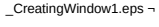
;CreatingDynamicallyLoadableBundles.rtfd;;¬ Next Section ;DisplayingAnAttentionPanel.rtfd;;¬ Previous Section

CreatingAWindowWithMultipleDisplays;¬Creating a window with multiple displays

1. **In a nib file, create a window with an empty box in the place where the display should change.**
2. **Add control objects that allow the user to change the display, and hook them to action methods in a controller class.**
3. **For each display, use Document arrow.eps ¬ New Modules arrow.eps ¬ New Empty to create a nib file containing a window with just that display.**
4. **Assign the controller to be the nib file's owner and connect one of its outlets to the display.**
5. **In the action method's implementation, load the appropriate nib file.**

One common interface style is to have a window whose display changes upon a user action, such as clicking a button. For example, Interface Builder's Inspector panel changes its display when you choose a different item in the pop-up list. Another example is Project Builder's Preferences panel. Both of these panels have infrequently-used displays, thus it makes sense to store these displays in nib files that are loaded only if needed.



The key to creating a window with multiple displays is the content view attribute. NSBox, NSScrollView, and NSWindow all have a content view attribute. The content view is the superview of all of the view objects, such as button and text fields, inside of the box, scroll view, or window. You can send a **setContentView:** message to a box to swap out the entire contents of the box and replace them with new contents. The rest of this task uses the window shown above to show you how to create a window with multiple displays.

You can define each of the window's displays in separate nib files. In the main nib file, place a box in the area

that you want to be changeable.

_CreatingWindow2.eps ¬

Also in the main nib file, define the controller class. Give the controller class an outlet for each view (in this example, Business Info, Personal Info, and Notes) plus outlets for the main window and the box on the main window. Also define an action for the controller class, named something like **setContents:**, and connect the pop-up list to that action.

Next, use the New Empty command to create a nib file for each of the displays that the window can show. In each of these nib files, create a window by dragging one from the Windows palette. Your application never displays these windows; they exist only to hold the view objects that the main window will display.

_CreatingWindow3.eps ¬

Tip: Use the Size display of the Inspector panel to make sure that this box is just smaller than the box on the main window. It also helps to make the auxiliary nib file's window the same size as the main nib file's window.

Now you need to connect this display to the controller class. Add the controller class to the nib file and assign it to the File's Owner object. Then, connect File's Owner to the box you just created.

_CreatingWindow4.eps ¬

In this example, we would create two more auxiliary nib files, one for the ªPersonal Infoº display and one for the ªNotesº display, in the same manner as the **BizInfo.nib** file shown above.

Note: If one of the displays is going to be used frequently, you might want to create it in an off-screen panel in the main nib file rather than creating a separate nib file and incurring the overhead of reading it in.

> ªGrouping objectsº in Chapter€2 describes how to group objects inside of a box. ;../../02_CreatingTheInterface/02_Composing/GroupingObjects.rtfd;;¬ ªSetting box (group) attributesº in Chapter€3 describes the box's Attributes display in the Inspector panel. ;../../02_CreatingTheInterface/03_SettingObjectAttributes/SettingBoxAttributes.rtfd;;¬

After all of the nib files have been created, implement the action method that you connected to the pop-up list.

In this example, the action method is named **setContents:**. Its implementation is shown here.

```
- (void)setContents:(id)sender
{
      switch((InfoType)[[sender selectedItem] tag]) {
        case BUSINESS:
          if (!bizView) {
              [NSBundle loadNibNamed:@"BizInfo" owner:self];
              [bizView retain];
          }
          [theBox setContentView:bizView];
          break;
        case PERSONAL:
          if (!persView) {
              [NSBundle loadNibNamed:@"PersInfo" owner:self];
              [persView retain];
          }
          [theBox setContentView:persView];
          break;
        case NOTES:
          if (!notesView) {
              [NSBundle loadNibNamed:@"Notes" owner:self];
              [notesView retain];
          }
          [theBox setContentView:notesView];
          break;
      }
}
```

Notes on the code: Based on the pop-up list's selection, this method loads the appropriate nib file, if necessary, then sets the contents of the box to the view defined in that nib file. As a view becomes the box's content view, it is removed from the window in its nib file. The **setContentView:** method releases the box's previous content view and retains the new one. To ensure that the previous content view isn't deallocated before the next user event,

you must retain each view. By preventing the views from being deallocated, you allow your users to switch back and forth between them. (Be sure to release the views in your class's **dealloc** method.)

> This method uses an enumerated type (InfoType) to give meaning to the pop-up list items' tags. For more information on using tags, see ªUsing tagsº in Chapter€3, ªSetting an Object's Attributes.º ;../../02_CreatingTheInterface/03_SettingObjectAttributes/UsingTags.rtfd;;¬

Finally, you need to have the Business Info display appear when the application starts up. To set this up, in the main nib file assign your Controller class to be the NSApplication delegate (NSApplication is the main nib file's owner). Implement the delegate method **applicationDidFinishLaunching:** as shown:

```
- (void)applicationDidFinishLaunching:(NSNotification *)notify
{
    [self setContents:popUp];
    [mainWindow makeKeyAndOrderFront:self];
}
```

Notes on the code: This method is invoked immediately after the NSApplication object has finished initializing itself. It invokes **setContents:** to load the nib file for the default display (Business Info) and set the contents of the box on the main window to be the view defined in that nib file. Then it displays the main window.

Tip: Make sure that the main window's ªVisible at launch timeº attribute is deselected. Otherwise, there will be a slight lag between the time the window appears on the screen and the time that the box's contents appear on screen.