Chapter 10

DRAG AND DROP

CHAPTER 10 DRAG AND DROP

Goal

To understand the objects and the steps in the drag and drop data transfer cycle and to apply this knowledge to a case study implementation—a file well.

Prerequisites

A practical understanding of pasteboards and the two elements of custom views: event handling and drawing, specifically image compositing.

Objectives

At the end of this section, you will be able to:

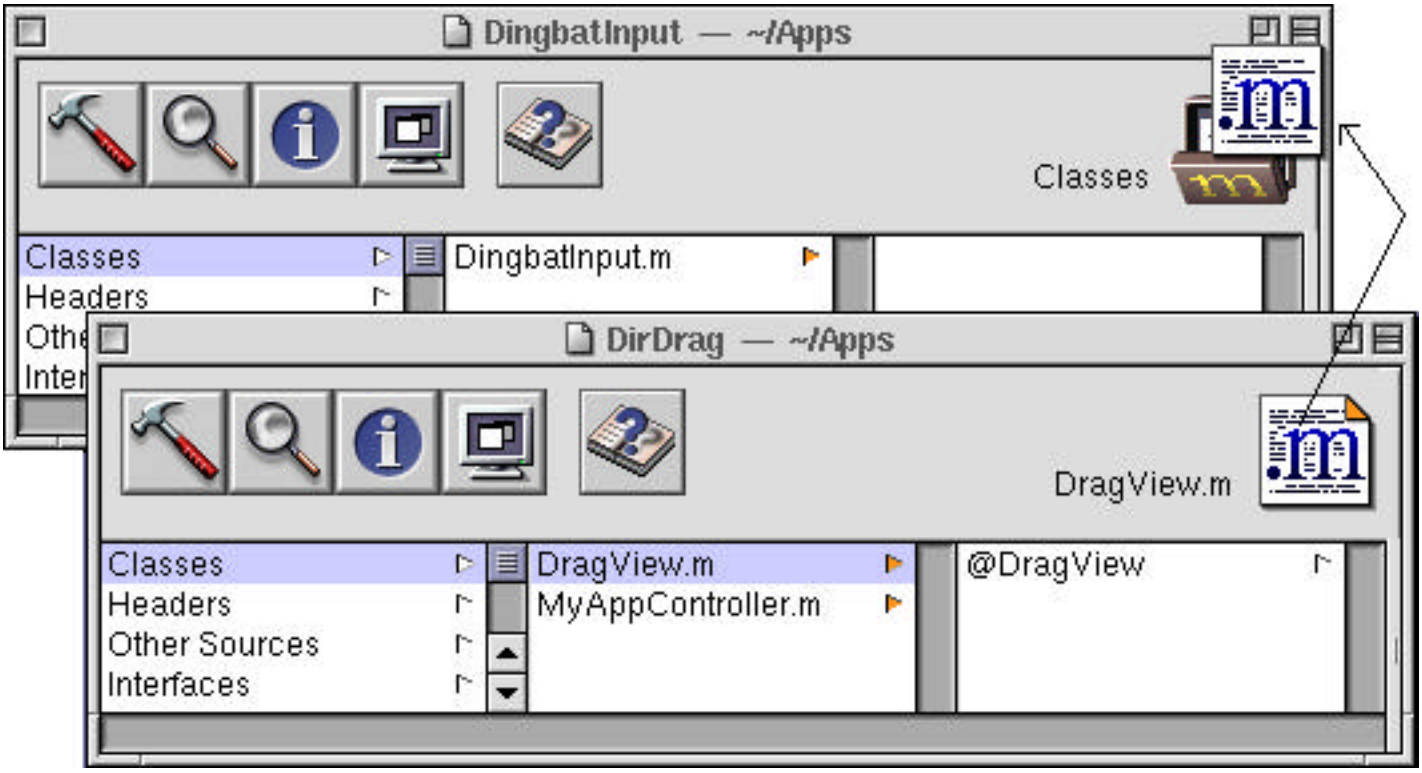
- List the objects and data transfer cycle involved with Drag and Drop
- Implement a custom view that is a dragging source
- Implement a custom view that is a dragging destination

Reading

NSDraggingSource protocol reference in the Application Kit

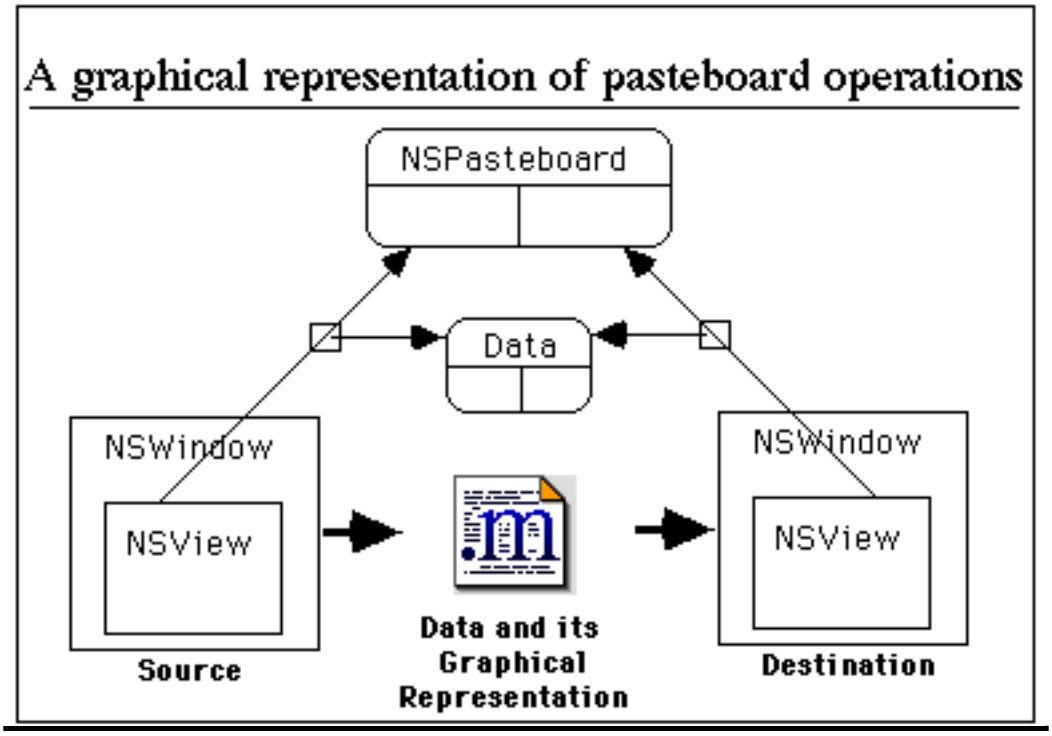
NSDraggingDestination protocol reference in the Application Kit

NSDraggingInfo protocol reference in the Application Kit



Drag and drop—object-oriented data transfer

Drag and drop is a familiar user interface aspect of most WIMP (Window Icon Menu Pointer) environments. An icon, graphically representing some kind of data, is selected from a source location, dragged across the screen to a destination location and dropped, signifying a data transfer. It is apt to view this user interface as object-oriented: simple user events such as mouse actions are applied to graphical objects and applications that know how to respond in a specific way, encapsulated behind the generic interface. Through polymorphism, each object to respond to the same messages—**mouseDown:**, **mouseDragged:**, **mouseUp:**—freeing the user of having to know how or of making a significant distinction among different object types. Drag and drop equips the user with an interface that is simple, intuitive, and very powerful. How does it work?

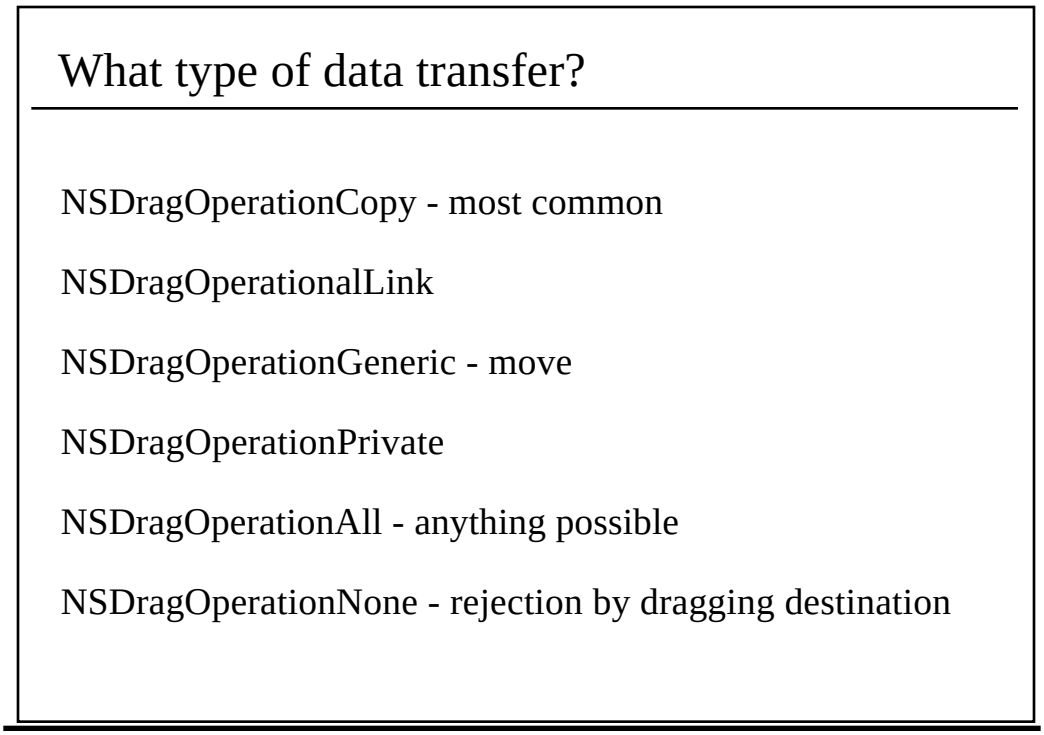


A graphical representation of pasteboard operations

Since drag and drop implies a data transfer between independent objects—in the same or across two different applications—it is natural that it involves a pasteboard. A data object is essentially passed from the source to the destination object via the NSDraggingPboard pasteboard instance. Graphical feedback vividly reinforces the transfer in the user's mind: the data object is represented as an icon, and typically, the destination illustrates that it is ready to accept the data and may even animate in some fashion once the data is actually dropped, accepted and incorporated into the application. While the basic mechanics of drag and drop are fairly straightforward and fully supported by the Application Kit, the way an application object uses the data it accepts or decides what data it will provide is unique to each application, sometimes involving complex internal behavior.

Because the drag and drop source and destinations occupy visual real estate, they must be subclasses of either NSWindow or NSView. Like pasteboard operations in general, the data object is dynamically typed and can be just about anything the source and destination agree upon. The moving image, the iconic representation of the data being transferred, is a fixed size sampled image of any type that NSImage understands.

But what kind of transfer is implied by a drag and drop: a copy, a move, a link? This is something the user may control through keyboard modifiers and is intimately based on what the dragging source and destination agree to support.



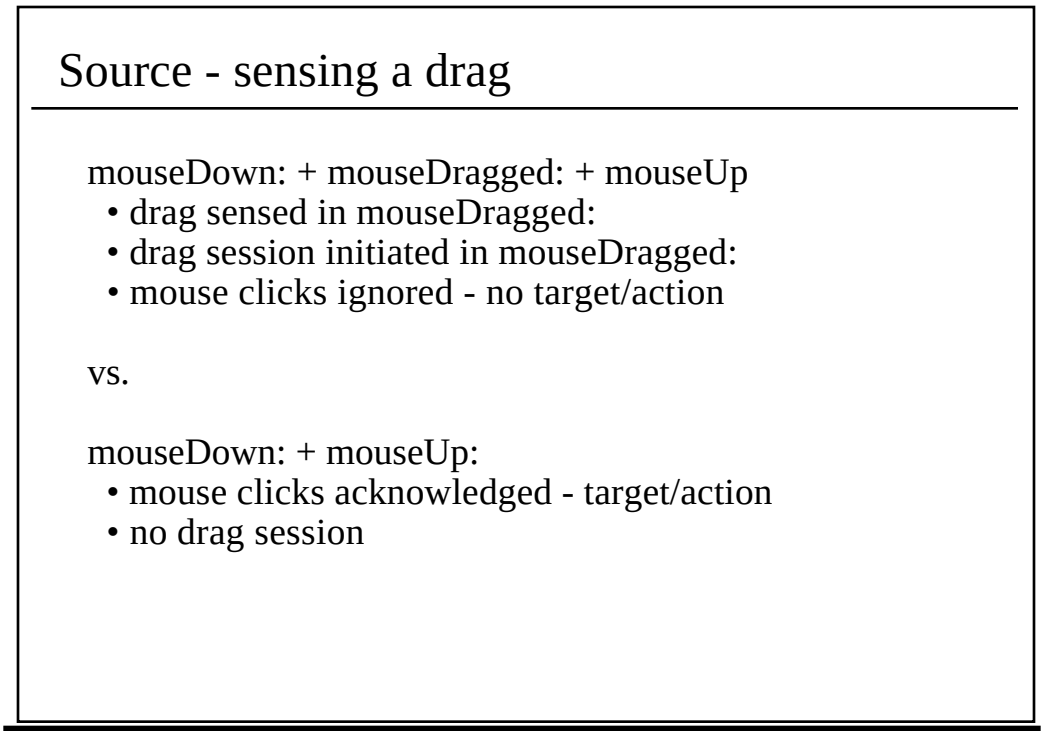
What type of data transfer?

The Application Kit provides a set of constants for a dragging source or destination to dynamically identify the type of transfer it supports. The source indicates which are possible. The destination determines which specific type it will perform given its capabilities, the type of data on the pasteboard, and possibly user input. The types can be combined in a bitmask to indicate more than one possibility. Your objects can also define their own private types.

- NSDragOperationCopy—the most common, it means that the data will be duplicated and copied to the destination.
- NSDragOperationLink—the same data instance will be shared by both source and destination.
- NSDragOperationGeneric—the destination decides. Usually a generic operation is a move, removing the original in the source, relocating it to the destination.
- NSDragOperationPrivate—the source and destination objects can agree on a non-standard sort of operation which is not one of the standards. It is a type private to the two participating objects.
- NSDragOperationAll—represents all types or'ed together. This type is usually used by the source and implies that any of the operations are possible and should be determined by the destination.
- NSDragOperationNone—no transfer possible. Returned by destinations to reject the pending operation.

File well—a practical example

While the data being transferred can be arbitrary—a string, one or more business objects, file contents—a common data type is a file name. Perhaps the most familiar drag and drop involves a file name icon and represents the movement of one or more files between applications including a file viewer itself. A single custom view can be designed to display the icon of a file it represents and implement both dragging source and destination behavior. The file icon can be dragged away and dropped over another location or dragged into and dropped on the view, thereby changing the file it graphically represents. An NSView subclass capable of doing this might be called a file well. Because it implements both the source and destination behavior and because file name data is easy to grasp and easy to handle, a file well provides an excellent study for the implementation of drag and drop.



Source—sensing a drag

The first thing a dragging source must implement is the ability to sense when a dragging request occurs. Many sources are controls, such as a file well, needing to distinguish a mouse click from a mouse drag. The former might trigger a target/action response while the latter initiates a drag and drop. Both events begin with a **mouseDown:** message and are not fully defined until the next mouse event—**mouseUp:** or **mouseDragged:**. Either one precludes the other. As such, the source object does not typically respond until it knows something is happening. Dragging sessions are usually initiated from **mouseDragged:**.

As you will see, initiating the drag session nonetheless requires the event object passed as an argument to the initial **mouseDown:** message. Source objects usually need to **retain** the mouse down event object, make use of it in **mouseDragged:** and **release** it in **mouseUp:** or the next **mouseDown:**.

Once a source has sensed a drag request from the user interface, it must provide the corresponding data and communicate that a drag and drop session is now in effect.

Source - providing the data in mouseDragged:

```
// Get Drag/Drop Named Pasteboard
NSPasteboard *pb =
    [NSPasteboard pasteboardWithName: NSDragPboard];

//Declare our data type, e.g., a filename
[pb declareTypes:
    [NSArray arrayWithObject: NSFilenamesPboardType]
    owner: self];

// Write the data to the Pasteboard
[pb setPropertyList;
    [NSArray arrayWithObject: filename]
    forType: NSFilenamesPboardType];
```

Source—providing the data in mouseDragged:

Usually from within **mouseDragged:**, the source object must now place the relevant data on the pasteboard. The Application Kit provides a single named pasteboard instance for the occasion—NSDragPboard. The source performs the standard pasteboard owner operations for writing data:

1. Get the NSDragPboard pasteboard instance.
2. Declare the data types to be provided. The file well uses the pre-defined NSFilenamesPboardType, an array of one or more filenames.
3. Write the data to the pasteboard. Property list data types, arrays or dictionaries of value objects, are written using **setPropertyList:forType:**.

As in any of pasteboard operation, the source can provide multiple representations and even provide them lazily.

Source - initiating the drag - mouseDragged:

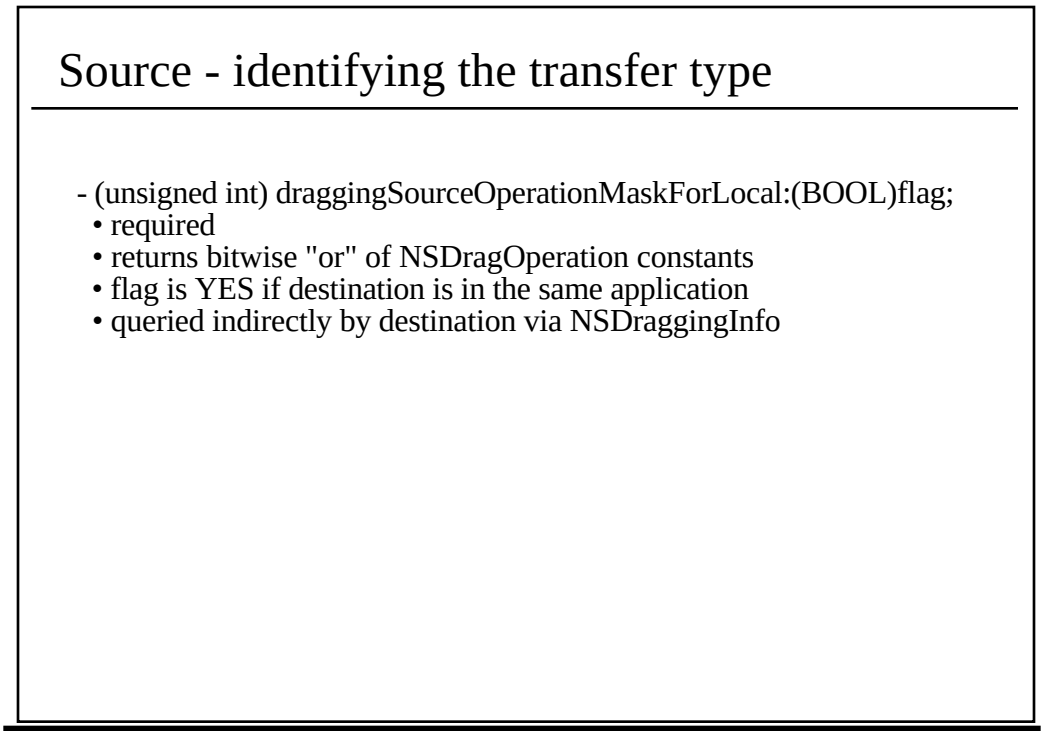
```
// Declarations for clarity.
// Typically non-nil instance variables
NSImage *myDataImage;
NSEvent *myLastMouseDownEvent;   // from mouseDown:
NSPoint myViewImageOrigin;

// Get Drag/Drop Named Pasteboard
NSPasteboard *pb =
    [NSPasteboard pasteboardWithName: NSDragPboard];

// Start the drag session
[self dragImage:      myDataImage
    at:               myViewImageOrigin
    origin:           myViewImageOrigin
    offset:           NSMakeSize(0,0)
    pasteboard:       pb
    source:           self
    slideBack:        YES];
```

Source—initiating the drag—mouseDragged:

Once the data is available on the pasteboard, the source initiates the drag session with a method it inherits from its superclass. This causes the icon to appear under the mouse cursor and notifies the relevant dragging destination as soon as the icon is dragged into its view. The **dragImage:** ... method takes the following parameters:

- dragImage:—an NSImage instance loaded with a sampled image file.
- at:—a point in the source's coordinate system indicating where the icon will be initially composited. This will usually correspond to the lower left corner of the original image in the source, the representation of data available for dragging. Values must be in the source's coordinate system.
- offset:—the x and y offset of the current mouse location relative to the initial **mouseDown:** location.
- event:—the original **mouseDown:** event object. If this message is sent from **mouseDragged:**, the source must retain the mouseDown event object and supply it here.
- pasteboard:—the pasteboard containing the source's data.
- source:—the source object itself, the "owner" of the drag session.
- slideBack:—whether the image should graphically slide back after unsuccessful or incomplete drop operations.

Once the source initiates a dragging session, all subsequent **mouseDragged:** and the final **mouseUp:** events are processed by the superclass to support drag and drop. The source object will not receive another mouse message until the next **mouseDown:**. In essence, the source will receive only one **mouseDragged:** event or one **mouseUp:** event, not both. The latter implies that there was no drag at all.



Source—identifying the transfer type

Finally, the source object must identify which NSDragOperations are possible. The source must implement **draggingSourceOperationMaskForLocal:**. It returns a bitmask listing one or more operations that are valid for the given operation. Dragging destinations invoke this message indirectly by querying the NSDraggingInfo object passed to them as a parameter. Look ahead to the destination pages for more detail.

Some applications will permit drag and drop operations between two objects within the application itself—between to documents or two views on the same window such as an icon shelf. The application may require special handling here. Some of these applications may not support dragging of data outside the application at all. In these cases, the answer will be conditional on whether the operation is local, within the application, or not, between this and another application. The flag argument makes this distinction.

To prohibit any kind of drag given the current state, the source may return NSDragOperationNone.

Destination - registering acceptable dragged types

```
// NSView Designated Initializer
- (id)initWithFrame: (Srect)frameRect
{
  [super initWithFrame: frameRect];

  // Declare what types we can accept
  [self registerForDraggedTypes:
    [NSArray arrayWithObject: NSFilenamesPboardType]]

  return self;
}
```

Destination—registering acceptable dragged types

Destination objects begin their life as potential participants in drag and drop sessions by registering the pasteboard types they are willing to accept. Whether NSWindow or NSView subclasses, a destination inherits a method for the purpose: **registerForDraggedTypes:**. Analogous to the list of types that a pasteboard owner provides, the destination specifies an array of one or more types it will accept. Registration must happen before any drag and drop operation is possible. Typically, it is done in the classes designated initializer. For the NSView subclass file well, it is **initWithFrame:**.

Destination - sensing and tracking a drag

Methods implemented by Destination
- (unsigned int)draggingEntered:(id<NSDraggingInfo>)sender;
- (unsigned int)draggingUpdated:(id)<NSDraggingInfo>)sender;
- (void)draggingExited:(id)<NSDraggingInfo>)sender;

What they do
- validate data type (e.g., on Pasteboard), transfer type
- update view's appearance
- message sender for more DraggingInfo
- return transfer type destination will use
- to fail, return NSDragOperationNone;

Destination—sensing and tracking a drag

Like the source, the destination needs to sense when the dragging mouse enters its real estate, signifying a potential data drop. There are three methods a drag destination must implement for this purpose:

» **draggingEntered:**—when the mouse first enters the destination view

» **draggingUpdated:**—each time the mouse is dragged within the view

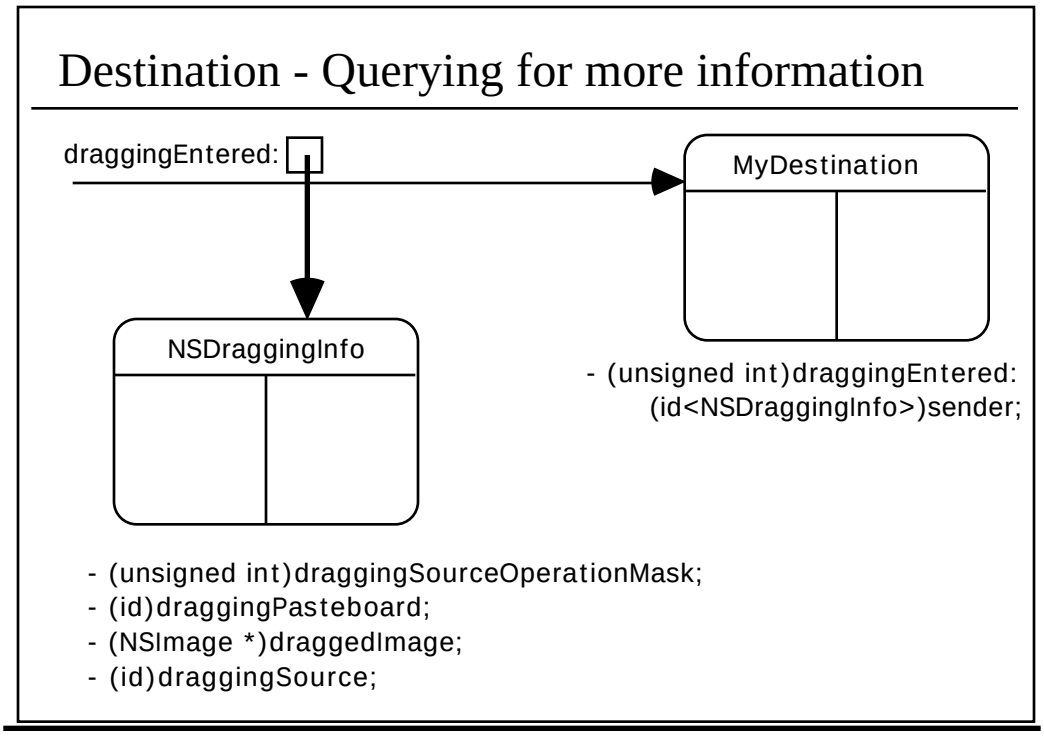
» **draggingExited:**—when the mouse completely leaves the view or stops dragging due to a **mouseUp:**.

These provide your object with the opportunity for two chores typical of dragging destinations:

» validate the potential data drop. Check the data on the pasteboard and the transfer type. If anything is unacceptable, return NSDragOperationNone, otherwise, return the type of operation the destination will perform.

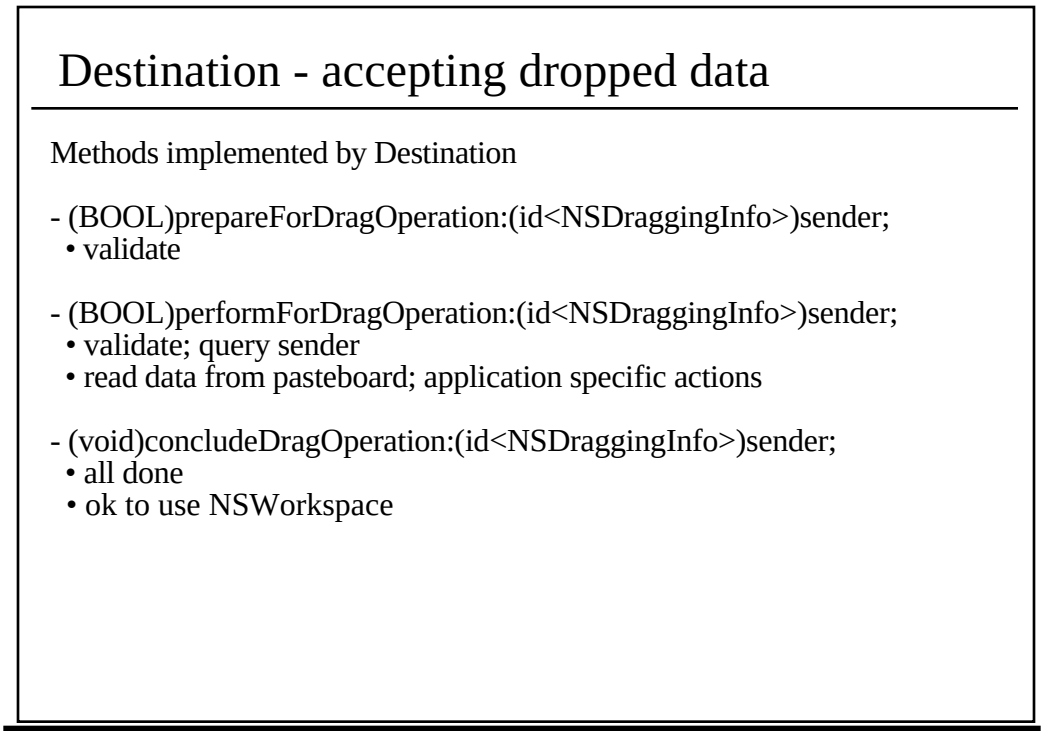
» update the destination’s appearance. Destination views often provide feedback that they will accept the data, that a drop is pending and applies directly to them. This may involve compositing the dragged image or providing some animated feedback, even tracking the mouse as it drags around within the view.

The single argument is an object that conforms to the NSDraggingInfo protocol and provides access to attributes about the session such as the data transfer type, the pasteboard, the dragged image and the source object if local.



Destination—querying for more information

A drag and drop destination will typically need to gather more information—to validate the transfer type and possibly the data on the pasteboard, to gain access to the dragged image, or to message a local dragging source object. All this information in available through the sender argument passed to all six of the destination messages in the NSDraggingDestination informal protocol. The sender conforms to the NSDraggingInfo protocol and is not the same as the dragging source which is typically in a completely different address space. These are some of the more commonly used methods.



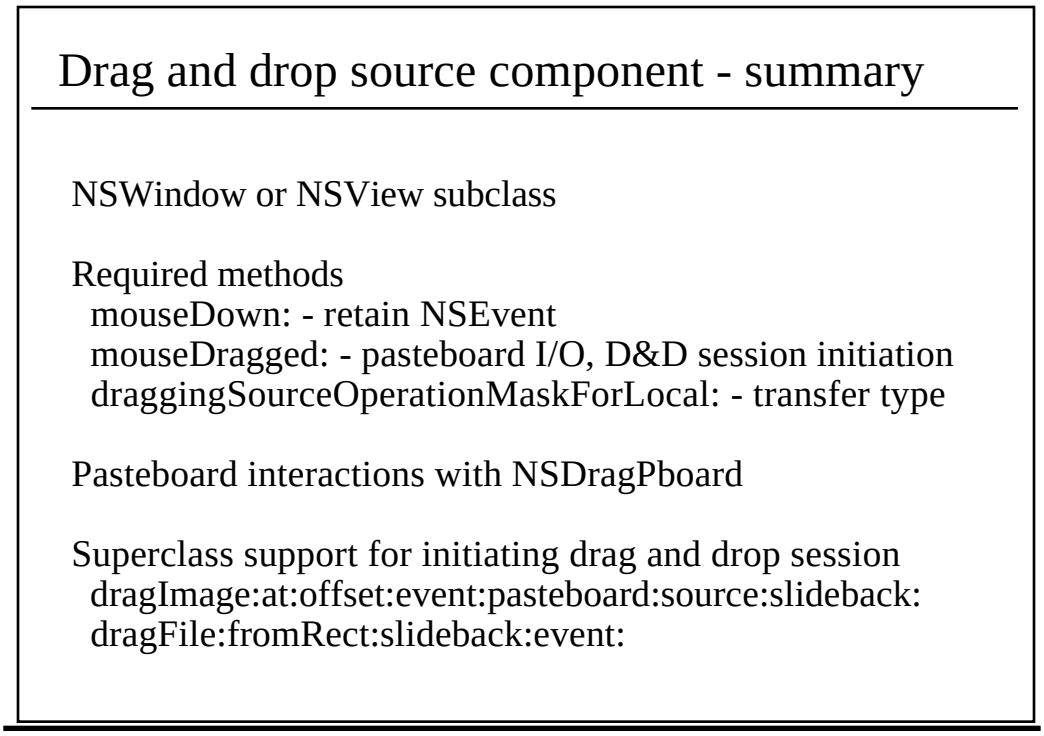
Destination—accepting dropped data

Once the user releases the mouse button over the destination that has not failed any of the mouse tracking methods, the drag turns into a drop and the destination accepts data from the pasteboard. There are three steps involved, each permitting unique actions typically required by most destinations:

» **prepareForDragOperation:**—chiefly for a final validation of the transfer type and the actual data. Once again, the destination can gather additional information about the context by messaging sender, an NSDraggingInfo object. The destination may fail this operation for any reason by returning NO. Typically, this would be due to the data on the pasteboard and/or the transfer type.

» **performDragOperation:**—intended for the actual data transfer into the destination object, it is here where the destination obtains the pasteboard, requests the valid types, reads the data, then finishes with specialized actions that incorporate the data into the application-specific context.The destination may fail this operation for any reason by returning NO. This might reflect an inability to read the data from the pasteboard or to successfully accommodate it in the application.

» **concludeDragOperation:**—any cleanup from the operation is performed here. File handling destinations such as a File Well are likely to need file system services via NSWorkspace. This can only be done here, not before in any of the earlier methods.

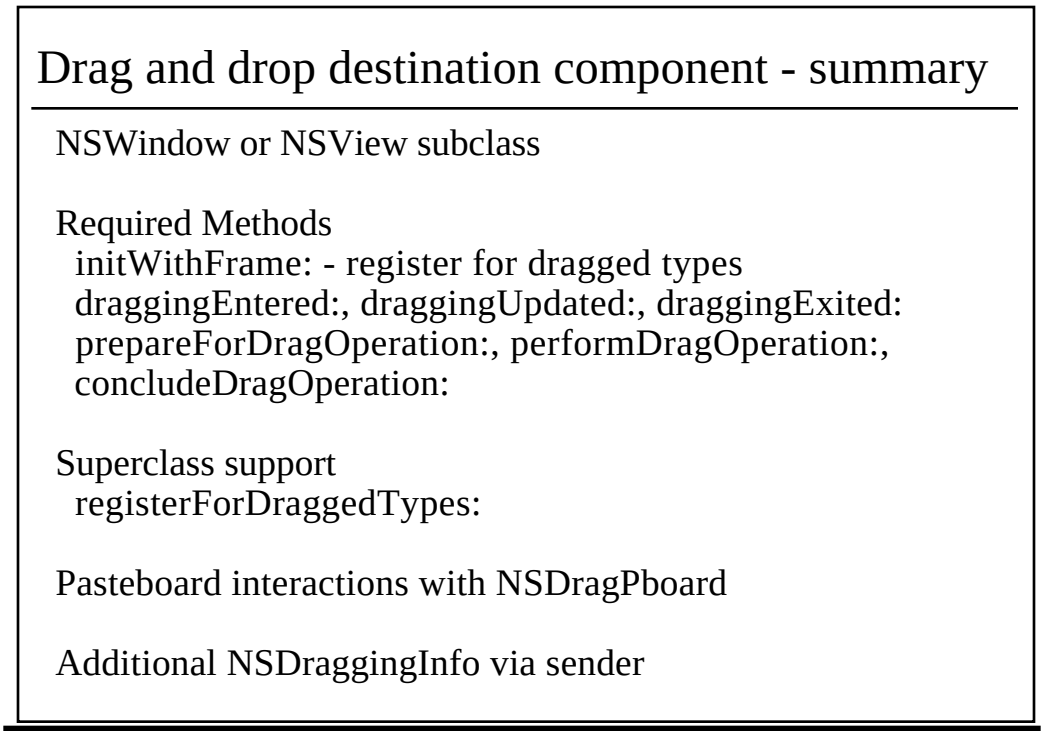


Drag and drop source component—summary

Drag and drop sources are NSWindow or NSView subclasses.

To sense a drag request and to report the supported transfer types, the source must override mouse event methods **mouseDown:** and **mouseDragged:** and implement the only required method in the NSDraggingSource informal protocol, **draggingSourceOperationMaskForLocal:**. Check the protocol documentation for additional methods a source may choose to implement.

All the action typically happens in **mouseDragged:** and involves transferring the data to the pasteboard and initiating the drag and drop session using one of the two superclass methods.



Drag and drop destination component—summary

Drag and Drop destinations are NSWindow or NSView subclasses. They register their unique list of acceptable data types as early as possible, typically in their designated initializer with a superclass method **registerForDraggedTypes:**. Once done, they wait for drag and drop to occur.

Dragging and dropping actions are communicated to the destination in six steps, each corresponding to a method the destination must implement:

- **draggingEntered:**
- **draggingUpdated:**
- **draggingExited:**
- **prepareForDragOperation:**
- **performDragOperation:**
- **concludeDragOperation:**

Each of these take a part in validating the data, updating the destination's appearance, transferring the data from the pasteboard into the application and cleaning up. All of these but the last can fail, prohibiting the drag and drop from taking place. All of them can gain additional information by messaging a sender argument that conforms to the NSDraggingInfo protocol.

Important ideas from this section

» Drag and Drop is an object-oriented graphical representation of data transfers using NSPasteboards.

» Drag and Drop involves two important custom objects: a dragging source and a dragging destination. Each of these objects can be either NSWindow or NSView subclasses.

» A dragging source must:

Properly sense a **mouseDragged:** event after retaining the associated **mouseDown:** event object

Provide typed data to the dragging pasteboard

Initiate a dragging session using NSWindow or NSView superclass methods

Identify the data transfer type when the dragging destination queries through the DraggingInfo object

» A dragging destination must:

Register for acceptable dragged types

Sense and track the dragging mouse

Accept dropped data, reading it from the pasteboard

Optionally query for additional information

Classes and protocols featured in this section

» NSDraggingSource—informal protocol

» NSDraggingDestination—informal protocol

» NSDraggingInfo—protocol

» NSPasteboard

REVIEW DRAG AND DROP

1. Name the key players in the drag and drop data transfer cycle.

2. What kinds of objects can act as a dragging source? How about a dragging destination? Does this make sense?

3. List a few possible things a dragging destination might check for when validating a pending drop.

4. How would a dragging source keep from becoming its own dragging destination?