Whew!   I haven't this much MiscMail in a long time. [Hi Robert]

I support the idea of forking off a separate thread to deal with NibController Objects/Categories for inclusion in MiscKit; we can then take it as a "given" in our further design.   I also agree that it is not important to our Doc design.   However, we *do* have a design contraint that single-view nib-based documents should be just as easy to implement under the new scheme as they old, so it is something to keep in mind.

[Don, were you referring to Mike's DocView handling or Nib handling when you said you liked the architecture?]

Anyway, let's try this definition thing again, with RTF.   If someone could turn this into CRC cards (or teach me how to use them; I've never done formal OOD!), I would be eternally grateful.

I.   The Model [state] is referred to as a Document.
It holds all the persistent information about a document (and probably most non-persistent info as well.).   It does not interact with the AppKit, and hence generally inherits from Object.   It notifies its *DocController* [DC] when it changes.   For our purposes, we can treat it as a single instance of class Object.

II.   The Document is owned (publicly) by a DocController [DC].

It is 'owned' in the sense that the Doc is created/loadedand freed/saved by the DC.   It is 'public' in the sense that other objects can send the Doc messages without having to go through the DC.   It is associated with a file to save the document into.   It is managed by *DocManager*, as before   It inherits from Object, as it is NOT part of the responder chain.

The important point is that the DC itself *has no window* - or indeed, any explicit AppKit dependence at all.   Put antoher way, the DC it has no UI, only an API.   One could conceivably use this mechanism to load in information which is changed and saved, but never directly accesible to the user.   Getting information to/from the user is handled instead by the *DocViewController (DVC)*.   This split, while perhaps not essential, does increase our flexibility by keeping our objects small and well focused.

So the DCs responsibilities are:
- · Create/Load the Document (app-specific)
- · Create and initialize the DVC (one of a few possibilities)
- · Save/Revert the Document
- · Maintain the association with a file
- · Notify the DVC when the Document changes

To customize the DC, all one needs to to is change the initialization method.   I suppose it might be possible to do this in IB or with a delegate or something, to save on subclassing.

III.   All document views and windows are handled by the DocViewController [DVC].

For simplicity, we will assume that all user interaction takes place via a single view contained in a single window. We will call this view (and its associated window) a DocView for convenience.   It is a subclass of View, and determined entirely by the application.   We don't need to know anything about it for MiscDoc, except to presume that it exists.   The DocView, once it is created, only needs to be able to send messages to the Document.   [The

DocWindow's delegate would presumably be the DC, in case it wanted to receive things like save:sender].

My assertion (if I am wrong, let me know) is that since the rest of the application doesn't care whether you have one or ten views, or whether they are peer-view or master-slave, this information should be encapsulated in a separate class. While related to the issue of documant creation and saving, it is a fundamentally different problem, and thus deserving of a separate class.

To be precise, I would think of DVC as more a protocol than a class, since the inheritance may vary depending on the kind. Different classes of it might be / have a single NibController, or dynamically generate scrollviews, or maintain one top-level and several subsidiary windows. The point is that the rest of the system does not need to know about it. Nor do you need to keep around the extra instance variables.

The DVC does need to know about the DocController (for the window delegate), and the DocManager (to know where to create the window). However, it can get that information from the Document, which is the only thing it needs to be initialized with.

Thus we have the following relationship among the controllers:

paste.eps

The DocManager creates DocControllers in response to New or Open requests. Each DC creates a Document (either directly, or from a file), and then a DVC initialized with the Document. The DVC then creates one or more DocViews, each of which has a connection to the Document. The Document is the DC's delegate, and the DC is the window's delegate. Or, rather perhaps the DVC is the window's delegate, and the DC is the DVC's delegate. Is that too convoluted?

The DocManager keeps track of the activeDocument (via the DocControllers). Each DC, however, has to keep track of the activeWindow (via the DVC). The DVC keeps track of all the DocViews, and can specify the mechanism for whether to close the document based on what DocViews are closed.

I realize this is pretty complicated. And, it won't kill me if you decide to merge the DC and the DVC, though I think the complexity of the connection is offset by the smaller, simpler classes. However, the important question is whether this is a good decompostion of the problem space. Which are classes, subclasses, categories, and protocols, can be decided at implementation time, right? I also assert that this network can, with a suitable implementation of defaults, be as simple to use in the single-nib single-view case as the MOKit classes. It would probably even be possible to make this backward compatible with them, but that could get ugly.

Well, anyway, I hope this clears things up a little bit. I'll be here briefly tomorrow, but after than won't be online until Monday the 21st. I hope to have some time then to put into this, but I can't say for sure. If you decide to go ahead and do it without me, that's fine, too.. I would recommend making the MiscDoc stuff "alpha" if you release it, so that we can do at least one more interation before we "freeze" the implementation.

Thanks for listening. Good luck,

- Ernie P.