Drag and Drop Programming Topics for Cocoa

Interapplication Communication

2006-06-28

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

Simultaneously published in the United States and Canada.

Contents

Introduction to Drag and Drop 7

Organization of This Document 7

Dragging Sources 9

Drag Operations 9
Drag Messages 10
The Dragged Image 10

Dragging Destinations 11

The Sender of Destination Messages 11
The Dragging Pasteboard 11
The Order of Destination Messages 11

Receiving Drag Operations 13

Dragging Files 15

Dragging File Paths 15
Dragging File URLs 16
Dragging File Contents 17
Dragging File Promises 18

Using Drag and Drop in Tables 21

Configuring Your Table View 21
Beginning a Drag Operation 22
Validating a Drag Operation 23
Accepting a Drop 23
Customizing Drag Behavior 24
Background Drags 24

Frequently Asked Questions 25

HFS Promise Drag 25
- How Do I Set a Custom Drag Image When Doing an HFS Promise Drag in Cocoa? 25
- How Do I Add Other Pasteboard Types to an HFS Promise Drag in Cocoa? 26

Cross-Application Drag and Drop 26
- When I Rebuild My application on Mac OS X version 10.2, My NSTableView Loses the Ability to Drag-And-Drop to Other Applications. How Do I Fix This? 26

Tables and Listings

Dragging Sources 9

Table 1 Available drag operations 9
Table 2 Drag operations selected with modifier keys 10

Using Drag and Drop in Tables 21

Listing 1 Registering the table's supported data types 21
Listing 2 Initiating a drag from a table. 22

Introduction to Drag and Drop

Cocoa gives you the ability to implement sophisticated drag-and-drop capabilities both within your application and between applications. This programming topic describes how you can implement drag-and-drop with just a few methods.

Organization of This Document

In the text here and in the dragging protocol descriptions, the term dragging session is the entire process during which an image is selected, dragged, released, and absorbed or rejected by the destination. A dragging operation is the action that the destination takes in absorbing the image when it is released. The dragging source is the object that "owns" the image that is being dragged; it is specified as an argument to the method that instigates the dragging session.

Dragging is a visual phenomenon. To be the source or destination of a dragging operation, an object must represent a portion of screen real estate; thus, only window and view objects can be the sources and destinations of drags. (Note that the source view is not necessarily the same objects as the dragging source defined above.) NSWindow and NSView provide methods that handle the user interface for dragging an object. You only need to implement a few methods from either the NSDraggingSource or NSDraggingDestination protocol, depending on whether your window or view subclass is the source or destination.

The dragging protocols are described in these articles:

- "Dragging Sources" (page 9)
- "Dragging Destinations" (page 11)

How to receive a drag is described in these articles:

- "Receiving Drag Operations" (page 13)
- "Dragging Files" (page 15)

Dragging support in table views is described in this article:

- Using Drag and Drop in Tables (page 21)

Commonly-asked questions about drag-and-drop are addressed in this article:

- "Frequently Asked Questions" (page 25)

Dragging Sources

A dragging session is initiated by the user clicking the mouse inside a window or view and moving the mouse. NSView and NSWindow implement the method `dragImage:at:offset:event:pasteboard:source:slideBack:` to handle the dragging session. You invoke this method in the `mouseDown:` or `mouseDragged:` method of your subclass of NSView or NSWindow. You provide an image to display during the drag, a pasteboard holding the data, and an object that acts as the "owner", or dragging source, of the data. During the dragging session, the dragging source is sent messages defined by the NSDraggingSource protocol to perform any necessary actions, described below.

Note: NSView also implements the convenience method `dragFile:fromRect:slideBack:event:` for when the dragged object represents a file. NSView then handles the image, pasteboard, and source messages itself.

Drag Operations

Only one of the NSDraggingSource methods needs to be implemented: `draggingSourceOperationMaskForLocal:`. This method declares what types of operations the source allows to be performed. Table 1 (page 9) lists the available drag operations. (In Java, the constants are defined in the NSDraggingInfo namespace and lack the `NS` prefix.) The method should return a bitwise-OR combination of the allowed types or `NSDragOperationNone` if no operations are allowed.

Table 1 Available drag operations

Dragging Operation	Meaning
`NSDragOperationCopy`	The data represented by the image can be copied.
`NSDragOperationLink`	The data can be shared.
`NSDragOperationGeneric`	The operation can be defined by the destination.
`NSDragOperationPrivate`	The operation is negotiated privately between the source and the destination.
`NSDragOperationMove`	The data can be moved.
`NSDragOperationDelete`	The data can be deleted.
`NSDragOperationEvery`	All of the above
`NSDragOperationAll`	Deprecated. Use `NSDragOperationEvery` instead.
`NSDragOperationNone`	No drag operations are allowed.

The allowed operations may differ if the drag is occurring entirely within your application or between your application and another. The flag passed to `draggingSourceOperationMaskForLocal:` indicates whether it is a local, or internal, drag.

The user can press modifier keys to further select which operation to perform. If the control, option, or command key is pressed, the source's operation mask is filtered to only contain the operations given in Table 2 (page 10). To prevent modifiers from altering the mask, your dragging source should implement `ignoreModifierKeysWhileDragging` and return `YES`.

Table 2 Drag operations selected with modifier keys

Modifier Key	Dragging Operation
Control	`NSDragOperationLink`
Option	`NSDragOperationCopy`
Command	`NSDragOperationGeneric`

Drag Messages

During the course of the drag, the source object is sent a series of messages to notify it of the status of the drag operation. At the very beginning of the drag, the source is sent the message `draggedImage:beganAt:`. Each time the dragged image moves, the source is sent a `draggedImage:movedTo:` message. Finally, when the user has released the mouse button and the destination has either performed the drop operation or rejected it, the source is sent a `draggedImage:endedAt:operation:` message. The *`operation`* argument is the drag operation the destination performed or `NSDragOperationNone` if the drag failed. (In Java, these method names are `startedDraggingImage`, `movedDraggingImage`, and `finishedDraggingImage`.)

The dragging source generally does not need to implement any of these methods. If you are going to support the `NSDragOperationMove` or `NSDragOperationDelete` operations, however, you do need to implement `draggedImage:endedAt:operation:` to remove the dragged data from the source. (Note that an `NSDragOperationDelete` operation is requested when dragging any object to the Trash icon in the dock.)

The Dragged Image

The image that is dragged in a dragging session is simply an image that represents the data on the pasteboard. Although a dragging destination can access the image, its primary concern is with the pasteboard data that the image represents—the dragging operation that a destination ultimately performs is on the pasteboard data, not on the image itself.

When the dragging session is started by using the NSView method `dragFile:fromRect:slideBack:event:`, NSView uses the file's Finder icon for the image. For your own custom drags, you need to construct a suitable image. Possibilities include a semi-transparent snapshot of the displayed data, such as the selected section of text, or a symbolic representation of the data, such as a table icon when dragging spreadsheet data.

Dragging Destinations

To receive drag operations, you must register the pasteboard types that your window or view will accept by sending the object a `registerForDraggedTypes:` message, defined in both NSWindow and NSView, and implement several methods from the NSDraggingDestination protocol. During a dragging session, a candidate destination receives NSDraggingDestination messages only if the destination is registered for a pasteboard type that matches the type of the pasteboard data being dragged. The destination receives these messages as an image enters, moves around inside, and then exits or is released within the destination's boundaries.

Although NSDraggingDestination is declared as an informal protocol, the NSWindow and NSView subclasses you create to adopt the protocol need only implement those methods that are pertinent. (The NSWindow and NSView classes provide private implementations for all of the methods.) Either a window object or its delegate may implement these methods; however, the delegate's implementation takes precedence if there are implementations in both places.

The Sender of Destination Messages

Each of the NSDraggingDestination methods sports a single argument: *sender*, the object that invoked the method. Within its implementations of the NSDraggingDestination methods, the destination can send NSDraggingInfo protocol messages to *sender* to get more information on the current dragging session, such as querying for the dragging pasteboard or the source's operations mask. In Java, *sender* is an NSDragDestination object, which implements the NSDraggingInfo interface.

The Dragging Pasteboard

Although a standard dragging pasteboard (obtained using `[NSPasteboard pasteboardWithName:NSDragPboard]`) is provided as a convenience in getting the pasteboard for dragging data, there is NO guarantee that this will be the pasteboard used in a cross-process drag. Thus, to guarantee getting the correct pasteboard, your code should use `[sender draggingPasteboard]`.

The Order of Destination Messages

The six NSDraggingDestination methods are invoked in a distinct order:

- As the image is dragged into the destination's boundaries, the destination is sent a `draggingEntered:` message. The method should return a value that indicates which dragging operation the destination will perform.
- While the image remains within the destination, a series of `draggingUpdated:` messages are sent. The method should return a value that indicates which dragging operation the destination will perform.

- If the image is dragged out of the destination, `draggingExited:` is sent and the sequence of NSDraggingDestination messages stops. If it re-enters, the sequence begins again (with a new `draggingEntered:` message).
- When the image is released, it either slides back to its source (and breaks the sequence) or a `prepareForDragOperation:` message is sent to the destination, depending on the value returned by the most recent invocation of `draggingEntered:` or `draggingUpdated:`.
- If the `prepareForDragOperation:` message returned `YES`, a `performDragOperation:` message is sent.
- Finally, if `performDragOperation:` returned `YES`, `concludeDragOperation:` is sent.

Receiving Drag Operations

This document shows sample code which allows a view (or window) to accept dragging sessions for several data types, performing different drag operations based on the dragged type. The sample implementation can accept either a color or a file. A dragged color will be copied while the file will be either linked or copied.

Before a view can receive a drag operation, you need to register the data types that it can accept by invoking its `registerForDraggedTypes:`, like this:

```
[self registerForDraggedTypes:[NSArray arrayWithObjects:
            NSColorPboardType, NSFilenamesPboardType, nil]];
```

Now, any time a dragging session that consists of either of these data types enters the views bounds, the view is sent a series of NSDraggingDestination messages. The code below is a simple example of the initial method that gets sent: `draggingEntered:`. The method obtains the dragging pasteboard and available drag operations from the *`sender`* object. If the pasteboard contains color data and the source object permits dragging, the method returns `NSDragOperationGeneric`, indicating that the destination permits dragging of the color data on the pasteboard. If the pasteboard contains a file name and the source object permits linking, the method returns `NSDragOperationLink`, indicating that the destination permits the link. If the source does not allow linking, the destination also checks if a copy operation is allowed instead, returning `NSDragOperationCopy` if so. Failing all these tests, the method returns `NSDragOperationNone`.

```
- (NSDragOperation)draggingEntered:(id <NSDraggingInfo>)sender {
    NSPasteboard *pboard;
    NSDragOperation sourceDragMask;

    sourceDragMask = [sender draggingSourceOperationMask];
    pboard = [sender draggingPasteboard];

    if ( [[pboard types] containsObject:NSColorPboardType] ) {
        if (sourceDragMask & NSDragOperationGeneric) {
            return NSDragOperationGeneric;
        }
    }
    if ( [[pboard types] containsObject:NSFilenamesPboardType] ) {
        if (sourceDragMask & NSDragOperationLink) {
            return NSDragOperationLink;
        } else if (sourceDragMask & NSDragOperationCopy) {
            return NSDragOperationCopy;
        }
    }
    return NSDragOperationNone;
}
```

As the dragging session continues, the destination is sent `draggingUpdated:` messages. You only need to implement this if the destination needs to know the current position of the dragged image, either to change the dragging operation or to update any visual feedback, such as an insertion point, you are providing. If not implemented, NSView assumes the dragging operation is unchanged from the `draggingEntered:` invocation. If the dragging session leaves the views bounds, the `draggingExited:` method is invoked. Implement this if you need to clean up after one of the preceding messages, such as removing the visual feedback.

When the image is dropped with a drag operation other than `NSDragOperationNone`, the destination is sent a `prepareForDragOperation:` message followed by `performDragOperation:` and `concludeDragOperation:`. You can cancel the drag by returning `NO` from either of the first two methods.

You do the bulk of the data handling in the `performDragOperation:` method; the other two methods are implemented only if necessary. The following code sample shows a possible implementation of this method. The method again checks the pasteboard for the available data and, if necessary, it also checks the dragging source's operation mask for the available operations.

```
- (BOOL)performDragOperation:(id <NSDraggingInfo>)sender {
    NSPasteboard *pboard;
    NSDragOperation sourceDragMask;

    sourceDragMask = [sender draggingSourceOperationMask];
    pboard = [sender draggingPasteboard];

    if ( [[pboard types] containsObject:NSColorPboardType] ) {
        // Only a copy operation allowed so just copy the data
        NSColor *newColor = [NSColor colorFromPasteboard:pboard];
        [self setColor:newColor];
    } else if ( [[pboard types] containsObject:NSFilenamesPboardType] ) {
        NSArray *files = [pboard propertyListForType:NSFilenamesPboardType];

        // Depending on the dragging source and modifier keys,
        // the file data may be copied or linked
        if (sourceDragMask & NSDragOperationLink) {
            [self addLinkToFiles:files];
        } else {
            [self addDataFromFiles:files];
        }
    }
    return YES;
}
```

Dragging Files

When dragging files, the dragging pasteboard can transfer the files in four different ways. The pasteboard can hold a list of file paths, a single URL, a file's complete contents, or a promise to create files at a location to be determined by the destination. Each style corresponds to a separate NSPasteboard data type and has a different method for reading and writing the dragged data. The following sections describe each style and how to handle them, whether you are the dragging source or destination.

Dragging File Paths

The most common and simplest method for dragging files is to transmit a list of the files' paths. The dragging source creates an NSArray containing NSString objects of all the paths of the files to be dragged and places the array onto the pasteboard with the data type `NSFilenamesPboardType`. The dragging destination then reads the array from the pasteboard and performs the requested operation using the paths that it holds.

To initiate a drag operation for a single file, you can use the NSView method `dragFile:fromRect:slideBack:event:` when the user clicks in the view representing the file. The first argument is the path of the file to drag. This method places the file's path onto the dragging pasteboard with the `NSFilenamesPboardType` pasteboard type and starts the drag operation.

To initiate a drag operation on multiple files, you need to use the NSView or NSWindow method `dragImage:at:offset:event:pasteboard:source:slideBack:`. You must place the array of file paths onto the pasteboard yourself, using the NSPasteboard method `setPropertyList:forType:`. The following sample code shows a possible implementation.

```
NSString *filePath1, *filePath2; // Assume these exist

- (void)mouseDown:(NSEvent *)theEvent
{
    NSImage *dragImage;
    NSPoint dragPosition;

    // Write data to the pasteboard
    NSArray *fileList = [NSArray arrayWithObjects:filePath1, filePath2, nil];
    NSPasteboard *pboard = [NSPasteboard pasteboardWithName:NSDragPboard];
    [pboard declareTypes:[NSArray arrayWithObject:NSFilenamesPboardType]
            owner:nil];
    [pboard setPropertyList:fileList forType:NSFilenamesPboardType];

    // Start the drag operation
    dragImage = [[NSWorkspace sharedWorkspace] iconForFile:filePath1];
    dragPosition = [self convertPoint:[theEvent locationInWindow]
                        fromView:nil];
    dragPosition.x -= 16;
    dragPosition.y -= 16;
    [self dragImage:dragImage
            at:dragPosition
```

```
            offset:NSZeroSize
            event:theEvent
            pasteboard:pboard
            source:self
            slideBack:YES];
}
```

After a drag operation is dropped, the dragging destination receives a `performDragOperation:` message. To extract the `NSFilenamesPboardType` data from the pasteboard, use the `propertyListForType:` method. Even if only one file is being dragged, the file's path is stored in an NSArray.

```
- (BOOL)performDragOperation:(id <NSDraggingInfo>)sender
{
    NSPasteboard *pboard = [sender draggingPasteboard];

    if ( [[pboard types] containsObject:NSFilenamesPboardType] ) {
        NSArray *files = [pboard propertyListForType:NSFilenamesPboardType];
        int numberOfFiles = [files count];
        // Perform operation using the list of files
    }
    return YES;
}
```

Dragging File URLs

Files can also be specified by their URLs. A file's URL is stored in a pasteboard with the type `NSURLPboardType`. Unlike the `NSFilenamesPboardType`, which holds an array of file paths, the `NSURLPboardType` type holds a single NSURL object. It is not possible to store more than one URL on the pasteboard using this pasteboard type, so you cannot drag more than one file with a URL.

To initiate a drag operation on a file using its URL, you need to use the NSView or NSWindow method `dragImage:at:offset:event:pasteboard:source:slideBack:`. You must place the file's URL onto the pasteboard yourself, using the NSURL method `writeToPasteboard:`. You must declare the `NSURLPboardType` before calling this method, though. This allows you to place both a file path and a file URL onto the pasteboard. The drag operation can then be dropped on destinations that registered for either drag type, or both. The following sample code shows a possible implementation for writing the data to the pasteboard.

```
// Write data to the pasteboard
NSURL *fileURL; // Assume this exists
NSPasteboard *pboard = [NSPasteboard pasteboardWithName:NSDragPboard];
[pboard declareTypes:[NSArray arrayWithObject:NSURLPboardType] owner:nil];
[fileURL writeToPasteboard:pboard];
```

See "Dragging File Paths" (page 15) for more complete sample code on starting the drag operation.

After a drag operation is dropped, the dragging destination receives a `performDragOperation:` message. To extract the `NSURLPboardType` data from the pasteboard, use the NSURL class method `URLFromPasteboard:`.

```
- (BOOL)performDragOperation:(id <NSDraggingInfo>)sender
{
    NSPasteboard *pboard = [sender draggingPasteboard];
```

```
    if ( [[pboard types] containsObject:NSURLPboardType] ) {
        NSURL *fileURL = [NSURL URLFromPasteboard:pboard];
        // Perform operation using the file's URL
    }
    return YES;
}
```

Dragging File Contents

Files do not need to be dragged using references to the file only; a file's contents can be placed directly onto the pasteboard and dragged using the `NSFileContentsPboardType`. Use this pasteboard type when you want to supply the contents of a file instead of its location in the file system. The destination can choose to extract the data directly to a location in the file system that it specifies or into a file wrapper in memory.

To initiate a drag operation on a file's contents, you need to use the NSView or NSWindow method `dragImage:at:offset:event:pasteboard:source:slideBack:`. You must place the file's contents onto the pasteboard yourself. You can write the data to the pasteboard using either the `writeFileContents:` method, which reads the data directly from the file system, or the `writeFileWrapper:` method, which reads the data from an NSFileWrapper object that you have already created. The following sample code shows a possible implementation for writing the data to the pasteboard.

```
// Write data to the pasteboard
NSString *filename; // Assume this exists
NSPasteboard *pboard = [NSPasteboard pasteboardWithName:NSDragPboard];
[pboard writeFileContents:filename];
```

See "Dragging File Paths" (page 15) for more complete sample code on starting the drag operation.

In addition to writing the file's contents to the pasteboard with the general type `NSFileContentsPboardType`, the `writeFileContents:` and `writeFileWrapper:` methods write the data with a more specific type based on the file's filename extension, if it exists. The drag destination can register for this more specific type instead of the generic type to restrict drags to files of a particular type, such as `mp3` or `mov` files. You can obtain the name of this specific pasteboard type by passing the filename extension to the `NSCreateFileContentsPboardType` function, which returns an NSString. The following code sample shows how a view could register to receive only QuickTime movie files.

```
NSString *pboardType = NSCreateFileContentsPboardType(@"mov");
NSArray *dragTypes = [NSArray arrayWithObject:pboardType];
[self registerForDraggedTypes:dragTypes];
```

After a drag operation is dropped, the dragging destination receives a `performDragOperation:` message. To extract the file contents from the pasteboard, use either the `readFileContentsType:toFile:` method, which copies the data from the pasteboard and writes it to the file path you specify, or the `readFileWrapper` method, which creates an NSFileWrapper object from the pasteboard data.

```
- (BOOL)performDragOperation:(id <NSDraggingInfo>)sender
{
    NSPasteboard *pboard = [sender draggingPasteboard];

    if ( [[pboard types] containsObject:NSFileContentsPboardType] ) {
        NSFileWrapper *fileContents = [pboard readFileWrapper];
        // Perform operation using the file's contents
    }
```

```
    return YES;
}
```

Dragging File Promises

In some cases, you may want to drag a file before it actually exists within the file system. You may have a new document that hasn't been saved, yet, or perhaps the file exists on a remote system, such as a web server, to which the dragging destination may not have access. (Typically, the destination is the Finder.) In these cases, the drag operation serves as a technique for specifying a location at which to save the new file. When the drag operation is dropped, the dragging destination tells the source where it wants the files saved and the dragging source creates the files. This type of file drag is called an HFS promise, because, in essence, the drag operation contains a promise from the source to the destination that the source will create the specified files if the drag operation is accepted. The data on the pasteboard has the type `NSFilesPromisePboardType`.

To initiate an HFS promise drag operation on one or more files, you need to use the NSView method `dragPromisedFilesOfTypes:fromRect:source:slideBack:event:`. The first argument is an array listing the file types of all the files the source promises to create. The types can be specified as filename extensions or as HFS file types encoded using the `NSFileTypeForHFSTypeCode` function. If a directory hierarchy is being dragged, only the top-level files and directories need to be listed in the type array. The `dragPromisedFiles...` method places the file type array onto the dragging pasteboard with the `NSFilesPromisePboardType` pasteboard type and starts the drag operation.

```
- (void)mouseDown:(NSEvent *)theEvent
{
    NSPoint dragPosition;
    NSRect imageLocation;

    dragPosition = [self convertPoint:[theEvent locationInWindow]
                        fromView:nil];
    dragPosition.x -= 16;
    dragPosition.y -= 16;
    imageLocation.origin = dragPosition;
    imageLocation.size = NSMakeSize(32,32);
    [self dragPromisedFilesOfTypes:[NSArray arrayWithObject:@"pdf"]
            fromRect:imageLocation
            source:self
            slideBack:YES
            event:theEvent];
}
```

When dragging HFS promises, the dragging source must also implement the `namesOfPromisedFilesDroppedAtDestination:` method. This method is invoked when the destination accepts the drag operation. The single argument is an NSURL object that identifies the location within the file system that the source should create the files. The method returns a list of the filenames (not full paths) of all the files the source promised to create. If a directory hierarchy is being dragged, only the top-level objects need to be listed in the returned array.

For short operations, you can create the promised files within the `namesOfPromisedFilesDroppedAtDestination:` method. For long operations, however, you should defer the creation of the files until later to avoid blocking the destination application. One technique is to cache the destination URL and create the files in your source's `draggedImage:endedAt:operation:`

method. Alternatively, you could spawn a background thread to create the files or delay the action on the current thread using an NSTimer, an NSNotificationQueue, or the NSObject method `performSelector:withObject:afterDelay:`.

Before the drag is actually dropped, a potential dragging destination does not have access to the filenames of the files being promised. Only the file types are available from the pasteboard. The destination can obtain the file types by requesting the pasteboard's `NSFilesPromisePboardType` data using the `propertyListForType:` method. The returned array contains the file types that the source passed to the `dragPromisedFiles...` method. The destination can then accept or reject a drag operation based on the contents of the types array.

After a drag operation is dropped, the dragging destination receives a `performDragOperation:` message. To specify the drop location and to obtain the filenames of the promised files, use the dragging information object's `namesOfPromisedFilesDroppedAtDestination:` method, passing the NSURL for the drop location as the one argument. The return value is an array of the filenames (not full paths) of the files that the source will create. The dragging destination must invoke this method only within `performDragOperation:` or else the source may create the files in the incorrect location.

```
NSURL *dropLocation; // Assume this exists

- (BOOL)performDragOperation:(id <NSDraggingInfo>)sender
{
    NSPasteboard *pboard = [sender draggingPasteboard];

    if ( [[pboard types] containsObject:NSFilesPromisePboardType] ) {
        NSArray *filenames = [sender
                namesOfPromisedFilesDroppedAtDestination:dropLocation];
        // Perform operation using the files' names, but without the
        // files actually existing yet
    }
    return YES;
}
```

Using Drag and Drop in Tables

The `NSTableView` class implements both the `NSDraggingSource` and `NSDraggingDestination` informal protocols. It manages most of the details of drag operations, sending messages to its data source when it begins or receives a drag operation. The minimum required steps for supporting drag and drop in a table are as follows:

1. Call the `registerForDraggedTypes:` method of your table view to specify the types of data your table supports.
2. Implement the `tableView:writeRowsWithIndexes:toPasteboard:` method in your data source to handle the beginning of a drag operation.
3. Implement the `tableView:validateDrop:proposedRow:proposedDropOperation:` method in your data source to validate the drop location.
4. Implement the `tableView:acceptDrop:row:dropOperation:` method in your data source to incorporate the dropped data.

Implementing the delegate methods in your data source object lets the `NSTableView` class know that your table supports drag and drop and also lets it know what to do with dragged or dropped data. If you omit one of these delegate methods from your data source, your table view will be unable to accept dropped data or initiate drags (depending on which method you omit). Other customizations to drag-and-drop behavior in table views are also possible but are not required.

The sections that follow provide additional information about how you implement the basic delegate methods and some specific customizations. For general information on how drag and drop works, see *Drag and Drop Programming Topics for Cocoa*.

Configuring Your Table View

The first step in configuring a table to support drag and drop is to tell it what data types your data source object understands. You do this by sending a `registerForDraggedTypes:` message to the `NSTableView` object. This method accepts an array of pasteboard types that your data source understands. If you want to support drag and drop operations only within your table, you can simply define a custom type. If you support data from several different sources, you might want to specify several different types.

The following example shows a sample implementation of the awakeFromNib method for a document. If you want to support drag and drop only within the table, you can simply register a custom type for your table data. If you wanted to accept other types of data, you would add the appropriate pasteboard types to the array.

Listing 1 Registering the table's supported data types

```
#define MyPrivateTableViewDataType @"MyPrivateTableViewDataType"
```

```
- (void)awakeFromNib
{
    [myTableView registerForDraggedTypes:
                        [NSArray arrayWithObject:MyPrivateTableViewDataType] ];

    // Other initialization...
}
```

Beginning a Drag Operation

When a drag operation begins, the table sends a `tableView:writeRowsWithIndexes:toPasteboard:` message to the data source. Your implementation of this method should place the data for the specified rows onto the provided pasteboard and return `YES`. If, for some reason, you do not want the drag operation to continue, your method should return `NO`.

Note: Prior to Mac OS X v10.4, table views initiated a drag by sending a `tableView:writeRows:toPasteboard:` message. In Mac OS X v10.4 and later, the use of this method is deprecated.

The following example shows a simple implementation of the `tableView:writeRowsWithIndexes:toPasteboard:` delegate method. This implementation assumes that drags and drops are confined to the table itself, so it simply copies the dragged row numbers to the pasteboard.

Listing 2 Initiating a drag from a table.

```
- (BOOL)tableView:(NSTableView *)tv writeRowsWithIndexes:(NSIndexSet *)rowIndexes
 toPasteboard:(NSPasteboard*)pboard
{
    // Copy the row numbers to the pasteboard.
    NSData *data = [NSKeyedArchiver archivedDataWithRootObject:rowIndexes];
    [pboard declareTypes:[NSArray arrayWithObject:MyPrivateTableViewDataType]
owner:self];
    [pboard setData:data forType:MyPrivateTableViewDataType];
    return YES;
}
```

If there is a lot of data to be placed onto the pasteboard, or if there are multiple data formats possible, you may provide the data lazily using promises. To do so, just tell the pasteboard object what types you support without providing the actual data. The pasteboard object will notify you at a later time if and when the data is actually needed. See the description of the `pasteboard:provideDataForType:` delegate method (Objective-C) or the `pasteboardProvideDataForType` delegate method (Java) in the `NSPasteboard` class for information about fulfilling a pasteboard promise.

In Mac OS X v10.4, support was added for handling file-promised drag operations in your data source object. To support this feature in a table view, you must first promise the the data to the pasteboard using the `NSFilesPromisePboardType` type in your `tableView:writeRowsWithIndexes:toPasteboard:` method. When a destination accepts the dropped file information, `NSTableView` calls through to the `tableView:namesOfPromisedFilesDroppedAtDestination:forDraggedRowsWithIndexes:` method of your data source to provide the files. Your implementation of this method should create the files and return an array containing the filenames (without path information).

The `NSTableView` class only supports local drags by default. When you try to drag table rows outside of the application, the table view's `draggingSourceOperationMaskForLocal:` method returns `NSDragOperationNone`. To allow interapplication drags in Mac OS X v10.4 and later, call the `setDraggingSourceOperationMask:forLocal:` method of `NSTableView`. (If your code supports versions of Mac OS X prior to v10.4, you must subclass `NSTableView` instead and override `draggingSourceOperationMaskForLocal:` to return an appropriate value.)

Validating a Drag Operation

When a drag operation enters a table view, the table view sends a `tableView:validateDrop:proposedRow:proposedDropOperation:` message to its data source. If this method is not implemented or if the method returns `NSDragOperationNone`, the drag operation is not allowed.

The last two parameters of the `tableView:validateDrop:proposedRow:proposedDropOperation:` method contain the proposed row insertion point and insertion behavior (`NSTableViewDropOn` or `NSTableViewDropAbove`). You can override these values in your delegate method implementation by sending a `setDropRow:dropOperation:` message to the table view.

Para

```
- (NSDragOperation)tableView:(NSTableView*)tv validateDrop:(id
<NSDraggingInfo>)info proposedRow:(NSInteger)row
proposedDropOperation:(NSTableViewDropOperation)op
{
    // Add code here to validate the drop
    NSLog(@"validate Drop");
    return NSDragOperationEvery;
}
```

Accepting a Drop

When a validated drag operation is dropped onto a table view, the table view sends a `tableView:acceptDrop:row:dropOperation:` message to its data source. The data source's implementation of this method should incorporate the data from the dragging pasteboard (obtained from the *`acceptDrop`* parameter) and use the other parameters to update the table. For example, if the drag operation type (also obtained from the *`acceptDrop`* parameter) was `NSDragOperationMove` and the drag originated from the table, you would want to move the row from its old location to the new one.

```
- (BOOL)tableView:(NSTableView *)aTableView acceptDrop:(id <NSDraggingInfo>)info
          row:(NSInteger)row dropOperation:(NSTableViewDropOperation)operation
{
    NSPasteboard* pboard = [info draggingPasteboard];
    NSData* rowData = [pboard dataForType:MyPrivateTableViewDataType];
   NSIndexSet* rowIndexes = [NSKeyedUnarchiver unarchiveObjectWithData:rowData];
    NSInteger dragRow = [rowIndexes firstIndex];

    // Move the specified row to its new location...
}
```

Customizing Drag Behavior

Dragging the cursor vertically in a table view can be interpreted as an attempt either to select a range of rows or to drag one or more rows elsewhere. The default behavior of `NSTableView` interprets vertical drags as the beginning of a drag operation but you can change this behavior using the `setVerticalMotionCanBeginDrag:` method of `NSTableView`. Horizontal drags always begin a drag operation.

When a drag operation begins, the table view constructs an image to represent the dragged rows. The default image is a copy of the rows. If you want a different image, you must subclass `NSTableView` and override the `dragImageForRows:event:dragImageOffset:` method to return your own `NSImage` object.

Background Drags

Table views and outline views allow you to drag entries while your application is in the background. This only affects clients using the row/item based dragging APIs. The behavior when dragging from such a table is the same as the `NSTextView` implementation of dragging. If the user clicks and drags on your table while your application is not active, a drag operation is initiated. If the user simply clicks on your table, your application becomes active and the current selection does not change.

Frequently Asked Questions

This document answers commonly asked questions about the drag and drop capabilities of the Application Kit. This includes information on HFS promise drags and special information regarding cross-application dragging from NSTableView objects.

Note: This document obsoletes Developer Technical Q&A's 1200, 1220, and 1300. Future updates to these questions will be posted to this programming topic.

This document covers the following subjects:

- "HFS Promise Drag" (page 25)
- "Cross-Application Drag and Drop" (page 26)

HFS Promise Drag

The questions addressed in this section include:

"How Do I Set a Custom Drag Image When Doing an HFS Promise Drag in Cocoa?" (page 25)
"How Do I Add Other Pasteboard Types to an HFS Promise Drag in Cocoa? " (page 26)

How Do I Set a Custom Drag Image When Doing an HFS Promise Drag in Cocoa?

NSView's `dragPromisedFilesOfTypes:fromRect:source:slideBack:event:` method doesn't provide a way to set the drag image to be used when dragging an HFS promise file. However, this method calls another NSView method, `dragImage:at:offset:event:pastboard:source:slideback:` in its implementation, which does provide a parameter for an NSImage to be used as the drag image. So, to set a custom drag image, simply override `dragImage:at:offset:event:pastboard:source:slideback:`, setup your custom image, invoke super's `dragImage:at:offset:event:pastboard:source:slideback:` and pass in your custom drag image.

For more information, see documentation on NSView.

How Do I Add Other Pasteboard Types to an HFS Promise Drag in Cocoa?

NSView's `dragPromisedFilesOfTypes:fromRect:source:slideBack:event:` method doesn't provide a means to add other pasteboard types to the HFS Promise data, because it doesn't expose the pasteboard that it uses. However, there is a workaround to add other pasteboard type data. `dragPromisedFilesOfTypes:fromRect:source:slideBack:event:` calls NSView's `dragImage:at:offset:event:pasteboard:source:slideBack:` method in its implementation. So, you can override `dragImage:at:offset:event:pasteboard:source:slideBack:`, add any additional pasteboard types you need, and then invoke super's `dragImage:at:offset:event:pasteboard:source:slideBack:` to allow everything to continue as before.

For more information, see documentation on NSView.

Cross-Application Drag and Drop

The questions addressed in this section include:

"When I Rebuild My application on Mac OS X version 10.2, My NSTableView Loses the Ability to Drag-And-Drop to Other Applications. How Do I Fix This?" (page 26)

When I Rebuild My application on Mac OS X version 10.2, My NSTableView Loses the Ability to Drag-And-Drop to Other Applications. How Do I Fix This?

A bug in NSTableView in Mac OS X version 10.2 and later causes cross-application drags to not work without additional code from the application developer. Drag-and-Drop within an application still works correctly.

You can work around the bug by subclassing NSTableView and overriding `draggingSourceOperationMaskForLocal:` to return the appropriate `NSDragOperation` (typically `NSDragOperationCopy`, depending upon what drag operation you want the drag-and-drop to perform). Only applications built on Mac OS X version 10.2 and later are affected; applications built on Mac OS X version 10.1.x are not affected.

In Mac OS X v10.4 you can work around this bug without subclassing. By default the NSTableView implementation of `draggingSourceOperationMaskForLocal:` disallows dragging to destinations outside of the application while allowing any type of drag within the same application. You can change this behavior by sending the table view a `setDraggingSourceOperationMask:forLocal:` message. Passing a value of `YES` as the second parameter indicates that specified mask applies when the destination object is in the same application. Passing a value of `NO` indicates that the specified mask applies when the destination object in an application outside the receiver's application. The masks are archived with the table view.

Document Revision History

This table describes the changes to *Drag and Drop Programming Topics for Cocoa*.

Date	Notes
2006-06-28	Added a link to "Using Drag and Drop in Table Views."
2005-08-11	Added information about setDraggingSourceOperationMask:forLocal: to the Frequently Asked Questions article.
2003-10-13	Added a "Frequently Asked Questions" (page 25) article to obsolete Technical Q&As 1200, 1220, and 1300.
2003-04-03	Dragging of color uses `NSDragOperationGeneric`, not `NSDragOperationCopy`. Updated sample code and descriptive text in "Receiving Drag Operations" (page 13) to use the correct drag operation.
2002-11-12	Revision history was added to existing topic. It will be used to record changes to the content of the topic.