Chapter 16

PASTEBOARDS

CHAPTER 16 PASTEBOARDS

Goal

To understand pasteboard mechanics and appreciate their widespread use in a variety of contexts.

Prerequisites

Familiarity with end-user pasteboard usage for tasks like cut, copy and paste.

Objectives

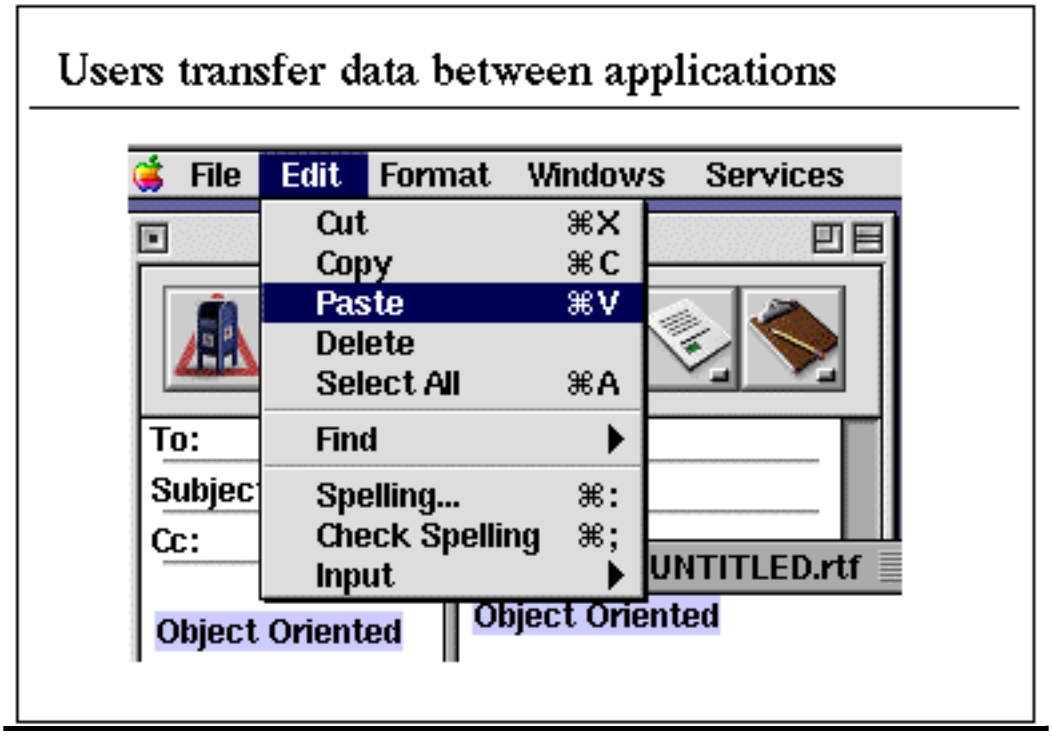
At the end of this section, you will be able to:

» Implement pasteboard owner and reader objects and outline each step of the data transfer cycle

» List features that use pasteboard

Reading

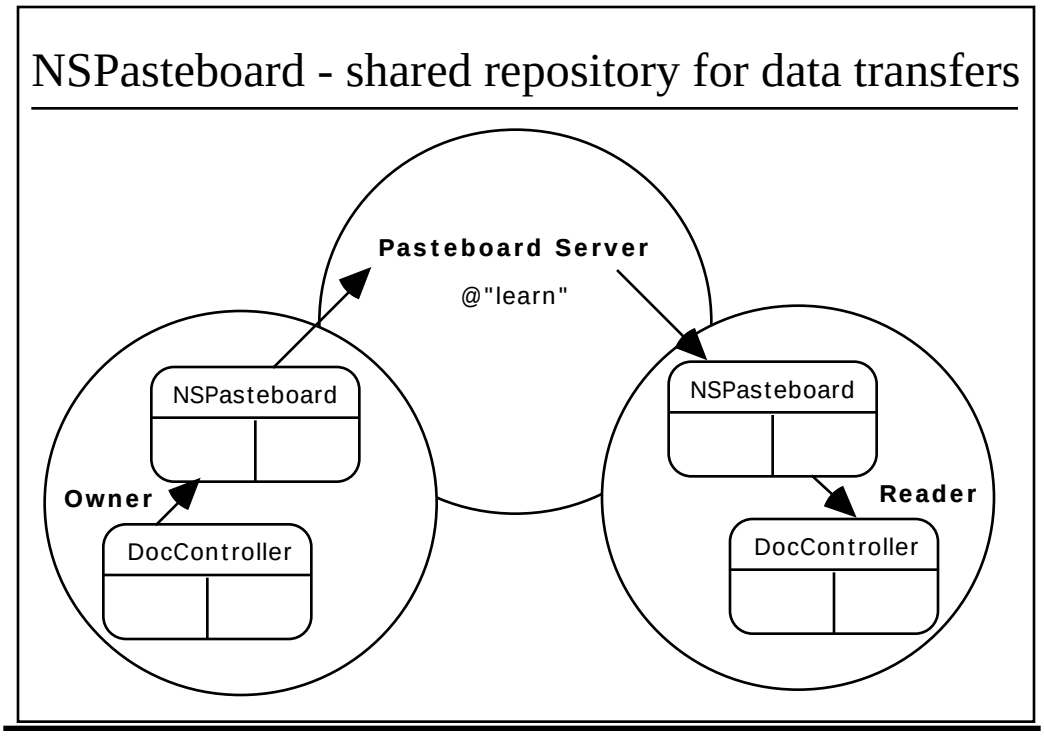
NSPasteboard class reference in the Applicaton Kit



Users transfer data between applications

A typical user runs a variety of applications simultaneously, each specialized for a certain task or domain—word processors, mail readers, database applications, spreadsheets, Web browsers and so on. Since most applications manage information in one form or another, the desire to share data between them is natural and useful. The ability to easily transfer data between applications is the mark of a well-integrated software environment.

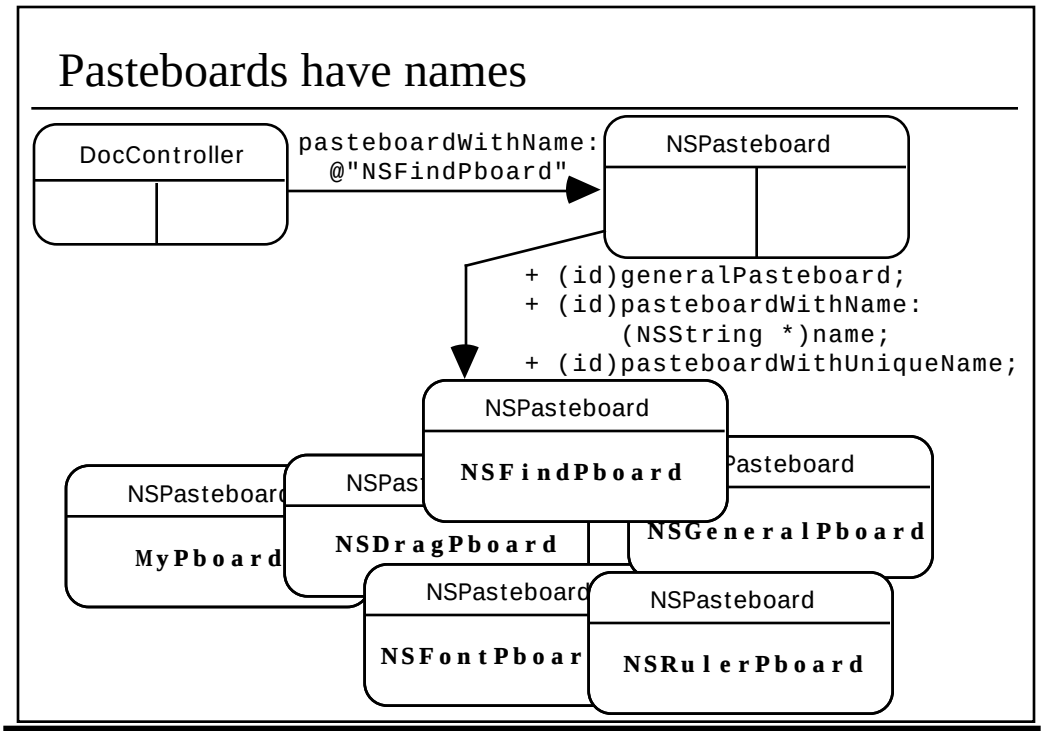
Most graphical environments provide a service for this, a holding place where one application can deposit data and another application pick it up. Called a pasteboard or clipboard, it facilitates data transfer not only between applications but between two objects within the same application. A TextView in one document can read data which has been copied from another TextView in another document in the same application. With an object-oriented interface in an environment that supports the general ability to objectify arbitrary data, pasteboards offer an open-ended, extensible technique for information sharing.



NSPasteboard—shared repository for data transfers

Whether the data is transferred between objects in the same application or two diverse applications, the interface is the same—the NSPasteboard object accesses a shared repository where writers and readers meet to exchange data. The writer, referred to as the pasteboard owner, deposits data on a pasteboard instance and moves on. The reader then accesses the pasteboard asynchronously, at some unspecified point in the future. By that time, the writer object may not even exist anymore. A user may likely have closed the source document or terminated the application. The case of moving data between two different applications and therefore two different address spaces requires that a third memory space get involved so the data persists in the mean time.

NSPasteboard provides access to a third address space—a pasteboard server process that is always running in the background. And it maintains an arbitrary number of individual pasteboards to distinguish among several concurrent on-going data transfers.

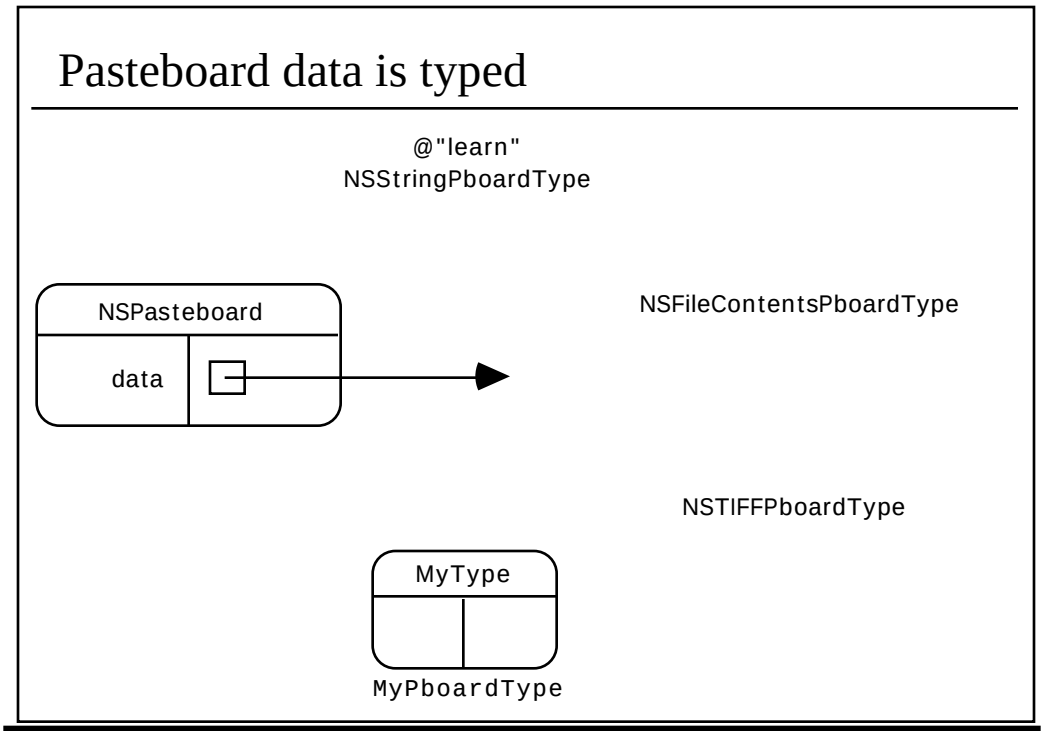


Pasteboards have names

The first step in any communication is to agree upon the meeting place. Pasteboard instances have names. Two objects begin an exchange with a common pasteboard by agreeing on its name. There are several standard pasteboards provided for well defined operations system-wide:

- NSGeneralPboard—for cut, copy and paste
- NSRulerPboard—for cut, copy and paste of rulers
- NSFontPboard—for cut, copy and paste of NSFont objects
- NSFindPboard—application-specific find panels can share a sought after text value. Find panels initialize from the pasteboard and update it with new user input
- NSDragPboard—for graphical drag and drop operations

You have the option of creating a special purpose pasteboard for exchanges that fall outside the standard set using **pasteboardWithName:**. In the manner of generating a guaranteed unique temporary filename, you can avoid contention by asking the pasteboard server for a unique name using **pasteboardWithUniqueName**.

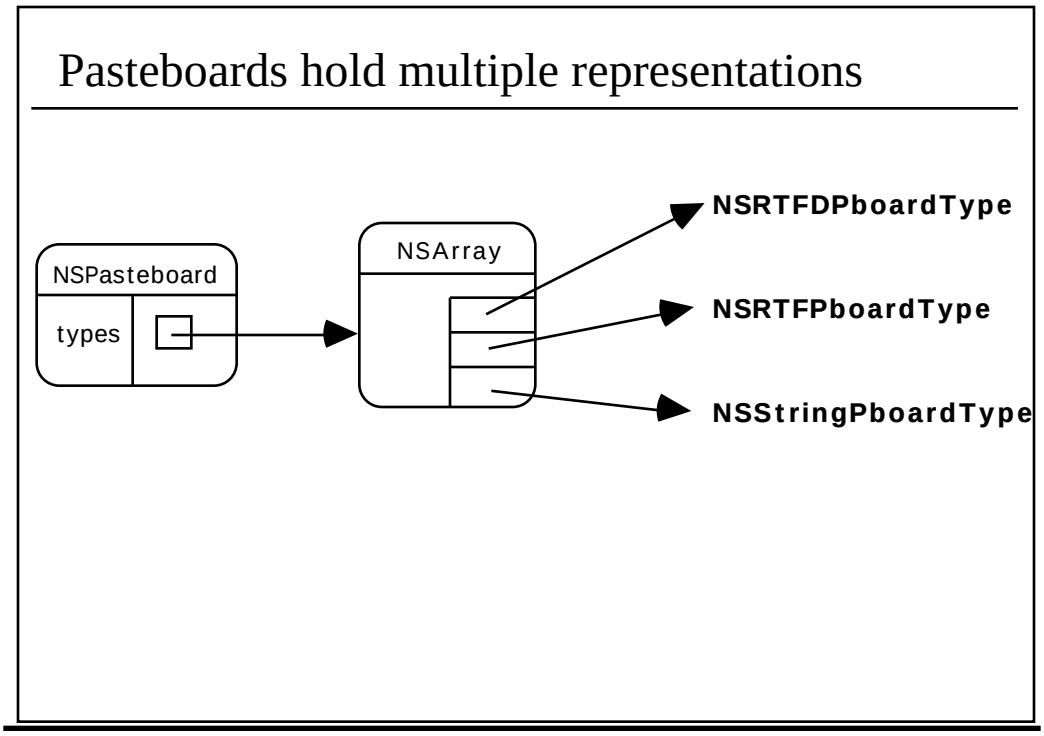


Pasteboard data is typed

Once a reader obtains the agreed upon named pasteboard instance, it must understand what type of data is available. The owner has to advertize what type of data it will write. Pasteboard data is generally an object instance whether a string, an arbitrarily complex object graph such as a dictionary of arrays or an instance of NSData, or an object wrapper for an arbitrary block of data. You can define your own special purpose data type. Any object written to and read from an NSPasteboard must conform to the NSCoding protocol. It must be able to archive and unarchive itself. Generally, pasteboard data consists of one or more value and/or collection classes.

Like the standard named pasteboards used for common operations, there are several commonly used standard system-defined data types:

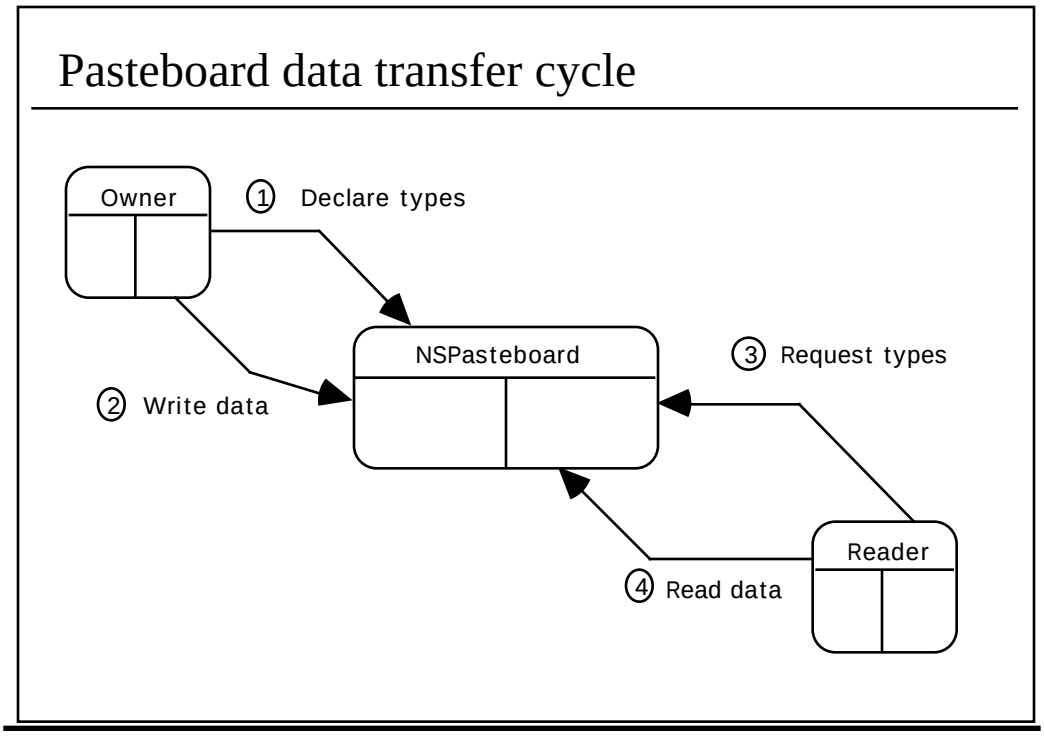
- NSStringPboardType, NSTabularTextPboardType
- NSFilenamesPboardType, NSFileContentsPboardType
- NSPostScriptPboardType
- NSTIFFPboardType
- NSRTFPboardType, NSRTFDPboardType
- NSFontPboardType
- NSRulerPboardType
- NSColorPboardType



Pasteboards hold multiple representations

Pasteboard operations are generally carried out between two anonymous and loosely-coupled applications. An editor, capable of handling rich text format, permits a user to select a region of text and copy it to the general pasteboard. Another application provides a simple NSTextView instance configured to only allow ASCII text. It permits the user to paste from the general pasteboard. Neither application has knowledge about the other and the kinds of data it can handle. It is impossible for the owner to determine which of several applications might show up as the next reader.

To maximize the potential for sharing, a pasteboard can hold multiple representations of the same data, each identified by a different pasteboard type string. Pasteboard owners should provide as many different representations as possible. In our example, the rich text editor might provide RTFD, RTF and NSString representations of the copied data. A reader, on the other hand, must search through the set of multiple representations to find the data type that best suits it capabilities. Generally, this means selecting the richest type available.



Pasteboard data transfer cycle

These are the objects that participate:

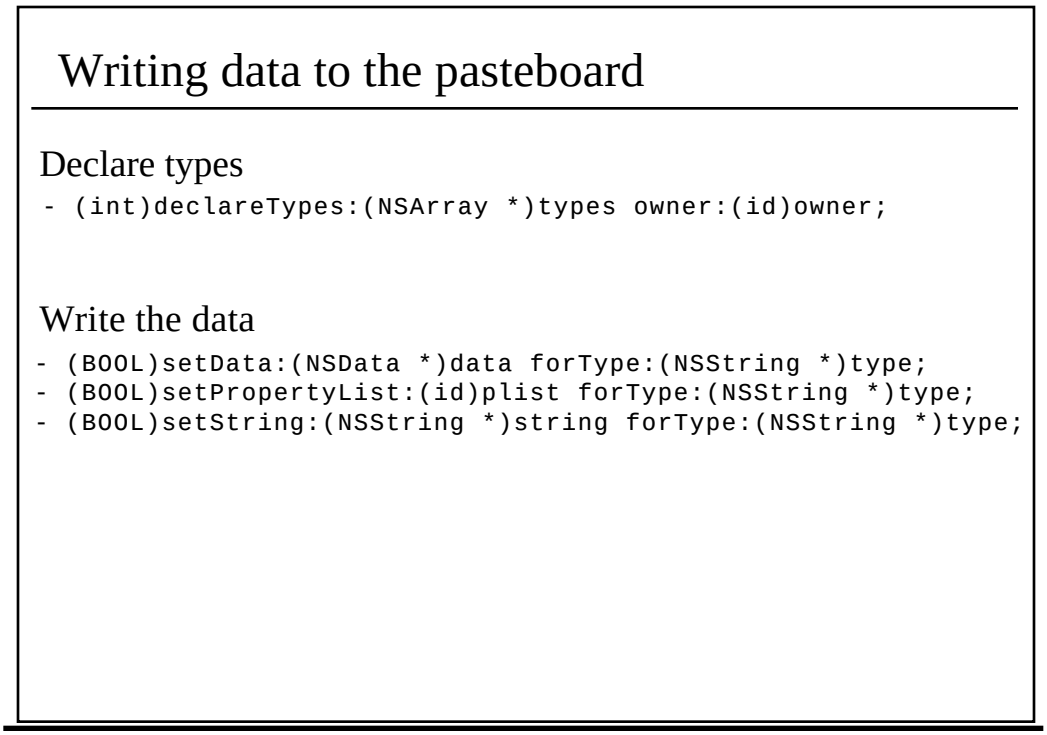
- Pasteboard instance
- Pasteboard typed data, generally multiple types at one time
- Owner
- Reader

These are the steps in the pasteboard data transfer cycle:

1. Owner—declares a set of data types it promises to write
2. Owner—writes each promised data type, one at a time
3. Reader—requests the set of data types provided by the owner
4. Reader—selects the most suitable type and reads from the pasteboard

What drives the cycle? User events. Through user interface controls such as menu items or keyboard equivalents, the user selects the object and triggers a write—commands like cut and copy. Asynchronously, at some indeterminate point in the future, the user selects a destination object and triggers a pasteboard read—commands like paste or link.

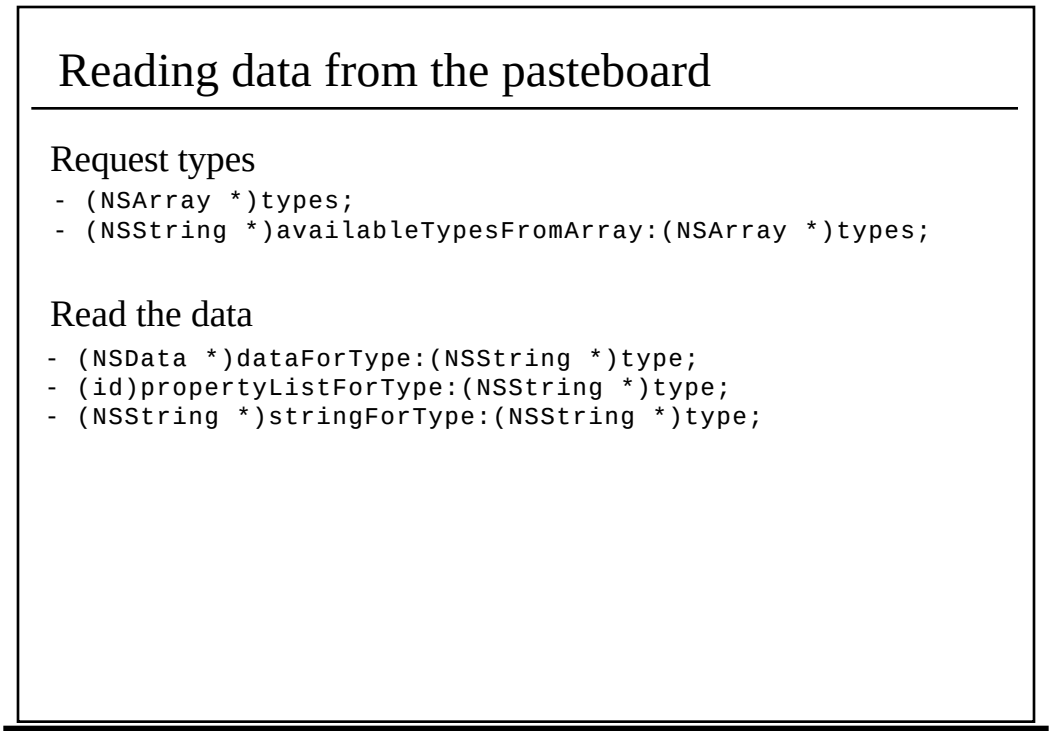
Drag and Drop always implies a pasteboard write and read, triggered by user generated mouse events and represented graphically.



Writing data to the pasteboard

The pasteboard writer, the owner, performs a write in two steps. First, the owner declares an array of pasteboard data types to be provided using **declareTypes:owner:**. Multiple types should be arranged in the array from richest to poorest since readers will enumerate through the array starting with index 0. The owner parameter may be nil and is only meaningful for lazy pasteboard owners covered later. Regardless, the pasteboard has a new owner and all previous data is cleared.

Next, the owner writes the data. Depending on the data type, the owner can use one of several messages. The owner sends one write message for each of the pasteboard data types it declared. **setPropertyList:** should be used for property list collections such as arrays and dictionaries.

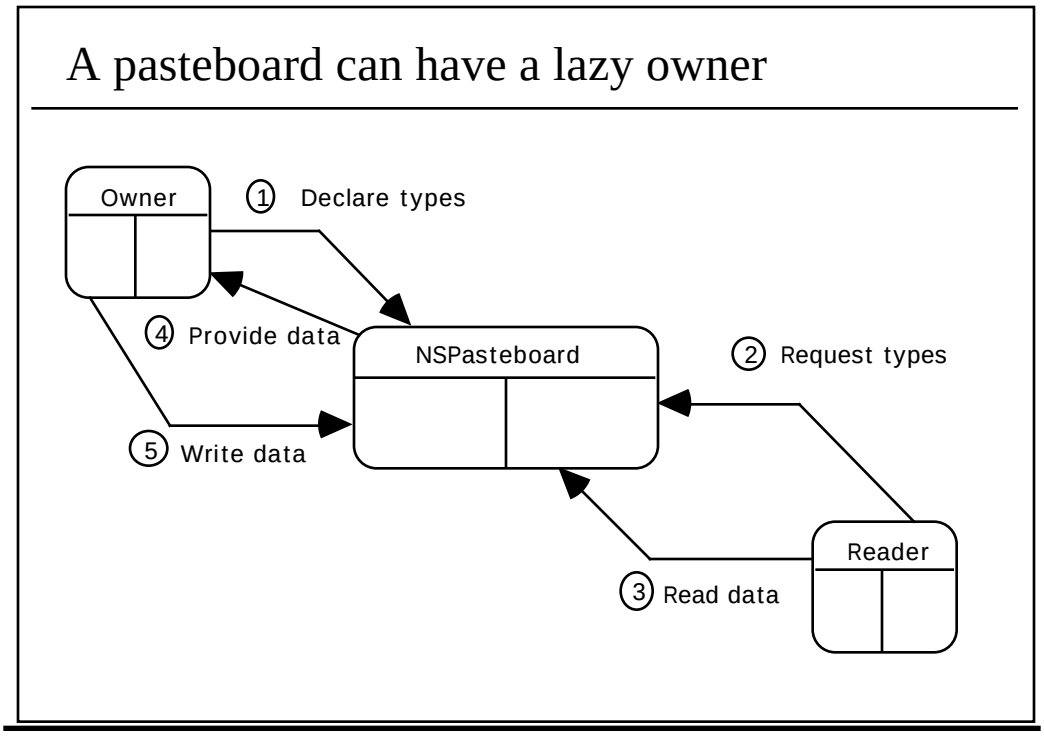


Reading data from the pasteboard

Like writing, reading involves two steps. The reader first requests the available types. Pasteboards can return an array of types for the reader to process, or can take a reader provided array of possible types and return the first that matches a type on the pasteboard. Because the reader wants to choose the richest possible type, both owners and readers should order their types array from richest to poorest. Where a set of types is completely diverse, none more rich that the other, the reader should simply order its array by preference. Sometimes the reader may require a more elaborate search algorithm in which case it can get the array and search itself.

Given a suitable type, the reader then reads the corresponding data, provided with methods matching the three writing methods used by the owner.

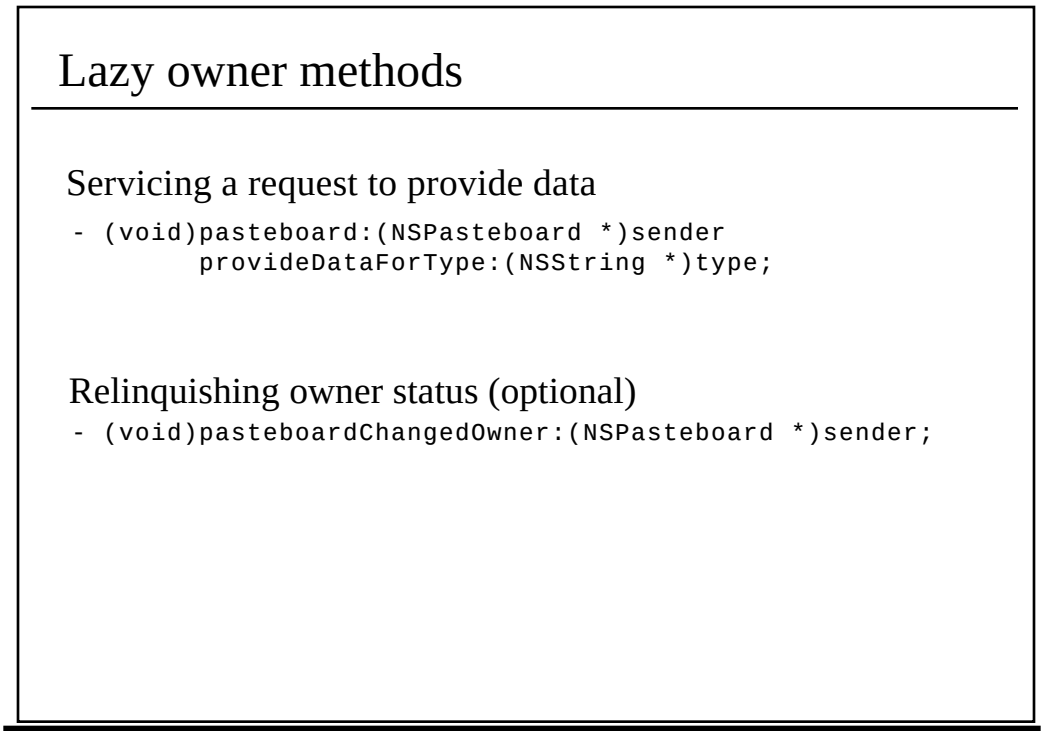
Objects instantiated by pasteboard reads are autoreleased much like unarchived objects.



A pasteboard can have a lazy owner

Owners capable of writing multiple representations can choose to be lazy—to wait until one representation is specifically requested by the reader. This enables the owner to defer the potentially time consuming and data intensive task of generating and writing each type. With only a single read operation, all but one of the representations will be ignored, the owner's efforts wasted.

This adds one new step at the end of the data transfer cycle. The pasteboard messages the owner, asking for the type the reader is now attempting to read. To qualify, the owner object must implement and respond to an additional message.



Lazy owner methods

Lazy owners must implement a method for the pasteboard to use when the reader attempts to read one of the promised data types. The parameters specify which pasteboard instance is active and which type is needed. If the owner deals with multiple pasteboards in a lazy fashion, it will have to map a pasteboard instance to the particular data instance selected in the past during the original pasteboard write. Once the owner provides the data, it can relinquish any data that was needed in support of the pending write.

A lazy owner object must not be released before this message unless the pasteboard owner changes.

Since laziness requires the owner object to maintain state for remembering the selected data—the user may select different data in the owner before performing the read, expecting the read to return the previously selected data—owners can implement a second method informing them that they need no longer be concerned. If and when it arrives, the owner can relinquish any saved state. The owner gets one or the other of these messages. A pasteboard's owner changes as soon as a new **declareTypes:owner:** message is sent to the same named pasteboard instance.

Pasteboards

Cut, copy and paste using NSGeneralPboard

Find panels using NSFindPboard

Interface Builder Palettes

Drag and Drop

Services

Pasteboard operations in other object
- NSTextView
- NSImage

Pasteboards

Pasteboard mechanics underlie many features. Some objects, such as NSTextView and NSImage, can write their contents to and read from a pasteboards directly.

Important ideas from this section

- Pasteboards provide a shared data repository through which two objects can transfer data.
- Pasteboard operations involve named pasteboards, typed data, often with multiple representations and two participating objects: the owner, also known as the writer, and the reader.
- Both owner and reader must inform the pasteboard the data types they support before writing or reading. The owner declares, the reader requests. The owner must write all the types it declares whereas the reader selects the most appropriate single type to read.
- Pasteboard owners can be lazy, deferring the write of any particular type until the reader requests it with a typed read operation.

Class featured in this section

NSPasteboard

REVIEW PASTEBOARDS

1. Why is pasteboard data typed?

2. Please describe the pasteboard data transfer cycle.

3. What does it mean to have a lazy pasteboard owner?

4. Assume you are currently a lazy pasteboard owner of a given pasteboard. You get the **dealloc** message. What, if any, are the implications?

EXERCISE 16.1 EXTENSIBLE, REUSABLE FIND PANEL

Most applications provide a find panel that supports string searching within a view. The frameworks do not include a generic reusable find panel component because the implementation and even the meaning of searching is specific to the first responder view. It is possible, however, to consider the problem generically and devise an approach that is reusable and extensible to a variety of contexts. The easiest place to start is with text view.

In this exercise, you study a simple find panel implementation. It incorporates a number of important aspects you have been covering—separate nib file component with File's Owner controller, first responder, target/action and string handling. Enhance the find panel to use the find pasteboard.

Objectives

After completing this exercise, you will be able to:

» Use pasteboards and types to transfer data between applications

» Describe a useful approach for implementing a reusable find panel component

Exercise

1. Make your own copy of the Find Panel project found in **ExerciseMaterials/Exercises/Provided**.

2. Build and run it. It is an editable text view with a find panel. Get a feel for how it works. The find panel options are found under the Edit menu.

3. Study and understand how the project is implemented. The project contains the main nib and a FindPanel subproject. Focus on the FindPanel object, **FindPanel.m**, the **FindPanel.nib** file and the **StringFinder.h** protocol. You can ignore the additional support pieces for now, such as StringStringFinder and TextViewStringFinder.
   Here is a simple description of the FindPanel's job:

   - FindPanel shows itself when asked
   - The user types in a string and presses Return or the "Next" button.
   - FindPanel examines the first responder. It asks, "is this object capable of searching?" If the object conforms to the StringFinder protocol, it is capable. The FindPanel uses the search message in the protocol to ask the object to find the user's string.
   - If found, the string is selected in the first responder object. If not, FindPanel displays the message, "Not Found".

4. Once you understand the basic construction of the FindPanel, consider how pasteboards fit in. The frameworks provide a global Find pasteboard. It allows a search string entered in one application to be seen by other applications. It is commonly the case that a user will want to search for the same thing in multiple places. Here is a simple description of how the Find pasteboard might be included in your FindPanel:

   - When FindPanel shows itself, it should fill in the search string text field with the string from the system-wide Find pasteboard—NSFindPboard. This requires a pasteboard read.
   - When the user enters a new string in the text field, FindPanel should propagate the string to the system-wide Find pasteboard. This requires a pasteboard write.

5. Implement pasteboard functionality in your FindPanel. Since a full-featured find panel might perform these reads and writes in multiple places, they should be implemented as separate methods. Encapsulate the pasteboard read and write operations in to separate methods and call them from the appropriate places in the code:

```
- (void)setTextFromPasteboard; (read)

- (void)setPasteboardFromText; (write)
```

6. Build and run your application. Run the Edit application concurrently. Verify that text entered in your find panel shows up in Edit's find panel. (To get each of the panel's to refresh themselves you may need to close and re-open them). Verify that text entered in Edit's find panel shows up in yours.

Enhancements

» Look at the find panel in Edit. It includes a “Previous” button and its menu items include “Previous” and “Enter selection”. Enhance your find panel to include these features.

» With a neatly packaged subproject and the StringFinder protocol, it is possible to add the FindPanel component to your Expense Report project. Exercise Materials provides a category on NSTableView that enhances it to conform to the StringFinder protocol. This enables a user to search for a string in a table view, row by row. In order to incorporate this in the Expense Report application, do the following:

Drag your FindPanel subproject into the your Expenses application.

Instantiate a FindPanel in the main nib and connect it to the Find submenu options. These are normally found under the Edit menu. Your Edit menu may not include these being a more simple default Edit menu provided when you created the nib file. You can delete the old Edit menu and add the full featured Edit menu from Interface Builder’s menu palette.

Copy **TableViewStringFinder.h** and **TableViewStringFinder.m** from **Provided** into your FindPanel subproject.

Build the application and test it.

» The FindPanel design assumes that the find controls—the text field and the push buttons—are on a panel which is loaded as a separate nib. Consider the Logger panel from an earlier exercise. What if you wanted to incorporate find controls on the same panel? This way, the Logger is self-sufficient, providing search capabilities whether or not the application provides a separate find panel. How could you adjust the design of the FindPanel object to deal with this case—hooking to controls on a pre-existing panel which is managed by another File’s Owner controller? The capability to do both would make the FindPanel object superior; it would be even more reusable.