
Sheet Programming Topics for Cocoa

User Experience: Windows & Views



2009-05-04



Apple Inc.
© 2002, 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Cocoa, Mac, Mac OS, and Objective-C are trademarks of Apple Inc., registered in the United States and other countries.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS

PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction to Sheets 7

Organization of This Document 7

About Sheets 9

When to Use Sheets 9
Sheets and Delegation 10
Working With Sheets 10

Types of Alerts 11

Using Alert Sheets 13

Using the UIAlertView Class 13
Using the Functional API 14

Displaying Alert Help 17

Using Custom Sheets 19

Presenting a Series of Sheets 21

Sheet Notifications 23

Positioning Sheets 25

Using Application-Modal Dialogs 27

Document Revision History 29

Figures and Listings

About Sheets 9

Figure 1 An example of a sheet 9

Types of Alerts 11

Figure 1 A standard alert with an application icon 11

Figure 2 A caution alert with the caution icon 11

Using Alert Sheets 13

Figure 1 Alert to confirm deletes 16

Listing 1 Creating and initializing the UIAlertView object 13

Listing 2 Displaying the alert sheet 13

Listing 3 Interpreting the result in the modal delegate method 14

Listing 4 The deleteSelectedRows: method implemented to present a sheet with two buttons 15

Displaying Alert Help 17

Listing 1 Setting the help button and delegate for an alert dialog 17

Listing 2 Implementing the delegate method for displaying alert help 17

Using Custom Sheets 19

Listing 1 Displaying a custom sheet 19

Listing 2 Closing a custom sheet 19

Listing 3 Did-end selector 20

Presenting a Series of Sheets 21

Figure 1 Alert sheet with three buttons 21

Listing 1 The deleteSelectedRows: method implemented to present a sheet with three buttons 21

Listing 2 Displaying the custom sheet 22

Positioning Sheets 25

Listing 1 Positioning a sheet right under a text field 25

Using Application-Modal Dialogs 27

- Listing 1 Displaying an application-modal dialog 27
- Listing 2 Closing an application-modal dialog 27

Introduction to Sheets

A sheet is simply a dialog attached to a specific window, ensuring that a user never loses track of which window the dialog belongs to. Sheets can be dialogs that require information from the user (such as a Save dialog) or they can be alerts that provide messages about error conditions or warn users of potentially hazardous actions (such as a Save Before Quitting alert).

Alert sheets are document-modal whereas standard alert dialogs are application-modal. A standard alert dialog appears in its own window (technically, an `NSPanel` object) and allows no user action in the application until the user dismisses the alert. See *Dialogs and Special Panels* for a discussion of standard alert dialogs.

The “Dialogs” in *Apple Human Interface Guidelines* section in *Apple Human Interface Guidelines* discusses sheets from the point of view of how they relate to other Mac OS X user interface objects.

Cocoa developers interested in using sheets in their application should read this document.

Organization of This Document

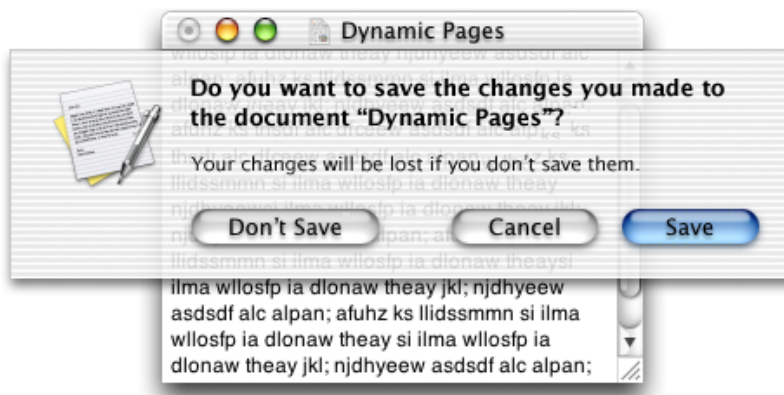
This programming describes sheets and how they work, and provides examples on how you can use sheets in your applications. It contains the following articles:

- [“About Sheets”](#) (page 9) provides basic information about sheets.
- [“Types of Alerts”](#) (page 11) describes the types of alerts and how they are used.
- [“Using Alert Sheets”](#) (page 13) describes how to present an alert sheet.
- [Displaying Alert Help](#) (page 17) describes how to display help information associated with an alert sheet.
- [“Using Custom Sheets”](#) (page 19) describes how to create and present a custom sheet.
- [“Presenting a Series of Sheets”](#) (page 21) describes how to present a series of sheets.
- [“Sheet Notifications”](#) (page 23) describes the notifications sent, along with their related delegate methods, while working with a sheet.
- [“Positioning Sheets”](#) (page 25) describes how to position a sheet within its window.
- [“Using Application-Modal Dialogs”](#) (page 27) describes how to create and present an application-modal dialog.

About Sheets

A sheet is simply a dialog attached to a specific window, ensuring that a user never loses track of which window the dialog belongs to. The ability to keep a dialog attached to its pertinent window enables users to take full advantage of the Mac OS X window layering model and also encourages modelessness; users can work on other documents or in other applications while a sheet is open.

Figure 1 An example of a sheet



A sheet is **document modal**—that is while it is open the user is prevented from doing anything else in the window, or document, until the dialog is dismissed. In contrast, a dialog that is **application modal** prevents the user from doing anything else within the application.

Managing the animation of sheets, and making sure the sheet appears properly if the parent window is narrower than the sheet or near the edge of the screen, is provided “for free” by Cocoa.

Cocoa provides an API for presenting sheets. Because sheets are document modal, these calls return immediately after presenting a sheet. Callback methods are used to let your application know when the user dismisses a sheet.

When to Use Sheets

Use sheets for dialogs specific to a document when the user interacts with the dialog and dismisses it before proceeding with work. Some examples of when to use sheets:

- A modal dialog that is specific to a particular document, such as saving or printing. The Cocoa classes `NSSavePanel` and `NSPrintPanel` present their views as sheets.
- A modal dialog that is specific to a single-window application that does not create documents. A single-window utility program might use a sheet to request acceptance of a licensing agreement from the user, for example.

- Other window-specific dialogs typically dismissed by the user before proceeding. Use a sheet when a dialog benefits from being attached to the window as a modal dialog, even if you might otherwise design the dialog as a modeless dialog.

Sheets and Delegation

Because sheets in Cocoa are document modal, when you display a sheet, normal program execution continues, allowing the user to do other things in other windows. This means your application must be prepared to handle user interaction when it occurs.

When you display a sheet, you specify selectors for the callback methods sent when the user dismisses the sheet, as well as the receiver of these messages, known as the **modal delegate**. The modal delegate is informed of the button the user clicked as a parameter in the method it receives.

Unlike other delegates in Cocoa, modal delegates in sheets are temporary and the relationship lasts only until the sheet is dismissed. The sheet's modal delegate is not retained by any document-modal method or function.

Working With Sheets

Sheets are laid out like any other dialog in Mac OS X. You are responsible for loading, showing, and closing sheets. While a sheet is displayed, events are handled by the Application Kit just as for any other window. Other sheet behavior, such as the animation when it appears and is dismissed, is handled automatically by the Application Kit.

Each sheet function takes as an argument a modal delegate and one or more callback methods. All calls specify a callback method that is sent before dismissing the sheet, sometimes referred to as the did-end selector. Some sheet functions also allow for a second callback method that is sent after dismissing the sheet, known as the did-dismiss selector. When you display a sheet you can optionally include a context-info argument. When the sheet ends, the modal delegate is sent the did-end selector, receiving as parameters the sheet window object, the result of running the sheet (either a Boolean value or a return code), and the context-info argument. context-info is a way for you to pass information from the start of the modal session to the end, and this information can be whatever you wish: a simple value, a structure, or an object.

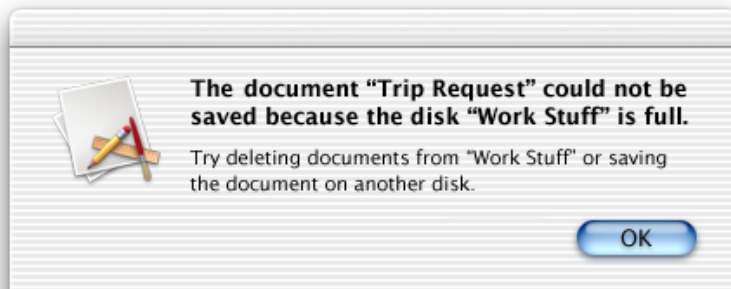
The did-end selector is sent before dismissing the sheet, providing you the opportunity to dismiss the sheet and the parent document window at the same time. For example, you might want to do this when dismissing a “do you want to save” alert before closing a window. If the user clicks Don't Save, you want to close both the document window and the sheet without the sheet effect. This can be accomplished by calling `[documentWindow close]` from the did-end selector. You can also dismiss just the sheet in this method by calling `[sheet orderOut: self]`. If you do not dismiss the sheet, it will be done for you on return from the did-end selector. You may be in a situation, however, where you want to immediately present another sheet, and this is best done from the did-end selector after dismissing the first sheet.

You may find it convenient to implement both the did-end selector and the did-dismiss selector, but it is not required.

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



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



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” in *Apple Human Interface Guidelines* section of *Apple Human Interface Guidelines*.

Using Alert Sheets

Alert sheets are used by almost every application. Cocoa makes it easy to create them. This task demonstrates how to quickly add an alert sheet to your application. See the “Dialogs” in *Apple Human Interface Guidelines* section of *Apple Human Interface Guidelines* for details on when a sheet should be used.

Cocoa gives you two ways to create and display alert sheets. You can use the Application Kit’s functional API for alert sheets, or you can use the methods of the `NSAlert` class. The latter approach is recommended for applications built for Mac OS X v10.3 and later; `NSAlert` not only brings the advantages of the object-oriented model, it introduces new features, such as the ability to display help related to the alert. This document explains both approaches.

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

Using the `NSAlert` Class

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

1. Create the `NSAlert` object through the standard Objective-C `alloc-and-init` procedure. Then send the required `NSAlert` “setter” messages to initialize the alert. [Listing 1](#) (page 13) gives an example of this.

Listing 1 Creating and initializing the `NSAlert` object

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

2. Invoke the `beginSheetModalForWindow:modalDelegate:didEndSelector:contextInfo:` method on the `NSAlert` object ([Listing 2](#)). This displays the sheet attached to the specified window.

Listing 2 Displaying the alert sheet

```
[alert beginSheetModalForWindow:[searchField window] modalDelegate:self
didEndSelector:@selector(alertDidEnd:returnCode:contextInfo:) contextInfo:nil];
```

The modal-delegate parameter must be an object that implements a method with the following signature:

```
- (void)alertDidEnd:(NSAlert *)alert returnCode:(NSInteger)returnCode
contextInfo:(void *)contextInfo;
```

The `UIAlertView` object invokes this method when the user clicks a button on the alert sheet to indicate his or her choice (which consequently dismisses the sheet). The modal delegate is not the recipient of any other delegation messages; it is the delegate *only* for the current modal session. The context-info parameter is for any data you wish to pass to the modal delegate. Instead of autoreleasing the `UIAlertView` object when you create it, you may release the object in this method, if you wish.

3. Implement the modal-delegate method to identify the user's choice and proceed accordingly (Listing 3).

Listing 3 Interpreting the result in the modal delegate method

```
- (void)alertViewDidEnd:(UIAlertView *)alertView returnCode:(NSInteger)returnCode
contextInfo:(void *)contextInfo {
    if (returnCode == UIAlertViewFirstButtonReturn) {
        [self deleteRecord:currentRec];
    }
}
```

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 `UIAlertViewFirstButtonReturn`. The second button that is added appears just to the left of the first and is identified by `UIAlertViewSecondButtonReturn`—and so forth for the third button.

As a convenience for compatibility with the older functional API (see “Using the Functional API” (page 14)), you can create `UIAlertView` objects with the class factory method `alertViewWithMessageText: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):

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

Using the Functional API

The application assumed for this example is a table of expense report entries similar to a spreadsheet. The `deleteSelectedRows:` method is sent when the user tries to delete selected rows in the expense report. `deleteSelectedRows:` asks the user's permission to delete the rows by displaying an alert sheet using the `NSBeginAlertSheet` function.

`NSBeginAlertSheet` has a fairly long list of parameters, but the function is not difficult to use. Here is a summary of the parameters:

title

The main text of the alert, which appears in the emphasized system font.

defaultButton

The label for the sheet's default button; the button label should correspond to the action that will result from pressing the button—for example, “Save,” “Erase,” or “Delete.”

alternateButton

The label for the sheet's alternate button. If you pass `nil`, only the *defaultButton* is displayed.

otherButton

The label for the sheet's other button. If you pass `nil`, only the *defaultButton* and *alternateButton* are displayed.

documentWindow

A reference to the `NSWindow` the sheet is attached to.

modalDelegate

A reference to the object acting as the sheet's delegate.

didEndSelector

A selector for a method implemented by *modalDelegate*. This method is sent when the modal session is ended, but before the sheet is dismissed. If you don't need this capability, pass `NULL`.

didDismissSelector

A selector for a method implemented by *modalDelegate*. This method is sent after the sheet is dismissed in the event your application might need to perform additional processing. If you don't need this capability, pass `NULL`.

contextInfo

Additional information you can define to pass to *modalDelegate* as a parameter of *didEndSelector* and *didDismissSelector*.

message

Optional additional text that appears in the sheet. The text appears in the small system font. This string can contain `printf`-style escape sequences.

optionalParameters

The `printf`-style parameters used to format *message*.

So, the implementation of `deleteSelectedRows:` would simply look as shown in Listing 4.

Listing 4 The `deleteSelectedRows:` method implemented to present a sheet with two buttons

```
- (BOOL)deleteSelectedRows: (NSWindow *)sender
{
    NSBeginAlertSheet(
        @"Do you really want to delete the selected rows?",
        // sheet message
        @"Delete", // default button label
        nil, // no third button
        @"Cancel", // other button label
        sender, // window sheet is attached to
        self, // we'll be our own delegate
        @selector(sheetDidEndShouldDelete:returnCode:contextInfo:),
        // did-end selector
        NULL, // no need for did-dismiss selector
        sender, // context info
        @"There is no undo for this operation.");
        // additional text

    // We don't know if the rows should be deleted until the user responds,
    // so don't.

    return NO;
}
```

When the user attempts to delete the selected expenses, the sheet shown in Figure 1 drops down from the window title bar.

Figure 1 Alert to confirm deletes

The implementation of the did-end selector `sheetDidEndShouldDelete:returnCode:contextInfo:`, sent when the user clicks a button is as follows:

```
- (void)sheetDidEndShouldDelete: (NSWindow *)sheet
    returnCode: (NSInteger)returnCode
    contextInfo: (void *)contextInfo
{
    if (returnCode == NSAlertDefaultReturn)
        // delete selected rows here
}
```

If the user clicks the Cancel button, the value of `returnCode` is `NSAlertOtherReturn`. If a third button were provided, its return value would be `NSAlertAlternateReturn`.

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.

Using Custom Sheets

One type of sheet is an alert sheet, which was discussed in [“Using Alert Sheets”](#) (page 13). If the type of information presented in alert sheets is not suitable for your application, you can create and present a custom sheet.

When working with custom sheets, you are responsible for displaying as well as dismissing the sheet. You display a sheet with the `beginSheet:...` method, and you end a sheet with the `endSheet:` method. Between these two methods, your sheet is operating modally.

You create your custom sheet using Interface Builder. It is important to remember to include a button to allow the user to dismiss the sheet when they are finished with it.

This discussion assumes that the sheet is in a separate nib file called `MyCustomSheet`. A Close button is defined on the sheet. The Close button is set to perform the `closeCustomSheet:` method when clicked.

The `showCustomSheet:` method displays the custom sheet modal to the window passed as a parameter. The arguments to `beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:` are similar to those for the `NSBeginAlertSheet` function, but the Close button on the custom sheet controls dismissing the sheet. A did-end selector is specified to handle any activity necessary before dismissing the sheet.

Listing 1 Displaying a custom sheet

```
- (void)showCustomSheet: (NSWindow *)window

// User has asked to see the custom display. Display it.
{
    if (!myCustomSheet)
//Check the myCustomSheet instance variable to make sure the custom sheet does
not already exist.
        [NSBundle loadNibNamed: @"MyCustomSheet" owner: self];

        [NSApp beginSheet: myCustomSheet
                 modalForWindow: window
                 modalDelegate: self
                 didEndSelector: @selector(didEndSheet:returnCode:contextInfo:)
                 contextInfo: nil];

        // Sheet is up here.
        // Return processing to the event loop
}
```

When the user clicks the Close button, the following method is executed (this was specified in the nib file when creating the sheet).

Listing 2 Closing a custom sheet

```
- (IBAction)closeMyCustomSheet: (id)sender
{
```

```
[NSApp endSheet:myCustomSheet];  
}
```

Control is sent to the did-end selector, which cleans up by closing the custom sheet. It is important to call `orderOut:` when finished with your sheet, or it is not removed from the screen.

Listing 3 Did-end selector

```
- (void)didEndSheet:(NSWindow *)sheet returnCode:(NSInteger)returnCode  
contextInfo:(void *)contextInfo  
{  
    [sheet orderOut:self];  
}
```

Presenting a Series of Sheets

Cocoa does not support the notion of cascading, or nested sheets. Per *Apple Human Interface Guidelines*, “when the user responds to a sheet, and another sheet for that document opens, the first sheet must close before the second one opens.”

Expanding on the sample in “Using Alert Sheets” (page 13), Figure 1 shows a third button added to the alert sheet displayed by `deleteSelectedRows:`. This button displays additional information to the user in the form of a custom sheet with additional information.

Figure 1 Alert sheet with three buttons



Listing 1 shows the implementation of the method that creates this alert.

Listing 1 The `deleteSelectedRows:` method implemented to present a sheet with three buttons

```
- (BOOL)deleteSelectedRows: (NSWindow *)sender
{
    NSBeginAlertSheet(
        @"Do you really want to delete the selected information?",
        // sheet message
        @"Delete", // default button label
        @"More Info", // allow user to check on the delete
        @"Cancel", // other button label
        sender, // window sheet is attached to
        self, // we'll be our own delegate
        @selector(sheetDidEndShouldDelete:returnCode:contextInfo:),
        // did-end selector
        NULL, // no need for did-dismiss selector
        sender, // context info
        @"There is no undo for this operation.",
        // additional text in sheet
        nil); // no parameters in message

    // We don't know if the rows should be deleted until the user responds,
    // so don't.

    return NO;
}
```

The `didEnd` selector is updated to handle the user requesting the additional information. Should the user request the information, the alert sheet is closed prior to presenting the custom sheet:

```
- (void)sheetDidEndShouldClose: (NSWindow *)sheet
    returnCode: (NSInteger)returnCode
    contextInfo: (void *)contextInfo
{
    if (returnCode == NSAlertAlternateReturn) {
        [sheet close];
        [self showMoreInfo: (NSWindow *)contextInfo];
    }
    if (returnCode == NSAlertDefaultReturn)
        // delete selected rows here
}
```

Creation and display of custom sheets is discussed in [“Using Custom Sheets”](#) (page 19).

In a separate nib, a window with the explanatory text and a Close button are defined. The Close button is set to perform the `showMoreInfo:` method when clicked. Implementation of the `showMoreInfo:` method would be the same as the `showCustomSheet:` method shown in [Listing 1](#) (page 19).

Listing 2 Displaying the custom sheet

```
- (void)showMoreInfo: (NSWindow *)window
// User has asked for more information about the delete. Display it.
{
    if (!moreInfoSheet)
        [NSBundle loadNibNamed: @"DeleteExpenseInfo" owner: self];

    [NSApp beginSheet: moreInfoSheet
        modalForWindow: window
        modalDelegate: nil
        didEndSelector: nil
        contextInfo: nil];
    [NSApp runModalForWindow: moreInfoSheet];
    // Sheet is up here.
    [NSApp endSheet: moreInfoSheet];
    [moreInfoSheet orderOut: self];
}
```

When the user clicks the Close button, the `showMoreInfo:` method is executed (this was specified in the nib file when creating the sheet), which stops the application’s modal display of the nested sheet.

Another example of a second sheet that appears as the result of the user clicking a button in a sheet is having a sheet with a progress indicator appear following a Save As dialog.

Sheet Notifications

`NSWindow` offers a set of notifications related to sheets, which it broadcasts on occurrences of a sheet opening or closing. Each notification is matched to a delegate method, so an `NSWindow`'s delegate is automatically registered for all notifications that it implements methods for.

`NSWindowWillBeginSheetNotification` is sent before a sheet is presented on a window and `NSWindowDidEndSheetNotification` after it is dismissed.

A window delegate should implement the following methods to receive the appropriate sheet notification:

- (void)windowWillBeginSheet:(NSNotification *)notification;
- (void)windowDidEndSheet:(NSNotification *)notification;

It is important to note that a window's delegate is *not* the same as the modal delegate specified as a parameter in the `NSBegin...Alert` calls. The modal delegate passed into the `NSBegin...Alert` calls is a delegate relationship that exists only until the sheet is dismissed.

Positioning Sheets

A sheet does not have to be welded to its default location just below the title bar. You can position where a sheet appears attached to a window by implementing the delegate method `window:willPositionSheet:usingRect:`, which an `NSWindow` object invokes just before it animates the sheet.

Important: The `window:willPositionSheet:usingRect:` method is available in Mac OS X version 10.3 and later.

The method does more than position the sheet on its window. It can also determine whether the sheet animation originates from a particular object or area of the window. With `window:willPositionSheet:usingRect:` you can, for example, have an alert sheet appear to emerge from the text field associated with the condition described in the alert; you could also position the sheet so that it is centered just under the text field. You might also implement this method to have the sheet placed just below a window's tool bar rather than its title bar.

Even though the method uses `NSRect` structures to specify the location of the sheet—one passed in as the default location and another the new (returned) location—the `NSRect` does not define the rectangle occupied by the sheet. (All sheets are of a standard size in relation to their window.) Instead the `NSRect` structure, particularly the structure's `origin` member (an `NSPoint` structure), specify where the top-left corner of the sheet is attached to the window (in window coordinates). The `size.width` member of the `NSRect` specifies the width of the initial animation (`size.height` is currently undefined).

The basic implementation of `window:willPositionSheet:usingRect:` is straightforward. It passes in an `NSRect` structure specifying the default location of the sheet. You return an `NSRect` structure specifying the new sheet location and the width of initial animation.

Listing 1 shows how a window delegate might implement `window:willPositionSheet:usingRect:` to have the sheet animate from a text field (`fooField`) and then become attached to the window right below the text field.

Listing 1 Positioning a sheet right under a text field

```
- (NSRect)window:(NSWindow *)window willPositionSheet:(NSWindow *)sheet
    usingRect:(NSRect)rect {
    NSRect fieldRect = [fooField frame];
    fieldRect.size.height = 0;
    return fieldRect;
}
```

Note that, as in the example, it is recommended that you set the `size.height` member of the returned `NSRect` to zero.

If the window delegate is managing multiple windows and multiple sheets, it should test the first and second arguments of the method to determine which window and sheet is involved and thus which sheet location is appropriate.

Using Application-Modal Dialogs

You may occasionally find it necessary to use an application-modal dialog rather than a document-modal sheet. Recall that an application-modal dialog prevents the user from doing anything else within the owning application, although the user can switch to another application.

Working with application-modal dialogs is very similar to working with custom sheets. When working with application-modal dialogs, you are responsible for displaying as well as dismissing the dialog. You display a dialog with the `beginSheet:...` method, and you end an application-modal dialog with the `endSheet:` method. Between these two methods, your dialog is operating application modally.

You create your dialog using Interface Builder. It is important to remember to include a button to allow the user to dismiss the dialog when they are finished with it.

This discussion assumes that the dialog is in a separate nib file called `MyCustomDialog`. A Close button is defined on the dialog. The Close button is set to perform the `closeMyCustomDialog:` method when clicked.

The `showCustomDialog:` method displays the dialog modal to the window passed as a parameter. The arguments to `beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo:` are similar to those for the `NSBeginAlertSheet` function, but the Close button on the dialog controls dismissing the dialog. After calling `beginSheet...` , the application continues to execute until it encounters `runModalForWindow:` . The user is still allowed to interact with the application-modal dialog (it wouldn't make sense not to allow this), but activity in the rest of the application is suspended while this dialog is presented.

Listing 1 Displaying an application-modal dialog

```
- (void)showCustomDialog: (NSWindow *)window
// User has asked to see the dialog. Display it.
{
    if (!myCustomDialog)
        [NSBundle loadNibNamed: @"MyCustomDialog" owner: self];

    [NSApp beginSheet: myCustomDialog
             modalForWindow: window
             modalDelegate: nil
             didEndSelector: nil
             contextInfo: nil];
    [NSApp runModalForWindow: myCustomDialog];
    // Dialog is up here.
    [NSApp endSheet: myCustomDialog];
    [myCustomDialog orderOut: self];
}
```

When the user clicks the Close button, the following method is executed (this was specified in the nib file when creating the dialog), which stops the application's modal display of the application-modal dialog.

Listing 2 Closing an application-modal dialog

```
- (IBAction)closeMyCustomDialog: (id)sender
```

```
{  
    [NSApp stopModal];  
}
```

Control is returned to `showCustomDialog:`, which cleans up by closing the dialog. It is important to call `orderOut:` when finished with your dialog, or it is not removed from the screen.

Document Revision History

This table describes the changes to *Sheet Programming Topics for Cocoa*.

Date	Notes
2009-05-04	Updated callback signatures to use NSInteger; removed Java information.
2006-09-05	Corrected typos; changed title from "Sheets."
2004-05-27	Updated "Using Custom Sheets" (page 19) to use a document-modal sheet rather than an application-modal dialog. Added "Using Application-Modal Dialogs" (page 27).
2003-07-31	Updated to cover the NSAlert class and the NSWindow method <code>window:willPositionSheet:usingRect:</code> , both introduced in Mac OS X version 10.3. Specifically, this includes:
	Addition of "Positioning Sheets" (page 25).
	Modification of "Using Alert Sheets" (page 13), "Introduction to Sheets" (page 7), and "Types of Alerts" (page 11) (renamed from "Types of Alert Sheets").
	Addition of link to Displaying Alert Help (page 17) in the Dialogs and Special Panels programming topic.
2003-04-01	"Using Custom Sheets and Cascading Sheets" split into "Using Custom Sheets" (page 19) and "Presenting a Series of Sheets" (page 21). Sample code in "Using Alert Sheets" (page 13) rewritten to better handle two versus three button sheet. Added "Using Alert Sheets in Java". Cleaned up numerous typos.
2002-11-12	Revision history was added to existing topic.

