NS_DEV_DOCFOR:objc_class:Pasteboard;, Pasteboard

Inherits From: Object

Declared In: appkit/Pasteboard.h

Class Description

Pasteboard objects transfer data to and from the pasteboard server, **pbs**. The server is shared by all running applications. It contains data that the user has cut or copied and may paste, as well as other data that one application wants to transfer to another. Pasteboard objects are an application's sole interface to the server and to all pasteboard operations.

Named Pasteboards

Data in the pasteboard server is associated with a name that indicates how it's to be used. Each set of named data is, in effect, a separate pasteboard, distinct from the others. An application keeps a separate Pasteboard object for each named pasteboard that it uses. There are five standard pasteboards in common use:

General pasteboard	The pasteboard that's used for ordinary cut, copy, and paste operations. It holds the contents of the last selection that's been cut or copied.
Font pasteboard	The pasteboard that holds font and character information and supports the Copy Font and Paste Font commands.
Ruler pasteboard	The pasteboard that holds information about paragraph formats in support of the Copy Ruler and Paste Ruler commands.
Find pasteboard	The pasteboard that holds information about the current state of the active application's Find panel. This information permits users to enter a search string into the Find panel, then switch to another application to conduct the search.
Drag pasteboard	The pasteboard that stores data to be manipulated as the result of a drag operation.

Each standard pasteboard is identified by a unique name designated by a global variable of type NXAtom:

NXGeneralPboard
NXFontPboard
NXRulerPboard
NXFindPboard
NXDragPboard

You can also create private pasteboards by asking for a Pasteboard object with any other name. The name of a private pasteboard can be passed to other applications to allow them to share the data it holds.

The Pasteboard class makes sure there's never more than one object for each named pasteboard. If you ask for a new object when one has already been created for the pasteboard, the existing one will be returned to you.

Data Types

Data can be placed in the pasteboard server in more than one representation. For example, an image might be provided both in Tag Image File Format (TIFF) and as encapsulated PostScript code (EPS). Multiple representations give pasting applications the option of choosing which data type to use. In general, an application taking data from the pasteboard should choose the richest representation it can handleÐrich text over plain ASCII, for example. An application putting data in the pasteboard should promise to supply it in as many data types as possible, so that as many applications as possible can make use of it.

Data types are identified by character strings containing a full type name. The following global variables (of type NXAtom) are string pointers for the standard NeXT pasteboard types. This list is not exhaustive; NEXTSTEP kits define some other data types that can only be read by kit objects and are not intended for general purpose data interchange.

Type	Description
NXAsciiPboardType	Plain ASCII text
NXPostScriptPboardType	Encapsulated PostScript code (EPS)
NXTIFFPboardType	Tag Image File Format (TIFF)
NXRTFPboardType	Rich Text Format (RTF)
NXSoundPboardType	The Sound object's pasteboard type
NXFilenamePboardType	ASCII text designating a file name
NXTabularTextPboardType	Tab-separated fields of ASCII text
NXFontPboardType	Font and character information
NXRulerPboardType	Paragraph formatting information
NXFileContentsPboardType	A representation of a file's contents
NXColorPboardType	NXColor data
NXSelectionPboardType	Describes a selection
NXDataLinkPboardType	Defines a link between documents

Types other than those listed can also be used. For example, your application may keep data in a private format that's richer than any of the types listed above. That format can also be used as a pasteboard type.

Reading and Writing Data

Typically, data is written to the pasteboard using **writeType:data:length:** and read using **readType:data:length:**. However, file contents and colors must be written using special pasteboard methods:

- Data of NXFileContentsPboardType, representing the contents of a named file, must be written using **writeFileContents:** and read using **readFileContentsType:toFile:**.
- NXColor data should be written using the **NXWriteColorToPasteboard()** function, and read using **NXReadColorFromPasteboard()**.

It's often convenient (and most memory-efficient) to prepare data for the pasteboard by writing data to a memory stream through functions such as **NXWrite()**, **NXPrintf()**, and **NXPutc()**. After the data has been written, the stream can be sent to the pasteboard server using **writeType:fromStream:**.

Similarly, you can get a memory stream for the data received from the pasteboard server via **readTypeToStream:** and use functions like **NXGetc()**, **NXRead()**, and **NXScanf()** to parse it. Objects can be archived to and from the pasteboard server using typed streams.

Errors

Except where errors are specifically mentioned in the method descriptions, any communications error with the pasteboard server raises an NX_pasteboardComm exception.

Instance Variables

id **owner**;

owner	The object responsible for putting data in the pasteboard.

Method Types

Creating and freeing a Pasteboard object
+ new
+ newName:
+ newUnique
- free
- freeGlobally

Getting data in different formats
+ newByFilteringFile:
+ newByFilteringData:ofType:
+ newByFilteringTypesInPasteboard:
+ typesFilterableTo:

Referring to a Pasteboard by name
+ newName:
- name

Writing data
- declareTypes:num:owner:
- addTypes:num:owner:
- writeType:data:length:
- writeType:fromStream:
- writeFileContents:

Discerning types
- types
- findAvailableTypeFrom:num:

Reading data
- changeCount
- readType:data:length:
- readTypeToStream:
- readFileContentsType:toFile:
- deallocatePasteboardData:length:

Class Methods

NS_DEV_DOCFOR:objc_method:[Pasteboard-alloc];, alloc

Generates an error message. This method cannot be used to create Pasteboard instances. Use **new** or **newName:** instead.

See also: + **new**, + **newName:**

NS_DEV_DOCFOR:objc_method:[Pasteboard-allocFromZone:];, allocFromZone:

Generates an error message. This method cannot be used to create Pasteboard instances. Use **new** or **newName:** instead.

See also: + **new**, + **newName:**

NS_DEV_DOCFOR:objc_method:[Pasteboard-new];, new

+ **new**

Returns the Pasteboard object for the selection pasteboard by passing NXGeneralPboard to the **newName:** method.

NS_DEV_DOCFOR:objc_method:[Pasteboard-newByFilteringData:ofType:];, newByFilteringData:ofType:

+ **newByFilteringData:**(NXData *)*data* **ofType:**(const char *)*type*

Creates and returns a new Pasteboard with a unique name that has, declared within it, data of every type that can be provided by the available filter services from *data*. The returned pasteboard also declares data of the supplied type *type*. No filter service is invoked until the data is actually requested, so invoking this method is reasonably inexpensive.

NS_DEV_DOCFOR:objc_method:[Pasteboard-newByFilteringFile:];, newByFilteringFile:

+ **newByFilteringFile:**(const char *)*filename*

Creates and returns a new Pasteboard with a unique name that has, declared within it, data of every type that can be provided by the available filter services from the file *filename*. No filter service is invoked until the data is actually requested, so invoking this method is reasonably inexpensive.

NS_DEV_DOCFOR:objc_method:[Pasteboard-newByFilteringTypesInPasteboard:];, newByFilteringTypesInPasteboard:

+ **newByFilteringTypesInPasteboard:**(Pasteboard *)*pboard*

Creates and returns a new Pasteboard with a unique name that has, declared within it, data of every type that can be provided by the available filter services from the data on pasteboard *pboard*. This process can be thought of as expanding the pasteboard, since the new pasteboard generally will contain more representations of the data on *pboard*.

This method returns *pboard* if *pboard* is a pasteboard returned by one of the **newByFiltering...** methods, so a pasteboard can't be expanded multiple times. This method only returns the original types and the types that can be created as a result of a single filter; the pasteboard will not have defined types that are the result of translation by multiple filters.

No filter service is invoked until the data is actually requested, so invoking this method is reasonably inexpensive.

NS_DEV_DOCFOR:objc_method:[Pasteboard-newName:];, newName:

+ **newName:**(const char *)*name*

Returns the Pasteboard object for the *name* pasteboard. A new object is created only if the application doesn't yet have a Pasteboard object for the specified name; otherwise, the existing one is returned. To get a standard pasteboard, *name* should be one of the following variables:

NXGeneralPboard
NXFontPboard
NXRulerPboard
NXFindPboard
NXDragPboard

Other names can be assigned to create private pasteboards for other purposes.

NS_DEV_DOCFOR:objc_method:[Pasteboard-newUnique];, newUnique

+ **newUnique**

Creates and returns a new Pasteboard with a name that is guaranteed to be unique with respect to other Pasteboards on the system. This method is useful for applications that implement their own interprocess communication using pasteboards.

NS_DEV_DOCFOR:objc_method:[Pasteboard-typesFilterableTo:];, typesFilterableTo:

+ (NXAtom *)**typesFilterableTo:**(const char *)*type*

Returns a null-terminated array of NXAtoms indicating the types that data of type *type* can be converted to by available filter services. The array contains the original type. The caller is responsible for freeing the returned array.

Instance Methods

NS_DEV_DOCFOR:objc_method:[Pasteboard-addTypes:num:owner:];, addTypes:num:owner:

- (int)**addTypes:**(const char *const *)*newTypes*
num:(int)*numTypes*
owner:*newOwner*

Adds additional types to the pasteboard. This method can be useful when multiple entities (such as a combination of application and library methods) contribute data for a single copy command. It should only be invoked after a **declareTypes:num:owner:** message has been sent for the same data. The owner for the new types may be different from the owner(s) of the previously declared data.

Returns the pasteboard's change count, or 0 in case of an error.

See also: - **changeCount**

NS_DEV_DOCFOR:objc_method:[Pasteboard-changeCount];, changeCount

- (int)**changeCount**

Returns the current change count for the pasteboard. The change count is a system-wide variable that increments every time the contents of the pasteboard changes (a new owner is declared). By examining the change count, an application can determine whether the current data in the pasteboard is the same as the data it last received.

An independent change count is maintained for each named pasteboard.

See also: - **declareTypes:num:owner:**

NS_DEV_DOCFOR:objc_method:[Pasteboard-deallocatePasteboardData:length:];, deallocatePasteboardData:length:

- **deallocatePasteboardData:**(char *)*data* **length:**(int)*numBytes*

This method should be used to deallocate the memory returned by **readType:data:length:**. Returns **self** if the memory is successfully deallocated, otherwise raises an NX_appkitVMError exception.

NS_DEV_DOCFOR:objc_method:[Pasteboard-declareTypes:num:owner:];,

declareTypes:num:owner:

- (int)**declareTypes:**(const char * const *)*newTypes*
num:(int)*numTypes*
owner:*newOwner*

Prepares the pasteboard for a change in its contents by declaring the new types of data it will contain and a new owner. This is the first step in responding to a user's copy or cut command and must precede the messages that actually write the data. A **declareTypes:num:owner:** message is tantamount to changing the contents of the pasteboard. It invalidates the current contents of the pasteboard and increments its change count.

numTypes is the number of types the new contents of the pasteboard may assume, and *newTypes* is an array of null-terminated strings that name those types. The types should be ordered according to the preference of the source application, with the most preferred type coming first (typically, the richest representation is first).

The *newOwner* is the object responsible for writing data to the pasteboard in all the types listed in *newTypes*. Data is written using the **writeType:data:length:** method. You can write the data immediately after declaring the types, or wait until it's required for a paste operation. If you wait, the owner will receive a **pasteboard:provideData:** message requesting the data in a particular type when it's needed. You might choose to write data immediately for the most preferred type, but wait for the others to see whether they'll be requested.

The *newOwner* can be NULL if data is provided for all types immediately. Otherwise, the owner should be an object that won't be freed. It should not, for example, be the View that displays the data if that View is in a window that might be closed.

Returns the pasteboard's change count.

See also: **- writeType:data:length:**, **- pasteboard:provideData:**, **-_addTypes:num:owner:**, **-_changeCount**

NS_DEV_DOCFOR:objc_method:[Pasteboard-findAvailableTypeFrom:num:];, findAvailableTypeFrom:num:

- (const char *)**findAvailableTypeFrom:**(const char *const *)*types*
num:(int)*numTypes*

Scans the types defined by *types* (which is an array of size *numTypes*) and returns the first type that matches a type declared on the pasteboard. A **types** or **findAvailableTypeFrom:num:** message should be sent before reading any data from the pasteboard.

NS_DEV_DOCFOR:objc_method:[Pasteboard-free];, free

- **free**

Frees the Pasteboard object. A Pasteboard object should not be freed if there's a chance that the application might want to use the named pasteboard again; standard pasteboards generally should not be freed at all.

NS_DEV_DOCFOR:objc_method:[Pasteboard-freeGlobally];, freeGlobally

- **freeGlobally**

Frees the Pasteboard object and the domain for its name within the pasteboard server. This means that no other application will be able to use the named pasteboard. A temporary, privately named pasteboard can be freed when it's no longer needed, but a standard pasteboard should never be freed globally.

NS_DEV_DOCFOR:objc_method:[Pasteboard-name];, name

- (NXAtom)**name**

Returns the name of the Pasteboard object.

See also: + **newName:**

NS_DEV_DOCFOR:objc_method:[Pasteboard-readFileContentsType:toFile:];, readFileContentsType:toFile:

- (char *)**readFileContentsType:**(const char *)*type* **toFile:**(const char *)*filename*

Reads data representing a file's contents from the pasteboard, and writes it to the file *filename*. Data of any file contents type should only be read using this method. *type* should generally be specified; if *type* is NULL, a type based on *filename*'s extention (as returned by **NXCreateFileContentsPboardType()**) is substituted. If data matching *type* isn't found on the pasteboard, data of type NXFileContentsPboardType is requested. Returns an allocated string with the name of the file that the data was actually written to.

You should send the **types** or **findAvailableTypeFrom:num:** message before reading any data from the pasteboard.

See also: - **writeFileContents:**

NS_DEV_DOCFOR:objc_method:[Pasteboard-readType:data:length:];, readType:data:length:

- **readType:**(const char *)*dataType*
data:(char **)*theData*
length:(int *)*numBytes*

Reads the *dataType* representation of the current contents of the pasteboard. *dataType* should be one of the types returned by the **types** method. The data is read by setting the pointer referred to by *theData* to the address of the data, and setting the integer referred to by *numBytes* to the length of the data in bytes.

If the data is successfully read, this method returns **self**. It returns **nil** if the contents of the pasteboard have changed (if the change count has been incremented by a **declareTypes:num:owner** message) since they were last checked with the **types** method. It also returns **nil** if the pasteboard server can't supply the data in timeÐfor example, if the pasteboard's owner is slow in responding to a **pasteboard:provideData:** message and the interprocess communication times out. All other errors raise an NX_pasteboardComm exception.

If **nil** is returned, the application should put up a panel informing the user that it was unable to carry out the paste operation. It shouldn't attempt to use the pointer referred to by *theData*, as it won't be valid.

The memory for the data that this method provides must eventually be freed by the caller using **deallocatePasteboardData:length:**; you should *not* attempt to free the returned memory using **vm_deallocate()** or **free()**. For example:

```
char *data;
int   length;

if ([myPasteboard readType:NXAsciiPboardType
                          data:&data length:&length])
{
    /* Use the data here, keeping it for as long as necessary */
    [myPasteboard deallocatePasteboardData:data length:length];
}
```

You should send the **types** or **findAvailableTypeFrom:num:** message before reading any data from the pasteboard.

See also: - **readTypeToStream:**

NS_DEV_DOCFOR:objc_method:[Pasteboard-readTypeToStream:];, readTypeToStream:

- (NXStream *)**readTypeToStream:**(const char *)*dataType*

Reads data from the pasteboard to a stream. This method uses the **readType:data:length:** method to read data of the type *dataType* from the pasteboard. It then opens a stream on the data, and returns the stream, or NULL if there is an error. Data returned with this method must eventually be freed using

```
NXCloseMemory(theStream, NX_FREEBUFFER)
```

You should *not* free the data using **deallocatePasteboardData:length:**. You should send the **types** or **findAvailableTypeFrom:num:** message before reading any data from the pasteboard.

See also: - **writeType:fromStream:**

NS_DEV_DOCFOR:objc_method:[Pasteboard-types];, types

- (const NXAtom *)**types**

Returns the list of the types that were declared for the current contents of the pasteboard. The list is an array of character pointers holding the type names, with the last pointer being NULL. Each of the pointers is of type NXAtom.

Types are listed in the same order that they were declared. A **types** or **findAvailableTypeFrom:num:** message should be sent before reading any data from the pasteboard.

See also: - **declareTypes:num:owner:**, - **readType:data:length:**, - **findAvailableTypeFrom:num:**, **NXUniqueString()**

NS_DEV_DOCFOR:objc_method:[Pasteboard-writeFileContents:];, writeFileContents:

- (BOOL)**writeFileContents:**(const char *)*filename*

Writes the contents of the file *filename* to the pasteboard, and declares the data to be of type NXFileContentsPboardType and also of a type appropriate for the file's extention (as returned by **NXCreateFileContentsPboardType()** when passed the files extention), if it has one. Returns YES if the data from *filename* was successfully written to the pasteboard, and NO otherwise.

See also: - **readFileContentsType:toFile:**

NS_DEV_DOCFOR:objc_method:[Pasteboard-writeType:data:length:];, writeType:data:length:

- **writeType:**(const char *)*dataType*
data:(const char *)*theData*
length:(int)*numBytes*

Writes data to the pasteboard server. *dataType* gives the type of data being written; it must be a type that was declared in the previous **declareTypes:num:owner:** message. *theData* points to the data to be sent to the pasteboard server, and *numBytes* is the length of the data in bytes.

A separate **writeType:data:length:** message is required for each data representation that's written to the server.

This method returns **self** if the data is successfully written. It returns **nil** if an object in another application has become the owner of the pasteboard. Any other error raises an NX_pasteboardComm exception.

See also: - **declareTypes:num:owner:**

NS_DEV_DOCFOR:objc_method:[Pasteboard-writeType:fromStream:];, writeType:fromStream:

- **writeType:**(const char *)*dataType* **fromStream:**(NXStream *)*stream*

Writes the type *dataType* to the pasteboard from the supplied stream *stream*. The stream must be readable. If the stream is seekable, it is seeked back to the beginning before the data is read; otherwise, data is read from the current position. In either case, all data to the end of the stream is read.

This method returns **self** if the data is successfully written. It returns **nil** if an object in another application has become the owner of the pasteboard. Any other error raises an NX_pasteboardComm exception.

See also: - **writeType:data:length:**

Method Implemented By The Owner

NS_DEV_DOCFOR:objc_method:[Pasteboard-pasteboardChangedOwner:];, pasteboardChangedOwner:

- **pasteboardChangedOwner:***sender*

Notifies a prior owner of the *sender* Pasteboard (and owners of representations on the pasteboard) that the pasteboard has changed owners. This method is optional and need only be implemented by pasteboard owners that need to know when they have lost ownership. The owner is not able to read the contents of the pasteboard when responding to this method. The owner should be prepared to receive this method at any time, even from within the **declareTypes:num:owner:** used to declare ownership.

NS_DEV_DOCFOR:objc_method:[Pasteboard-pasteboard:provideData:];, pasteboard:provideData:

- **pasteboard:***sender* **provideData:**(NXAtom)*type*

Implemented by the owner (previously declared in a **declareTypes:num:owner:** message) to provide promised data. The owner receives a **pasteboard:provideData:** message from the *sender* Pasteboard when the data is required for a paste operation; *type* gives the type of data being requested. The requested data should be written to *sender* using the **writeType:data:length:** method.

pasteboard:provideData: messages may also be sent to the owner when the application is shut down through Application's **terminate:** method. This is the method that's invoked in response to a Quit command. Thus the user can copy something to the pasteboard, quit the application, and still paste the data that was copied.

A **pasteboard:provideData:** message is sent only if *type* data hasn't already been supplied. Instead of writing all data types when the cut or copy operation is done, an application can choose to implement this method to provide the data for certain types only when they're requested.

If an application writes data to the pasteboard in the richest, and therefore most preferred, type at the time of a cut or copy operation, its **pasteboard:provideData:** method can simply read that data from the pasteboard, convert it to the requested *type*, and write it back to the pasteboard as the new type.

See also: - **declareTypes:num:owner:**, - **writeType:data:length:**