Preferences

Library: None, this API is defined by the Preferences application

Header File Directory: /NextDeveloper/Headers/apps

Import: apps/Preferences.h

Introduction

The Preferences application lets the user customize system features to agree with personal preferences. By clicking each button in turn at the top of the Preferences window, the user can reveal groups of controls for setting mouse, keyboard, font, and other preferences. Programmatically, these displays are provided by modules that Preferences loads into itself. With the API described in this chapter, you can create additional modules that can be added to the ones that are commonly displayed in Preferences.

A Preferences module contains three components: a TIFF image for the button that represents the new display, a nib file containing the interface for the display, and a file containing the code linking the interface to the Preferences application. When Preferences€begins running, it locates modules to be loaded by searching these locations in€the order listed:

~/Library/Preferences
/LocalLibrary/Preferences
/NextLibrary/Preferences
/NextApps/Preferences.app

It looks for bundles with names of the form ª*MyModule*.preferencesº. When it locates such a bundle, it loads the executable code from the bundle and adds a new button to the scrolling list at the top of the Preferences window. When a user clicks the button, the new module's interface is displayed in the lower portion of the Preferences window. Notice that Preferences checks its own file package for modules; this is in fact how it loads the modulesÐMouse Preferences, Keyboard Preferences, Localization Preferences, and so onÐthat appear on all systems.

The Preferences application and loadable module communicate through the API found in **/NextDeveloper/Headers/apps/Preferences.h**. This API consists of the declarations of the Layout class and a category of Application. The Layout class is an abstract superclass that defines the owner of the module's interface. The methods declared in the Application category make it easier for your module to load its interface and to control Preferences' menu commands.

Building a Preferences Module

Building a module is easy, especially since you're provided with a template (in **/NextApps/Preferences.app/Template.bproj**) to be modified. This template module contains:

File	Description

PB.project	Project file for the loadable module
Template.h	Class interface file
Template.m	Class implementation file
Template.tiff	TIFF image for button in Preferences window
English.lproj/Template.nib	Nib file contain user-interface for this module
Makefile	Instructions used by the **make** utility

To build a Preferences module, make a copy of the template directory and rename the components of the new directory to reflect the nature of the module. For example, for a module that lets the user specify a mantra to be played continuously in the background, you might use these names:

```
Mantra.bproj/
    Mantra.h
    Mantra.m
    Mantra.tiff
    English.lproj
        Mantra.nib
```

Add each of these files to the project in the appropriate place and remove the references to the files having the root name ªTemplateº. Next, using Project Builder's Attributes display, change the name of the project to ªMantraº. Finally, open the class files and replace any reference to ªTemplateº with ªMantraº.

At this point, you can build the project and test the template module. Using Project Builder, build the project. When the process is complete, rename the resultant ªMantra.bundleº file to ªMantra.preferencesº and double-click it. Preferences will load the sample module.

Now that process is clear, you can begin adapting the Mantra module to its specific purpose by modifying the project's nib, class, and TIFF files.

Some Requirements and Considerations

Preferences modules are bundle files and so must adhere to the naming requirements for bundles. Specifically, the bundle file package and the executable file within the package must have the same root name. For the Mantra example above, this implies that if the file package is named ªMantra.preferencesº, the executable file within it is named ªMantraº. (See the description of the NXBundle class for more information.)

Preferences also uses this root name to identify the TIFF image for the button that's added to the Perference window and to identify the principal class within the bundle's executable file. (Thus, the example has ªMantra.tiffº, and the class is named ªMantraº.)

Since the code you write is loaded into the Preferences application, there's a potential for name collisions. For example, if you create a Preferences module called ªMouse.preferencesº (which would of course define the Mouse class, **Mouse.tiff**, and **Mouse.nib**), these components would conflict with those in the standard module **/NextApps/Preferences.app/Mouse.preferences**. To be safe, the root name for your module could have a distinctive prefix, for example.

Finally, the subclass of Layout within your module must be the principal class of the bundleÐthat is, the object file containing the code for this class must be listed first on the **ld** command line that created the bundle. The easiest way to specify this is within Project Builder's Files display. Make sure the the class file (for example, **Mantra.m**) is the first entry under ªClassesº. If it isn't, Control-drag the class file to the top of the list.