Dialogs and Special Panels

Cocoa > User Experience

2009-02-04

Contents

Figures and Listings

Introduction to Dialogs and Special Panels

This topic describes alert dialogs and other special panels.

An alert notifies users about some event and, optionally, lets them choose how the application responds to that event. Alerts come in two forms: as a free-floating dialog and as a sheet. An alert dialog is application-modal and unattached to any window; it refers to a condition affecting the application as a whole. An alert sheet is a dialog that is attached to a particular window (usually representing a document) and is modal to that window. You can find additional information about alert sheets in the programming topic *Sheet Programming Topics for Cocoa*.

The “Dialogs” in *Apple Human Interface Guidelines* chapter in *Aqua Human Interface Guidelines* also provides information about the various kinds of alert dialogs and the conditions suited to their use.

Organization of This Document

This topic has the following tasks:

- “Displaying Alert Dialogs” (page 11) describes how to display alert dialogs.
- “Customizing Alert Dialogs” (page 13) discusses how to add accessory views and suppression buttons to alert dialogs and manage them.
- “Displaying Alert Help” (page 15) describes how to display help information related to alert dialogs.

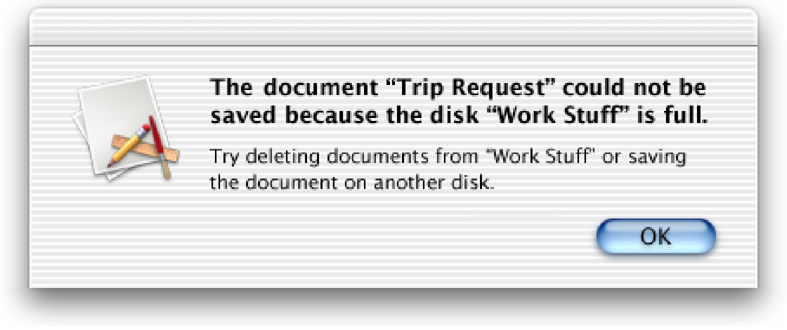
And here’s where you can find information on other types of special panels:

- Creating a Font Panel
- Choosing Colors With Color Wells and Color Panels
- Using a Save Panel
- Using an Open Panel
- Using a Print Panel
- Using a Page Setup Panel

Types of Alerts

There are two types of alerts, standard and caution. Most alerts should be standard alerts, which display the application icon of the current application, as shown in Figure 1.

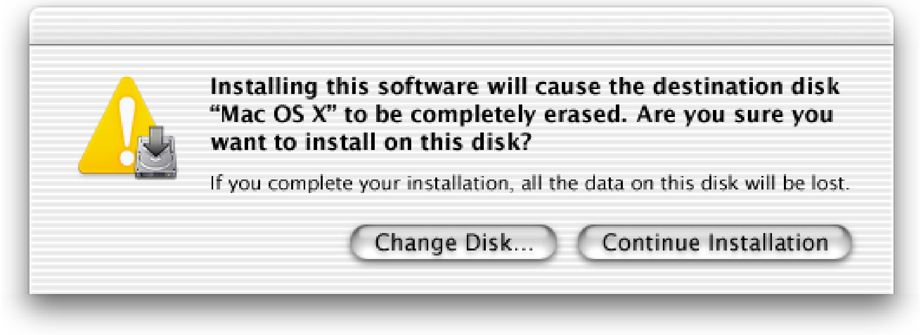
Figure 1 A standard alert with an application icon


The document "Trip Request" could not be saved because the disk "Work Stuff" is full.

Try deleting documents from "Work Stuff" or saving the document on another disk.

Use a caution alert only to warn the user when a possible side effect of the current task is the inadvertent destruction of data. A caution alert displays a caution icon badged with application icon, as shown in Figure 2.

Figure 2 A caution alert with the caution icon


Installing this software will cause the destination disk "Mac OS X" to be completely erased. Are you sure you want to install on this disk?

If you complete your installation, all the data on this disk will be lost.

How you specify the alert type varies according to programmatic interface:

- **NSAlert**. Send `setAlertStyle:` to an NSAlert object with an argument of `NSWarningAlertStyle` or `NSInformationalAlertStyle` to specify a standard alert. Send the same message with an argument of `NSCriticalAlertStyle` to specify a caution alert.
- **Functional API**. Use the `NSBeginAlertSheet` function to display a standard alert and `NSBeginCriticalAlertSheet` to display a caution alert.

Caution alerts should be used only as specified in the "Alerts" section of *Apple Human Interface Guidelines*.

Displaying Alert Dialogs

Alert dialogs inform the user of an event and present buttons that let the user choose how to proceed. You may implement alert dialogs by invoking methods of the `NSAlert` class and by calling special functions.

Important: The `NSAlert` class is available in Mac OS X version 10.3 and later.

Using the NSAlert API is the preferred approach if you are writing applications that run on Mac OS v10.3 (or later). You thereby gain the advantages of an object-oriented model as well as additional features, such as the ability to display help information related to the alert dialog.

Using the NSAlert Class

Using the `NSAlert` API to display an alert dialog involves three simple steps: creating and initializing an `NSAlert` instance, running the dialog, and interpreting and acting on the user's choice.

1. Create the `NSAlert` object though the standard Objective-C `alloc`-and-`init` procedure. Then send the required `NSAlert` "setter" messages to initialize the alert. Listing 1 (page 11) gives an example of this.

Listing 1 Creating and initializing the NSAlert object

```
NSAlert *alert = [[NSAlert alloc] init];
[alert addButtonWithTitle:@"OK"];
[alert addButtonWithTitle:@"Cancel"];
[alert setMessageText:@"Delete the record?"];
[alert setInformativeText:@"Deleted records cannot be restored."];
[alert setAlertStyle:NSWarningAlertStyle];
```

2. Invoke the `runModal` method on the `NSAlert` object.

3. Test the result return from `runModal` and proceed accordingly.

   Listing 2 (page 11) illustrates the last two steps.

Listing 2 Running the dialog and interpreting the result

```
if ([alert runModal] == NSAlertFirstButtonReturn) {
    // OK clicked, delete the record
    [self deleteRecord:currentRec];
}
[alert release];
```

The return code is an `enum` constant identifying the button on the dialog that the user clicked. The first button added to the dialog (which, in left-to-right scripts, is the one closest to the right edge) is identified by `NSAlertFirstButtonReturn`. The second button that is added appears just to the left of the first and is identified by `NSAlertSecondButtonReturn` —and so forth for the third button.

Make sure that you release the `NSAlert` instance.

You can also create an `NSAlert` object directly from an `NSError` object using the `alertWithError:` method. You can then modally run the alert, thereby presenting the error information to the user. The method uses the localized description, recovery suggestion, and recovery options encapsulated by the `NSError` object for the alert's message text, informative text, and button titles, respectively.

As a convenience for compatibility with the older functional API (see "Using the Functional APIs" (page 12)), you can create `NSAlert` objects with the class factory method `alertWithMessageText:defaultButton:alternateButton:otherButton:informativeTextWithFormat:`. This method allows you to retain the earlier constants used to identify the button clicked. Here is an example of how you might invoke this method (with the previous example in mind):

```
NSAlert *alert = [NSAlert alertWithMessageText:@"Delete the record?"
    defaultButton:@"OK" alternateButton:@"Cancel"
    otherButton:nil informativeTextWithFormat:
    @"Deleted records cannot be restored."];
```

Using the Functional APIs

You can also call functions to create and display alerts. There are three kinds of functions:

- To create and run an alert panel, use `NSRunAlertPanel`, `NSRunCriticalAlertPanel`, or `NSRunInformationalAlertPanel`.
- To create and run an alert sheet, use `NSBeginAlertSheet`, `NSBeginCriticalAlertSheet`, or `NSBeginInformationalAlertSheet`.
- To create an alert panel, use `NSGetAlertPanel`, `NSGetCriticalAlertPanel`, or `NSGetInformationalAlertPanel`.

Customizing Alert Dialogs

Beginning with Mac OS X v10.5 (Leopard), `NSAlert` includes methods for displaying and managing accessory views and managing the suppression check box. The following sections describe how to use these features.

Managing Accessory Views

An alert may have an accessory view, a view that contains controls and possibly other objects related to the alert. For example, an alert could warn a user that the file they're copying will replace an existing file of the same name; it might have a checkbox for an accessory view that, if checked, results in the application saving the existing file under a slightly different name (for example, buy appending "_save" to the filename).

You set an alert dialog's accessory view using the `setAccessoryView:` method. You may create the view programmatically, but a more common approach is to compose the view in Interface Builder. In the header file for the custom class, declare an outlet for the accessory view, and then connect this outlet in Interface Builder. By default, `NSAlert` positions the accessory view below the alert's informative text and the suppression button (if any) and above the alert buttons, left-aligned with the informative text.

Listing 1 illustrates how to display an alert dialog that has both an accessory view and a suppression button, and how to handle the user's response in both cases. Comments call out lines that are of particular interest to one or the other feature. (You can learn about suppression buttons in "Managing the Suppression Button" (page 14).)

Listing 1 Displaying an alert with an accessory view and a suppression button

```
static BOOL runAgain = YES; // Suppression button: static var holds current value of
suppression button

- (void)showRecordDeleteAlert:(id)sender {
   if (runAgain == NO) // Suppression button: if user doesn't want to see alert, return
        return;
    NSAlert *alert = [[NSAlert alloc] init];
    [alert addButtonWithTitle:@"Delete"];
    [alert addButtonWithTitle:@"Extend"];
    [alert setMessageText:@"Delete the record?"];
    [alert setInformativeText:@"Deleted records cannot be restored.
           You may extend the valid-until date if you wish."];
    [alert setAlertStyle:NSWarningAlertStyle];
    [alert setShowsSuppressionButton:YES]; // Suppression button: show it
    [alert setAccessoryView:myView];  // Accessory view: "my" accessed via an outlet
connection
    NSInteger result = [alert runModal];

    if ( result == NSAlertFirstButtonReturn ) {
        // "Delete" clicked
        [self deleteRecord:currentRec];
    } else if ( result == NSAlertSecondButtonReturn ) {  // Accessory view: handle
user-specified data
        // "Extend" clicked
        NSDate *chosenDate = [myDatePicker dateValue];
        if ([chosenDate laterDate:[NSDate date]] == chosenDate) {
            [self setValidDate:chosenDate ofRecord:currentRec];
        }
    }
    runAgain = (BOOL)![[alert suppressionButton] state]; // Suppression button: get
state of button
    [alert release];
}
```

If you want to customize the location of the accessory view, first call `layout` and then do any special positioning and sizing of the accessory view prior to running the alert. This sequence overrides the default behavior where `NSAlert` lazily lays out the accessory view just before showing the panel. You should call `layout` only after you have finished configuring the alert with message, informative text, buttons, and, if desired, suppression button. If you do any custom layouts, be advised that the default layout of the alert could change in future releases.

Managing the Suppression Button

Alert dialogs may include a suppression button, which is a checkbox with a default title (localized) of "Do not show this message again." (A checkbox is a type of button.) If the user selects the checkbox, the application should not display the alert again during the current application session. The suppression button is a feature intended to improve the user experience by allowing the application to suppress alerts about relatively trivial matters. The application is responsible for determining whether an alert should be displayed based on the user's choice; in other words, suppression doesn't happen automatically.

You add a suppression button to an alert dialog by sending the `NSAlert` object a `setShowsSuppressionButton:` message with an argument of `YES`. Before running the alert dialog, you can fetch the suppression button by calling `suppressionButton` and customize the button in certain ways. For example, you could customize the default title using the following nested message expression:

```
[[alert suppressionButton] setTitle:@"My new button title."];
```

You can also call `suppressionButton` and then set the initial state of the button or fetch its current state using the `setState:` and `state` methods of `NSButton`. After an alert is dismissed, you use the latter method to determine whether the user checked the suppression checkbox. The code example in Listing 1 (page 13) illustrates a simple way—that is, the use of a static variable—to record the value of the suppression button and later check it before running the alert dialog again. However, a more useful approach might be to store the state of the suppression button in user defaults and later check it before showing the alert again.

By default, the suppression button is positioned below the informative text and above the accessory view (if any) and the alert buttons; it is left-aligned with the informative text. However do not count on the placement of this button, since it might change in the future. If you need a checkbox for purposes other than alert suppression, it is recommended you create an accessory view that is, or contains, a checkbox.

Displaying Alert Help

The NSAlert class includes several methods that enable you to display help information related to an alert dialog or sheet. You can either use the application's NSHelpManager object to find and display information using the Help Viewer application, or you can provide your own means for displaying help information.

Important: The NSAlert class is available in Mac OS X version 10.3 and later.

An alert dialog or sheet advertises that help is available with a round question-mark button. You request the display of this button by sending `setShowsHelp:` to the NSAlert object with an argument of `YES`. To actually display the help, you have two options:

- Specify a help anchor, which the NSHelpManager object can use to find the help text to display in Help Viewer.

  Specify the help anchor by invoking NSAlert's `setHelpAnchor:` method.
- Set a delegate for the NSAlert object and implement the delegate method `alertShowHelp:`. The delegate is responsible for displaying help information related to the alert.

Listing 1 shows how you might initialize an NSAlert object for the second help option.

Listing 1 Setting the help button and delegate for an alert dialog

```
NSAlert *alert = [[NSAlert alloc] init];
// other initializations here ...
[alert setShowsHelp:YES];
[alert setDelegate:self];
```

Listing 2 illustrates an implementation of the NSAlert `alertShowHelp:` delegate method.

Listing 2 Implementing the delegate method for displaying alert help

```
- (BOOL)alertShowHelp:(NSAlert *)alert {
    NSString *path = [[NSBundle mainBundle] pathForResource:@"Help"
ofType:@"html"];
    BOOL flag = [[NSWorkspace sharedWorkspace] openFile:path];
    return flag;
}
```

If your application has more than one alert dialog or sheet for which it displays help, it should test the NSAlert object passed into this method to determine the help text to display. Always return `YES` unless the display of help did not succeed.

Document Revision History

This table describes the changes to *Dialogs and Special Panels*.

Date	Notes
2009-02-04	Updated for Mac OS X v10.5: added "Customizing Alert Dialogs," which describes the suppression checkbox and accessory views.
2003-08-01	Updated for NSAlert, introduced in Mac OS X v. 10.3: added "Displaying Alert Help" (page 15) and updated "Displaying Alert Dialogs" (page 11) (title changed from "Using Alert Panels"). Also added definition of application-modal alert dialogs (versus alert sheets) in introduction.
2002-11-12	Revision history was added to existing topic. It will be used to record changes to the content of the topic.