snapshot API to show the actual changes themselves in any uncommitted updates. When a user does an undo, you might take the user to a page that explodes this change information in all its glory. The user could toggle undo/redo back and forth on such a page and get a clear indication of the scope of the undo in question. If things looked OK, then they could call saveChanges on the EC.

PS: some important docs I want to bring to light in case you Don't use undo/redo much...

Notes

EOEditingContext's undo support is arbitrarily deep; you can undo an object repeatedly until you restore it to the state it was in when it was first created or fetched into its editing context. Even after saving, you can undo a change. To support this feature, the NSUndoManager can keep a lot of data in memory.

For example, whenever an object is removed from a relationship, the corresponding editing context creates a snapshot of the modified, source object. The snapshot, which has a reference to the removed object, is referenced by the editing context and by the undo manager. The editing context releases the reference to the snapshot when the change is saved, but the undo manager doesn't. It continues holding the snapshot, so it can undo the deletion if requested.

If the typical usage patterns for your application generate a lot of change processing, you might want to limit the undo feature to keep its memory usage in check. For example, you could clear an undo manager whenever its editing context saves. To do so, simply send the undo manager a removeAllActions message (or a removeAllActionsWithTarget message with the editing context as the argument). If your application doesn't need undo at all, you can avoid any undo overhead by setting the editing context's undo manager to null with setUndoManager.

More Info

If you are using Undo/Redo, it is important to read the Memory Management section.