C H A P T E R 2

Drag Manager Programmer’s Guide

Contents

Drag Manager Programmer's Guide

This chapter describes how your application can use the Drag Manager to drag items that reside within your application. By using the Drag Manager, you can allow items from your application to be directly dragged to other applications and you can receive items from other applications.

The Drag Manager is not available in all versions of system software. Use the `Gestalt` function, described in the chapter "Gestalt Manager" of *Inside Macintosh: Operating System Utilities*, to determine whether the Drag Manager is present.

Read this chapter if you want your application to be able to drag items either within your own application's windows or between your application and other applications. You might want to allow the user of your application to drag selections of your documents to the Finder to create "clippings" from your documents. You might also want to allow selections from other applications to be dragged directly into your documents.

About the Drag Manager

Many Macintosh applications typically allow their users to drag objects within their documents. The Finder, for example, allows users to move files and folders anywhere within the file system using a simple drag-and-drop user interface.

The Drag Manager is the part of the Macintosh Toolbox that facilitates dragging objects within the Macintosh user interface. The Drag Manager provides routines that handle the user interface for dragging an object from, within, or to one of your application's windows. The Drag Manager can be used whenever an object is dragged within your application.

This chapter describes how your application can use the Drag Manager to facilitate drag and drop of objects within your documents. The Drag Manager also provides your application with the ability to receive dragged items from other applications as well as allowing other applications to receive items dragged from your application.

This document does not discuss the actual human interface guidelines for drag and drop. Please see the separate document "Drag and Drop Human Interface Guidelines."

IMPORTANT
The Drag Manager is not available in all versions of system software. Use the `Gestalt` function to determine if the Drag Manager is available for use. ▲

The Drag Process

The Drag Manager divides the drag and drop user interface into three discrete steps. The steps are starting a drag, tracking a drag, and finishing a drag. The Drag Manager divides the action of dragging into these three steps to allow for the possibility that different applications will be involved in each of the steps.

In the simplest case, the user may drag an item wholly within one of your document windows. In this case, your application starts the drag, tracks the movement of the item through the document window, and then accepts the item when the user releases the item in the window.

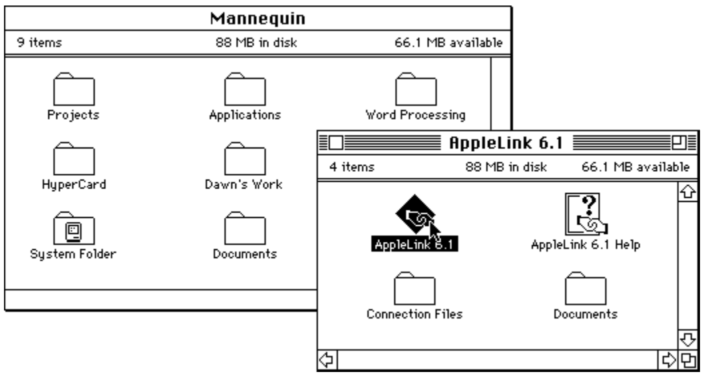
In a more ambitious scenario where the user drags an item from one application to another, the source application starts the drag; potentially several other applications become involved as the user moves the mouse on the screen while searching for a place to drop the item; and finally a different application may be involved when the user drops the item at its final destination.

The remainder of this section describes each of the these three steps in greater detail.

Starting a Drag

The first step in a drag-and-drop action occurs when the user clicks on a selected object and begins to move the mouse without first releasing the mouse button. The Drag Manager includes a `WaitMouseMoved` function that allows you to easily determine if the mouse has moved far enough to start a drag.

Figure 2-1 Starting a Drag



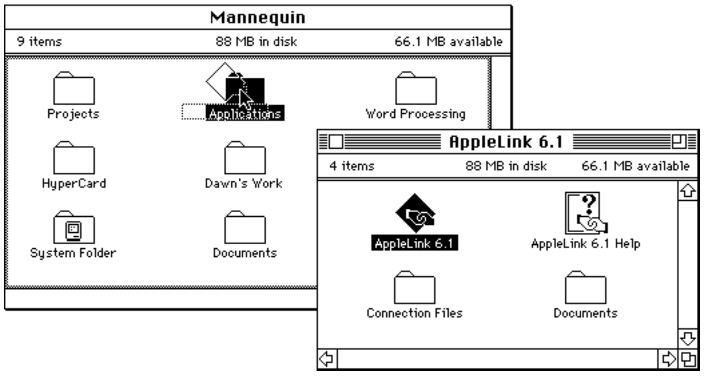
Your application must create a new **drag reference** which is used by your application to refer to a specific drag process in subsequent calls to the Drag Manager. Use the `NewDrag` function to create a new drag reference. After creating a new drag reference, your application provides the Drag Manager with a description of the selected item or items being dragged. You describe the selection being dragged by creating a list of **drag item flavors** that represent the different data formats that each drag item may be produced in. The Drag Manager includes the `AddDragItemFlavor` function for adding drag item flavors to a drag reference.

After preparing the Drag Manager to drag your application's selection, your application begins the drag by the Drag Manager's `TrackDrag` function.

Tracking a Drag

While the user drags a selection on the screen, the drag is "tracked" through each window the the cursor moves over. Destination highlighting occurs while tracking a drag after the cursor has left the source location. Also, destination feedback may occur if a container under the cursor can accept the selection (such as folders in the Finder).

Figure 2-2 Tracking a Drag



While the drag is being tracked on the screen by the Drag Manager, the Drag Manager calls routines provided by your application to track the drag through your windows. These routines are called **drag tracking handlers**

Your drag tracking handlers can inspect the description of the items that are being dragged and highlight parts of the application's windows accordingly. Routines are provided that allow you to inspect the data within the selection being dragged, and routines are provided to create and remove drag highlighting.

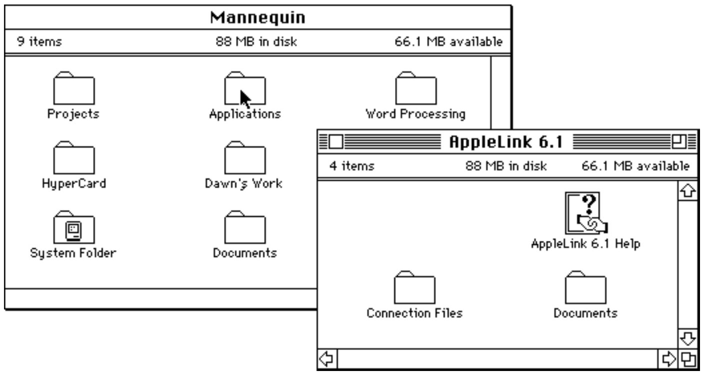
You can also provide the Drag Manager with routines that override the Drag Manager's standard behaviors, such as to provide a different appearance for the dotted outline or to modify the keyboard or mouse inputs.

Finishing a Drag

When the user releases the mouse button, the Drag Manager calls another routine provided by your application called a **drag receive handler** Your application's drag receive handler is responsible for accepting the drop and performing the actual data transaction that is required to place the selection at its final destination.

Your drag receive handler can inspect and request the data types contained within the drag by using the`GetFlavorData` function provided by the Drag Manager.

Figure 2-3 Finishing a Drag



Drag Items

When dragging a selection from your application, each distinct object being dragged is a **drag item** The following list contains examples of single drag items:

- any icon in the Finder
- a selection in a bitmap drawing application
- a selection of objects in an object-oriented graphics application
- a continuous range of text in a word processor (even if the text selection contains a picture)

There are many circumstances that result in multiple drag items being dragged simultaneously:

- any group of icons in the Finder
- a discontinuous text selection (resulting from using the Command key)

When deciding how to break your application's selections into drag items, keep in mind that when dragging to the Finder, each drag item results in a separate “clipping” icon. One of the best heuristics is to draw distinctions from your application's richest data format (which may be your own internal data format).

Drag Item Flavors

Many items that can be dragged (such as text, pictures or sounds) can be represented using several different data formats. The Drag Manager introduces the concept of **drag item flavors** to allow Drag Manager clients to send and receive objects in the most preferable data format that both the sender and receiver can understand.

When you start a drag, you use drag item flavors to inform the Drag Manager of each of the data formats that you could provide to the receiver of the drag. The Drag Manager provides an `AddDragItemFlavor` function to add flavors to drag items before starting a drag.

For example, a text selection of the string "Welcome to Macintosh." is represented in standard 'TEXT' format as:

```
Welcome to Macintosh.
```

The standard 'TEXT' format does not include any information such as the text's font or size. The standard Macintosh styled TextEdit data format 'styl' supplements the 'TEXT' data structure by providing font and style information. The 'styl' data for the same selection (in hexadecimal) is:

```
0000: 0001 0000 0000 000E
0008: 000A 0015 0000 000C
0010: 0000 0000 0000
```

Another popular data format is rich text format. The RTF data format includes much more information about the fonts and styles, and also includes information about the source document's margins, page size, columns, etc. The same text selection is represented in RTF format as:

```
{\rtf1\mac\deff2 \windowctrl\ftnbj\fracwidth
  \sectd \linemod0\linex0\cols1\endnhere
  \pard\plain {\f21 Welcome to Macintosh.}
}
```

There are many other ways to represent this string. Most importantly, if the user drags a selection entirely within one of your documents, you might want to transfer the data in your application's own internal data format.

There is no way to know where the user intends to drag a selection when starting a drag. The user may want to drag within one of your windows, between two of your windows or to a different application's window. Different destinations may prefer different data formats. In the text selection example, your own application might prefer its own internal data format. Another sophisticated word processor may not understand your internal data format, but may prefer RTF over styled text. A simple TextEdit field, such as the Comments field of the Finder's Get Info window may only be able to accept the plain text.

Each flavor has its own set of flags associated with it. These flags are used by the Drag Manager and its clients to provide additional information about each drag item flavor. The following flags may be set for each flavor:

Flavor flag descriptions

`flavorSenderOnly`
: This flag is set by the sender if the flavor should only be available to the sender of the drag. Flavors that are marked with this flag do not appear to any other application other than the sender.

`flavorSenderTranslated`
: Set if the flavor data is translated by the sender. This attribute is useful if the receiver needs to determine if the sender is performing its own translation to generate this data type. The Finder does not save translated types into clipping files.

`flavorNotSaved`
: Set by the sender if the flavor data should not be stored by the receiver. This flag is useful for marking flavor data that will become stale after the drag is completed. Receivers that store data should not store flavors that are marked with this flag. Flavor types marked with this flag are not stored by the Finder in clipping files.

`flavorSystemTranslated`
: Set if the flavor data is provided by the Translation Manager. If this flavor is requested, the Drag Manager will obtain the required data type from the sender and then it will use the Translation Manager to provide the data that the receiver requested. Flavor types marked with this flag are not stored by the Finder in clipping files.

Drag Handlers

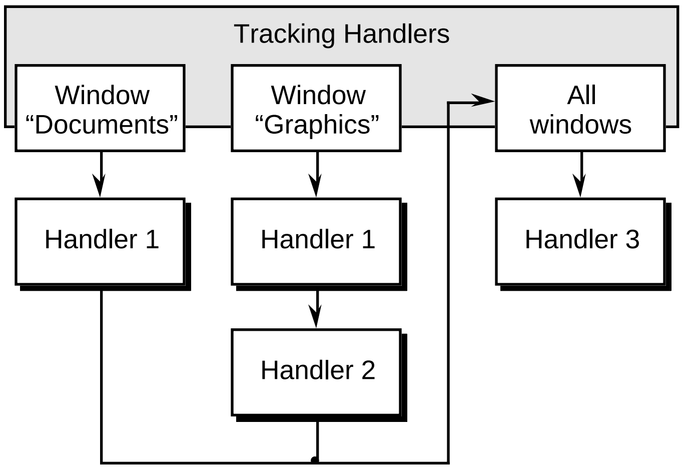
You register with the Drag Manager callback routines that the Drag Manager calls to allow your application to implement dragging. The Drag Manager uses two different types of callback routines, called**drag handlers** and **drag procedures** Drag handlers are routines that are installed on windows that the Drag Manager uses when dragging over that window. Drag procedures are routines that are used by the Drag Manager during a drag regardless of which window the user may be dragging over. The Drag Manager allows you to install the following drag handlers on your application's windows:

- a **drag tracking**handler that the Drag Manager calls when the user drags a selection through one of your application's windows. This allows you to track the drag within the window
- a **drag receive**handler that the Drag Manager calls when the user finishes a drag in one of your application's windows

The Drag Manager provides a pair of `InstallHandler` and `RemoveHandler` routines that allow you to register handlers of each of these two types with windows in your application. You can register a different set of handlers to be used for each window in your application. You can also register with the Drag Manager a set of handlers to be used when a window does not have its own handlers.

If you assign more than one handler of the same type on the same window, the Drag Manager calls each of these handler routines in the order that they were installed. Figure 2-4 shows an example of the tracking handler registry for an application that has installed the same handler for its "Graphics" and "Documents" window, an additional handler for its "Graphics" window, and a handler to be used for all windows in the application including the "Graphics" and "Documents" windows. When the Drag Manager tracks a drag through the "Documents" window, Handler 1 is called followed by Handler 3 being called. When the Drag Manager tracks a drag through the "Graphics" window, Handlers 1, 2 and 3 three are called, in order. Finally, if the Drag Manager tracks a drag through any other window in the application, only Handler 3 is called.

Figure 2-4 Example "Tracking" Handler Registry



In the next three sections, the drag tracking and drag receive handler types are described in more detail.

Drag Tracking

While the user drags a collection of items on the screen, as the mouse passes through one of your application's windows, the Drag Manager calls your `DragTrackingHandler` to allow you to track the drag through your windows.

The Drag Manager sends your `DragTrackingHandler` tracking status messages as the user moves the mouse. Your `DragTrackingHandler` receives the following messages from the Drag Manager:

- an **enter handler** message when the focus of a drag enters a window that is handled by your `DragTrackingHandler` from any window that is not handled by the same `DragTrackingHandler`
- an **enter window** message when the focus of a drag enters any window that is handled by your `DragTrackingHandler`
- an **in window** message as the user drags within a window handled by your `DragTrackingHandler`
- a **leave window** message when the focus of a drag leaves any window that is handled by your `DragTrackingHandler`
- a **leave handler** message when the focus of a drag enters a window that is not handled by your `DragTrackingHandler`

When you receive any of these messages from the Drag Manager, you can use several routines provided by the Drag Manager that allow your application to determine what is being dragged. This includes counting the number of drag items, counting the number of flavors in a drag item and getting the type and flags for each flavor in a drag item. Using the information returned by these functions, your application can determine if a portion of a window should highlight when the user drags through the window.

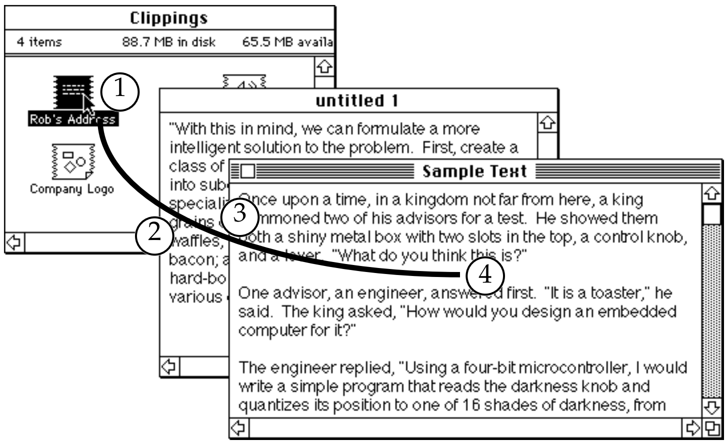
The **in window** message is where most highlighting occurs. You can test the current position of the mouse and highlight different areas of your window accordingly.

The **enter window** and **leave window** messages always occur in pairs. These messages are useful for determining the point at which the mouse enters or leaves a window.

The **enter handler** and **leave handler** messages also occur in pairs. These messages only occur when the drag moves between windows that are handled by different handler routines. These messages are useful for allocating and releasing memory that you might need when tracking within a set of windows.

Figure 2-5 shows an example of a user dragging a clipping from the Finder through two windows of a word processing application. The following example demonstrates what tracking messages are sent to the Finder and application during a drag:

Figure 2-5 Example Drag Tracking Path Through Multiple Applications and Windows



"With this in mind, we can formulate a more intelligent solution to the problem. First, create a class of
into sub
speciali
grains
waffles,
bacon; a
hard-bo
various

Once upon a time, in a kingdom not far from here, a king summoned two of his advisors for a test. He showed them both a shiny metal box with two slots in the top, a control knob, and a lever. "What do you think this is?"

One advisor, an engineer, answered first. "It is a toaster," he said. The king asked, "How would you design an embedded computer for it?"

The engineer replied, "Using a four-bit microcontroller, I would write a simple program that reads the darkness knob and quantizes its position to one of 16 shades of darkness, from

1. The user clicks and drags the clipping and the Finder starts a drag. The Finder receives an enter handler message followed by an enter window message. As the user drags within the Finder's "Clippings" window, the Finder receives multiple in window messages.
2. When the user drags into the word processor's "untitled 1" window, the Finder receives a leave window message followed by a leave handler message. The word processing application then receives an enter handler message followed by an enter window message. While the user drags within the application's "untitled 1" window, the application receives in window messages.
3. Assuming that both of the word processor's windows are handled by the same `DragTrackingHandler`, when the user drags into the "Sample Text" window, the word processing application receives a leave window message followed by an enter window message. It does not receive any enter/leave handler messages since the same handler routine is used for both windows. As the user drags within the application's "Sample Text" window, the application receives in window messages.
4. When the user releases the mouse button, the data transaction occurs by calling the word processing application's receive drop handler routine. Following the data transaction, the application receives a leave window message followed by a leave handler message. The drag is now complete and both the Finder's and word processor's event loops continue as they did before the drag and drop action.

Receiving Data

When the user drops a collection of items in one of your application's windows, the Drag Manager calls any`DragReceiveHandler` routines that are installed on the destination window. This call allows you to request the drag item flavors that your application wishes to accept.

Your `DragReceiveHandler` can inspect the available flavors by using the `CountDragItems`, `GetDragItemReferenceNumber`, `CountDragItemFlavors`, `GetFlavorType` and `GetFlavorFlags` functions.

The `DragReceiveHandler` may receive data from the sender of the drag by calling the `GetFlavorData` function.

Drag Procedures

In addition to installing drag handler routines on windows for a drag, you can supply the Drag Manager with several different kinds of **drag procedures**. Drag procedures are used by the Drag Manager during a drag regardless of which window the user may be dragging over. You do not need to provide the Drag Manager with drag procedures unless you wish to override the default behavior. Only the sender of a drag can specify drag procedures to be used during a drag. The Drag Manager allows you to install the following drag procedures in a given drag:

- a **send data** procedure that the Drag Manager calls when the receiver application requests a drag item flavor that the Drag Manager does not currently have the data cached for
- a **drag input** procedure that the Drag Manager calls when sampling the mouse position and keyboard state to allow the application program to override the current state of the input devices
- a **drag drawing** procedure that the Drag Manager calls to allow your application to assume responsibility for drawing the drag region on the screen

Sending Data

The Drag Manager caches the flavor data for any flavors that were added to a drag with the `AddDragItemFlavor` function. If a receiver calls the `GetFlavorData` function to get a flavor's data, the Drag Manager simply returns the cached data to the caller.

If your application passes `NIL` as the pointer to the flavor data when adding a new flavor with the `AddDragItemFlavor` function, the Drag Manager does not cache any data in the new flavor. In this case, when a receiver requests the data by calling `GetFlavorData`, the Drag Manager will call the drag's send data procedure to get the data from the sender.

This mechanism allows your application to add all of the various drag item flavor types that could be provided to a receiver upon request, but doesn't require the sender to spend the time and memory required to generate the data. This is usually a consideration when the sender must perform expensive computations to produce the data or if the resulting data requires a great deal of memory to store.

A drag send data procedure is only required when one or more flavors were added to a drag without specifying a pointer to the flavor data when calling the `AddDragItemFlavor` function.

Overriding Standard Input

The Drag Manager allows your application to provide a drag input procedure that is called by the Drag Manager each time the Drag Manager samples the mouse and keyboard.

The drag input procedure gets passed the current mouse location, mouse button state and keyboard modifier status. A drag input procedure can either slightly modify these parameters or completely change them.

For example, a drag input procedure can inhibit or force specific modifier keys, can control the state of the mouse button, and can control the coordinates of the cursor.

Overriding Standard Drawing

The Drag Manager provides the sender application with a way to override the standard drag region drawing behavior. This is done be specifying a drag drawing procedure for a given drag operation.

When a drag drawing procedure is given for a drag, the Drag Manager sends your drag drawing procedure a sequence of messages that allow you to assume responsibility for drawing the drag region or similar feedback on the screen. Your `DragDrawingProc` receives the following messages from the Drag Manager:

- a **drag region begin** message when a drag is beginning and it is time for your `DragDrawingProc` to allocate memory and initialize any data structures it needs to function properly
- a **drag region draw** message when the drag region has moved or needs to be redrawn on the screen
- a **drag region hide** message when all or part of the drag region needs to be removed from the screen
- a **drag region idle** message when the drag region has not moved and does not need to be redrawn
- a **drag region end** message when the drag has ended and it is time for your `DragDrawingProc` to deallocate any memory it may have allocated

Using the Drag Manager

You use the Drag Manager to let the user drag items in your application. The Drag Manager will allow items to be dragged between windows of your application and also between other applications that the user is currently using. The Drag Manager is also used to drag items both to and from the Macintosh Finder.

Before items can be dragged into or out of one of your application's windows, you must register a set of drag handlers for the Drag Manager to use when you application is involved in dragging. A drag handler is a callback routine that the Drag Manager will call when the Drag Manager needs to send your application a message about a drag occurring within your application.

A drag and drop action by the user is broken down into three discrete steps. The steps are first to **pick up** the item or items being dragged, then to **track** the selection being dragged through application windows as the user searches for a place to drop the selection, and finally to **drop** the item or items at the user's chosen destination.

This section explains in detail how you use the Drag Manager to:

- install and remove drag handlers to and from the Drag Manager's handler registry for your application's windows
- recognize the start of a drag operation
- create new drag reference to be used in a drag operation
- prepare the Drag Manager with drag items and drag item flavors
- provide a drag send procedure to the Drag Manager
- start a drag
- track a drag through your application's windows
- receive a drop and accept the contents of a drag
- send data to the receiver of a drag that originated from one of your application's windows

Installing and Removing Drag Handlers

You register a drag handler with the Drag Manager using the `InstallHandler` functions. There is a separate `InstallHandler` function for each kind of handler. These functions are `InstallTrackingHandler` and `InstallReceiveHandler`.

Each of the `InstallHandler` functions takes a pointer to the window that you want to associate the handler with. If you supply `NIL` as the window pointer, the Drag Manager will register the handler in the special area that is used when a drag occurs in any window in your application. Handlers installed in this special area are called **default**

handlers.

A reference constant may be passed to each of the `InstallHandler` functions. This value is stored by the Drag Manager and is forwarded to your handler routine when it is called. You can use this reference constant to provide additional information to your handler routine, such as a pointer to a data structure used by your handler.

Listing 2-1 shows how to use the `InstallHandler` functions to install a default receive handler and a default tracking handler for your application.

Listing 2-1 Installing Default Drag Handlers

```
OSErr MyInitDragManager()

{   OSErr       result;

 result = InstallTrackingHandler(MyDefaultTrackingHandler, 0L,
                       &myGlobals);

    if (result != noErr)
        return(result);

    result = InstallReceiveHandler(MyDefaultReceiveHandler, 0L, &myGlobals);

    return(result);
}
```

The function `MyInitDragManager` defined in Listing 2-1 calls `InstallTrackingHandler` and `InstallReceiveHandler` to install default tracking and receive handlers for your application. In the window parameter, `0L` (`NIL`) is passed to specify that these handlers should be installed as default handlers. A pointer to the application's global variables is passed in the reference constant parameter.

Listing 2-2 shows how to use the `InstallHandler` functions to install handlers for a specific window.

Listing 2-2 Installing Drag Handlers for Individual Windows

```
OSErr MyDoNewWindow(WindowPtr *newWindow)

{   OSErr       result;
    WindowPtr   theWindow;

    if (!(theWindow = GetNewWindow(kMyWindowID, 0L, -1L))) {
        return(resNotFound);
    }
    if (result = InstallTrackingHandler(MyTrackingHandler,
                                    theWindow, &myGlobals)) {
        DisposeWindow(theWindow);
        return(result);
    }

    if (result = InstallReceiveHandler(MyReceiveHandler,
                                   theWindow, &myGlobals)) {
        DisposeWindow(theWindow);
        RemoveTrackingHandler(MyTrackingHandler, theWindow);
        return(result);
    }

    *newWindow = theWindow;
    return(noErr);
}
```

The function `MyDoNewWindow` defined in Listing 2-2 calls both of the `InstallHandler` functions to install a pair of drag handlers for the window that it creates. In `MyDoNewWindow`, the window pointer is passed to the `InstallHandler` functions.

In the scenario created in the last two example functions, the Drag Manager will use the `MyDefaultTrackingHandler` and `MyDefaultReceiveHandler` functions for all windows in your application. The Drag Manager will also use the `MyTrackingHandler` and `MyReceiveHandler` functions for windows that were specifically created by `MyDoNewWindow`.

To remove a drag handler from the Drag Manager's handler registry, call the corresponding `RemoveHandler` functions. Listing 2-3 shows how to remove drag handlers.

Listing 2-3 Removing Drag Handlers from Individual Windows

```
OSErr MyDoCloseWindow(WindowPtr theWindow)

{
    RemoveTrackingHandler(MyTrackingHandler, theWindow);
    RemoveReceiveHandler(MyReceiveHandler, theWindow);

    DisposeWindow(theWindow);

    return(noErr);
}
```

The function `MyDoCloseWindow` defined in Listing 2-3 demonstrates the use of the `RemoveHandler` functions. The same handler address and window pointer used to install a handler is used to remove a handler. If nil (NIL) is used as the window pointer, the Drag Manager will attempt to remove the default handler with that address.

Recognizing the Start of a Drag

When the user clicks on an item or a selection of items in your application and begins to move the mouse without first releasing the mouse button, the user is making a gesture to begin dragging the selected items.

The Drag Manager provides a function `WaitMouseMoved` that you can use to determine if the mouse has moved far enough after a `mouseDown` event to start a drag. Listing 2-4 show how to determine if a `mouseDown` event should result in a drag.

Listing 2-4 Handling a Mouse Down Event with Dragging

```
OSErr MyDoMouseDown(EventRecord *theEvent)

{   OSErr       result = noErr;
    short       thePart;
    WindowPtr   theWindow;
    Boolean     onItem;

    thePart = FindWindow(theEvent->where, &theWindow);

    switch(thePart) {
        case inContent:
            if (theWindow == FrontWindow()) {
                MyDoContentClick(theEvent, theWindow, &onItem);
                if (onItem && WaitMouseMoved(theEvent->where)) {
                    result = MyDoStartDrag(theEvent, theWindow);
                }
            } else {
                SelectWindow(theWindow);
            }

        case ...

    }

    return(result);
}
```

The function `MyDoMouseDown` defined in Listing 2-4 shows a simplified mouse down event service routine. Only the code for handling an `inContent` part code from `FindWindow` is shown. The `MyDoContentClick` function either selects, extends the selection or deselects an item in the application's document window. The `onItem` parameter to `MyDoContentClick` returns `true` if the mouse down event occurred on a draggable item.

If the `mouseDown` event occurred on a draggable object, the `WaitMouseMoved` function is then called, which is a Drag Manager function that waits for the mouse button to be released or the mouse to move from its mouse down location. `WaitMouseMoved` returns `true` if the mouse moved and it returns `false` if the mouse button is released before the mouse moved.

The `MyDoStartDrag` function, which is defined later in Listing 2-5, is called if the user gestures to start a drag.

Performing a Drag

To perform a drag, you need to first create a new drag reference by calling the `NewDrag` function. The `NewDrag` function returns a reference number that you use to refer to a specific drag process in subsequent function calls to the Drag Manager.

After creating a new drag reference, the drag item flavors that describe the contents of the drag are added to the drag by calling the Drag Manager's `AddDragItemFlavor` function.

Specific callback procedures can be added to the drag that the Drag Manager will call in response to several Drag Manager events. These callback procedures allow your application to defer the sending of data to the receiver of the drag, or to change the input or drawing behaviors of the Drag Manager.

When all of the data describing the items contained in the drag has been given to the Drag Manager, call `TrackDrag` to actually perform the drag. After a drag is performed, the `DisposeDrag` function is used to release the memory associated with a drag process.

Listing 2-5 demonstrates each of these steps by showing the implementation of the `MyDoStartDrag` function that is called by the `MyDoMouseDown` function defined in the previous section.

Listing 2-5 Performing a Drag

```
OSErr MyDoStartDrag(EventRecord *theEvent, WindowPtr theWindow)

{   OSErr           result;
    DragReference   theDrag;
    RgnHandle       dragRegion;

    if (result = NewDrag(&theDrag)) {
        return(result);
    }

    if (result = MyDoAddFlavors(theWindow, theDrag)) {
        DisposeDrag(theDrag);
        return(result);
    }

    dragRegion = NewRgn();

    if (result = MyGetDragRegion(theWindow, dragRegion, theDrag)) {
        DisposeDrag(theDrag);
        return(result);
    }

    if (result = SetDragSendProc(theDrag, MySendDataProc, 0L)) {
        DisposeDrag(theDrag);
        return(result);
    }

    result = TrackDrag(theDrag, theEvent, dragRegion);

    DisposeRgn(dragRegion);
    DisposeDrag(theDrag);

    return(result);
}
```

The `MyDoStartDrag` function that is defined in Listing 2-5 first creates a new drag by calling the `NewDrag` function. It then calls the `MyDoAddFlavors` function, which is defined in Listing 2-6, to add the application's drag item flavors to the drag. The drag region for the drag is created by calling the application's `MyGetDragRegion` function, which is defined in Listing 2-7. The `SetDragSendProc` function is then called to allow the application to prepare and send data to a receiver at the end of the drag operation. The `TrackDrag` function is called to perform the drag. Finally, the `DisposeDrag` function is called to release all of the memory used to perform the drag.

Adding Drag Item Flavors

In the next program listing, the `MyDoAddFlavors` function is defined, which demonstrates how a set of drag item flavors are added to a drag. The drag item flavors describe the contents of a drag to the Drag Manager and to any potential receiver of the drag.

To add drag item flavors to a drag, use the `AddDragItemFlavor` function. The `AddDragItemFlavor` function requires a drag reference number to add the flavor to. You also provide an item reference number when adding flavors. You may specify any item numbers when adding items. Use the same item number for adding flavors to the same item. Using different item numbers results in new items being created.

Listing 2-6 shows how to add drag item flavors to a drag.

Listing 2-6 Adding Drag Item Flavors

```
OSErr MyDoAddFlavors(WindowPtr theWindow, DragReference theDrag)

{   MyDocumentItem  *theItem;

    theItem = MyGetFirstSelectedItem(theWindow);

    while (theItem) {

        AddDragItemFlavor(theDrag, (ItemReference) theItem, 'DATA',
                          theItem->dataPtr, theItem->dataSize,
                          flavorSenderOnly);

        AddDragItemFlavor(theDrag, (ItemReference) theItem, 'TEXT',
                          0L, 0L, 0);

        if (theItem->hasStyles) {
            AddDragItemFlavor(theDrag, (ItemReference) theItem, 'styl',
                              0L, 0L, 0);
        }

        theItem = theItem->nextSelectedItem;
    }
}
```

The `MyDoAddFlavors` function defined in Listing 2-6 uses the Drag Manager's `AddDragItemFlavor` function to add either two or three flavors to the drag for each item that is selected in the window.

This function goes through a loop of all of the selected items in the given window. The `AddDragItemFlavor` function is used to add the first flavor to the drag. This first flavor is of the application's own internal data type `'DATA'`. A pointer to the data and the data's size is given to the `AddDragItemFlavor` function. The data given to the `AddDragItemFlavor` function is copied (or cached) into the given drag by the Drag Manager. The `flavorSenderOnly` flag is set for this flavor to make the `'DATA'` flavor visible only to the sending application.

The item reference number used for the first `'DATA'` flavor and the following flavors is derived from `theItem` pointer used by the application. Since each `MyDocumentItem` element will have a unique address, the pointer to these elements may be used as unique item reference when adding new items to a drag.

The second call to `AddDragItemFlavor` uses the same document item pointer as the drag item reference number. Since this is the same item number used in the last call, the second flavor is added to the same drag item. This flavor is of type `'TEXT'`.

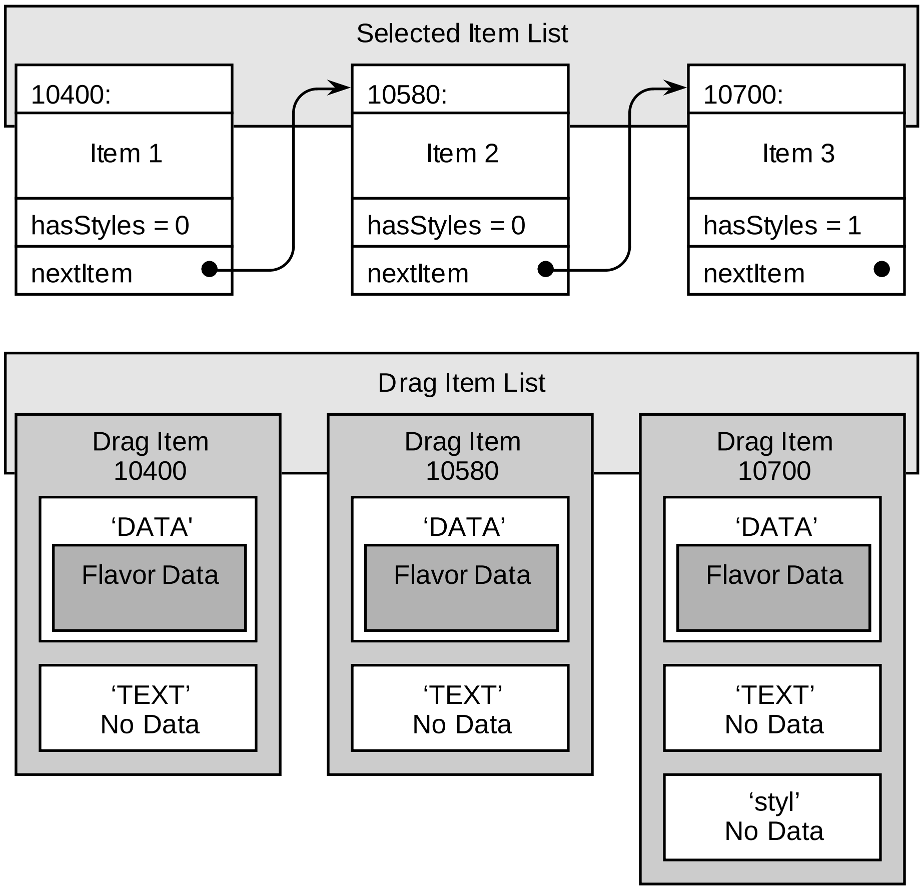
Suppose that you do not want to provide the plain text data to the Drag Manager unless this flavor is specifically requested by the receiver of a drag. A `NIL` pointer and zero size is passed to `AddDragItemFlavor`. By passing `NIL`, the Drag Manager will mark the flavor as not being cached in the drag and will call the drag's `DragSendDataProc` if the data is requested.

In our example, an item in the selection may have text styles, and if it does, it also adds a '`styl`' flavor. Again, the same item reference number is used to add the flavor to the same drag item. The flavor data is not provided; it will only be created by the `DragSendDataProc` if needed.

The `MyDoAddFlavors` function loops to the next selected item in its list. When it adds the flavors for the next item, it will be using a different item number (since the address of the next item is different), which will result in a new item being created.

To illustrate the effect of calling the `MyDoAddFlavors` function defined above, Figure 2-6 shows an example list of selected items and the resulting drag items and drag item flavors.

Figure 2-6 Drag Items and Drag Item Flavors from Application Example



Creating the Drag Region

In the next program listing, the `MyGetDragRegion` function is defined, which demonstrates how to create the drag region for a drag. The drag region is the region drawn by the Drag Manager in a dithered 50% gray pattern that follows the mouse on the screen during the drag.

Listing 2-7 shows how to create a drag region for the drag.

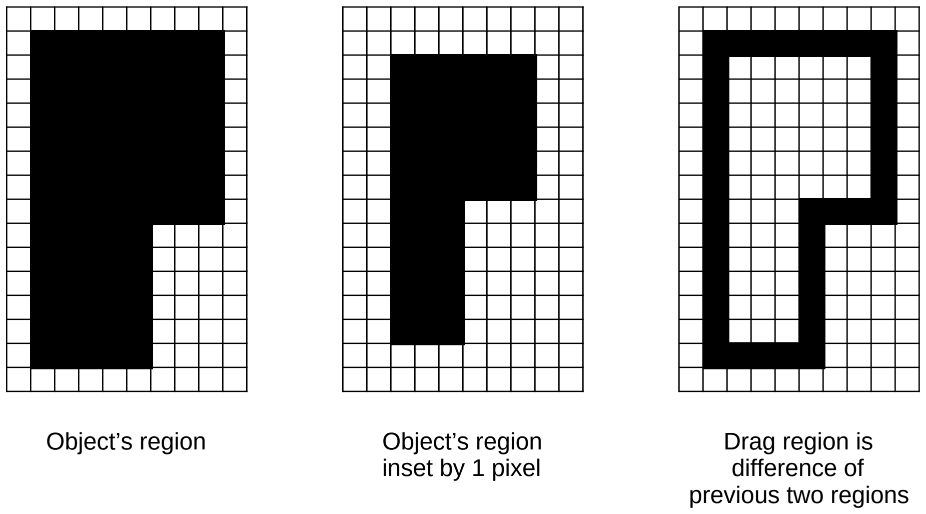
Listing 2-7 Creating a drag region

```
OSErr MyGetDragRegion(WindowPtr theWindow, RgnHandle dragRegion,
                      DragReference theDragRef)

{   MyDocumentItem  *theItem;
    RgnHandle       tempRgn;
    Point           globalPoint;

    theItem = MyGetFirstSelectedItem(theWindow);
    tempRgn = NewRgn();

    globalPoint.v = globalPoint.h = 0;
    LocalToGlobal(&globalPoint);

    while (theItem) {

        CopyRgn(theItem->theRegion, tempRgn);
        InsetRgn(tempRgn, 1, 1);
        DiffRgn(theItem->theRegion, tempRgn, tempRgn);

        OffsetRgn(tempRgn, globalPoint.h, globalPoint.v);
        UnionRgn(tempRgn, dragRegion, dragRegion);

        SetDragItemBounds(theDrag, (ItemReference) theItem,
                          &(**tempRgn).rgnBBox);

        theItem = theItem->nextSelectedItem;
    }

    DisposeRgn(tempRgn);
    return(noErr);
}
```

The `MyGetDragRegion` function defined in Listing 2-7 loops through all of the selected items in the given window. For each selected item in the window, the region of the item is added to the `dragRegion` and the item's bounding rectangle is set by using the Drag Manager's `SetDragItemBounds` function.

The function uses `CopyRgn` to copy the item's region into `tempRgn`. The `tempRgn` is inset by one pixel and then subtracted from the original region with `DiffRgn`. Performing these three steps creates a region that has the same outline as the original region but is only one pixel thick. Figure 2-7 demonstrates the effect of this procedure on the region.

Figure 2-7 Creating a drag region


Object's region
Object's region inset by 1 pixel
Drag region is difference of previous two regions

Each of the individual drag regions that are created for each item being dragged is offset from local coordinates to global screen coordinates by the `OffsetRgn` call. Each item's drag region is added to the final drag region with the `UnionRgn` call. It is this composite region of each item's individual drag region that is returned by this function and used in the call to `TrackDrag`.

The `MyGetDragRegion` function also calls the Drag Manager's `SetDragItemBounds` function for each item in the drag. `SetDragItemBounds` is used to provide the bounding rectangle of each of the individual items in the drag. This rectangle is also specified in global screen coordinates. During a drag, Drag Manager clients may ask for the bounding rectangle of any drag item by using the `GetDragItemBounds` function. The `GetDragItemBounds` function returns the item's bounds relative to the current mouse location.

Tracking a Drag

During a drag, as the user moves the mouse on the screen, searching for a destination for the drag items, the Drag Manager sends a sequence of tracking messages to the tracking handlers that are registered for the window that the mouse is over.

Your tracking handler is responsible for providing all of the feedback to the user that the group of items being dragged can be accepted into the current destination. Your tracking handler can inspect the drag item flavors contained in a drag and highlight your application's windows or part of your application's windows in response to data that your application can accept.

Listing 2-8 shows an example of a very simple tracking handler. This tracking handler highlights the destination window if each of the drag items contains either the application's own 'DATA' flavor or the 'TEXT' flavor. It also calls the application's `MyTrackItemUnderMouse` function that could be defined to highlight other parts of the window as the mouse moves through the window.

Listing 2-8 Example Tracking Handler

```
OSErr MyTrackingHandler(DragTrackingMessage theMessage, WindowPtr theWindow,
                        void *handlerRefCon, DragReference theDrag)

{   GlobalsPtr      myGlobals = (GlobalsPtr) handlerRefCon;
    Point           mouse, localMouse;
    DragAttributes  attributes;
    RgnHandle       hiliteRgn;

    GetDragAttributes(theDrag, &attributes);

    switch(theMessage) {

        case dragTrackingEnterHandler:
            break;

        case dragTrackingEnterWindow:
            myGlobals->canAcceptDrag = IsMyTypeAvailable(theDrag);
            myGlobals->inContent = false;
            break;

        case dragTrackingInWindow:
            if (!myGlobals->canAcceptDrag)
                break;

            GetDragMouse(theDrag, &mouse, 0L);
            localMouse = mouse;
            GlobalToLocal(&localMouse);

            if (attributes & dragHasLeftSenderWindow) {
                if (PtInRect(localMouse, &(**(myGlobals->theTE)).viewRect)) {

                    if (!myGlobals->inContent) {
                        RectRgn(hiliteRgn = NewRgn(),
                                &(**(myGlobals->theTE)).viewRect);
                        ShowDragHilite(theDrag, hiliteRgn, true);
                        DisposeRgn(hiliteRgn);
                        myGlobals->inContent = true;
                    }

                } else {

                    if (myGlobals->inContent) {
                        HideDragHilite(theDrag);
                        myGlobals->inContent = false;
                    }

                }
            }

            MyTrackItemUnderMouse(localMouse, theWindow);
            break;

        case dragTrackingLeaveWindow:
            if (myGlobals->canAcceptDrag && myGlobals->inContent) {
                HideDragHilite(theDrag);
            }
            myGlobals->canAcceptDrag = false;
            break;

        case dragTrackingLeaveHandler:
            break;
    }

    return(noErr);
}
```

The `MyTrackingHandler` function defined in Listing 2-8 switches on the given message from the Drag Manager. This example does not require any setup or memory allocation when the handler is entered or left, so the `dragTrackingEnterHandler` and the `dragTrackingLeaveHandler` messages are ignored.

When `MyTrackingHandler` receives the `dragTrackingEnterWindow` message, it calls the application's `IsMyTypeAvailable` function, which is defined in Listing 2-9. It returns either `true` or `false`, depending on whether a type is available in each of the drag items that the application window can accept. The result of this function is stored in the application's global variable `canAcceptDrag`. Another global variable `inContent` is used to keep track of whether the mouse is inside the area of the window that can be highlighted during a drag.

When the `dragTrackingInWindow` message is received, if the window can accept the drag, `GetDragMouse` is called to get the mouse location. The code then checks to make sure that the drag has left the source window. The *Drag and Drop Human Interface Guidelines* specify that drag highlighting should only occur after the mouse has left the source window. The local mouse coordinate is then checked against the region that will highlight and either `ShowDragHilite` or `HideDragHilite` is then called to show or hide the highlighting. Finally, the application's `MyTrackItemUnderMouse` is called. Presumably, `MyTrackItemUnderMouse` would use the given `localMouse` location to determine if the mouse is over an object that must also be highlighted.

When the `dragTrackingLeaveWindow` message is received, if the window can accept the drag and the highlighting is still visible, `HideDragHilite` is called to remove the window highlighting.

Determining What is Being Dragged

To determine what drag items and drag item flavors are available in a drag, use the `CountDragItems`, `GetDragItemReferenceNumber`, `CountDragItemFlavors`, `GetFlavorType` and `GetFlavorFlags` functions.

Given a drag reference number, `CountDragItems` returns the number of drag items contained in the drag. The `GetDragItemReferenceNumber` function returns an item reference number given a drag item's index. Given an item reference number, `CountDragItemFlavors` returns the number of drag item flavors in a drag item. The `GetFlavorType` function returns the type of a flavor given the flavor's index. The `GetFlavorFlags` function returns the flavor flags of a flavor.

Listing 2-9 shows the `IsMyTypeAvailable` function which demonstrates the use of these functions to determine if at least one of the application's flavor types is available in each item being dragged.

Listing 2-9 Determining What Flavors Are Available

```
Boolean IsMyTypeAvailable(DragReference theDrag)

{   short           items, index;
    FlavorFlags     theFlags;
    ItemReference   theItem;
    OSErr           result;

    CountDragItems(theDrag, &items);

    for (index = 1; index <= items; index++) {
        GetDragItemReferenceNumber(theDrag, index, &theItem);

        result = GetFlavorFlags(theDrag, theItem, 'DATA', &theFlags);
        if ((result == noErr) && (theFlags & flavorSenderOnly)) {
            continue;
        }

        result = GetFlavorFlags(theDrag, theItem, 'TEXT', &theFlags);
        if (result == noErr) {
            continue;
        }

        return(false);
    }

    return(true);
}
```

The IsMyTypeAvailable function defined in Listing 2-9 counts the number of items in the drag and begins a loop through each of the items. It continues with the next drag item when it encounters either a flavor of type 'DATA' or a flavor of type 'TEXT'. The IsMyTypeAvailable function returns false if it encounters a drag item that does not contain at least one of these two flavors. It returns true after it verifies that the last drag item contains either a 'DATA' or 'TEXT' flavor.

Note that when IsMyTypeAvailable checks for the 'DATA' flavor, it also checks to make sure that the flavorSenderOnly flag is set, which guarantees that the private 'DATA' flavor has come directly from this application, and not some other application.

The result that this function returns is used my the MyTrackingHandler function defined in Listing 2-8 to determine if the window highlighting should be drawn.

Receiving a Drop

When the user has chosen a final destination for the items being dragged, the Drag Manager calls the destination window's receive drop handlers to allow your application to request the drag item flavors that it wishes to accept.

Your receive drop handler gets a pointer to the destination window, the handler's reference constant, and the drag reference. Your receive drop handler can user the Drag Manager's CountDragItems, GetDragItemReferenceNumber, CountDragItemFlavors, GetFlavorType, GetFlavorFlags and GetFlavorData functions to determine what items, flavors, and data are contained in the drag.

Listing 2-10 shows an example receive handler that iterates through each of the items in the received drag. If an item contains the application's internal 'DATA' flavor, the data is inserted directly into the document. If there is no 'DATA' flavor in the item, the handler checks for a 'TEXT' flavor. If a 'TEXT' flavor exists, it attempts to get data for both 'styl' and 'TEXT' types. The text data is inserted into the document regardless of the existence of the optional styl type.

Listing 2-10 Example Receive Handler

```
OSErr MyReceiveHandler(WindowPtr theWindow, void *handlerRefCon,
                       DragReference theDrag)

{   GlobalsPtr      myGlobals = (GlobalsPtr) handlerRefCon;
    Point           mouse;
    short           items, index;
    ItemReference   theItem;
    FlavorFlags     theFlags;
    Size            dataSize, stylSize;
    char            *theData, *theStyl;
    OSErr           result;

    GetDragMouse(theDrag, &mouse, 0L);

    CountDragItems(theDrag, &items);

    for (index = 1; index <= items; index++) {
        GetDragItemReferenceNumber(theDrag, index, &theItem);

        result = GetFlavorFlags(theDrag, theItem, 'DATA', &theFlags);
        if ((result == noErr) && (theFlags & flavorSenderOnly)) {

            GetFlavorDataSize(theDrag, theItem, 'DATA', &dataSize);
            theData = NewPtr(dataSize);
            GetFlavorData(theDrag, theItem, 'DATA', theData, dataSize, 0L);

            MyInsertDataAtPoint(theData, dataSize, mouse, theWindow);

            DisposePtr(theData);

            continue;
        }

        result = GetFlavorFlags(theDrag, theItem, 'TEXT', &theFlags);
        if (result == noErr) {

            theStyl = 0L;
            if (GetFlavorDataSize(theDrag, theItem,
                                  'styl', &stylSize) == noErr) {

                theStyl = NewPtr(stylSize);
                GetFlavorData(theDrag, theItem, 'styl',
                              theStyl, stylSize, 0L);
            }

            GetFlavorDataSize(theDrag, theItem, 'TEXT', &dataSize);
            theData = NewPtr(dataSize);
            GetFlavorData(theDrag, theItem, 'TEXT', theData, dataSize, 0L);

            MyInsertStylTextAtPoint(theData, dataSize,
                                    theStyl, stylSize, mouse, theWindow);

            DisposePtr(theData);
            if (theStyl) {
                DisposePtr(theStyl);
            }
        }

    return(noErr);
}
```

The MyReceiveHandler function defined in Listing 2-10 counts the number of items in the drag and begins a loop through each of the items. It checks for the existence of a 'DATA' flavor by using the GetFlavorFlags function. If this flavor exists in the item, the size of the flavor's data is obtained by calling GetFlavorDataSize, memory is allocated for the data and GetFlavorData is used to get the flavor data from the Drag Manager. The application's MyInsertDataAtPoint function is called to insert the data into the document at the given mouse point.

If the 'DATA' type is not available in the given item, GetFlavorFlags is called to check for the existence of the 'TEXT' flavor type. If there is a 'TEXT' flavor in the item, the handler then also checks for a 'styl' flavor. If this flavor exists, the styl data is copied into the theStyl buffer. The 'TEXT' data is copied into the theData buffer. The application's MyInsertStylTextAtPoint function is used to insert the TEXT data with the optional styl information into the document at the given mouse point.

The MyReceiveHandler continues with each item in the drag by inserting acceptable data into the destination document window.

Providing Flavor Data on Demand

If the receiver of a drop requests a flavor whose data has not been cached by the Drag Manager (as is the case for the 'TEXT' and 'styl' flavors in our example), the Drag Manager calls the drag's DragSendDataProc to obtain the data when needed.

The Drag Manager calls your DragSendDataProc with the requested flavor type, your handler's reference constant, and the item and drag reference numbers. In your DragSendDataProc, call the SetDragItemFlavorData function to provide the requested flavor data to the Drag Manager. Listing 2-11 shows an example send data procedure.

Listing 2-11 Example Send Data Procedure

```
OSErr MySendDataProc(FlavorType theType, void *dragSendRefCon,
                     ItemReference theItem, DragReference theDrag)

{   MyDocumentItem      *myItem;
    Handle              myData;
    OSErr               result;

    myItem = (MyDocumentItem *) theItem;

    switch(theType) {
        case 'TEXT':
            myData = MyConvertItemToText(myItem);
            HLock(myData);
            result = SetDragItemFlavorData(theDrag, theItem, 'TEXT', *myData,
                                           GetHandleSize(myData), 0L);
            HUnlock(myData);
            DisposeHandle(myData);
            break;

        case 'styl':
            myData = MyConvertItemToStyl(myItem);
            HLock(myData);
            result = SetDragItemFlavorData(theDrag, theItem, 'styl', *myData,
                                           GetHandleSize(myData), 0L);
            HUnlock(myData);
            DisposeHandle(myData);
            break;

        default:
            result = badDragFlavorErr;
            break;
    }

    return(result);
}
```

The `MySendDataProc` function defined in Listing 2-11 provides both the 'TEXT' and 'styl' flavors to the Drag Manager. The routine uses the item reference number as a pointer to the application's `MyDocumentItem` data structure (this pointer was used when adding the drag item flavors with `AddDragItemFlavor`). The routine calls its own `MyConvertItemToText` and `MyConvertItemToStyl` functions to get the needed data. The Drag Manager's `SetDragItemFlavorData` function is then called to pass the requested data to the Drag Manager.

Drag Manager Reference

This section describes the Drag Manager's constants, data structures and routines.

The "Constants" section describes the constants received from the Drag Manager and used when calling Drag Manager routines. The "Data Structures" section shows the data structures used to refer to drags, drag items, drag item flavors, and special drag item flavor data. The "Drag Manager Routines" section describes Drag Manager routines for installing and removing drag handlers, creating and disposing of drag references, adding drag item flavors to a drag, providing drag callback routines, tracking a drag, getting drag item information, getting drag status information, window highlighting and Drag Manager related utilities. The "Application-Defined Routines" section describes both the drag handler and drag callback functions.

Constants

The constants described in this section are received from the Drag Manager and used when calling Drag Manager routines.

Gestalt Selector and Response Bits

You can determine if the Drag Manager is available by calling the `Gestalt` function with the selector `gestaltDragMgrAttr`.

```
#define gestaltDragMgrAttr      'drag'  // Drag Manager attributes
```

The `Gestalt` function returns information by setting or clearing bits in the response parameter. The following constant defines the bit currently used:

```
#define gestaltDragMgrPresent   0       // Drag Manager is present
```

Constant description

`gestaltDragMgrPresent`
: Set if the Drag Manager is installed. When the Drag Manager is installed, all of the features contained within this document are available for use by your application.

You can determine if the new TextEdit function `TEGetHiliteRgn` is available by calling the `Gestalt` function with the selector `gestaltTEAttr`.

```
#define gestaltTEAttr            'teat' // TextEdit attributes
```

The `Gestalt` function returns information by setting or clearing bits in the response parameter. The following constant defines the bit currently used:

```
#define gestaltTEHasGetHiliteRgn 0      // TEGetHiliteRgn present
```

Constant description

`gestaltTEHasGetHiliteRgn`
: Set if the TextEdit function `TEGetHiliteRgn` is available. This new TextEdit function was introduced with the Drag Manager.

Flavor Flags

The following constants are used to provide additional attribute information about drag item flavors. These constants are used when calling the `AddFlavor` functions and can be obtained using the `GetFlavorFlags` function.

```
#define flavorSenderOnly            0x00000001
#define flavorSenderTranslated      0x00000002
#define flavorNotSaved              0x00000004
#define flavorSystemTranslated      0x00000100
```

Constant descriptions

`flavorSenderOnly`
: Set by the sender if the flavor should only be available to the sender of a drag. If this flag is set when adding the flavor to a drag, no Drag Manager clients other than the sender can receive this flavor.

`flavorSenderTranslated`
: Set by the sender if the flavor data is translated by the sender. This flag is useful to a receiver if the receiver needs to determine if the sender is performing its own translation to generate this data type. Typically, receivers that store dragged data without interpreting each data type do not store translated types. Flavor types marked with this flag are not stored by the Finder in clipping files.

`flavorNotSaved`
: Set by the sender if the flavor data should not be stored by the receiver. This flag is useful for marking flavor data that will become stale after the drag has completed. Receivers that store dragged data should not store flavors that are marked with this flag. Flavor types marked with this flag are not stored by the Finder in clipping files.

`flavorSystemTranslated`
: Set if the flavor data is provided by the Translation Manager. If this flavor is requested, the Drag Manager will obtain any required data types from the sender and then it will use the Translation Manager to provide the data that the receiver requested. Typically, receivers that store dragged data without interpreting each data type do not store translated types. Flavor types marked with this flag are not stored by the Finder in clipping files.

Drag Attributes

The following constants are used to provide additional attribute information about a drag that is in progress. The attribute flags provide information about the window and application that the drag is currently occurring in. During a drag, the current drag attributes can be obtained by calling the `GetDragAttributes` function.

```
#define dragHasLeftSenderWindow         0x00000001
#define dragInsideSenderApplication     0x00000002
#define dragInsideSenderWindow          0x00000004
```

Constant descriptions

`dragHasLeftSenderWindow`
: Set if the drag has left the source window since the beginning of the drag. This flag is useful for providing window highlighting after the user has moved the mouse outside of the source window.

`dragInsideSenderApplication`
: Set if the drag is currently in any window that belongs to the application that started the drag.

`dragInsideSenderWindow`
: Set if the drag is currently in the same window that the drag started from.

Special Flavor Kinds

The following constants are used to identify special flavor kinds that are defined by the Drag Manager.

```
#define flavorTypeHFS           'hfs '
#define flavorTypePromiseHFS    'phfs'
#define flavorTypeDirectory     'diry'
```

Constant descriptions

`flavorTypeHFS`
: The flavor type for an HFS file system object. The Finder uses HFS flavors when dragging existing file system objects. The HFS flavor data is defined by the `HFSFlavor` structure defined below.

`flavorTypePromiseHFS`
: The flavor type for promising an HFS file system object to the receiver of the drag. This flavor type can be used when a file could be created if the destination of the drag can accept file system objects. The `PromiseHFSFlavor` structure defined below is used to access the data in this flavor type.

`flavorTypeDirectory`
: The flavor type for a AOCE directory specification. Refer to the AOCE documentation for definition of the `DSSpec` data structure.

Zoom Acceleration

The following constants are used when specifying an `acceleration` constant to either the `ZoomRects` or `ZoomRegion` functions.

```
#define zoomNoAcceleration         0
#define zoomAccelerate             1
#define zoomDecelerate             2
```

Constant descriptions

`zoomNoAcceleration`
: Use linear interpolation for each frame of animation between the source and destination.

`zoomAccelerate`
: Increment the step size for each frame of animation between the source and destination. This option produces the visual appearance of the animation speeding up as it approaches the destination.

`zoomDecelerate`
: Decrement the step size for each frame of animation between the source and destination. This option produces the visual appearance of the animation slowing down as it approaches the destination.

Data Structures

This section describes the data structures that you use to identify drags, drag items, drag item flavors and special drag item flavor data.

Drag Reference

The Drag Reference is a reference to a drag object. Before calling any other Drag Manager routine, you must first create a new Drag Reference by calling the `NewDrag` function. The Drag Reference that is returned by `NewDrag` is used in all subsequent calls to the Drag Manager. Use the `DisposeDrag` function to dispose of a Drag Reference after you are finished using it.

IMPORTANT
The meaning of the bits in a drag reference is internal to the Drag Manager. You should not attempt to interpret the value of the drag reference.▲

```
typedef unsigned long DragReference;
```

Drag Item Reference

The Drag Item Reference is a reference number used to refer to a single item in a drag. Drag Item Reference numbers are created by the sender application when adding drag item flavor information to a drag. Drag Item Reference numbers are created by and should only be interpreted by the sender application.

```
typedef unsigned long ItemReference;
```

Flavor Type

The Flavor Type is a four character type that describes the format of drag item flavor data. The Flavor Type has the same function as a scrap type; it designates the format of the associated data. Any scrap type or resource type may be used.

Four character types consisting of only lower-case letters are reserved by Apple. You can be guaranteed a unique type by using your registered application signature.

```
typedef ResType FlavorType;
```

HFS Flavor

The Drag Manager defines a special flavor type for dragging file system objects. The HFS drag item flavor is used when dragging document and folder icons in the Finder. The HFS drag item flavor data structure is defined by the `HFSFlavor` data type.

```
typedef struct HFSFlavor {
    OSType          fileType;         // file type
    OSType          fileCreator;      // file creator
    unsigned short  fdFlags;          // Finder flags
    FSSpec          fileSpec;         // file system specification
} HFSFlavor;
```

Field descriptions

`fileType`	The file type of the object.
`fileCreator`	The file creator of the object.
`fdFlags`	The Finder flags of the object (Finder flags are defined in the "Finder Interface" chapter of Inside Macintosh).
`fileSpec`	The FSSpec record for the object.

Promise HFS Flavor

The Drag Manager defines a special data flavor for promising file system objects. The Promise HFS flavor is used when you wish to create a new file when dragging to the Finder. The flavor consists of an array of the following `PromiseHFSFlavor` structures, with the first entry being the preferred file type you would like to create, and subsequent array entries being file types in descending preference. This structure allows you to create the file in your `DragSendDataProc`, and provide the `FSSpec` for the new file at that time.

```
typedef struct PromiseHFSFlavor {
    OSType          fileType;         // file type
    OSType          fileCreator;      // file creator
    unsigned short  fdFlags;          // Finder flags
    FlavorType      promisedFlavor;   // FSSpec flavor
} HFSFlavor;
```

Field descriptions

`fileType`	The potential file type of the object.
`fileCreator`	The potential file creator of the object.
`fdFlags`	The expected Finder flags of the object (Finder flags are defined in the "Finder Interface" chapter of Inside Macintosh).
`promisedFlavor`	The `FlavorType` of a separate promised flavor to contain the `FSSpec` for the new file. Call `AddDragItemFlavor` and promise this separate flavor if you wish to create the file in your `DragSendDataProc`. After providing an `FSSpec` in this flavor, the Finder will move the new file to the drop location. If you wish to create the file before the drag and provide the `FSSpec` data up front, create the new file in the Temporary Items folder so it does not prematurely appear in an open Finder window.

Drag Manager Routines

This section describes the Drag Manager routines you can use to start a drag from your application, gain control when the user drags an object into one of your application's windows, support the drag and drop user interface, and send and receive data as part of a drop transaction.

Installing and Removing Drag Handler Routines

You can use the Drag Manager to install or remove drag handler routines for your entire application or for one of your application's windows. The Drag Manager provides a pair of install/remove functions for each of the two different handler types.

InstallTrackingHandler

Use the `InstallTrackingHandler` function to install a tracking handler routine for the Drag Manager to use while the user drags through your application's windows.

```
pascal OSErr InstallTrackingHandler
                          (DragTrackingHandler trackingHandler,
                           WindowPtr theWindow,
                           void *handlerRefCon);
```

`trackingHandler`
Pointer to a `DragTrackingHandler` routine.

`theWindow` A pointer to the window to install the drag tracking handler for. When the cursor moves into this window during a drag, the Drag Manager sends tracking messages to the tracking handler routine. If this parameter is `NIL`, the tracking handler receives messages for all open windows in your application.

`handlerRefCon`
A reference constant that will be forwarded to your drag tracking handler routine when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your drag tracking handler.

DESCRIPTION

The `InstallTrackingHandler` function installs a tracking handler for one of your application's windows. Installing a tracking handler allows your application to track the user's movements through your application's windows during a drag. You may install more than one drag tracking handler on a single window.

The Drag Manager sequentially calls all of the drag tracking handlers installed on a window when the user moves the cursor over that window during a drag.

By specifying a value of `NIL` in `theWindow`, the tracking handler is installed in the default handler space for your application. Drag tracking handlers installed in this way are called when the user moves the mouse over any window that belongs to your application.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`duplicateHandlerErr`	-1855	Handler already exists

InstallReceiveHandler

Use the `InstallReceiveHandler` function to install a drag receive handler routine for the Drag Manager to use when the user releases the mouse button while dragging over one of your application's windows.

```
pascal OSErr InstallReceiveHandler
                              (DragReceiveHandler receiveHandler,
                               WindowPtr theWindow,
                               void *handlerRefCon);
```

`receiveHandler`
Pointer to a `DragReceiveHandler` routine.

`theWindow` A pointer to the window to install the receive drop handler for. When a drop occurs over this window, the Drag Manager calls this routine to allow your application to accept the drag. If this parameter is `NIL`, the receive handler is called regardless of which window the drop occurred in your application.

`handlerRefCon`
A reference constant that will be forwarded to your receive drop handler routine when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your drag receive handler.

DESCRIPTION

The `InstallReceiveHandler` function installs a drag receive handler for one of your application's windows. Installing a drag receive handler allows your application to accept a drag by getting drag item flavor data from the Drag Manager when the user releases the mouse button while dragging over one of your application's windows. You may install more than one drag receive handler on a single window.

The Drag Manager sequentially calls all of the drag receive handlers installed on a window when a drop occurs in that window.

By specifying a value of `NIL` in `theWindow`, the drag receive handler is installed in the default handler space for your application. Drag receive handlers installed in this way are called when a drop occurs in any window that belongs to your application.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`duplicateHandlerErr`	-1855	Handler already exists

RemoveTrackingHandler

Use the `RemoveTrackingHandler` function to remove a tracking handler routine from one of your application's windows.

```
pascal OSErr RemoveTrackingHandler
                              (DragTrackingHandler trackingHandler,
                               WindowPtr theWindow);
```

`trackingHandler`
Pointer to a `DragTrackingHandler` routine.

`theWindow` A pointer to the window to remove the drag tracking handler from. If this parameter is `NIL`, the given handler will be removed from the default handler space for your application.

DESCRIPTION

The `RemoveTrackingHandler` function removes a drag tracking handler from one of your application's windows.

By specifying a value of `NIL` in `theWindow`, the tracking handler is removed from the default handler space for your application.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`duplicateHandlerErr`	-1856	Handler not found

RemoveReceiveHandler

Use the `RemoveReceiveHandler` function to remove a drag receive handler routine from one of your application's windows.

```
pascal OSErr RemoveReceiveHandler
                              (DragReceiveHandler receiveHandler,
                               WindowPtr theWindow);
```

`receiveHandler`
Pointer to a `DragReceiveHandler` routine.

`theWindow` A pointer to the window to remove the drag receive handler from. If this parameter is `NIL`, the given handler will be removed from the default handler space for your application.

DESCRIPTION

The `RemoveReceiveHandler` function removes a drag receive handler from one of your application's windows.

By specifying a value of `NIL` in `theWindow`, the drag receive handler is removed from the default handler space for your application.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`duplicateHandlerErr`	-1856	Handler not found

Creating and Disposing of Drag References

You create a drag reference whenever your application wishes to start a drag. A drag reference is a token that is used in all subsequent calls to Drag Manager routines to refer to a particular drag.

NewDrag

Use the `NewDrag` function to create a new drag reference token.

```
pascal OSErr NewDrag (DragReference *theDragRef);
```

`theDragRef` The drag reference, which `NewDrag` fills in before returning.

DESCRIPTION

The `NewDrag` function allocates a new drag object for your application to use with the Drag Manager and returns a token to it in `theDrag` parameter. Use this drag reference in subsequent calls to the Drag Manager to identify the drag. This drag reference is required when adding drag item flavors and calling `TrackDrag`. Your installed drag handlers receive this drag reference so you can call other Drag Manager routines within your drag handlers.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory

DisposeDrag

Use the `DisposeDrag` function to dispose of a drag reference token and its associated data when a drag has been completed or if the drag reference is no longer needed.

```
pascal OSErr DisposeDrag (DragReference theDragRef);
```

`theDragRef` The drag reference of the drag object to dispose of.

DESCRIPTION

The DisposeDrag function disposes of the drag object that is identified by the given drag reference token. If the drag reference contains any drag item flavors, the memory associated with the drag item flavors is disposed of as well.

You should call DisposeDrag after a drag has been performed using TrackDrag or if a drag reference was created but is no longer needed.

RESULT CODES

noErr	0	No error
paramErr	-50	Parameter error
badDragRefErr	-1850	Unknown drag reference

Adding Drag Item Flavors

You can use the set of AddFlavor routines to create drag items and to provide the data types for each item. There is a special AddFlavor routine to add an HFS FSSpec flavor to a drag item.

AddDragItemFlavor

Use the AddDragItemFlavor function to create drag items and to add a data flavor to a drag item.

```
pascal OSErr AddDragItemFlavor (DragReference theDragRef,
                                ItemReference theItemRef,
                                FlavorType theType,
                                void *dataPtr,
                                Size dataSize,
                                FlavorFlags theFlags);
```

theDragRef	A drag reference.
theItemRef	The drag item reference to add the flavor to. You create new drag items by providing unique item reference numbers. By using the same item reference number as in a previous call to AddDragItemFlavor, the flavor is added to an existing item. You may use any item reference number when adding flavors to items.
theType	The data type of the flavor to add. This may be any four-character scrap type. Use you application's signature for a unique type for your own internal use.
dataPtr	A pointer to the flavor data to add.

`dataSize`	The size, in bytes, of the flavor data to add.
`theFlags`	A set of attributes to set for this flavor.

DESCRIPTION

The `AddDragItemFlavor` function adds a drag item flavor to a drag item. A new drag item is created if the given item reference number is different than any other item reference numbers. When adding multiple flavors to the same item, supply the same item reference number.

In many cases it is easiest to use index numbers as item reference numbers (1, 2, 3...). Item reference numbers are only used as unique “key” numbers for each item. Item reference numbers do not need to be given in order, nor must they be sequential. Depending on your application, it might be easier to use your own internal memory addresses as item reference numbers (as long as each item being dragged has a unique item reference number).

Sometimes it is preferable to defer the creation of a particular data type until a receiver has specifically requested it (possibly if a lengthy translation is required). This can be done by passing `NIL` in the `data` parameter when adding a drag item flavor. Flavors that are added in this way will cause the Drag Manager to call the drag’s send data procedure if the flavor is requested to get the data from your application. See the section “Application Defined Routines” for information on writing a send data procedure for a drag.

You must add all of the drag item flavors to a drag before calling `TrackDrag`. Once `TrackDrag` is called, receiving applications may not operate properly if new drag items or drag item flavors are added.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`badDragRefErr`	-1850	Unknown drag reference
`duplicateFlavorErr`	-1853	Flavor type already exists

SetDragItemFlavorData

Use the `SetDragItemFlavorData` function to set the data or part of the data contained within an existing flavor.

```
pascal OSErr SetDragItemFlavorData (DragReference theDragRef,
                                    ItemReference theItemRef,
                                    FlavorType theType,
                                    void *dataPtr,
                                    Size dataSize,
                                    unsigned long dataOffset);
```

`theDragRef`	A drag reference.
`theItemRef`	A drag item reference of the item that contains the flavor you wish to set all or part of the data for.
`theType`	The data type of the existing flavor to set all or part of the data for.
`dataPtr`	A pointer to the flavor data.
`dataSize`	The size, in bytes, of the flavor data.
`dataOffset`	The offset, in bytes, into the flavor record to place the data specified by the `dataPtr` and `dataSize` parameters.

DESCRIPTION

The `SetDragItemFlavorData` function sets all or part of a given flavor's data. The data pointed to by `dataPtr` with the size given in `dataSize` is placed into the flavor record at the offset specified by `dataOffset`.

This function is commonly used in a scenario where a flavor's data is not added to the flavor when the flavor is created using `AddDragItemFlavor`. When the sender's `DragSendDataProc` is called, `SetDragItemFlavorData` can be used to provide the requested data to the Drag Manager. This method is useful when the data needs to be translated by the sender and would be to expensive to compute the data until required.

By using the `dataOffset` parameter, small pieces of the data may be placed into the flavor with each call to `SetDragItemFlavorData`.

This function, unlike the `AddFlavor` functions, may be called both before and during a drag.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference
`badDragFlavorErr`	-1852	Unknown flavor type

Providing Drag Callback Procedures

You provide drag callback procedures to the Drag Manager when you want to override the default behavior of the Drag Manager. You can override the mechanisms in the Drag Manager that provide data to a drop receiver, that sample the mouse and keyboard, and that draw the standard "dotted outline" drag feedback.

SetDragSendProc

Use the `SetDragSendProc` function to set the send data procedure for the Drag Manager to use with a particular drag.

```
pascal OSErr SetDragSendProc (DragReference theDragRef,
                              DragSendDataProc sendProc,
                              void *dragSendRefCon);
```

`theDragRef`	The drag reference that `SetDragSendProc` will set the drag send procedure for.
`sendProc`	The send data routine that will be called by the Drag Manager when the receiver of a drop requests the flavor data of a flavor that has not been cached by the Drag Manager.
`dragSendRefCon`	A reference constant that will be forwarded to your drag send procedure when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your drag send procedure.

DESCRIPTION

The `SetDragSendProc` function sets the drag send procedure for the given drag reference. A drag's drag send procedure is called by the Drag Manager when the receiver of a drop requests the data of a flavor and the requested flavor data is not currently cached by the Drag Manager.

The Drag Manager caches drag item flavor data when the flavor was added to a drag by calling `AddDragItemFlavor`. If `NIL` is passed to `AddDragItemFlavor` as the data pointer, the flavor data is not cached and the Drag Manager will attempt to cache the data by calling the drag send procedure.

You do not need to provide a drag send procedure if your application never passes `NIL` to `AddDragItemFlavor` when adding a drag item flavor to a drag.

Details for how to write a drag send procedure are covered in the "Application-Defined Routines" section below.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference

SetDragInputProc

Use the `SetDragInputProc` function to set the drag input procedure for the Drag Manager to use with a particular drag.

```
pascal OSErr SetDragInputProc (DragReference theDragRef,
                               DragInputProc inputProc,
                               void *dragInputRefCon);
```

`theDragRef`	The drag reference that`SetDragInputProc` will set the drag input procedure for.
`inputProc`	The drag input routine that will be called by the Drag Manager whenever the Drag Manager requires the location of the mouse, the state of the mouse button, and the status of the modifier keys.
`dragInputRefCon`	A reference constant that will be forwarded to your drag input procedure when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your drag input procedure.

DESCRIPTION

The `SetDragInputProc` function sets the drag input procedure for the given drag reference. A drag's drag input procedure is called by the Drag Manager whenever the Drag Manager requires the location of the mouse, the state of the mouse button, and the status of the modifier keys on the keyboard. The Drag Manager typically calls this routine once per cycle through the Drag Manager's main drag tracking loop.

Your drag input procedure may either modify the current state of the mouse and keyboard to slightly alter dragging behavior or entirely replace the input data to drive the drag completely by itself.

Details for how to write a drag input procedure are covered in the "Application-Defined Routines" section below.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference

SetDragDrawingProc

Use the `SetDragDrawingProc` function to set the drag drawing procedure for the Drag Manager to use with a particular drag.

```
pascal OSErr SetDragDrawingProc (DragReference theDragRef,
                                 DragDrawingProc drawingProc,
                                 void *dragDrawingRefCon);
```

`theDragRef`	The drag reference that `SetDragDrawingProc` will set the drag drawing procedure for.
`drawingProc`	The drag drawing routine that will be called by the Drag Manager to draw, move and hide the "dotted outline" drag feedback on the screen during a drag.
`dragDrawingRefCon`	A reference constant that will be forwarded to your drag drawing procedure when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your drag drawing procedure.

DESCRIPTION

The `SetDragDrawingProc` function sets the drag drawing procedure for the given drag reference. A drag's drag drawing procedure is called by the Drag Manager when the Drag Manager needs to draw, move or hide the "dotted outline" drag feedback on the screen.

Your drag drawing procedure can implement any type of drag feedback, such as dragging a bitmap of the object being dragged.

Details for how to write a drag drawing procedure are covered in the "Application-Defined Routines" section below.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference

Performing a Drag

You can use the `TrackDrag` function to start a drag from within your application.

TrackDrag

Use the `TrackDrag` function to drag an item or collection of items from your application.

```
pascal OSErr TrackDrag (DragReference theDragRef,
                        const EventRecord *theEvent,
                        RgnHandle theRegion);
```

`theDragRef`	A drag reference token to perform the drag operation with.
`theEvent`	The `mouseDown` event record that your application received that resulted in starting a drag.
`theRegion`	A region that represents the item or items being dragged. Note that under normal circumstances, the drag region should only include the pixels that represent the outline of the items being dragged. The Drag Manager draws the region on the screen by using calling `PaintRgn` (not `FrameRgn`).

DESCRIPTION

The `TrackDrag` function performs a drag operation with a particular drag reference given the `mouseDown` event and a drag region.

The Drag Manager follows the cursor on the screen with the "dotted outline" drag feedback and sends tracking messages to applications that have registered drag tracking handlers. The drag item flavor information that was added to the drag using the `AddDragItemFlavor` functions is available to each application that becomes active during a drag.

When the user releases the mouse button, the Drag Manager calls any receive drop handlers that have been registered on the destination window. An application's receive drop handler(s) are responsible for accepting the drag and transferring the dragged data into their application.

The `TrackDrag` function returns `noErr` in situations where the user selected a destination for the drag and the destination received data from the Drag Manager. If the user drops over a non-aware application or the receiver does not accept any data from the Drag Manager, the Drag Manager automatically provides a "zoom back" animation and returns `userCanceledErr`.

SPECIAL CONSIDERATIONS

During the call to `TrackDrag`, your application's context is temporarily switched out when the Drag Manager calls a different application's tracking and receive handlers. Do not depend on your application's context to be active for the entire duration of a drag.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`userCanceledErr`	-128	Drag was canceled
`badDragRefErr`	-1850	Unknown drag reference

Getting Drag Item Information

The Drag Manager provides a set of functions that allow you to get information about the drag items and drag item flavors that have been added into a drag reference.

CountDragItems

Use the `CountDragItems` function to determine how many drag items are contained in a drag reference.

```
pascal OSErr CountDragItems (DragReference theDragRef,
                             unsigned short *numItems);
```

`theDragRef`	A drag reference.
`numItems`	The `CountDragItems` function returns the number of drag items in the given drag reference in the `numItems` parameter.

DESCRIPTION

The `CountDragItems` function returns the number of drag items in a drag reference in the `numItems` parameter.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference

GetDragItemReferenceNumber

Use the `GetDragItemReferenceNumber` function to determine the item reference number of a specific item in a drag reference.

```
pascal OSErr GetDragItemReferenceNumber
                                       (DragReference theDragRef,
                                        unsigned short index,
                                        ItemReference *theItemRef);
```

`theDragRef`	A drag reference.
`index`	The index of an item in a drag to get the item reference number for.
`theItemRef`	The `GetDragItemReferenceNumber` function returns the item reference

number of the item with the specified index in the `theItemRef` parameter.

DESCRIPTION

The `GetDragItemReferenceNumber` function returns the item reference number of the item with the specified index in the `theItemRef` parameter.

If `index` is zero or larger than the number of items in the drag, `badDragItemErr` is returned by `GetDragItemReferenceNumber`.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference

CountDragItemFlavors

Use the `CountDragItemFlavors` function to determine how many drag item flavors are contained within a drag item.

```
pascal OSErr CountDragItemFlavors (DragReference theDragRef,
                                    ItemReference theItemRef,
                                    unsigned short *numFlavors);
```

`theDragRef`	A drag reference.
`theItemRef`	An item reference number.
`numFlavors`	The `CountDragItemFlavors` function returns the number of drag item flavors in the specified drag item in the `numFlavors` parameter.

DESCRIPTION

The `CountDragItemFlavors` function returns the number of drag item flavors in the specified drag item in the `numFlavors` parameter.

When `CountDragItemFlavors` is called by an application other than the sender, the flavors that are marked with the `flavorSenderOnly` flavor flag are not included in the count.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference

GetFlavorType

Use the `GetFlavorType` function to determine the type of a specific flavor in a drag item.

```
pascal OSErr GetFlavorType (DragReference theDragRef,
                            ItemReference theItemRef,
                            unsigned short index,
                            FlavorType *theType);
```

`theDragRef`	A drag reference.
`theItemRef`	An item reference number.
`index`	The index of a flavor in the specified item to get the flavor type of.
`theType`	The `GetFlavorType` function returns the type of the specified flavor in the `theType` parameter.

DESCRIPTION

The `GetFlavorType` function returns the type of the specified flavor in the `theType` parameter.

If `index` is zero or larger than the number of flavors in the item, `badDragFlavorErr` is returned by `GetFlavorType`.

If a flavor is marked with the `flavorSenderOnly` flavor flag, it is only visible to the sender application. If `GetFlavorType` is called by any application other than the sender, flavors that are visible only to the sender will not be returned.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference
`badDragFlavorErr`	-1852	Bad flavor index

GetFlavorFlags

Use the `GetFlavorFlags` function to get the flags for a specific flavor in a drag item.

```
pascal OSErr GetFlavorFlags (DragReference theDragRef,
                             ItemReference theItemRef,
                             FlavorType theType,
                             FlavorFlags *theFlags);
```

`theDragRef`	A drag reference.
`theItemRef`	An item reference number.
`theType`	The flavor type of the flavor to get the attributes of.
`theFlags`	The `GetFlavorFlags` function returns the attributes of the specified flavor in the `theFlags` parameter.

DESCRIPTION

The `GetFlavorFlags` function returns the flags of the specified flavor in the `theFlags` parameter.

If a flavor is marked with the `flavorSenderOnly` flavor flag, it is only visible to the sender application. If `GetFlavorFlags` is called by any application other than the sender, the flags for flavors that are visible only to the sender will not be returned.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference
`badDragFlavorErr`	-1852	Unknown flavor type

GetFlavorDataSize

Use the `GetFlavorDataSize` function to get the size of the flavor data for a specific flavor in a drag item.

```
pascal OSErr GetFlavorDataSize (DragReference theDragRef,
                                ItemReference theItemRef,
                                FlavorType theType,
                                Size *dataSize);
```

`theDragRef`	A drag reference.

`theItemRef`	An item reference number.
`theType`	The flavor type of the flavor to get the data size of.
`dataSize`	The `GetFlavorDataSize` returns the size of the specified drag item flavor data in the `dataSize` parameter.

DESCRIPTION

Before calling `GetFlavorData` (defined below), you may want to first determine the size of the data contained within a flavor. The `GetFlavorDataSize` function returns the specified flavor's data size in the `dataSize` parameter.

Note that calling `GetFlavorDataSize` on a flavor that requires translation will force that translation be performed in order to determine the data size. Since translation may require a significant amount of time and memory during processing, call `GetFlavorDataSize` only when absolutely necessary.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference
`badDragFlavorErr`	-1852	Unknown flavor type
`cantGetFlavorErr`	-1854	Error while trying to get flavor data

GetFlavorData

Use the `GetFlavorData` function to get all or part of the flavor data for a specific flavor in a drag item.

```
pascal OSErr GetFlavorData (DragReference theDragRef,
                            ItemReference theItemRef,
                            FlavorType theType,
                            void *dataPtr,
                            Size *dataSize,
                            unsigned long *dataOffset);
```

`theDragRef`	A drag reference.
`theItemRef`	An item reference number.
`theType`	The flavor type of the flavor to get the flavor data from.
`dataPtr`	Specifies where the `GetFlavorData` function should copy the requested flavor data. Your application is responsible for allocating the memory

for the flavor data and for setting the `dataSize` parameter to the number of bytes that you have allocated for the data.

`dataSize` Contains the size of the data (in bytes) that you have allocated memory for and wish to receive from the flavor. When `GetFlavorData` returns, `dataSize` will contain the actual number of bytes copied into the buffer specified by `dataPtr`.

If you specify a `dataSize` that is smaller than the amount of data in the flavor, the data is copied into your buffer and `dataSize` is unchanged when `GetFlavorData` returns.

If you specify a `dataSize` that is larger than the amount of data in the flavor, only the amount of data in the flavor is copied into your buffer and the `dataSize` parameter will contain the actual number of bytes copied when `GetFlavorData` returns.

`dataOffset` The offset (in bytes) into the flavor record to begin copying data from into the supplied buffer pointed to by `dataPtr`.

DESCRIPTION

The `GetFlavorData` function returns all or part of a flavor's data in a data buffer supplied by the `dataPtr` parameter.

You can first determine the size of a flavor by calling the `GetFlavorDataSize` function.

If you wish to receive the flavor data in smaller pieces than the entire size of the data, you can set the `dataSize` to be as large as your buffer and call `GetFlavorData` multiple times while incrementing the `dataOffset` by the size of your buffer.

You can determine when you have reached the end of the flavor's data when the `dataSize` parameter returns a number of bytes lower than you provided.

If the `dataOffset` was larger than the amount of data contained within the flavor, `0` (zero) will be returned in the `dataSize` parameter denoting that no data was copied into your buffer.

Note that calling `GetFlavorData` on a flavor that requires translation will force that translation to occur in order to return the data.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference

`badDragFlavorErr`	-1852	Unknown flavor type
`cantGetFlavorErr`	-1854	Error while trying to get flavor data

GetDragItemBounds

Use the `GetDragItemBounds` function to determine the bounding rectangle of a drag item.

```
pascal OSErr GetDragItemBounds (DragReference theDragRef,
                                ItemReference theItemRef,
                                Rect *itemBounds);
```

`theDragRef`	A drag reference.
`theItemRef`	An item reference number.
`itemBounds`	The `GetDragItemBounds` function returns the bounding rectangle of the specified item in global coordinates in the `itemBounds` parameter.

DESCRIPTION

The `GetDragItemBounds` returns the bounding rectangle of the specified item. The rectangle is provided in global coordinates.

`GetDragItemBounds` always returns the rectangle relative to the current pinned mouse position. You can use the `GetDragItemBounds` function in your tracking or receive handlers to determine the current or dropped location of each item in the drag.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference

SetDragItemBounds

Use the `SetDragItemBounds` function to set the bounding rectangle of a drag item.

```
pascal OSErr SetDragItemBounds (DragReference theDragRef,
                                ItemReference theItemRef,
                                const Rect *itemBounds);
```

`theDragRef`	A drag reference.
`theItemRef`	An item reference number.

`itemBounds`	The bounding rectangle to set for the given drag item. This rectangle is specified in global coordinates relative to the mouse down position.

DESCRIPTION

The `SetDragItemBounds` function sets the bounding rectangle for a given drag item. The rectangle is specified in global coordinates relative to the mouse down position that is given to the `TrackDrag` function. Your application would normally want to call `SetDragItemBounds` on each drag item before starting a drag with `TrackDrag`.

If you do not set the bounds of an item, the rectangle returned by `GetDragItemBounds` is an empty rectangle centered under the pinned mouse location.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference

Getting Drag Status Information

The Drag Manager provides a set of functions that allow you to get information about a drag that is currently in progress.

GetDragAttributes

Use the `GetDragAttributes` function to get the current set of drag attribute flags.

```
pascal OSErr GetDragAttributes
                              (DragReference theDragRef,
                               DragAttributes *attributes);
```

`theDragRef`	A drag reference.
`attributes`	The `GetDragAttributes` function returns the drag attribute flags for the given drag reference in the `attributes` parameter.

DESCRIPTION

The `GetDragAttributes` function returns the drag attribute flags for the given drag reference in the `attributes` parameter.

If `GetDragAttributes` is called during a drag, the current set of `DragAttributes` is returned. If `GetDragAttributes` is called after a drag, the set of `DragAttributes` that were set at drop time is returned.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`badDragRefErr`	-1850	Unknown drag reference

GetDragMouse

Use the `GetDragMouse` function to get the current mouse and pinned mouse locations.

```
pascal OSErr GetDragMouse (DragReference theDragRef,
                           Point *mouse,
                           Point *pinnedMouse);
```

`theDragRef`	A drag reference.
`mouse`	The `GetDragMouse` function returns the current mouse location in the `mouse` parameter. The mouse location is given in global screen coordinates.
`pinnedMouse`	The `GetDragMouse` function returns the current pinned mouse location in the `pinnedMouse` parameter. The pinned mouse location is the mouse location that is used to draw the drag region on the screen. The `pinnedMouse` location is different than the `mouse` location when the cursor is being constrained in either dimension by a tracking handler. The pinned mouse location is given in global screen coordinates.

DESCRIPTION

The `GetDragMouse` function returns the mouse location in the `mouse` parameter and the pinned mouse location in the `pinnedMouse` parameter. All coordinates are given in global screen coordinates.

The pinned mouse location is the mouse location used to draw the drag region on the screen. Tracking handlers may constrain the mouse by setting the pinned mouse location to be different than the current mouse location by using the `SetDragMouse` function.

You may pass `NIL` into the `mouse` or `pinnedMouse` parameters if you wish to disregard either of these return values.

Calling `GetDragMouse` before using the drag in a `TrackDrag` call returns (0, 0) as both the `mouse` and `pinnedMouse` locations.

If `GetDragMouse` is called during a drag, the current `mouse` and `pinnedMouse` are returned. If `GetDragMouse` is called after a drag completes, the `mouse` and `pinnedMouse` at the drop location are returned.

RESULT CODES

`noErr`	0	No error
`badDragRefErr`	-1850	Unknown drag reference

SetDragMouse

Use the `SetDragMouse` function to set the current pinned mouse location.

```
pascal OSErr SetDragMouse (DragReference theDragRef,
                           Point pinnedMouse);
```

`theDragRef`	A drag reference.
`pinnedMouse`	The coordinates to set the pinned mouse location. The pinned mouse location is specified in global screen coordinates.

DESCRIPTION

The `SetDragMouse` function sets the current pinned mouse location. The pinned mouse location is the location used to draw the drag region on the screen. You can use the `SetDragMouse` function to "constrain" the mouse while dragging through one of your application's windows.

To constrain the mouse within one of your application's windows, call `SetDragMouse` from within your tracking handler when you receive `dragTrackingInWindow` messages. The Drag Manager updates the position of the drag region on the screen after each time your tracking handlers are called.

RESULT CODES

`noErr`	0	No error
`badDragRefErr`	-1850	Unknown drag reference

GetDragOrigin

Use the `GetDragOrigin` function to get the `mouseDown` location that started the given drag.

```
pascal OSErr GetDragOrigin (DragReference theDragRef,
                            Point *initialMouse);
```

`theDragRef` A drag reference.

`initialMouse`
The `GetDragOrigin` function returns the `mouseDown` location that started the given drag in the `initialMouse` parameter. The initial mouse location is given in global screen coordinates.

DESCRIPTION

The `GetDragOrigin` function returns the `mouseDown` location that started the given drag. The initial mouse location is returned in global screen coordinates.

`GetDragOrigin` may be called to return the initial mouse location both during and after a drag.

RESULT CODES

`noErr`	0	No error
`badDragRefErr`	-1850	Unknown drag reference

GetDragModifiers

Use the `GetDragModifiers` function to get the current set of keyboard modifiers.

```
pascal OSErr GetDragModifiers (DragReference theDragRef,
                               short *modifiers,
                               short *mouseDownModifiers,
                               short *mouseUpModifiers);
```

`theDragRef` A drag reference.

`modifiers` The `GetDragModifiers` function returns the current keyboard modifiers in the `modifiers` parameter.

`mouseDownModifiers`
The `GetDragModifiers` function returns the keyboard modifiers at mouseDown time in the `mouseDownModifiers` parameter.

`mouseUpModifiers`
The `GetDragModifiers` function returns the keyboard modifiers at mouseUp time in the `mouseUpModifiers` parameter.

DESCRIPTION

The `GetDragModifiers` function returns the set of modifier keys that are currently pressed and that were pressed at `mouseDown` time and at `mouseUp` time.

You may pass `NIL` into the `modifiers`, `mouseDownModifiers` or `mouseUpModifiers` parameters if you wish to disregard any of these return values.

Calling `GetDragModifiers` before using the drag in a `TrackDrag` call returns zero in all of the `modifier` parameters. Calling `GetDragModifiers` during a drag, but while the drag is still tracking returns zero in the `mouseUpModifiers` parameter. Calling `GetDragModifiers` in a receive handler or after the drag has completed returns all of the `modifier` parameters.

RESULT CODES

`noErr`	0	No error
`badDragRefErr`	-1850	Unknown drag reference

GetDropLocation

Use the `GetDropLocation` function to get an AppleEvent descriptor of the drop location.

```
pascal OSErr GetDropLocation
                              (DragReference theDragRef,
                               AEDesc *dropLocation);
```

`theDragRef` A drag reference.

`dropLocation`
The `GetDropLocation` function returns an AppleEvent descriptor of the drop location in the `dropLocation` parameter. The drop location is only valid after the receiver has set the drop location by calling `SetDropLocation`.

DESCRIPTION

The `GetDropLocation` function returns an AppleEvent descriptor describing the drop location in the `dropLocation` parameter.

If the destination is in the Finder, the Finder sets the drop location to be an alias to the location in the file system that received the drag. Refer to the Finder Interface chapter of Inside Macintosh for more information about aliases to desktop objects.

If the receiver of the drag has not set a drop location by calling the `SetDropLocation` function, `typeNull` will be returned in the descriptor.

`GetDropLocation` may be called both during a drag as well as after a drag has completed.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory to duplicate descriptor
`badDragRefErr`	-1850	Unknown drag reference

SetDropLocation

Use the `SetDropLocation` function to set the AppleEvent descriptor for the drop location for a drag.

```
pascal OSErr SetDropLocation
                              (DragReference theDragRef,
                               const AEDesc *dropLocation);
```

`theDragRef` A drag reference.

`dropLocation`
The AppleEvent descriptor of the drop location to set.

DESCRIPTION

The `SetDropLocation` function is used to set the AppleEvent descriptor of the drop location of a drag. Typically, this function is called by a receive handler before attempting to get any flavor data by using the `GetFlavorDataSize` or `GetFlavorData` functions. When a sender application's drag send data procedure is called to provide flavor data to a receiver, `GetDropLocation` can then be called to determine the drop location while providing data to the sender.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory to duplicate descriptor
`badDragRefErr`	-1850	Unknown drag reference

Window Highlighting Utilities

You can use the `ShowDragHilite`, `HideDragHilite`, and `UpdateDragHilite` functions to highlight parts of your application's windows during a drag. You can also use the

DragPreScroll and DragPostScroll functions if you intend to scroll parts of your window that contain drag highlighting.

ShowDragHilite

Use the ShowDragHilite function to highlight an area of your window during a drag. Your tracking handler routine should call this if a drop is allowed at the current mouse position.

```
pascal OSErr ShowDragHilite (DragReference theDragRef,
                             RgnHandle hiliteFrame,
                             Boolean inside);
```

theDragRef	The drag reference of the drag currently in progress.
frame	A QuickDraw region of the frame of the window, pane, or shape you wish to highlight. This region should be in the window's local coordinate system.
inside	If true, the highlighting will be drawn inside the frame shape. Otherwise it will be drawn outside the frame shape. Note that in either case, the highlight will not include the boundary edge of the frame.

DESCRIPTION

The ShowDragHilite procedure creates a standard drag and drop highlight in your window. You can only have one highlight showing at a time, and if you call this routine when a highlight is currently visible, the first one is removed before the newly requested highlight is shown.

The highlight that is drawn is defined by the hiliteFrame and inside parameters. The hiliteFrame defines the shape of the highlighting to draw, the inside parameter determines whether the highlighting is drawn on the outside or inside of the hiliteFrame region. This allows you to easily highlight inside a window frame or a pane, or to highlight outside of a container or object in your window. ShowDragHilite uses a two pixel thick line when drawing the highlight.

ShowDragHilite assumes that the highlighting should be drawn in the current port. Your application should make sure that the correct port is set before calling ShowDragHilite. Also, highlighting drawn by ShowDragHilite is clipped to the current port. Make sure that the port's clip region is appropriately sized to draw the highlighting.

The Drag Manager maintains the currently highlighted portion of your window if you use the HideDragHilite and UpdateDragHilite functions. If you intend to scroll the window that contains the highlighting, you can use the DragPreScroll and DragPostScroll functions to properly update the drag highlighting.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`badDragRefErr`	-1850	Unknown drag reference

HideDragHilite

This routine removes highlighting created with the `ShowDragHilite` function.

```
pascal OSErr HideDragHilite (DragReference theDragRef);
```

`theDragRef` The drag reference that is currently showing a drag highlight.

DESCRIPTION

Use the `HideDragHilite` function to remove any highlighting from your window that was shown using the `ShowDragHilite` function.

`HideDragHilite` assumes that the highlighting should be erased from the current port. Your application should make sure that the correct port is set before calling `HideDragHilite`. Also, highlighting erased by `HideDragHilite` is clipped to the current port. Make sure that the port's clip region is appropriately sized to remove the highlighting.

RESULT CODES

`noErr`	0	No error
`badDragRefErr`	-1850	Unknown drag reference

DragPreScroll

When scrolling part of your window when drag highlighting is showing, use the `DragPreScroll` function to remove any drag highlighting that would be scrolled away from the `hiliteFrame` given to `ShowDragHilite`.

```
pascal OSErr DragPreScroll (DragReference theDragRef,
                            short dH,
                            short dV);
```

`theDragRef` The drag reference.

`dH`	The horizontal distance you intend to scroll.
`dV`	The vertical distance you intend to scroll.

DESCRIPTION

The `DragPreScroll` function prepares your window or pane for scrolling. Use this function if you plan to scroll part of your window using `ScrollRect` or `CopyBits`.

Scrolling part of your window may inadvertently move part of the drag highlighting with it. `DragPreScroll` is optimized to remove from the screen only the parts of the highlighting that will be scrolled away from the `hiliteFrame` region. After calling `DragPreScroll` with the `dH` and `dV` that you are going to scroll, you can then scroll your window followed by a call to `DragPostScroll` which redraws any necessary highlighting after the scroll.

If you use an offscreen port to draw your window into while scrolling, it is recommended that you simply use the `HideDragHilite` and `ShowDragHilite` functions to preserve drag highlighting in your offscreen port. The `DragScroll` functions are optimized for onscreen scrolling.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`badDragRefErr`	-1850	Unknown drag reference

DragPostScroll

Use the `DragPostScroll` function to restore the drag highlight after scrolling part of your window using the `DragPreScroll` function.

```
pascal OSErr DragPostScroll (DragReference theDragRef);
```

`theDragRef`	The drag reference.

DESCRIPTION

The `DragPostScroll` function restores the drag highlight after you scroll part of your window. This routine must be called following a call to `DragPreScroll`.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`badDragRefErr`	-1850	Unknown drag reference

UpdateDragHilite

Use the `UpdateDragHilite` function to update a portion of the drag highlight that was drawn over by your application.

```
pascal OSErr UpdateDragHilite (DragReference theDragRef,
                               RgnHandle updateRgn);
```

`theDragRef`	The drag reference.
`updateRgn`	A region that needs to be updated. Typically the port's `updateRgn`.

DESCRIPTION

The `UpdateDragHilite` function redraws the portion of the drag highlight which intersects the given `updateRgn`. Use this function if your application draws into the highlighted portion of your window during a drag. For example, dragging over a folder icon in the Finder causes the Finder to redraw the folder icon in its darkened (selected) color. The Finder calls `UpdateDragHilite` to redraw any portion of the drag highlight that may have intersected with the folder icon.

You must guarantee, however, that any current highlighting within the `updateRgn` has been completely erased or is clipped out. If this routine is asked to highlight over an area which is still highlighted, it will be redrawn incorrectly.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`badDragRefErr`	-1850	Unknown drag reference

Drag Manager Utilities

You can use the `WaitMouseMoved` function to determine after a mouseDown event if a drag should be started, the `ZoomRects` and `ZoomRegion` functions to provide "zooming" animation similar to the Finder's in your application, and `TEGetHiliteRgn` to get the QuickDraw highlight region from the current selection in a TextEdit record.

WaitMouseMoved

When your application receives a mouseDown event on a draggable object, call `WaitMouseMoved` to determine if you should begin to drag the object.

```
pascal Boolean WaitMouseMoved (Point initialMouse);
```

`initialMouse`
: The point where a mouseDown event occurred. The `initialMouse` location is given in global screen coordinates.

DESCRIPTION

The `WaitMouseMoved` function waits for either the mouse to move from the given `initialMouse` location or for the mouse button to be released. If the mouse moves away from the `initialMouse` location before the mouse button is released, `WaitMouseMoved` returns `true`. If the mouse button is released before the mouse moved from the `initialMouse` location, `WaitMouseMoved` returns `false`.

ZoomRects

Use the `ZoomRects` function to animate a rectangle into a second rectangle. This routine provides the same visual effect that the Finder uses to open windows and applications.

```
pascal OSErr ZoomRects (const Rect *fromRect,
                        const Rect *toRect,
                        short zoomSteps,
                        ZoomAcceleration acceleration);
```

`fromRect`	Specifies the starting rectangle to animate from, in global coordinates.
`toRect`	Specifies the ending rectangle to animate to, in global coordinates.
`zoomSteps`	Specifies the number of animation steps that are shown between the source and destination rectangles. The minimum number of `zoomSteps` is `4`. If less than `4` are specified, `4` will be used. The maximum number of `zoomSteps` is `25`. If more than `25` are specified, `25` will be used.
`acceleration`	Specifies how the intermediate animation steps will be calculated. Can accept the constants `zoomNoAcceleration`, `zoomAccelerate`, or `zoomDecelerate`. Using `zoomNoAcceleration` makes the distance between steps from the source to the destination equal. Using `zoomAccelerate` makes each step from the source to the destination increasingly larger, making the animation appear to speed up as it approaches the destination. Using `zoomDecelerate` makes each step from the source to the destination smaller, making the animation appear to slow down as it approaches the destination.

DESCRIPTION

The `ZoomRects` function animates a movement between two rectangles on the screen. It does this by drawing gray dithered rectangles incrementally toward the destination rectangle.

`ZoomRects` draws on the entire screen, outside of the current port. It does not change any pixels on the screen after it has completed its animation. It also preserves the current port and the port's settings.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error

ZoomRegion

Use the `ZoomRegion` function to animate a region's outline from one screen location to another. This routine provides the same visual feedback that the Finder uses to "zoom" icons when performing a Clean Up operation.

```
pascal OSErr ZoomRegion (RgnHandle region,
                         Point zoomDistance,
                         short zoomSteps,
                         ZoomAcceleration acceleration);
```

`region` A region to animate.

`zoomDistance`
The horizontal and vertical distance from the starting point that the region will animate to.

`zoomSteps` Specifies the number of animation steps that are shown between the source and destination regions. The minimum number of `zoomSteps` is `4`. If less than `4` are specified, `4` will be used. The maximum number of `zoomSteps` is `25`. If more than `25` are specified, `25` will be used.

`acceleration`
Specifies how the intermediate animation steps will be calculated. Can accept the constants `zoomNoAcceleration`, `zoomAccelerate`, or `zoomDecelerate`. Using `zoomNoAcceleration` makes the distance between steps from the source to the destination equal. Using `zoomAccelerate` makes each step from the source to the destination increasingly larger, making the animation appear to speed up as it approaches the destination. Using `zoomDecelerate` makes each step from the source to the destination smaller, making the animation appear to slow down as it approaches the destination.

DESCRIPTION

The `ZoomRegion` function animates a region from one location to another on the screen. It does this by drawing gray dithered regions incrementally toward the destination region.

`ZoomRegion` draws on the entire screen, outside of the current port. It does not change any pixels on the screen after it has completed its animation. It also preserves the current port and the port’s settings.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Parameter error

TextEdit Utilities

The `TEGetHiliteRgn` can be used to get the QuickDraw highlight region from the current selection in a TextEdit record. This TextEdit utility is useful for determining what areas of a TextEdit field can be dragged by the user.

TEGetHiliteRgn

Use the `TEGetHiliteRgn` function to get the QuickDraw highlight region from the current selection in a TextEdit record.

```
pascal OSErr TEGetHiliteRgn (RgnHandle region,
                             TEHandle hTE);
```

`region`	The `TEGetHiliteRgn` function computes the QuickDraw region of the current selection in the given TextEdit handle. This region is placed into the `region` parameter that you have already allocated. This region is in your window’s local screen coordinates.
`hTE`	A TextEdit handle.

DESCRIPTION

The `TEGetHiliteRgn` function returns in the `region` parameter the region of the current selection in the given TextEdit handle.

`TEGetHiliteRgn` does not allocate a new region. You must create a new region with `NewRgn` before calling `TEGetHiliteRgn`. Also, the previous contents of the region are replaced by the TextEdit selection region.

If the given TextEdit handle does not currently have a selection, `TEGetHiliteRgn` returns an empty region.

RESULT CODES

`noErr`	0	No error
`memFullErr`	-108	Not enough memory

Application-Defined Routines

This section describes the application-defined routines whose addresses you pass to the Drag Manager. You can define routines that the Drag Manager calls during a drag to implement the different aspects of dragging both into and out of your application's windows.

Drag Handler Routines

Most of the application's dragging functionality is implemented through the use of drag handlers. The Drag Manager calls your application's drag handlers while the user drags a collection of items through one of your application's windows, and when the user drops the items into one of your application's windows.

DragTrackingHandler

A drag tracking handler has the following syntax:

```
pascal OSErr DragTrackingHandler (DragTrackingMessage message,
                                  WindowPtr theWindow,
                                  void *handlerRefCon,
                                  DragReference theDragRef);
```

`message`	A tracking message from the Drag Manager.
`theWindow`	A pointer to the window that the mouse is currently over.
`handlerRefCon`	A reference constant that was provided to `InstallTrackingHandler` when this handler was installed.
`theDragRef`	The drag reference of the drag.

DESCRIPTION

When the user drags an item or collection of items through a window, the Drag Manager calls any `DragTrackingHandler` functions that have been installed on that window. Your `DragTrackingHandler` can determine the contents of the drag by calling the drag item information functions, such as `CountDragItems`, `CountDragItemFlavors`, `GetFlavorType` and `GetFlavorFlags` and highlight or modify the objects in your window accordingly.

You use the `message` parameter to determine what action your `DragTrackingHandler` should take. The `message` parameter may be one of the following values:

Message descriptions

`dragTrackingEnterHandler`
: You will receive a call with this message when the focus of a drag enters a window that is handled by your `DragTrackingHandler`. If the user moves the drag directly to another window that is handled by the same `DragTrackingHandler`, a second `dragTrackingEnterHandler` message is not received. Your `DragTrackingHandler` only receives this message when the drag enters the domain of your procedure after leaving another.

`dragTrackingEnterWindow`
: You will receive a call with this message when a drag enters any window that is handled by your `DragTrackingHandler`. This message is sent to your `DragTrackingHandler` for each window that the drag may enter. You will always receive this message within a pair of `dragTrackingEnterHandler` and `dragTrackingLeaveHandler` calls.

`dragTrackingInWindow`
: You will receive calls with this message as the user is dragging within a window handled by your `DragTrackingHandler`. You can use this message to track the dragging process through your window. You will always receive this message within a pair of `dragTrackingEnterWindow` and `dragTrackingLeaveWindow` calls.

: You would typically draw the majority of your window highlighting and track objects in your window when you receive this message from the Drag Manager.

`dragTrackingLeaveWindow`
: You will receive a call with this message when a drag leaves any window that is handled by your `DragTrackingHandler`. You are guaranteed to receive this message after receiving a corresponding `dragTrackingEnterWindow`. You will always receive this message within a pair of `dragTrackingEnterHandler` and `dragTrackingLeaveHandler` calls.

`dragTrackingLeaveHandler`
: You will receive a call with this message when the focus of a drag enters a window that is not handled by your `DragTrackingHandler`. You are

guaranteed to receive this message after receiving a corresponding `dragTrackingEnterHandler`.

When the Drag Manager calls your `DragTrackingHandler`, the port will always be set to the window that the mouse is over.

SPECIAL CONSIDERATIONS

The Drag Manager guarantees that your application's A5 world and low-memory environment is properly set up for your application's use. Therefore, you can allocate memory, and use your application's global variables. You can also rely on low-memory globals being valid.

You can call `WaitNextEvent` or any other Event Manager routines from within your `DragTrackingHandler`. This includes calling any routines that may call the Event Manager, such as `ModalDialog` or `Alert`. Note that the Process Manager's process switching mechanism is disabled during calls to your handler. If your application is not frontmost when calling these routines, your application will not be able to switch forward. This may result in a situation where a modal dialog appears behind the front process but will not be able to come forward in order to interact with the user.

DragReceiveHandler

A drag receive handler has the following syntax:

```
pascal OSErr DragReceiveHandler (WindowPtr theWindow,
                                  void *handlerRefCon,
                                  DragReference theDragRef);
```

`theWindow`	A pointer to the window that the drop occurred in.
`handlerRefCon`	A reference constant that was provided to `InstallReceiveHandler` when this handler was installed.
`theDragRef`	The drag reference of the drag.

DESCRIPTION

When the user releases a drag in a window, the Drag Manager calls any `DragReceiveHandler` functions that have been installed on that window. You can get the information about the data that is being dragged to determine if your window will accept the drop by using the drag information functions provided by the Drag Manager.

After your `DragReceiveHandler` decides that it can accept the drop, use the `GetFlavorData` function to get the needed data from the sender of the drag.

When the Drag Manager calls your `DragReceiveHandler`, the port will be set to the window that the drop occurred in.

If you want to provide an optional AppleEvent descriptor of the drop location for the sender, use the `SetDropLocation` function to set the drop location descriptor before calling the sender with the `GetFlavorData` or `GetFlavorDataSize` functions.

If you return any result code other than `noErr` from your `DragReceiveHandler`, the Drag Manager will "zoomback" the drag region to the source location and return `userCanceledErr` from `TrackDrag`.

If the drag is dropped into a location that cannot accept the drag (such as the window title bar or window scroll bars) or no acceptable data types were available, your `DragReceiveHandler` should return `dragNotAcceptedErr`, which will cause the Drag Manager to provide the "zoomback" animation described above.

SPECIAL CONSIDERATIONS

The Drag Manager guarantees that your application's A5 world and low-memory environment is properly set up for your application's use. Therefore, you can allocate memory, and use your application's global variables. You can also rely on low-memory globals being valid.

You can call `WaitNextEvent` or any other Event Manager routines from within your `DragReceiveHandler`. This includes calling any routines that may call the Event Manager, such as `ModalDialog` or `Alert`. Note that the Process Manager's process switching mechanism is disabled during calls to your handler. If your application is not frontmost when calling these routines, your application will not be able to switch forward. This may result in a situation where a modal dialog appears behind the front process but will not be able to come forward in order to interact with the user.

Drag Callback Procedures

There are several Drag Manager functions that can be overridden by setting any of several drag callback procedures for any given drag. The available drag callback procedures override the standard flavor data sending, mouse and keyboard sampling, and drag region drawing functions.

DragSendDataProc

A drag send data procedure has the following syntax:

```
pascal OSErr DragSendDataProc (FlavorType theType,
                               void *dragSendRefCon,
                               ItemReference theItemRef,
                               DragReference theDragRef);
```

`theType`	A flavor type being requested by a drop receiver.
`dragSendRefCon`	A reference constant that was provided when `SetDragSendProc` was called to install this procedure.
`theItemRef`	The item reference of the item that the flavor data is being requested from.
`theDragRef`	The drag reference of the drag.

DESCRIPTION

The Drag Manager calls your `DragSendDataProc` when drag item flavor data is requested by a drop receiver if the drag item flavor data is not already cached by the Drag Manager. Use the `SetDragItemFlavorData` function to give the Drag Manager the requested data.

The Drag Manager caches all drag item flavor data that was given in the `data` pointer when the flavor was added using the `AddDragItemFlavor` function. If the `data` pointer is `NIL` when the flavor is added, the Drag Manager will call the `DragSendDataProc` to get the data only if a receiver requests the data using the `GetFlavorData` or `GetFlavorDataSize` functions.

A second scenario where the `DragSendDataProc` is called is when a drop receiver requests a flavor that is translated by the Translation Manager and the source data (which would be a different type than actually requested by the receiver) is not already cached by the Drag Manager.

You can use the `GetDropLocation` function to get the AppleEvent descriptor of the drop location from within your `DragSendDataProc`. It is optional for the receiver to provide a drop location descriptor. If the receiver does not provide the drop location descriptor, `typeNull` will be returned by the `GetDropLocation` function.

You do not need to provide a `DragSendDataProc` if you do not ever pass `NIL` as the `data` pointer when calling `AddDragItemFlavor`.

SPECIAL CONSIDERATIONS

The Drag Manager guarantees that your application's A5 world and low-memory environment is properly set up for your application's use. Therefore, you can allocate memory, and use your application's global variables. You can also rely on low-memory globals being valid.

You can call `WaitNextEvent` or any other Event Manager routines from within your `DragTrackingHandler`. This includes calling any routines that may call the Event Manager, such as `ModalDialog` or `Alert`. Note that the Process Manager's process switching mechanism is disabled during calls to your handler. If your application is not frontmost when calling these routines, your application will not be able to switch forward. This may result in a situation where a modal dialog appears behind the front process but will not be able to come forward in order to interact with the user.

DragInputProc

A drag input procedure has the following syntax:

```
pascal OSErr DragInputProc (Point *mouse,
                            short *modifiers,
                            void *dragInputRefCon,
                            DragReference theDragRef);
```

`mouse` On entry, the `mouse` parameter contains the physical location of the mouse. On exit, your `DragInputProc` returns the desired current mouse location in the `mouse` parameter. The `mouse` location is specified in global screen coordinates.

`modifiers` On entry, the `modifiers` parameter contains the physical state of the keyboard modifiers and mouse button. On exit, your `DragInputProc` returns the desired state of the keyboard modifiers and mouse button. The `modifiers` parameter is specified using the same format and constants as the Event Manager's `EventRecord.modifiers` field.

`dragInputRefCon`
A reference constant that was provided when `SetDragInputProc` was called to install this procedure.

`theDragRef` The drag reference of the drag.

DESCRIPTION

Each time the Drag Manager samples the mouse and keyboard, it calls the `DragInputProc` (if one has been set by calling `SetDragInputProc`) to provide a way to modify or completely change the mouse and keyboard input to the Drag Manager.

When the `DragInputProc` is called, the `mouse` and `modifiers` parameters contain the actual values from the physical input devices. Your `DragInputProc` may modify these values in any way. For example, your `DragInputProc` may simply inhibit the control key modifier bit from being set or it may completely replace the mouse coordinates with those generated some other way to drive the drag itself.

Note that the Drag Manager uses the `btnState` flag in the `modifiers` parameter to determine when the mouse button has been released to finish a drag.

SPECIAL CONSIDERATIONS

Your application's context is not available when your `DragInputProc` is called by the Drag Manager. If you need access to your application's global variables, you will need to setup and restore your application's A5 world yourself.

You cannot call `WaitNextEvent` or any other Event Manager routines in your `DragInputProc`. This restriction includes calling any routines that may call the Event Manager, such as `ModalDialog` or `Alert`.

DragDrawingProc

A drag drawing procedure has the following syntax:

```
pascal OSErr DragDrawingProc (DragRegionMessage message,
                              RgnHandle showRegion,
                              Point showOrigin,
                              RgnHandle hideRegion
                              Point hideOrigin,
                              void *dragDrawingRefCon,
                              DragReference theDragRef);
```

`message`	A drag region drawing message from the Drag Manager.
`showRegion`	A region containing the drag region as it should be drawn or is currently visible on the screen. The `showRegion` parameter has slightly different meanings depending on the `message` passed to your `DragDrawingProc`. The `showRegion` is always given in global screen coordinates.
`showOrigin`	The point corresponding to the original `mouseDown` location in the drag region within the given `showRegion`. The `showOrigin` is always given in global screen coordinates.
`hideRegion`	A region containing the drag region as it should be erased from the screen. The `hideRegion` parameter has slightly different meanings depending on the `message` passed to your `DragDrawingProc`. The `hideRegion` is always given in global screen coordinates.
`hideOrigin`	The point corresponding to the original `mouseDown` location in the drag region within the given `hideRegion`. The `hideOrigin` is always given in global screen coordinates.
`dragDrawingRefCon`	A reference constant that was provided when `SetDragDrawingProc` was called to install this procedure.

`theDragRef` The drag reference of the drag.

DESCRIPTION

If your application set a custom drawing procedure for a drag using the `SetDragDrawingProc` function, the Drag Manager calls your `DragDrawingProc` to perform all drag region drawing operations.

The Drag Manager tracks the drag region as it appears on the screen and as it should follow the mouse. All drag region operations are performed on the region given to the `TrackDrag` function. Drag region drawing is managed by sending the `DragDrawingProc` messages to show and hide pieces of the drag region.

The Drag Manager has its own drag region port that is used to do all drag region drawing during a drag. This port is set to the current port before calling your `DragDrawingProc`. The drag region port is for your `DragDrawingProc`'s exclusive use during a drag. You may modify its fields and depend on its contents between calls to your `DragDrawingProc`.

You use the `message` parameter to determine what action your `DragDrawingProc` should take. The `message` parameter may be one of the following values:

Message descriptions

`dragRegionBegin`

You will receive a call with this message when a drag is being started and it is time to initialize your drawing procedure. You should not draw anything to the screen when you receive this message.

The `showRegion` contains the drag region that was passed to the `TrackDrag` function and the `showOrigin` contains the `mouseDown` location that was given to the `TrackDrag` function. This location is the origin of the drag region.

The `hideRegion` is `NIL` when your `DragDrawingProc` receives this message.

`dragRegionDraw`

You will receive a call with this message when you should move your drag region from the area of the screen defined by the `hideRegion` to the area of the screen defined by the `showRegion`.

The `showRegion` contains the drag region that was passed to the `TrackDrag` function, offset to the current pinned mouse location. This region represents the area of the screen that must be drawn into.

The `hideRegion` contains the drag region as it is currently visible on the screen from the last call with a `dragRegionDraw` message. This region represents the area of the screen that must be restored. Any part of the drag region that was previously obscured by a call with the `dragRegionHide` message is not included in this `hideRegion`.

`dragRegionHide`

You will receive calls with this message when you should remove part of the drag region from the screen. You will receive this message when the drag has ended or when part of the region must be obscured for drawing operations to occur underneath the drag region.

The `showRegion` is `NIL` when your `DragDrawingProc` receives this message.

The `hideRegion` contains the part of the currently visible drag region that must be removed from the screen.

`dragRegionIdle`

You will receive calls with this message when the drag region has not moved on the screen and no drawing is necessary. You can use this message if animation of the drag region is necessary.

The `showRegion` contains the drag region as it is currently visible on the screen.

The `hideRegion` is `NIL` when your `DragDrawingProc` receives this message.

`dragRegionEnd`

You will receive a call with this message when the drag has completed and it is time to deallocate any allocations made from within the `DragDrawingProc`. Your `DragDrawingProc` will have already received a `dragRegionHide` message to hide the entire drag region before receiving this message. After you receive this message, your `DragDrawingProc` will not be called again for the duration of the drag.

Both the `showRegion` and the `hideRegion` are `NIL` when your `DragDrawingProc` receives this message.

SPECIAL CONSIDERATIONS

Your application's context is not available when your `DragDrawingProc` is called by the Drag Manager. If you need access to your application's global variables, you will need to setup and restore your application's A5 world yourself.

You cannot call `WaitNextEvent` or any other Event Manager routines in your `DragDrawingProc` . This restriction includes calling any routines that may call the Event Manager, such as `ModalDialog` or `Alert`.

Summary of the Drag Manager

Pascal Summary

Constants

```
CONST
   { Gestalt Constants }
   gestaltDragMgrAttr          = 'drag';    { Drag Manager attributes }
   gestaltDragMgrPresent       = 0;         { Drag Manager is present }

   gestaltTEAttr               = 'teat';    { TextEdit attributes }
   gestaltTEHasGetHiliteRgn    = 0;         { TextEdit has TEGetHiliteRgn }

   { Flavor Flags }
   flavorSenderOnly            = $00000001; { flavor available to sender only }
   flavorSenderTranslated      = $00000002; { flavor translated by sender }
   flavorNotSaved              = $00000004; { flavor should not be saved }
   flavorSystemTranslated      = $00000100; { flavor translated by system }

   { Drag Attributes }
   dragHasLeftSenderWindow      = $00000001; { drag has left source window }
   dragInsideSenderApplication = $00000002; { drag is in the source app }
   dragInsideSenderWindow       = $00000004; { drag is in the source window }

   { Special Flavor Type }
   flavorTypeHFS               = 'hfs ';    { flavor type for HFS data }
   flavorTypePromiseHFS        = 'phfs';    { flavor type for promised HFS }
   flavorTypeDirectory         = 'diry';    { flavor type for AOCE directory }

   { Drag Tracking Handler Messages }
   dragTrackingEnterHandler    = 1;         { drag has entered handler }
   dragTrackingEnterWindow     = 2;         { drag has entered window }
   dragTrackingInWindow        = 3;         { drag is moving within window }
   dragTrackingLeaveWindow     = 4;         { drag has exited window }
   dragTrackingLeaveHandler    = 5;         { drag has exited handler }

   { Drag Drawing Handler Messages }
   dragRegionBegin             = 1;         { initialize drawing }
   dragRegionDraw              = 2;         { draw drag feedback }
   dragRegionHide              = 3;         { hide drag feedback }
   dragRegionIdle              = 4;         { drag feedback idle time }
   dragRegionEnd               = 5;         { end of drawing }

   { Zooming Constants }
   zoomNoAcceleration          = 0;         { use linear interpolation }
   zoomAccelerate              = 1;         { ramp up step size }
   zoomDecelerate              = 2;         { ramp down step size }
```

Data Types

Drag Manager Data Types

```
TYPE
   DragReference = LONGINT;
   ItemReference = LONGINT;

   FlavorType = ResType;
   FlavorFlags = LONGINT;
   DragAttributes = LONGINT;

   DragTrackingMessage = INTEGER;
   DragRegionMessage = INTEGER;

   ZoomAcceleration = INTEGER;
```

Special Flavor Data Types

```
   HFSFlavor = RECORD
      fileType:       OSType;            { file type }
      fileCreator:    OSType;            { file creator }
      fdFlags:        INTEGER;           { Finder flags }
      fileSpec:       FSSpec;            { file system specification }
   END;

   PromiseHFSFlavor = RECORD
      fileType:       OSType;            { file type }
      fileCreator:    OSType;            { file creator }
      fdFlags:        INTEGER;           { Finder flags }
      promisedFlavor: FlavorType;        { promised flavor containing FSSpec }
   END;
```

Drag Manager Routines

Installing and Removing Drag Handlers

```
FUNCTION InstallTrackingHandler
                              (trackingHandler : DragTrackingHandler;
                               theWindow : WindowPtr;
                               handlerRefCon : UNIV Ptr) : OSErr;

FUNCTION InstallReceiveHandler(receiveHandler : DragReceiveHandler;
                               theWindow : WindowPtr;
                               handlerRefCon : UNIV Ptr) : OSErr;

FUNCTION RemoveTrackingHandler(trackingHandler : DragTrackingHandler;
                               theWindow : WindowPtr) : OSErr;
```

```
FUNCTION RemoveReceiveHandler (receiveHandler : DragReceiveHandler;
                               theWindow : WindowPtr) : OSErr;
```

Creating and Disposing of Drag References

```
FUNCTION NewDrag              (VAR theDragRef : DragReference) : OSErr;

FUNCTION DisposeDrag          (theDragRef : DragReference) : OSErr;
```

Adding Drag Item Flavors

```
FUNCTION AddDragItemFlavor    (theDragRef : DragReference;
                               theItemRef : ItemReference;
                               theType : FlavorType;
                               dataPtr : UNIV Ptr;
                               dataSize : Size;
                               theFlags : FlavorFlags) : OSErr;

FUNCTION SetDragItemFlavorData(theDragRef : DragReference;
                               theItemRef : ItemReference;
                               theType : FlavorType;
                               dataPtr : UNIV Ptr;
                               dataSize : Size;
                               dataOffset : LONGINT) : OSErr;
```

Providing Drag Callback Procedures

```
FUNCTION SetDragSendProc      (theDragRef : DragReference;
                               sendProc : DragSendDataProc;
                               dragSendRefCon : UNIV Ptr) : OSErr;

FUNCTION SetDragInputProc     (theDragRef : DragReference;
                               inputProc : DragInputProc;
                               dragInputRefCon : UNIV Ptr) : OSErr;

FUNCTION SetDragDrawingProc   (theDragRef : DragReference;
                               drawingProc : DragDrawingProc;
                               dragDrawingRefCon : UNIV Ptr) : OSErr;
```

Performing a Drag

```
FUNCTION TrackDrag            (theDragRef : DragReference;
                               theEvent : EventRecord;
                               theRegion : RgnHandle) : OSErr;
```

Getting Drag Item Information

```
FUNCTION CountDragItems       (theDragRef : DragReference;
                               VAR numItems : INTEGER) : OSErr;

FUNCTION GetDragItemReferenceNumber
                              (theDragRef : DragReference;
                               index : INTEGER;
                               VAR theItemRef : ItemReference) : OSErr;
```

```
FUNCTION CountDragItemFlavors
                              (theDragRef : DragReference;
                               theItemRef : ItemReference;
                               VAR numFlavors : INTEGER) : OSErr;

FUNCTION GetFlavorType        (theDragRef : DragReference;
                               theItemRef : ItemReference;
                               index : INTEGER;
                               VAR theType : FlavorType) : OSErr;

FUNCTION GetFlavorFlags       (theDragRef : DragReference;
                               theItemRef : ItemReference;
                               theType : FlavorType;
                               VAR theFlags : FlavorFlags) : OSErr;

FUNCTION GetFlavorDataSize    (theDragRef : DragReference;
                               theItemRef : ItemReference;
                               theType : FlavorType;
                               VAR dataSize : Size) : OSErr;

FUNCTION GetFlavorData        (theDragRef : DragReference;
                               theItemRef : ItemReference;
                               theType : FlavorType;
                               dataPtr : UNIV Ptr;
                               VAR dataSize : Size;
                               dataOffset : LONGINT) : OSErr;

FUNCTION GetDragItemBounds    (theDragRef : DragReference;
                               theItemRef : ItemReference;
                               VAR itemBounds : Rect) : OSErr;

FUNCTION SetDragItemBounds    (theDragRef : DragReference;
                               theItemRef : ItemReference;
                               itemBounds : Rect) : OSErr;
```

Getting and Setting Drag Status Information

```
FUNCTION GetDragAttributes    (theDragRef : DragReference;
                               VAR attributes : DragAttributes) : OSErr;

FUNCTION GetDragMouse         (theDragRef : DragReference;
                               VAR mouse : Point;
                               VAR pinnedMouse : Point) : OSErr;

FUNCTION SetDragMouse         (theDragRef : DragReference;
                               pinnedMouse : Point) : OSErr;

FUNCTION GetDragOrigin        (theDragRef : DragReference;
                               VAR initialMouse : Point) : OSErr;

FUNCTION GetDragModifiers     (theDragRef : DragReference;
                               VAR modifiers : INTEGER;
                               VAR mouseDownModifiers : INTEGER;
                               VAR mouseUpModifiers : INTEGER) : OSErr;

FUNCTION GetDropLocation      (theDragRef : DragReference;
                               VAR dropLocation : AEDesc) : OSErr;
FUNCTION SetDropLocation     (theDragRef : DragReference;
                               dropLocation : AEDesc) : OSErr;
```

Window Highlighting Utilities

```
FUNCTION ShowDragHilite      (theDragRef : DragReference;
                              hiliteFrame : RgnHandle;
                              inside : BOOLEAN) : OSErr;
FUNCTION HideDragHilite      (theDragRef : DragReference) : OSErr;
FUNCTION DragPreScroll       (theDragRef : DragReference;
                              dH : INTEGER;
                              dV : INTEGER) : OSErr;
FUNCTION DragPostScroll      (theDragRef : DragReference) : OSErr;
FUNCTION UpdateDragHilite    (theDragRef : DragReference;
                              updateRgn : RgnHandle) : OSErr;
```

Drag Manager Utilities

```
FUNCTION WaitMouseMoved      (initialMouse : Point) : BOOLEAN;
FUNCTION ZoomRects           (fromRect : Rect;
                              toRect : Rect;
                              zoomSteps : INTEGER;
                              acceleration : ZoomAcceleration) : OSErr;
FUNCTION ZoomRegion          (region : RgnHandle;
                              zoomDistance Point;
                              zoomSteps : INTEGER;
                              acceleration : ZoomAcceleration) : OSErr;
```

TextEdit Utilities

```
FUNCTION TEGetHiliteRgn      (region : RgnHandle;
                              hTE : TEHandle) : OSErr;
```

Application Defined Routines

Drag Handler Routines

```
FUNCTION DragTrackingHandler (message : DragTrackingMessage;
                              theWindow : WindowPtr;
                              handlerRefCon : Ptr;
                              theDragRef : DragReference) : OSErr;
FUNCTION DragReceiveHandler  (theWindow : WindowPtr;
                              handlerRefCon : Ptr;
                              theDragRef : DragReference) : OSErr;
```

Drag Callback Procedures

```
FUNCTION DragSendDataProc     (theType : FlavorType;
                               dragSendRefCon : Ptr;
                               theItemRef : ItemReference;
                               theDragRef : DragReference) : OSErr;

FUNCTION DragInputProc        (VAR mouse : Point;
                               VAR modifiers : INTEGER;
                               dragInputRefCon : Ptr;
                               theDragRef : DragReference) : OSErr;

FUNCTION DragDrawingProc      (message : DragRegionMessage;
                               showRegion : RgnHandle;
                               showOrigin : Point;
                               hideRegion : RgnHandle;
                               hideOrigin : Point;
                               dragDrawingRefCon : Ptr;
                               theDragRef : DragReference) : OSErr;
```

C Summary

Constants

```
/* Gestalt Constants */
#define gestaltDragMgrAttr          'drag'  // Drag Manager attributes
#define gestaltDragMgrPresent       0       // Drag Manager is present

#define gestaltTEAttr               'teat'  // TextEdit attributes
#define gestaltTEHasGetHiliteRgn    0       // TextEdit has TEGetHiliteRgn

/* Flavor Flags */
#define flavorSenderOnly         0x00000001 // flavor available to sender only
#define flavorSenderTranslated   0x00000002 // flavor translated by sender
#define flavorNotSaved           0x00000004 // flavor should not be saved
#define flavorSystemTranslated   0x00000100 // flavor translated by system

/* Drag Attributes */
#define dragHasLeftSenderWindow     0x00000001 // drag has left source window
#define dragInsideSenderApplication 0x00000002 // drag is in the source app
#define dragInsideSenderWindow      0x00000004 // drag is in the source window

/* Special Flavor Type */
#define flavorTypeHFS               'hfs '  // flavor type for HFS data
#define flavorTypePromiseHFS        'phfs'  // flavor type for promised HFS
#define flavorTypeDirectory         'diry'  // flavor type for AOCE directory
```

```
/* Drag Tracking Handler Messages */
enum {
    dragTrackingEnterHandler        = 1,    // drag has entered handler
    dragTrackingEnterWindow         = 2,    // drag has entered window
    dragTrackingInWindow            = 3,    // drag is moving within window
    dragTrackingLeaveWindow         = 4,    // drag has exited window
    dragTrackingLeaveHandler        = 5     // drag has exited handler
};

/* Drag Drawing Handler Messages */
enum {
    dragRegionBegin                 = 1,    // initialize drawing
    dragRegionDraw                  = 2,    // draw drag feedback
    dragRegionHide                  = 3,    // hide drag feedback
    dragRegionIdle                  = 4,    // drag feedback idle time
    dragRegionEnd                   = 5     // end of drawing
};

/* Zooming Constants */
enum {
    zoomNoAcceleration              = 0,    // use linear interpolation
    zoomAccelerate                  = 1,    // ramp up step size
    zoomDecelerate                  = 2     // ramp down step size
};
```

Data Types

Drag Manager Data Types

```
typedef unsigned long DragReference;
typedef unsigned long ItemReference;

typedef ResType FlavorType;
typedef unsigned long FlavorFlags;
typedef unsigned long DragAttributes;

typedef short DragTrackingMessage;
typedef short DragRegionMessage;

typedef short ZoomAcceleration;
```

Special Flavor Data Types

```
typedef struct HFSFlavor {
    OSType          fileType;             // file type
    OSType          fileCreator;          // file creator
    unsigned short  fdFlags;              // Finder flags
    FSSpec          fileSpec;             // file system specification
} HFSFlavor;
```

```
typedef struct PromiseHFSFlavor {
    OSType          fileType;             // file type
    OSType          fileCreator;          // file creator
    unsigned short  fdFlags;              // Finder flags
    FlavorType      promisedFlavor;       // promised flavor containing FSSpec
} HFSFlavor;
```

Drag Manager Routines

Installing and Removing Drag Handlers

```
pascal OSErr InstallTrackingHandler
                              (DragTrackingHandler trackingHandler,
                               WindowPtr theWindow,
                               void *handlerRefCon);

pascal OSErr InstallReceiveHandler
                              (DragReceiveHandler receiveHandler,
                               WindowPtr theWindow,
                               void *handlerRefCon);

pascal OSErr RemoveTrackingHandler
                              (DragTrackingHandler trackingHandler,
                               WindowPtr theWindow);

pascal OSErr RemoveReceiveHandler
                              (DragReceiveHandler receiveHandler,
                               WindowPtr theWindow);
```

Creating and Disposing of Drag References

```
pascal OSErr NewDrag          (DragReference *theDragRef);

pascal OSErr DisposeDrag      (DragReference theDragRef);
```

Adding Drag Item Flavors

```
pascal OSErr AddDragItemFlavor(DragReference theDragRef,
                               ItemReference theItemRef,
                               FlavorType theType,
                               void *dataPtr,
                               Size dataSize,
                               FlavorFlags theFlags);

pascal OSErr SetDragItemFlavorData
                              (DragReference theDragRef,
                               ItemReference theItemRef,
                               FlavorType theType,
                               void *dataPtr,
                               Size dataSize,
                               unsigned long dataOffset);
```

Providing Drag Callback Procedures

```
pascal OSErr SetDragSendProc  (DragReference theDragRef,
                               DragSendDataProc sendProc,
                               void *dragSendRefCon);

pascal OSErr SetDragInputProc (DragReference theDragRef,
                               DragInputProc inputProc,
                               void *dragInputRefCon);

pascal OSErr SetDragDrawingProc
                              (DragReference theDragRef,
                               DragDrawingProc drawingProc,
                               void *dragDrawingRefCon);
```

Performing a Drag

```
pascal OSErr TrackDrag        (DragReference theDragRef,
                               const EventRecord *theEvent,
                               RgnHandle theRegion);
```

Getting Drag Item Information

```
pascal OSErr CountDragItems   (DragReference theDragRef,
                               unsigned short *numItems);

pascal OSErr GetDragItemReferenceNumber
                              (DragReference theDragRef,
                               unsigned short index,
                               ItemReference *theItemRef);

pascal OSErr CountDragItemFlavors
                              (DragReference theDragRef,
                               ItemReference theItemRef,
                               unsigned short *numFlavors);

pascal OSErr GetFlavorType    (DragReference theDragRef,
                               ItemReference theItemRef,
                               unsigned short index,
                               FlavorType *theType);

pascal OSErr GetFlavorFlags   (DragReference theDragRef,
                               ItemReference theItemRef,
                               FlavorType theType,
                               FlavorFlags *theFlags);

pascal OSErr GetFlavorDataSize(DragReference theDragRef,
                               ItemReference theItemRef,
                               FlavorType theType,
                               Size *dataSize);

pascal OSErr GetFlavorData    (DragReference theDragRef,
                               ItemReference theItemRef,
                               FlavorType theType,
                               void *dataPtr,
                               Size *dataSize,
                               unsigned long dataOffset);

pascal OSErr GetDragItemBounds(DragReference theDragRef,
                               ItemReference theItemRef,
                               Rect *itemBounds);

pascal OSErr SetDragItemBounds(DragReference theDragRef,
                               ItemReference theItemRef,
                               const Rect *itemBounds);
```

Getting and Setting Drag Status Information

```
pascal OSErr GetDragAttributes(DragReference theDragRef,
                                DragAttributes *attributes);

pascal OSErr GetDragMouse     (DragReference theDragRef,
                                Point *mouse,
                                Point *pinnedMouse);

pascal OSErr SetDragMouse     (DragReference theDragRef,
                                Point pinnedMouse);

pascal OSErr GetDragOrigin    (DragReference theDragRef,
                                Point *initialMouse);

pascal OSErr GetDragModifiers (DragReference theDragRef,
                                short *modifiers,
                                short *mouseDownModifiers,
                                short *mouseUpModifiers);

pascal OSErr GetDropLocation  (DragReference theDragRef,
                                AEDesc *dropLocation);

pascal OSErr SetDropLocation  (DragReference theDragRef,
                                const AEDesc *dropLocation);
```

Window Highlighting Utilities

```
pascal OSErr ShowDragHilite   (DragReference theDragRef,
                                RgnHandle hiliteFrame,
                                Boolean inside);

pascal OSErr HideDragHilite   (DragReference theDragRef);

pascal OSErr DragPreScroll    (DragReference theDragRef,
                                short dH,
                                short dV);

pascal OSErr DragPostScroll   (DragReference theDragRef);

pascal OSErr UpdateDragHilite (DragReference theDragRef,
                                RgnHandle updateRgn);
```

Drag Manager Utilities

```
pascal Boolean WaitMouseMoved (Point initialMouse);
pascal OSErr ZoomRects        (const Rect *fromRect,
                               const Rect *toRect,
                               short zoomSteps,
                               ZoomAcceleration acceleration);

pascal OSErr ZoomRegion       (RgnHandle region,
                               Point zoomDistance,
                               short zoomSteps,
                               ZoomAcceleration acceleration);
```

TextEdit Utilities

```
pascal OSErr TEGetHiliteRgn   (RgnHandle region,
                               TEHandle hTE);
```

Application Defined Routines

Drag Handler Routines

```
pascal OSErr DragTrackingHandler
                              (DragTrackingMessage message,
                               WindowPtr theWindow,
                               void *handlerRefCon,
                               DragReference theDragRef);

pascal OSErr DragReceiveHandler
                              (WindowPtr theWindow,
                               void *handlerRefCon,
                               DragReference theDragRef);
```

Drag Callback Procedures

```
pascal OSErr DragSendDataProc (FlavorType theType,
                               void *dragSendRefCon,
                               ItemReference theItemRef,
                               DragReference theDragRef);

pascal OSErr DragInputProc    (Point *mouse,
                               short *modifiers,
                               void *dragInputRefCon,
                               DragReference theDragRef);

pascal OSErr DragDrawingProc  (DragRegionMessage message,
                               RgnHandle showRegion,
                               Point showOrigin,
                               RgnHandle hideRegion,
                               Point hideOrigin,
                               void *dragDrawingRefCon,
                               DragReference theDragRef);
```

Assembly-Language Summary

Constants

```
; Gestalt Selector and Response Constants

gestaltDragMgrAttr            EQU    'drag'    ; Drag Manager attributes
gestaltDragMgrPresent         EQU    0         ; Drag Manager is present

gestaltTEAttr                 EQU    'teat'    ; TextEdit attributes
gestaltTEHasGetHiliteRgn      EQU    0         ; TextEdit has TEGetHiliteRgn


; Flavor Flags

flavorSenderOnly              EQU    $00000001 ; flavor available to sender only
flavorSenderTranslated        EQU    $00000002 ; flavor is translated by sender
flavorNotSaved                EQU    $00000004 ; flavor should not be saved
flavorSystemTranslated        EQU    $00000100 ; flavor is translated by system


; Drag Attributes

dragHasLeftSenderWindow       EQU    $00000001 ; drag has left source window
dragInsideSenderApplication   EQU    $00000002 ; drag is in the source app
dragInsideSenderWindow        EQU    $00000004 ; drag is in the source window


; Special Flavor Types

flavorTypeHFS                 EQU    'hfs '    ; flavor type for HFS data
flavorTypePromiseHFS          EQU    'phfs'    ; flavor type for promised HFS
flavorTypeDirectory           EQU    'diry'    ; flavor type for AOCE directory


; Drag Tracking Handler Messages

dragTrackingEnterHandler      EQU    1         ; drag has entered handler
dragTrackingEnterWindow       EQU    2         ; drag has entered window
dragTrackingInWindow          EQU    3         ; drag is moving within window
dragTrackingLeaveWindow       EQU    4         ; drag has exited window
dragTrackingLeaveHandler      EQU    5         ; drag has exited handler


; Drag Drawing Procedure Messages

dragRegionBegin               EQU    1         ; initialize drawing
dragRegionDraw                EQU    2         ; draw drag feedback
dragRegionHide                EQU    3         ; hide drag feedback
dragRegionIdle                EQU    4         ; drag feedback idle time
```

```
dragRegionEnd                EQU    5          ; end of drawing


; Zoom Acceleration

zoomNoAcceleration           EQU    0          ; use linear interpolation
zoomAccelerate               EQU    1          ; ramp up step size
zoomDecelerate               EQU    2          ; ramp down step size
```

Data Structures

HFS Flavor Record

0	`fileType`	long	file type
4	`fileCreator`	long	file creator
8	`fdFlags`	word	Finder flags
10	`fileSpec`	70 bytes	file system specification

Promised HFS Flavor Record

0	`fileType`	long	file type
4	`fileCreator`	long	file creator
8	`fdFlags`	word	Finder flags
10	`promisedFlavor`	long	promised flavor containing FSSpec

Trap Macros

Trap Macros Requiring Routine Selector
`_DragDispatch`

Selector	Routine
$0001	`InstallTrackingHandler`
$0002	`InstallReceiveHandler`
$0003	`RemoveTrackingHandler`
$0004	`RemoveReceiveHandler`
$0005	`NewDrag`
$0006	`DisposeDrag`
$0007	`AddDragItemFlavor`
$0009	`SetDragItemFlavorData`
$000A	`SetDragSendProc`

$000B	`SetDragInputProc`
$000C	`SetDragDrawingProc`
$000D	`TrackDrag`
$000E	`CountDragItems`
$000F	`GetDragItemReferenceNumber`
$0010	`CountDragItemFlavors`
$0011	`GetFlavorType`
$0012	`GetFlavorFlags`
$0013	`GetFlavorDataSize`
$0014	`GetFlavorData`
$0015	`GetDragItemBounds`
$0016	`SetDragItemBounds`
$0017	`GetDropLocation`
$0018	`SetDropLocation`
$0019	`GetDragAttributes`
$001A	`GetDragMouse`
$001B	`SetDragMouse`
$001C	`GetDragOrigin`
$001D	`GetDragModifiers`
$001E	`ShowDragHilite`
$001F	`HideDragHilite`
$0020	`DragPreScroll`
$0021	`DragPostScroll`
$0022	`UpdateDragHilite`
$0023	`WaitMouseMoved`
$0024	`ZoomRects`
$0025	`ZoomRegion`

`_TEDispatch`

Selector	**Routine**
$000F	`TEGetHiliteRgn`

Result Codes

`noErr`	0	No error
`paramErr`	-50	Parameter error
`memFullErr`	-108	Not enough memory
`badDragRefErr`	-1850	Unknown drag reference
`badDragItemErr`	-1851	Unknown drag item reference
`badDragFlavorErr`	-1852	Unknown flavor type
`duplicateFlavorErr`	-1853	Flavor type already exists
`cantGetFlavorErr`	-1854	Error while trying to get flavor data
`duplicateHandlerErr`	-1855	Handler already exists
`duplicateHandlerErr`	-1856	Handler not found
`dragNotAcceptedErr`	-1857	Drag was not accepted by receiver