| | |
|---|---|
| | free |
| Setting up the application+ workspace | |
| | loadNibFile:owner: |
| | loadNibFile:owner:withNames: |
| | loadNibFile:owner:withNames:fromZone: |
| | loadNibSection:owner: |
| | loadNibSection:owner:withNames: |
| | loadNibSection:owner:withNames:fromHeader: |
| | loadNibSection:owner:withNames:fromZone: |
| | loadNibSection:owner:withNames:fromHeader: fromZone: |
| | appName |
| | setMainMenu: |
| | mainMenu |
| Responding to notification | applicationWillLaunch: |
| | applicationDidLaunch: |
| | applicationDidTerminate: |
| Changing the active application | |
| | activeApp |
| | becomeActiveApp |
| | activate: |
| | activateSelf: |
| | isActive |
| | resignActiveApp |
| | deactivateSelf |
| Running the event loop | run |
| | isRunning |
| | stop: |
| | runModalFor: |
| | stopModal |
| | stopModal: |
| | abortModal |
| | beginModalSession:for: |

| | |
|---|---|
| | peekNextEvent:into:waitFor:threshold: |
| Journaling | setJournalable: |
| | isJournalable |
| | masterJournaler |
| | slaveJournaler |
| Handling user actions and events | |
| | applicationDefined: |
| | hide: |
| | isHidden |
| | unhide |
| | unhide: |
| | unhideWithoutActivation: |
| | powerOff: |
| | powerOffIn:andSave: |
| | rightMouseDown: |
| | unmounting:ok: |
| Sending action messages | sendAction:to:from: |
| | tryToPerform:with: |
| | calcTargetForAction: |
| Remote messaging | setAppListener: |
| | appListener |
| | setAppSpeaker: |
| | appSpeaker |
| | appListenerPortName |
| | replyPort |
| Managing Windows | appIcon |
| | findWindow: |
| | getWindowNumbers:count: |
| | keyWindow |
| | mainWindow |
| | makeWindowsPerform:inOrder: |
| | setAutoupdate: |
| | updateWindows |
| | windowList |
| | miniaturizeAll: |
| | preventWindowOrdering |
| Managing the Windows menu | setWindowsMenu: |
| | windowsMenu |
| | arrangeInFront: |
| | addWindowsItem:title:filename: |
| | changeWindowsItem:title:filename: |
| | removeWindowsItem: |
| | updateWindowsItem: |
| Managing Panels | showHelpPanel: |
| | orderFrontDataLinkPanel: |
| Managing the Services menu | setServicesMenu: |
| | servicesMenu |
| | registerServicesMenuSendTypes:andReturnTypes: |
| | validRequestorForSendType:andReturnType: |
| Managing screens | mainScreen |

| | |
|---|---|
| | openTempFile:ok: |
| | fileOperationCompleted: |
| Responding to devices | mounted: |
| | unmounted: |
| Printing | setPrintInfo: |
| | printInfo |
| | runPageLayout: |
| Color | orderFrontColorPanel: |
| | setImportAlpha: |
| | doesImportAlpha |
| Terminating the application | terminate: |
| Assigning a delegate | setDelegate: |
| | delegate |

run

(int)activate:(int)contextNumber

Makes the application identified by contextNumber the active application. The argument contextN
context number of the application to be activated. Normally, you shouldn't invoke this method th
responsible for proper activation. Returns the PostScript context number of application that was pr

isActive, activateSelf:, deactivateSelf

(int)activateSelf:(BOOL)flag

Makes the receiving application the active application. If flag is NO, the application is activated o
application is currently active. Normally, this method is invoked with flag set to NO. When the W
launches an application, it deactivates itself, so activateSelf:NO allows the application to become a
for it to launch, but the application remains unobtrusive if the user activates another application. If
application will always activate. Regardless of the setting of flag, there may be a time lag before t
you should not assume that the application will be active immediately after sending this message.

Note that you can make one of your Windows the key window without changing the active applica
makeKeyWindow message to a Window, you simply ensure that the Window will be the key wind
application is active.

You should rarely need to invoke this method. Under most circumstances the Application Kit take
activation. However, you might find this method useful if you implement your own methods for in
communication. This method returns the PostScript context number of the previously active applic

activeApp, activate:, deactivateSelf, makeKeyWindow (Window)

(int)activeApp

Returns the active application's PostScript context number. If no application is active, returns zero

isActive, activate:

addWindowsItem:aWindow
title:(const char *)aString
filename:(BOOL)isFilename

Adds an item to the Windows menu corresponding to the Window aWindow. If isFilename is NO
literally in the menu. If isFilename is YES, aString is assumed to be a converted name with the na
the path (the way Window's setTitleAsFilename: method shows a title). If an item for aWindow a
Windows menu, this method has no effect. You rarely invoke this method because an item is plac
menu for you whenever a Window's title is set. Returns self.

changeWindowsItem:title:filename:, setTitle: (Window), setTitleAsFilename: (Window)

appIcon

(int)applicationDidLaunch:(const char *)appName

Notification from the Workspace Manager that the application whose name is appName has launch
messages the Application will receive if it has previously sent the Workspace Manager the messag
beginListeningForApplicationStatusChanges.

If the delegate implements the method app:applicationDidLaunch:, that message is sent to it.  If the
implement it, the method is handled by the Application subclass object (if you created one).  The r
integer your application defines and interprets it.  If you neither provide a delegate method nor ove
default definition simply returns 0.

app:applicationDidLaunch: (Application delegate method),  beginListeningForApplicationStatusC
(NXWorkspaceRequest protocol)

(int)applicationDidTerminate:(const char *)appName

Notification from the Workspace Manager that the application whose name is appName has termin
messages the Application will receive if it has previously sent the Workspace Manager the messag
beginListeningForApplicationStatusChanges.

If the delegate implements the method app:applicationDidTerminate:, that message is sent to it.  If
implement it, the method is handled by the Application subclass object (if you created one).  The r
integer your application defines and interprets it.  If you neither provide a delegate method nor ove
default definition simply returns 0.

app:applicationDidTerminate: (Application delegate method),  beginListeningForApplicationStatu
(NXWorkspaceRequest protocol)

(int)applicationWillLaunch:(const char *)appName

Notification from the Workspace Manager that the application whose name is appName is about to
the messages the Application will receive if it has previously sent the Workspace Manager the mes
beginListeningForApplicationStatusChanges.

If the delegate implements the method app:applicationWillLaunch:, that message is sent to it.  If th
implement it, the method is handled by the Application subclass object (if you created one).  The r
integer your application defines and interprets it.  If you neither provide a delegate method nor ove
default definition simply returns 0.

app:applicationWillLaunch: (Application delegate method),  beginListeningForApplicationStatusC
(NXWorkspaceRequest protocol)

appListener

Returns the Application object's  ListenerÐthe  object that will receive messages sent to the port th
application's  name.  If you don't  send a setAppListener: message before your application starts run
Listener is created for you.  (Note, however, that to communicate with the Workspace Manager to
files, you should send messages to the object that represents the Workspace Manager, returned by t
method it responds to the NXWorkspaceRequest protocol.)

checkInAs: (Listener), appName, NXPortFromName()

(const char *)appName

Returns the name under which the Application object has been registered for defaults. This name i
messaging unless the messaging name was changed by overriding appListenerPortName.

appListenerPortName

appSpeaker

Returns the Application object's Speaker. You can use this object to send messages to other applic

setSendPort: (Speaker)

arrangeInFront:sender

Arranges all of the windows listed in the Windows menu in front of all other windows. Windows
application but not listed in the Windows menu are not ordered to the front. Returns self.

removeWindowsItem:, makeKeyAndOrderFront: (Window)

becomeActiveApp

Sends the appDidBecomeActive: message to the Application object's delegate. This method is inv
application is activated. You never send a becomeActiveApp message directly, but you can overri
subclass. Returns self.

activateSelf:, appDidBecomeActive: (delegate method)

(NXModalSession *)beginModalSession:(NXModalSession *)session for:theWindow

Prepares the application for a modal session with theWindow. In other words, prepares the applica
events get to it only if they occur in theWindow. If session is NULL, an NXModalSession is alloc
given storage is used. (The sender could declare a local NXModalSession variable for this purpos
the key window and ordered to the front.

beginModalSession:for: should be balanced by endModalSession:. If an exception is raised, begin
arranges for proper cleanup. Do not use NX_DURING constructs to send an endModalSession: m
an exception. Returns the NXModalSession pointer that's used to refer to this session.

runModalSession:, endModalSession:

calcTargetForAction:(SEL)theAction

filename:(BOOL)isFilename

Changes the item for aWindow in the Windows menu to aString.  If aWindow doesn't have an iten
menu, this method adds the item.  If isFilename is NO, aString appears literally in the menu.  If isF
aString is assumed to be a converted name with the file's name preceding the path (the way Windo
setTitleAsFilename: places a title).  Returns self.

addWindowsItem:title:filename:,  setTitle: (Window),  setTitleAsFilename: (Window)

(const NXScreen *)colorScreen

Returns the screen that can best represent color.  This method will always return a screen, even if n
present.

(DPSContext)context

Returns the Application object's Display PostScript context.

(NXEvent *)currentEvent

Returns a pointer to the last event the Application object retrieved from the event queue.  A pointe
also passed with every event message.

getNextEvent:waitFor:threshold:,  peekNextEvent:waitFor:threshold:

deactivateSelf

Deactivates the application if it's active.  Normally, you shouldn't invoke this method the Applicat
for proper deactivation.  Returns self.

activeApp,  activate:,  activateSelf:

delayedFree:theObject

Frees theObject by sending it the free message after the application finishes responding to the curr
gets the next event.  If this method is performed during a modal loop, theObject is freed after the m
Returns self.

perform:with:afterDelay:cancelPrevious: (DelayedPerform informal protocol)

delegate

Returns the Application object's delegate.

setDelegate:

endModalSession:(NXModalSession *)session

Cleans up after a modal session. The argument session should be from a previous invocation of be

runModalSession:, beginModalSession:for:

(int)fileOperationCompleted:(int)operation

Notification from the Workspace Manager that the file operation identified by operation has compl
the integer returned by the method that requested the file operation, to wit performFileOperation:so
options: (part of NXWorkspaceRequest protocol).

If the delegate implements the method app:fileOperationCompleted:, that message is sent to it. If t
implement it, the method is handled by the Application subclass object (if you created one). The r
integer your application defines and interprets it. If you neither provide a delegate method nor ove
default definition simply returns 0.

findWindow:(int)windowNum

Returns the Window object that corresponds to the window number windowNum. This method is
finding the Window object associated with a particular event.

windowNum (Window)

focusView

Returns the View whose focus is currently locked, or nil if no View's focus is locked.

lockFocus (View)

free

Closes all the Application object's windows, breaks the connection to the Window Server, and free
object.

(NXEvent *)getNextEvent:(int)mask

Gets the next event from the Window Server and returns a pointer to its event record. This methoc
getNextEvent:waitFor:threshold: with an infinite timeout and a threshold of NX_MODALRESPTH

getNextEvent:waitFor:threshold, run, runModalFor:, currentEvent

(NXEvent *)getNextEvent:(int)mask
        waitFor:(double)timeout

to do this in response to a mouse-down event in order to track the mouse while it's down. In this c
mask to accept mouse-dragged, mouse-entered, mouse-exited, or mouse-up events.

level determines what other procedures should be performed when the event queue is examined. T
procedures to deal with timed-entries, procedures to handle messages received on ports, or procedu
from files. Any such procedure that needs to be called will be called if its priority (specified when
registered) is equal to or higher than level.

In general, modal responders should pass NX_MODALRESPTHRESHOLD for level. The main r
threshold of NX_BASETHRESHOLD, allowing all procedures (except those registered with prior
invoked if needed.

peekNextEvent:waitFor:threshold:, run, runModalFor:

getScreens:(const NXScreen **)list count:(int *)numScreens

Gets screen information for every screen connected to the system. A pointer to an array of NXScre
in the variable indicated by list, and the number of NXScreen structures in that array is placed in th
numScreens. The list of NXScreen structures belongs to the Application object it should not be alt
self.

getScreenSize:(NXSize *)theSize

Gets the size of the main screen, in units of the screen coordinate system, and places it in the struct
theSize. Returns self.

getWindowNumbers:(int **)list count:(int *)numWindows

Gets the window numbers for all the Application object's Windows. A pointer to a non-NULL-ter
placed in the variable indicated by list. The number of entries in this array is placed in the integer
numWindows. The order of window numbers in the array is the same as their order in the Window
which is their front-to-back order on the screen. The application is responsible for freeing the list a
Returns self.

hide:sender

Collapses the application's graphicsÐincluding all its windows, menus, and panelsÐinto a single
hide: message is usually sent using the Hide command in the application's main Menu. Returns se

unhide:

(const char *)hostName

Returns the name of the host machine on which the Window Server that serves the Application obj
method returns the name that was passed to the receiving Application object through the NXHost c

(BOOL)isHidden

Returns YES if the application is currently hidden, and NO if it isn't.

(BOOL)isJournalable

Returns YES if the application can be journaled, and NO if it can't.  By default, applications can b
is handled by the NXJournaler class.

setJournalable:

(BOOL)isRunning

Returns YES if the application is running, and NO if the stop: method has ended the main event lo

run,  stop:,  terminate:

keyWindow

Returns the key Window, that is, the Window that receives keyboard events.  If there is no key Wi
Window belongs to another application, this method returns nil.

mainWindow,  isKeyWindow (Window)

loadNibFile:(const char *)filename owner:anOwner

Loads interface objects from a NeXT Interface Builder (nib) file.  The argument anOwner is the ob
ªFile's  Ownerº  in Interface Builder's  File window.  The objects and their names are read from the
storage allocated from the default zone.

Objects that were archived in the nib file (standard objects from an Interface Builder palette) are se
and awake messages other objects are instantiated and are sent an init message.

Returns non-nil if the file filename is successfully opened and read, and nil otherwise.

Invoking loadNibFile:owner: is equivalent to invoking loadNibFile:owner:withNames:fromZone: 
argument values indicate that names should also be loaded and that memory should be allocated fr

loadNibFile:owner:withNames:fromZone:, NXDefaultMallocZone(),  awake (Object),  init (Objec

loadNibFile:(const char *)filename
owner:anObject
withNames:(BOOL)flag

Loads interface objects from a NeXT Interface Builder (nib) file.  The argument anOwner is the ob
ªFile's  Ownerº  in Interface Builder's  File window.  The objects are read from the specified interfa

loadNibFile:owner:withNames:fromZone:, NXDefaultMallocZone(), awake (Object), init (Obje

loadNibFile:(const char *)filename
owner:anOwner
withNames:(BOOL)flag
fromZone:(NXZone *)zone

Loads interface objects from a NeXT Interface Builder (nib) file. The argument anOwner is the ob
ªFile's Ownerº in Interface Builder's File window. The objects are read into memory allocated fro
YES, the objects' names are also loaded. Names must be loaded if you use NXGetNamedObject()
but are not otherwise required. Objects that were archived in the nib file (standard objects from an
palette) are sent finishUnarchiving and awake messages other objects are instantiated and are sent

Returns non-nil if the file filename is successfully opened and read.

awake (Object), init (Object)

loadNibSection:(const char *)name owner:anOwner

Loads interface objects and their names from the source identified by name. To find the source, th
follows:

·First, for a section named name within the _ _NIB segment of the application's executable file. (T
versions of Interface Builder routinely put nib sections, but not where Project Builder puts them
will be here only if the applications was compiled by an earlier version of Interface Builder.)

·Second, if no such section exists, the method searches certain language directories within the mai
name name and type ªnib,º andÐif it finds oneÐloads the interface objects from there. It sear
directories that the user specified for this application, or (if none) those specified by the user's d
preferences (see systemLanguages).

·Third, if there's no file named name in the main bundle's relevant language directories, it looks fo
name and type ªnibº in the main bundle (but outside the ª. lprojº directories).

The argument anOwner is the object that corresponds to the ªFile's Ownerº object in Interface Bui
The loaded objects are allocated memory from the default zone.

Objects that were archived in the nib file (standard objects from an Interface Builder palette) are se
and awake messages other objects are instantiated and are sent an init message.

Returns non-nil if the section or file is successfully opened and read.

Invoking loadNibSection:owner: is equivalent to invoking loadNibSection:owner:withNames:from
additional arguments indicate that names should also be loaded and that memory should be allocate
zone.

NXDefaultMallocZone(), + mainBundle (NXBundle), getPath:forResource:ofType: (NXBundle),
(Object)

loadNibSection:(const char *)name
owner:anOwner
withNames:(BOOL)flag

Invoking loadNibSection:owner:withNames is equivalent to invoking loadNibSection:owner:withN
the additional argument indicates that memory should be allocated from the default zone.

awake (Object), init (Object)

```
loadNibSection:(const char *)name
        owner:anOwner
        withNames:(BOOL)flag
        fromHeader:(const struct mach_header *)header
```

Loads interface objects from a section within a dynamically loaded object fileÐthat is, from a file
application's main bundle. The argument header identifies the file, as returned by the function obj
argument name identifies a named section within the file's _ _NIB segment. When no such file ex
searches the executable file's bundle, first within its language subdirectories, as described above fo
owner: instance method.

The argument anOwner is the object that corresponds to the ªFile's Ownerº object in Interface Bui
Memory for the loaded objects is allocated from the default zone. When flag is YES, the objects'
Names must be loaded if you use NXGetNamedObject() to get at the objects, but are not otherwise

Objects that were archived in the nib file (standard objects from an Interface Builder palette) are se
and awake messages other objects are instantiated and are sent an init message.

A class can use this method in its finishLoading class method to load interface data objects require
stored separately (for example, because the same interface objects are also used by other classes).

Returns non-nil if the section or file is successfully opened and read.

Invoking loadNibSection:owner:withNames:fromHeader: is equivalent to invoking loadNibSection
fromHeader:fromZone: when the additional arguments indicate that names should also be loaded a
be allocated from the default zone.

awake (Object), init (Object)

```
loadNibSection:(const char *)name
        owner:anOwner
        withNames:(BOOL)flag
        fromHeader:(const struct mach_header *)header
        fromZone:(NXZone *)zone
```

Loads interface objects from a section within a dynamically loaded object fileÐthat is, from a file
application's main bundle. The argument header identifies the file, as returned by the function obj
argument name identifies a named section within the file's _ _NIB segment. When no such file ex
searches the executable file's bundle, first within its language subdirectories, as described above fo
owner: instance method.

The argument anOwner is the object that corresponds to the ªFile's Ownerº object in Interface Bui
Memory for the loaded objects is allocated from the zone specified by zone. When flag is YES, th
also loaded. Names must be loaded if you use NXGetNamedObject() to get at the objects, but are
Objects that were archived in the nib file (standard objects from an Interface Builder palette) are se
and awake messages other objects are instantiated and are sent an init message.

A class can use this method in its finishLoading class method to load interface data objects require
stored separately (for example, because the same interface objects are also used by other classes).

Returns non-nil if the section is successfully opened and read.

Loads interface objects and their names from the source identified by name. The source may be a
executable file, or a file within the application bundle, as described above for the loadNibSection:o

The argument anOwner is the object that corresponds to the ªFile's Ownerº object in Interface Bui
When flag is YES, the objects' names are also loaded. Names must be loaded if you use NXGetNa
the objects, but are not otherwise required. Memory for the loaded objects is allocated from the zo
Objects that were archived in the nib file (standard objects from an Interface Builder palette) are se
and awake messages other objects are instantiated and are sent an init message.

Returns non-nil if the section or file is successfully opened and read, and nil otherwise.

loadNibSection:owner:withNames:fromHeader:fromZone:, awake (Object), init (Object)

mainMenu

Returns the Application object's main Menu.

(const NXScreen *)mainScreen

Returns the main screen. If there is only one screen, that screen is returned. Otherwise, this metho
key window's screen. If there is no key window, it attempts to return the main menu's screen. If t
this method returns the screen that contains the screen coordinate system origin.

screen (Window)

mainWindow

Returns the main Window. This method returns nil if there is no main window, if the main windov
application, or if the application is hidden.

keyWindow, isMainWindow (Window)

makeWindowsPerform:(SEL)aSelector inOrder:(BOOL)flag

Sends the Application object's Windows a message to perform the aSelector method. The messag
Window in turn until one of them returns YES this method then returns that Window. If no Windo
method returns nil.

If flag is YES, the Application object's Windows receive the aSelector message in the front-to-bac
appear in the Window Server's window list. If flag is NO, Windows receive the message in the or
Application object's window list. This order generally reflects the order in which the Windows we

The method designated by aSelector can't take any arguments.

masterJournaler

Returns the Application object's master journaler. Journaling is handled by the NXJournaler class

slaveJournalar:

Invoked by the Workspace Manager when the device identified by fullPath has completed mountin
directly send a mounted: message. This is one of the messages the Application will receive if it ha
Workspace Manager the message beginListeningForDeviceStatusChanges.

If the delegate implements the method app:mounted:, that message is sent to it. If the delegate doe
method is handled by the Application subclass object (if you created one). The return value is an a
application defines and interprets it. If you neither provide a delegate method nor override in a sub
definition simply returns 0.

unmounting:ok:, unmounted:

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

Responds to a remote message requesting the application to open a file. openFile:ok: is typically s
from the Workspace Manager, although an application can send it directly to another application.
object's delegate is queried with appAcceptsAnotherFile: and if the result is YES, it's sent an app:
If the delegate doesn't respond to either of these messages, they're sent to the Application object (i

The variable pointed to by flag is set to YES if the file is successfully opened, NO if the file is not
and 1 if the application does not accept another file. Returns zero.

app:openFile:type: (delegate method), openTempFile:ok:, openFile:ok: (Speaker)

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

Same as the openFile:ok: method, but app:openTempFile:type: is sent. Returns 0.

app:openTempFile:type: (delegate method), openTempFile:ok: (Speaker)

orderFrontColorPanel:sender

Displays the color panel. Returns self.

orderFrontDataLinkPanel:sender

Displays the data link panel. It does this by sending an orderFront: message to the shared instance
(if need be, creating a new one). Returns self.

(NXEvent *)peekAndGetNextEvent:(int)mask

This method is similar to getNextEvent:waitFor:threshold: with a zero timeout and a threshold of
NX_MODALRESPTHRESHOLD.

getNextEvent:waitFor:threshold, run, runModalFor:, currentEvent, peekNextEvent:into:

into:(NXEvent *)eventPtr
waitFor:(float)timeout
threshold:(int)level

This method is similar to getNextEvent:waitFor:threshold: except the matching event isn't remove
nor is it placed in currentEvent instead, it's copied into storage pointed to by eventPtr.

If no matching event is found, NULL is returned otherwise, eventPtr is returned.

getNextEvent:waitFor:threshold:, run, runModalFor:, currentEvent

powerOff:(NXEvent *)theEvent

A powerOff: message is generated when a power-off event is sent from the Window Server. As a
Workspace Manager and login window should respond to this event. If the application was launch
Manager, this method does nothing instead, the Application object will wait for the powerOffIn:an
the Workspace Manager. If the application wasn't launched from the Workspace Manager, this me
delegate a powerOff: message, assuming there's a delegate and it implements the method. Applica
launched from the Workspace Manager are not fully supported, and are not guaranteed any amoun
this message. However, applications launched from the Workspace Manager can request additiona
from within the app:powerOffIn:andSave method. Returns self.

app:powerOffIn:andSave: (delegate method), powerOffIn:andSave:

(int)powerOffIn:(int)ms andSave:(int)aFlag

You never invoke this method directly it's sent from the Workspace Manager. The delegate or you
Application will be given the chance to receive the app:powerOffIn:andSave message. The aFlag
particular meaning and can be ignored. This method raises an exception, so it never returns.

app:powerOffIn:andSave: (delegate method)

preventWindowOrdering

Suppresses the usual window ordering behavior entirely. Most applications will not need to use th
Application Kit support for dragging will call it when dragging is initiated.

printInfo

Returns the Application object's global PrintInfo object. If none exists, a default one is created.

registerServicesMenuSendTypes:(const char *const *)sendTypes andReturnTypes:(const char

Registers pasteboard types that the application can send and receive in response to service requests
a Services menu, a menu item is added for each service provider that can accept one of the specifie
one of the specified return types. This method should typically be invoked at application startup ti
that can use services is created. It can be invoked more than once its purpose is to ensure that ther
every service that the application may use. The individual items will be dynamically enabled and

automatically added again, so you must use Window's setExcludedFromWindowsMenu: method i
remain excluded from the Windows menu. Returns self.

changeWindowsItem:title:filename:, setExcludedFromWindowsMenu: (Window)

(port_t)replyPort

Returns the Application object's reply port. This port is allocated for you automatically by the run
default reply port which can be shared by all the Application object's Speakers.

setReplyPort: (Speaker)

resignActiveApp

This method is invoked immediately after the application is deactivated. You never send resignAc
directly, but you could override this method in your Application object to notice when your applica
Alternatively, your delegate could implement appDidResignActive:. Returns self.

deactivateSelf:, appDidResignActive: (delegate method)

rightMouseDown:(NXEvent *)theEvent

Pops up the main Menu. Returns self.

run

Initiates the Application object's main event loop. The loop continues until a stop: or terminate: m
Each iteration through the loop, the next available event from the Window Server is stored, and is
sending the event to the Application object using sendEvent:

A run message should be sent as the last statement from main(), after the application's objects have
Returns self if terminated by stop:, but never returns if terminated by terminate:.

runModalFor:, sendEvent:, stop:, terminate:, appDidInit: (delegate method)

(int)runModalFor:theWindow

Establishes a modal event loop for theWindow. Until the loop is broken by a stopModal, stopMod
message, the application won't respond to any mouse, keyboard, or window-close events unless th
theWindow. If stopModal: is used to stop the modal event loop, this method returns the argument
If stopModal is used, it returns the constant NX_RUNSTOPPED. If abortModal is used, it returns
NX_RUNABORTED. This method is functionally similar to the following:

can continue processing between method invocations. When you invoke this method, events for th
session are dispatched as normal this method returns when there are no more events. You must inv
frequently enough that the window remains responsive to events.

If the modal session was not stopped, this method returns NX_RUNCONTINUES. If stopModal w
result of event procession, NX_RUNSTOPPED is returned. If stopModal: was invoked, this meth
passed to stopModal:. The NX_abortModal exception raised by abortModal isn't caught.

beginModalSession:, endModalSession, stopModal:, stopModal, runModalFor:

runPageLayout:sender

Brings up the Application object's Page Layout panel, which allows the user to select the page size
Returns self.

(BOOL)sendAction:(SEL)aSelector to:aTarget from:sender

Sends an action message to an object. If aTarget is nil, the Application object looks for an object t
messageÐthat is, for an object that implements a method matching aSelector. It begins with the fi
window. If the first responder can't respond, it tries the first responder's next responder and contin
responder links up the Responder chain. If none of the objects in the key window's responder chai
message, the Application object attempts to send the message to the key Window's delegate.

If the delegate doesn't respond and the main window is different from the key window, NXApp be
responder in the main window. If objects in the main window can't respond, the Application objec
message to the main window's delegate. If still no object has responded, NXApp tries to handle th
NXApp can't respond, it attempts to send the message to its own delegate.

Returns YES if the action is applied otherwise returns NO.

sendEvent:(NXEvent *)theEvent

Sends an event to the Application object. You rarely send sendEvent: messages directly although y
override this method to perform some action on every event. sendEvent: messages are sent from th
run method). sendEvent is the method that dispatches events to the appropriate responders the App
application events, the Window indicated in the event record handles window related events, and m
are forwarded to the appropriate Window for further dispatching. Returns self.

setAutoupdate:

servicesMenu

Returns the Application object's Services menu. Returns nil if no Services menu has been created

setServicesMenu:

setAppListener:aListener

Sets the Application object's Speaker. If you don't send a setAppSpeaker: message before the App
initializes, a default Speaker is created for you. This method doesn't free the Application object's
object.

appWillInit: (delegate method)

setAutoupdate:(BOOL)flag

Turns on or off automatic updating of the application's windows. (Until this message is sent, auto
enabled.) When automatic updating is on, an update message is sent to each of the application's W
event has been processed. This can be used to keep the appearance of menus and panels synchroni
application. Returns self.

updateWindows

setDelegate:anObject

Sets the Application object's delegate. The notification messages that a delegate can expect to rec
end of the Application class specification. The delegate doesn't need to implement all the methods

delegate

setImportAlpha:(BOOL)flag

Determines whether your application will accept translucent colors in objects it receives. This affe
the View method acceptsColor:atPoint:, or by NXColorPanel's dragColor:withEvent:fromView:.
internal programmatic manipulations of colors.

A pixel may be described by its color (values for red, blue, and green) and also by its opacity, mea
called alpha. When alpha is 1.0, a color is completely opaque and thus hides anything beneath it.
1, the effective color is derived partly from the color of the object itself and partly from the color o
it. When flag is YES, the application accepts a color that includes an alpha coefficient, and forces
for a source where alpha was not specified. In addition, when flag is YES, a ColorPanel opened w
includes an opacity slider.

When the Application has received a setImportAlpha: message with flag set to NO, all imported co
an alpha value of NX_NOALPHA, and there's no opacity slider in the ColorPanel. The default sta
alpha.

This method has the same effect as the NXColorPanel method setShowAlpha:. The only differenc
setImportAlpha: even before an NXColorPanel has been instantiated. Since the two methods set th
each can reverse the effect of the other.

Returns self.

doesImportAlpha, doesShowAlpha (NXColorPanel), setShowAlpha: (NXColorPanel)

setJournalable:(BOOL)flag

mainMenu

setPrintInfo:info

Sets the Application object's global PrintInfo object. Returns the previous PrintInfo object, or nil i

printInfo

setServicesMenu:aMenu

Makes aMenu the Application object's Services menu. Returns self.

servicesMenu

setWindowsMenu:aMenu

Makes aMenu the Application object's Windows menu. Returns self.

windowsMenu

showHelpPanel:sender

Shows the application's Help panel. If no Help panel yet exists, the method first creates a default I
delegate implements app:willShowHelpPanel:, notifies it. Returns self.

slaveJournaler

Returns the Application object's slave journaler if one exists, or nil if not. The slave journaler is c
your application if these two conditions are met:

·Your application allows journaling (see setJournalable:)

·Some application running concurrently with yours (or your application itself) starts a journaling s

See the NXJournaler class specification for more information.

masterJournalar:

stop:sender

Stops the main event loop. This method will break the flow of control out of the run method, there
main() function. A subsequent run message will restart the loop.

If this method is applied during a modal event loop, it will break that loop but not the main event l

terminate:, run, runModalFor:, runModalSession:

stopModal:(int)returnCode

Just like stopModal except argument returnCode allows you to specify the value that runModalFor
self.

stopModal, runModalFor:, abortModal

(const char *const *)systemLanguages

Returns a list of the names of languages in order of the user's preference. If your application will
language preference, this method is the way to discover what the preferences are. The return is a N
pointers to NULL-terminated strings.

If the user has recorded preferences specific to the application now in use, the method returns them
recorded no preferences for the application, but has recorded a global preference, the method retur
preferences. (Note that just because the user has recorded a preference doesn't mean than the lang
installed on the host that is executing the application.) If this method returns NULL, the user has

terminate:sender

Terminates the application. (This is the default action method for the application's Quit menu item
terminate: invokes appWillTerminate: to notify the delegate that the application will terminate. If
returns nil, terminate: returns self control is returned to the main event loop, and the application isn
Otherwise, this method frees the Application object and calls exit() to terminate the application. N
put final cleanup code in your application's main() function it will never be executed.

stop, appWillTerminate: (delegate method), exit()

(BOOL)tryToPerform:(SEL)aSelector with:anObject

Aids in dispatching action messages. The Application object tries to perform the method aSelecto
Responder method tryToPerform:with:. If the Application object doesn't perform aSelector, the de
opportunity to perform it using its inherited Object method perform:with:. If either the Applicatio
Application object's delegate accept aSelector, this method returns YES otherwise it returns NO.

tryToPerform:with: (Responder), respondsTo: (Object), perform:with: (Object)

(int)unhide

Responds to an unhide message sent from Workspace Manager. You shouldn't invoke this metho
instead. Returns zero.

unhide:

unhide:sender

invoking this method to make the receiving application active if there is no active application.  Ret

hide:,  activateSelf:

(int)unmounted:(const char *)fullPath

Invoked by the Workspace Manager when it has completed unmounting the device identified by fu
directly send an unmounted: message.   This is one of the messages the Application will receive if
the Workspace Manager the message beginListeningForDeviceStatusChanges.

If the delegate implements the method app:unmounted:, that message is sent to it.  If the delegate d
the method is handled by the Application subclass object (if you created one).  The return is an arb
application defines and interprets it.  If you neither provide a delegate method nor override in a sub
definition simply returns 0.

mounted:,  unmounting:ok:

(int)unmounting:(const char *)fullPath ok:(int *)flag

Invoked and sent to all active applications when the Workspace Manager has received a request to
identified by fullPath.  This serves to warn applications that may be making use of the device.  You
send unmounting:ok: messages.

The method sets flag to point to YES to indicate that the Application assents to unmounting, and N

If the delegate implements the method app:unmounting:, that message is sent to it, and flag is set t
returns.  If the delegate doesn't  implement app:unmounting:, the method is handled by the Applica
you created one).  The default behavior is to close all files on the device, and if the current workin
device, to change the current working directory to the user's  home directory.

The return value is an arbitrary integer your application defines and interprets it.  If you neither pr
nor override in a subclass, the default definition simply returns 0.

updateWindows

Sends an update message to the Application object's  visible Windows.  When automatic updating
method is invoked automatically in the main event loop after each event.  An application can also s
messages at other times to have Windows update themselves.

If the delegate implements appWillUpdate:, that message is sent to the delegate before the window
Similarly, if the delegate implements appDidUpdate:, that message is sent to the delegate after the
Returns self.

setAutoupdate:,  appWillUpdate: (delegate method),  appDidUpdate: (delegate method)

updateWindowsItem:aWindow

Updates the item for aWindow in the Windows menu to reflect the edited status of aWindow.  You
this method because it is invoked automatically when the edited status of a Window is set.  Return

changeWindowsItem:title:filename:,  setDocEdited: (Window)

Messages to perform this method are initiated by the Services menu.

validRequestorForSendType:andReturnType: (Responder), registerServicesMenuSendTypes:and
writeSelectionToPasteboard:types: (Object), readSelectionFromPasteboard: (Object)

windowList

Returns the List object used to keep track of all the Application object's Windows, including Menu
In the current implementation, this list also contains global (shared) Windows.

windowsMenu

Returns the Application object's Windows menu. Returns nil if no Windows menu has been creat

app:sender applicationDidLaunch:(const char *)appName

Implement this method to respond to an applicationDidLaunch: message sent from the Workspace
Application object), informing it that an application named appName has launched. This is one of
Application will receive if it has previously sent the Workspace Manager the message
beginListeningForApplicationStatusChanges.

applicationDidLaunch:

app:sender applicationDidTerminate:(const char *)appName

Implement this method to respond to an applicationDidTerminate: message sent from the Workspa
(an Application object), informing it that an application named appName has terminated. This is o
Application will receive if it has previously sent the Workspace Manager the message
beginListeningForApplicationStatusChanges.

applicationDidTerminate:

app:sender applicationWillLaunch:(const char *)appName

Implement this method to respond to an applicationWillLaunch: message sent from the Workspace
Application object), informing it that an application named appName is about to launch. This is o
Application will receive if it has previously sent the Workspace Manager the message
beginListeningForApplicationStatusChanges.

applicationWillLaunch:

app:sender fileOperationCompleted:(int)operation

Implement this method to respond to a mounted: message sent from the Workspace Manager to se
object), informing it that a device (for example a floppy disk or an optical disk) has been mounted.
messages the Application will receive if it has previously sent the Workspace Manager the messag
beginListeningForDeviceStatusChanges.

mounted:

```
    (int)app:sender
         openFile:(const char *)filename
         type:(const char *)aType
```

Invoked from within openFile:ok: after it has been determined that the application can open anothe
should attempt to open the file of type type and name filename, returning YES if the file is success
otherwise. (Although a file's type may by convention be reflected in its name, type is not a synony
filename should not exclude part of the name just because it can sometimes be inferred from type.)

This method is also invoked from within openTempFile:ok: if neither the delegate nor the Applica
to app:openTempFile:type:

openFile:ok:, openTempFile:ok:, app:openFileWithoutUI:type:, app:openTempFile:type:

```
    (NXDataLinkManager *)app:sender
         openFileWithoutUI:(const char *)filename
         type:(const char *)type
```

Sent to the delegate when sender (an Application) requests that the file of type type and name filen
linked file. The file is to be opened without bringing up its application's user interface that is, wor
under programmatic control of sender, rather than under keyboard control of the user.

Returns a pointer to the NXDataLinkManager that will coordinate data flow between the two appli

app:openFile:type:

```
    (int)app:sender
         openTempFile:(const char *)filename
         type:(const char *)aType
```

Invoked from within openTempFile:ok: after it has been determined that the application can open a
method should attempt to open the file filename with the extension aType, returning YES if the fil
opened, and NO otherwise.

By design, a file opened through this method is assumed to be temporary it's the application's resp
the file at the appropriate time.

openFile:ok:, openTempFile:ok:

```
    app:sender powerOffIn:(int)ms andSave:(int)aFlag
```

Invoked from the powerOffIn:andSave: method after the Workspace Manager receives a power-of
invoked only if the application was launched from the Workspace Manager. The argument ms is t
milliseconds to wait before powering down or logging out. The argument aFlag has no particular

Implement this method to respond to an unmounted: message sent from the Workspace Manager to
Application object), informing it that the device identified by fullPath has been unmounted. This i
the Application will receive if it has previously sent the Workspace Manager a beginListeningForD
message.

unmounted, app:mounted:

(int)app:sender unmounting:(const char *)fullPath

Invoked when the device mounted at fullPath is about to be unmounted. This method is invoked fr
and is invoked only if the application was launched from the Workspace Manager. The Applicatio
should do whatever is necessary to allow the device to be unmounted. Specifically, all files on the
closed and the current working directory should be changed if it's on the device.

unmounting:ok:, app:unmounted:

app:sender willShowHelpPanel:panel

Implement this to respond to notice that sender (an Application) has received a showHelpPanel: m
put up the Help panel identified by panel. The return value doesn't matter.

showHelpPanel:

(BOOL)appAcceptsAnotherFile:sender

Invoked from within Application's openFile:ok: and openTempFile:ok: methods, this method shou
okay for the application to open another file, and NO if isn't. If neither the delegate nor the Applic
to the message, then the file shouldn't be opened.

openFile:ok:, openTempFile:ok:

appDidBecomeActive:sender

Implement to respond to notification sent from the Workspace Manager immediately after the App
active.

applicationDidLaunch:

appDidHide:sender

Invoked immediately after the application is hidden.

hide:, unhide:, appDidUnhide: (delegate method)

appDidInit:sender

becomeActiveApp, resignActiveApp

appDidUnhide:sender

Invoked immediately after the application is unhidden.

hide:, unhide:, appDidHide: (delegate method)

appDidUpdate:sender

Invoked immediately after the Application object updates its Windows.

updateWindows, updateWindowsItem:, appWillUpdate: (delegate method)

applicationDefined:(NXEvent *)theEvent

Invoked when the application receives an application-defined (NX_APPDEFINED) event. See the
method under ªInstance Methods,º above.

appWillInit:sender

Invoked before the Application object is initialized. This method is invoked before the Applicatio
its Listener and Speaker objects and before any app:openFile:type: messages are sent to your deleg
object's Listener and Speaker objects will be created for you immediately after invoking this meth
previously created.

appDidInit: (delegate method), appListener, appSpeaker

appWillTerminate:sender

Invoked from within the terminate: method immediately before the application terminates. If this
application is not terminated, and control is returned to the main event loop. If you want to allow t
terminate, you should put your clean up code in this method and return non-nil.

terminate:

appWillUpdate:sender

Invoked immediately before the Application object updates its Windows.

updateWindows, updateWindowsItem:, appDidUpdate: (delegate method)

powerOff:(NXEvent *)theEvent