SwapItDemo

by Greg Burd

Overview

This example is provided because I believe this to be the best way to learn, and I am a fanatic on support. The code for SwapView does not in itself do an inspector, as used in many applications, by itself. The swap view only does the actual swapping of views (hence the name :-). So I felt an apt example would be something that finishes out the 'inspector' functionality.

Important classes within SwapItDemo

AppDelegate class

This is the Application delegate (connected as such in IB), and as such responds to the - appDidInit:sender method. There it loads the info panel and inspector panel. I wait to alloc, and init the InspectorController until absolutely necessary. That way memory and processor time is only used when requested. Someone might not ever use the inspector so why alloc it. This has a -free method because all objects should clean up properly, there is no need to do so in this object, but it is the thought that counts.

** new in 2.0 **
I have added a menu for command keys. This finishes out the basic functionality of an inspector. To do this I had to add one method to the AppDelegate. This - showInspector:sender method will get the title of the title in the menu and pass that to the InspectorController (later I talk about the new method in there to handle the other end of this).

InspectorController class
** new in 2.0 **
I put all the separate inspector panels in one .nib so as to make it extremely easy to use in your code (and easier for me to make, and to leave something for you to do). This class needs very little modification to make it do all that other 'large' applications do. The one area that is left to you is the separate inspector panels are included in this .nib, this should be changed so each panel has its own .nib, and the FilesOwner will load the nib section and control everything. This is also useful for having one inspector, a content inspector, that can inspect different things such as text, images, files, etc. Have all the related panels for the content inspector selection in one nib, and then simply return the correct id for the situation. There is one other small element to finish out the separation of the inspector panels. In the - whatPanel call which the delegate will answer (here it is the InspectorController) you should have outlets for the controlling classes of the separated nib sections which contain the inspectors. So what panel will

determine which inspector has been requested, and then checks to see if it is NULL, and then askes that object to return the id of the inspector panel to be swapped in.

Example: **(in the delegate to the SwapView)**

```
- whatPanel
{
    if(theTitle) {

    ...

    if (!strcmp(theTitle,"Info..."))
            /* see if the infoController is NULL */
            if(!infoController) {
                    /* if not then create it now */
                    infoController = [[Info-Inspector alloc] init]];
            /* and ask it to return the correct panel id */
            return [infoController whatPanel];

    if (!strcmp(theTitle,"First Inspector"))
            /* see if the firstController is NULL */
            if(!firstController) {
                    /* if not then create it now */
```

```
            firstController = [[First-Inspector alloc] init]];
        /* and ask it to return the correct panel id */
        return [firstController whatPanel];

...
```

This allows you to load the nib when the user moves to that inspector. The first time that particular inspector is selected it will be slightly slower, then from then on out there should be no slow down. This means that you will only use the memory to hold the inspector panels that the user has been using and not waste memory that the user never needs. This maintains a PopUp which has its target as self, and action as popUp. When we get a pop up call the matrix of the pop up is still visible, and there are two methods of discovering the chosen string. By looking at the matrix's selected cell and its stringValue, or waiting for the pop up button to return and then look at the button title. I choose the former for reasons of speed (this is changed from 1.0). The real meat of this class is in the delegate method -whatPanel. When this is called, I look at pointer to a string, and switch accordingly. For all the work seen on the screen there is very little to do in this class. Most of the work is in SwapView.
When the user requests an inspector via command keys, the AppController gets the action and in turn sends a call to InspectorController to `-inspectName:(char *)str`. At that point InspectorController sets up its pointer to a string and then

swaps out the views.

Interesting Stuff

AppController

How to wait to load nib sections until it is necessary.

InspectorView

How to set up the target action of a pop up. How to deal with the pop up and the fact that the matrix will remain on screen until the action method returns. How to properly answer the delegate method -whatPanel.

Greg Burd
gburd@nmsu.edu

-greg