NS_DEV_DOCFOR:objc_class:Speaker;, Speaker

Inherits From: Object

Declared In: appkit/Speaker.h

Class Description

The Speaker class, with the Listener class, puts an Objective C interface on Mach messaging. Mach messages are the way that applications communicate with each other; they're how remote procedure calls (RPCs) are implemented in the Mach operating system.

A remote message is initiated by sending a Speaker instance the very same Objective C message you want delivered to the remote application. The Speaker translates the message into the correct Mach message format and dispatches it to the receiving application's port. A Listener in the receiving application reads the message from the port queue and translates in back into an Objective C message, which it tries to delegate to another object.

If the Speaker expects information back from the Listener, it will wait to receive a reply.

Every application must have at least one Speaker and one Listener, if for no other reason but to communicate with the Workspace Manager. If you don't create a Speaker in start-up code and register it as the application's Speaker (with the **setAppSpeaker:** method), the Application object, when it receives a **run** message, will create one for you.

For a general discussion of the Speaker-Listener interaction, see the Listener class. The descriptions here add Speaker-specific information, but don't repeat any of the basic information presented there. In particular, the discussion here doesn't explain the structure of remote messages or the distinction between input and output argument types.

Sending Remote Messages

Before sending a remote message, it's necessary only to provide variables where output informationÐinformation returned to the Speaker by the receiving applicationÐcan be returned by reference, and to tell the Speaker which port to send the message to.

The example below shows a typical use of the Speaker class:

```
int     msgDelivered, fileOpened;
id      mySpeaker = [[Speaker alloc] init];
port_t  thePort = NXPortFromName("SomeApp", NULL);
                            /* Gets the public port for SomeApp */

if (thePort != PORT_NULL) {
    [mySpeaker setSendPort:thePort];
                            /* Sets the Speaker to send its
                             * next message to SomeApp's port */
    msgDelivered = [mySpeaker openFile:"/usr/foo" ok:&fileOpened];
                            /* Sends the message, here a message
                             * to open the "/usr/foo" file. */
    if (msgDelivered == 0) {
        if (fileOpened == YES)
            . . .
        else
            . . .
    }
}
. . .
```

```
[mySpeaker free];           /* Frees the Speaker
                             * when it's no longer needed. */
port_deallocate(task_self(), thePort);
                            /* Frees the application's
                             * send rights to the port. */
```

The **NXPortFromName()** function returns the port registered with the network name server under the name passed in its first argument. The second argument names the host machine; when it's NULL, as in the example above, the local host is assumed.

To find the port of the Workspace Manager, the constant NX_WORKSPACEREQUEST can be passed as the first argument to **NXPortFromName()**. For example:

```
port_t  workspacePort;
workspacePort = NXPortFromName(NX_WORKSPACEREQUEST, NULL);
```

A Speaker can be dedicated to sending remote messages to a single application, in which case its destination port may need to be set only once. Or a single Speaker can be used to send messages to any number of applications, simply by resetting its port.

It's important to reset the destination port of the Speaker registered as the **appSpeaker** before each remote message. The Application Kit uses the **appSpeaker** to keep in contact with the Workspace Manager and so may reset its port behind your application's back.

Return Values

Each method that initiates a remote message returns an **int** that indicates whether the message was successfully transmitted or not.

- If the message couldn't be delivered to the receiving application, the return value will be one of the Mach error codes defined in the **mach/message.h**.
- If the message was delivered, but the Listener didn't recognize it or couldn't delegate it to a responsible object, the return value is -1.
- If the message was successfully delivered, recognized, and delegated, 0 is returned.

A Mach error code is also returned if the Speaker times out while waiting for a return message.

Copying Output Data

The validity of all output arguments is guaranteed until the next remote message is sent. Then the memory allocated for a character string or a byte array will be freed automatically. If you want to save an output string or an array, you must copy it. When the amount of data is large, you can use the **NXCopyOutputData()** function to take advantage of the out-of-line data feature of Mach messaging. This function is passed the index of the output argument to be copied (the combination of a pointer and an integer for a byte array counts as a single argument) and returns a pointer to an area obtained through the **vm_allocate()** function. This pointer must be freed with **vm_deallocate()**, rather than **free()**. Note that the size of the area allocated is rounded up to the next page boundary, and so will be at least one page. Consequently, it is more efficient to **malloc()** and copy amounts up to about half the page size.

Note: The application is responsible for deallocating all ports received when they're no longer needed.

Instance Variables

port_t **sendPort**;
port_t **replyPort**;
int **sendTimeout**;
int **replyTimeout**;
id **delegate**;

sendPort	The port to which the Speaker sends remote messages.
replyPort	The port where the Speaker receives return messages from the Listener of the remote application.
sendTimeout	How long the Speaker will wait for a remote message to be delivered at the port of the receiving application.
replyTimeout	How long the Speaker will wait, after a remote message is delivered, to receive a return message from the other application.
delegate	The Speaker's delegate, which is generally unused.

Method Types

Initializing a new Speaker instance	- init
Freeing a Speaker	- free
Setting up a Speaker	- setSendTimeout: - sendTimeout - setReplyTimeout: - replyTimeout
Managing the ports	- setSendPort: - sendPort - setReplyPort: - replyPort
Standard remote methods	- openFile:ok: - openTempFile:ok:
Providing for program control	- msgCalc: - msgCopyAsType:ok: - msgCutAsType:ok: - msgDirectory:ok: - msgFile:ok: - msgPaste: - msgPosition:posType:ok: - msgPrint:ok: - msgQuit: - msgSelection:length:asType:ok: - msgSetPosition:posType:andSelect:ok: - msgVersion:ok:
Sending remote messages	- performRemoteMethod: - performRemoteMethod:with:length: - selectorRPC:paramTypes:... - sendOpenFileMsg:ok:andDeactivateSelf: - sendOpenTempFileMsg:ok:andDeactivateSelf:
Assigning a delegate	- setDelegate: - delegate
Archiving	- read: - write:

Instance Methods

NS_DEV_DOCFOR:objc_method:[Speaker-delegate];, delegate

- **delegate**

Returns the Speaker's delegate.

See also: - **setDelegate:**

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

- **free**

Frees the memory occupied by the Speaker object, but does not deallocate its ports.

NS_DEV_DOCFOR:objc_method:[Speaker-init];, init

- **init**

Initializes a newly allocated Speaker instance. The new object's **sendTimeout** and **replyTimeout** are both set to 30,000 milliseconds, its **sendPort** and **replyPort** are both PORT_NULL, and its delegate is **nil**. Returns **self**.

NS_DEV_DOCFOR:objc_method:[Speaker-msgCalc:];, msgCalc:

- (int)**msgCalc:**(int *)*flag*

Sends a remote message asking the receiving application to perform any calculations necessary to update its current window. *flag* points to an integer that will be set to YES if the calculations will be performed, and to NO if they won't.

NS_DEV_DOCFOR:objc_method:[Speaker-msgCopyAsType:ok:];, msgCopyAsType:ok:

- (int)**msgCopyAsType:**(const char *)*aType* **ok:**(int *)*flag*

Sends a remote message asking the receiving application to copy its current selection to the pasteboard as *aType* data. *flag* is the address of an integer that will be set to YES if the selection is copied, and to NO if it isn't.

NS_DEV_DOCFOR:objc_method:[Speaker-msgCutAsType:ok:];, msgCutAsType:ok:

- (int)**msgCutAsType:**(const char *)*aType* **ok:**(int *)*flag*

Sends a remote message requesting the receiving application to delete the current selection and put it in the pasteboard as *aType* data. *flag* points to an integer that will be set to YES if the request is carried out, and to NO if it isn't.

NS_DEV_DOCFOR:objc_method:[Speaker-msgDirectory:ok:];, msgDirectory:ok:

- (int)**msgDirectory:**(char *const *)*fullPath* **ok:**(int *)*flag*

Sends a remote message asking the receiving application for its current directory. See the Listener class for information on the two arguments.

See also: - **msgDirectory:ok:** (Listener)

NS_DEV_DOCFOR:objc_method:[Speaker-msgFile:ok:];, msgFile:ok:

- (int)**msgFile:**(char *const *)*fullPath* **ok:**(int *)*flag*

Sends a remote message asking the receiving application for its current document (the file displayed in the main window). See the Listener class for information on the two arguments.

See also: - **msgFile:ok:** (Listener)

NS_DEV_DOCFOR:objc_method:[Speaker-msgPaste:];, msgPaste:

- (int)**msgPaste:**(int *)*flag*

Sends a remote message asking the receiving application to replace its current selection with the contents of the pasteboard, just as if the user had chosen the Paste command in the Edit menu. *flag* is the address of an integer that will be set to YES if the receiving application will carry out the request, and to NO if it won't.

NS_DEV_DOCFOR:objc_method:[Speaker-msgPosition:posType:ok:];, msgPosition:posType:ok:

- (int)**msgPosition:**(char *const *)*aString*
posType:(int *)*anInt*
ok:(int *)*flag*

Sends a remote message asking the receiving application for information about its current selection. See the Listener class for information on the three arguments.

See also: - **msgPosition:posType:ok:** (Listener)

NS_DEV_DOCFOR:objc_method:[Speaker-msgPrint:ok:];, msgPrint:ok:

- (int)**msgPrint:**(const char *)*fullPath* **ok:**(int *)*flag*

Sends a remote message asking the receiving application to print the *fullPath* file, then close it. *flag* points to an integer that will be set to YES if the file will be printed, and to NO if it won't.

NS_DEV_DOCFOR:objc_method:[Speaker-msgQuit:];, msgQuit:

- (int)**msgQuit:**(int *)*flag*

Sends a remote message requesting the receiving application to quit. *flag* points to an integer that will be set to YES if the receiving application quits, and to NO if it doesn't.

NS_DEV_DOCFOR:objc_method:[Speaker-msgSelection:length:asType:ok:];, msgSelection:length:asType:ok:

- (int)**msgSelection:**(char *const *)*bytes*
length:(int *)*numBytes*
asType:(const char *)*aType*
ok:(int *)*flag*

Sends a remote message asking the receiving application to provide its current selection as *aType* data. See the Listener class for information on the four arguments.

See also: - **msgSelection:length:asType:ok:** (Listener)

NS_DEV_DOCFOR:objc_method:[Speaker-msgSetPosition:posType:andSelect:ok:];, msgSetPosition:posType:andSelect:ok:

- (int)**msgSetPosition:**(const char *)*aString*
posType:(int)*anInt*
andSelect:(int)*sflag*

ok:(int *)*flag*

Sends a remote message asking the receiving application to scroll its current document (the one displayed in the main window) so that the portion represented by *aString* is visible. See the Listener class for information on permitted argument values.

See also: - **msgSetPosition:posType:andSelect:ok:** (Listener)

NS_DEV_DOCFOR:objc_method:[Speaker-msgVersion:ok:];, msgVersion:ok:

- (int)**msgVersion:**(char *const *)*aString* **ok:**(int *)*flag*

Sends a remote message asking the receiving application for its current version. See the Listener class for information on the arguments.

See also: - **msgVersion:ok:** (Listener)

NS_DEV_DOCFOR:objc_method:[Speaker-openFile:ok:];, openFile:ok:

- (int)**openFile:**(const char *)*fullPath* **ok:**(int *)*flag*

Sends a remote message requesting another application to open the *fullPath* file. Before the message is sent, the sending application is deactivated to allow the application that will open the file to become the active application.

If the Workspace Manager is sent this message, it will find an appropriate application to open the file based on the file name extension. It will launch that application if necessary.

flag is the address of an integer that the receiving application will set to YES if it opens the file, and to NO if it doesn't.

See also: - **openFile:ok:** (Application)

NS_DEV_DOCFOR:objc_method:[Speaker-openTempFile:ok:];, openTempFile:ok:

- (int)**openTempFile:**(const char *)*fullPath* **ok:**(int *)*flag*

Sends a remote message requesting another application to open a temporary file. The file is specified by an absolute pathname, *fullPath*. Before the message is sent, the sending application is deactivated to allow the application that will open the file to become the active application.

Using this method instead of **openFile:ok:** lets the receiving application know that it should delete the file when it no longer needs it.

See also: - **openTempFile:ok:** (Application)

NS_DEV_DOCFOR:objc_method:[Speaker-performRemoteMethod:];, performRemoteMethod:

- (int)**performRemoteMethod:**(const char *)*methodName*

Sends a remote message to perform the *methodName* method. The method must be one that takes no arguments. **performRemoteMethod:** is analogous to Object's **perform:** method in that it permits you to send an arbitrary message.

This method has the same return values as other methods that send remote messages: 0 on success, a Mach error code if the message couldn't be delivered, and -1 if it was delivered but wasn't understood or couldn't be delegated.

See also: -**_selectorRPC:paramTypes:**

NS_DEV_DOCFOR:objc_method:[Speaker-performRemoteMethod:with:length:];,

performRemoteMethod:with:length:

- (int)**performRemoteMethod:**(const char *)*methodName*
with:(const char *)*data*
length:(int)*numBytes*

Sends a remote message to perform the *methodName* method and passes it the *data* byte array containing *numBytes* of data. This method is similar to Object's **perform:with:** method in that it permits you to send an arbitrary message with one argument.

performRemoteMethod:with:length: has the same return values as other methods that send remote messages: 0 on success, a Mach error code if the message couldn't be delivered, and -1 if it was delivered but wasn't understood or couldn't be delegated.

See also: **-_selectorRPC:paramTypes:**

NS_DEV_DOCFOR:objc_method:[Speaker-read:];, read:

- **read:**(NXTypedStream *)*stream*

Reads the Speaker from the typed stream *stream*. The Speaker's send-port and reply-port will both be PORT_NULL. Returns **self**.

See also: **- write:**

NS_DEV_DOCFOR:objc_method:[Speaker-replyPort];, replyPort

- (port_t)**replyPort**

Returns the port where the Speaker expects to receive return messages. If this method returns PORT_NULL, the default, the Speaker will use the port returned by Application's **replyPort** method.

See also: **- replyPort** (Application), **- setReplyPort:**

NS_DEV_DOCFOR:objc_method:[Speaker-replyTimeout];, replyTimeout

- (int)**replyTimeout**

Returns how many milliseconds the Speaker will wait, after delivering a remote message to another application, for a return message to arrive back from the other application.

See also: **- setReplyTimeout:**

NS_DEV_DOCFOR:objc_method:[Speaker-selectorRPC:paramTypes:];, selectorRPC:paramTypes:

- (int)**selectorRPC:**(const char *)*methodName*
paramTypes:(char *)*params,*
...

Sends a remote message to perform the *methodName* method with an arbitrary number of arguments. This is the general routine for sending remote messages and is used by most of the more specific Speaker methods. For example, an **openFile:ok:** message could be sent as follows:

```
int      msgDelivered, wasOK;

msgDelivered = [mySpeaker selectorRPC:"openFile:ok:"
                          paramTypes:"cI","/usr/foo",
                          &wasOK]
```

params is a character string, ªcIº in the example above, that describes the arguments to the method. Each argument is represented by a single character that encodes its type. (A single character, ªbº or ªBº, represents the two Objective C arguments of a byte array.) See the Listener class for an

explanation of these codes.

The actual arguments that will be passed to *methodName* are listed after *params*.

This method has the same return values as other methods that send remote messages: 0 on success, a Mach error code if the message couldn't be delivered, -1 if it was delivered but wasn't understood or couldn't be delegated, and NX_INCORRECTMESSAGE if the RPC succeeds but the selector is not implemented at the other end.

NS_DEV_DOCFOR:objc_method:[Speaker-sendOpenFileMsg:ok:andDeactivateSelf:];, sendOpenFileMsg:ok:andDeactivateSelf:

- (int)**sendOpenFileMsg:**(const char *)*fullPath*
ok:(int *)*flag*
andDeactivateSelf:*(BOOL)deactivateFirst*

Initiates an **openFile:ok:** remote message, which could also be initiated by sending an **openFile:ok:** message directly to the Speaker. However, when a Speaker receives an **openFile:ok:** message, it first deactivates the application in order to allow the receiving application to become active when it opens the file.

In contrast, this way of sending an **openFile:ok:** remote message gives the sending application control over whether it will deactivate before dispatching the message. If *deactivateFirst* is YES, this method works just like **openFile:ok:**. If *deactivateFirst* is NO, the sending application will remain the active application.

See also: - **openFile:ok:**

NS_DEV_DOCFOR:objc_method:[Speaker-sendOpenTempFileMsg:ok:andDeactivateSelf:];, sendOpenTempFileMsg:ok:andDeactivateSelf:

- (int)**sendOpenTempFileMsg:**(const char *)*fullPath*
ok:(int *)*flag*
andDeactivateSelf:*(BOOL)deactivateFirst*

Initiates an **openTempFile:ok:** remote message, which could also be initiated by sending an **openTempFile:ok:** message directly to the Speaker. However, when a Speaker receives an **openTempFile:ok:** message, it first deactivates the application in order to allow the receiving application to become active when it opens the file.

In contrast, this way of sending an **openTempFile:ok:** remote message gives the sending application control over whether it will deactivate before dispatching the message. If *deactivateFirst* is YES, this method works just like **openTempFile:ok:**. If *deactivateFirst* is NO, the sending application will remain the active application.

See also: - **openTempFile:ok:**

NS_DEV_DOCFOR:objc_method:[Speaker-sendPort];, sendPort

- (port_t)**sendPort**

Returns the port the Speaker will send remote messages to.

See also: - **setSendPort:**

NS_DEV_DOCFOR:objc_method:[Speaker-sendTimeout];, sendTimeout

- (int)**sendTimeout**

Returns how many milliseconds the Speaker will wait for its remote message to be delivered to the port of the receiving application. The Speaker caches this value as its **sendTimeout** instance variable. If it's 0, there's no time limit.

See also: - **setSendTimeout:**

NS_DEV_DOCFOR:objc_method:[Speaker-setDelegate:];, setDelegate:

- **setDelegate:***anObject*

Makes *anObject* the Speaker's delegate. The default delegate is **nil**. However, before processing the first event, Application's **run** method makes the Application object, **NXApp**, the delegate of the Speaker registered as the **appSpeaker**. If there is no **appSpeaker**, the **run** method creates one, registers it, and sets its delegate to be NXApp.

Unlike a Listener, a Speaker doesn't expect anything from its delegate.

See also: - **delegate**, - **setAppSpeaker:** (Application)

NS_DEV_DOCFOR:objc_method:[Speaker-setReplyPort:];, setReplyPort:

- **setReplyPort:**(port_t)*aPort*

Makes *aPort* the port where the Speaker receives return messages. If the Speaker sends a remote message with output arguments, it will supply the receiving application with send rights to this port, then wait for a return message containing the output data it requested.

If *aPort* is PORT_NULL, the Speaker will use a port supplied by the Application object in response to a **replyPort** message. Since return messages are read from the port as they arrive (synchronously), a number of different Speakers can share the same port.

At start-up, before the **run** method gets the application's first event, it sets the port of the Speaker registered as the **appSpeaker** to the port returned by Application's **replyPort** method.

See also: - **replyPort**, - **replyPort** (Application)

NS_DEV_DOCFOR:objc_method:[Speaker-setReplyTimeout:];, setReplyTimeout:

- **setReplyTimeout:**(int)*ms*

Sets, to *ms* milliseconds, how long the Speaker will wait to receive a reply from the application it sent a remote message. The Speaker expects a reply when the remote message it sends contains output arguments for information to be supplied by the receiving application. If *ms* is 0, there will be no time limit; the Speaker will wait until a return message is received or there's a transmission error. The default is 30,000 milliseconds.

See also: - **replyTimeout**

NS_DEV_DOCFOR:objc_method:[Speaker-setSendPort:];, setSendPort:

- **setSendPort:**(port_t)*aPort*

Makes *aPort* the port that the Speaker will send remote messages to. The default is PORT_NULL. A single Speaker can send remote messages to a variety of applications simply by setting a different port before each message.

The **NXPortFromName()** function can be used to find the public port of another application, as explained in the class description above.

See also: - **sendPort**

NS_DEV_DOCFOR:objc_method:[Speaker-setSendTimeout:];, setSendTimeout:

- **setSendTimeout:**(int)*ms*

Sets, to *ms* milliseconds, how long the Speaker will persist in attempting to deliver a message to the port of the receiving application. If *ms* is 0, there will be no time limit; the Speaker will wait until the message is successfully delivered or there's a transmission error. The default is 30,000 milliseconds.

See also: - **sendTimeout**

NS_DEV_DOCFOR:objc_method:[Speaker-write:];, write:

- **write:**(NXTypedStream *)*stream*

Writes the receiving Speaker to the typed stream *stream*. Returns **self**.

See also: - **read:**