

2

Application Kit

Classes

ActionCell

Inherits From: Cell : Object

Configuring an ActionCell

- | | |
|--|---|
| - setEnabled: (BOOL) <i>flag</i> | Sets whether the ActionCell reacts to mouse events |
| - setBezeled: (BOOL) <i>flag</i> | Adds or removes the ActionCell's bezel |
| - setBordered: (BOOL) <i>flag</i> | Adds or removes the ActionCell's border |
| - setAlignment: (int) <i>mode</i> | Sets the ActionCell's text alignment to <i>mode</i> |
| - setFloatingPointFormat: (BOOL) <i>autoRange</i> | Sets the ActionCell's floating point format |
| left: (unsigned int) <i>leftDigits</i> | |
| right: (unsigned int) <i>rightDigits</i> | |
| - setFont: <i>fontObject</i> | Sets the ActionCell's Font to <i>fontObject</i> |

- **setIcon:**(const char *)*iconName*

Sets the ActionCell's icon to the NXImage named *iconName*

Manipulating ActionCell Values

- (double)**doubleValue**

Returns the ActionCell's contents as a **double**

- (float)**floatValue**

Returns the ActionCell's contents as a **float**

- (int)**intValue**

Returns the ActionCell's contents as an **int**

- **setStringValue:**(const char *)*aString*

Sets the ActionCell's contents to a copy of *aString*

- **setStringValueNoCopy:**(char *)*aString*
 shouldFree:(BOOL)*flag*

Sets the ActionCell's contents to a *aString*; will free the string when freed if *flag* is YES

- (const char *)**stringValue**

Returns the ActionCell's contents as a string

Displaying

- **drawSelf:**(const NXRect *)*cellFrame*
 inView:*controlView*

Draws the ActionCell in *controlView*

- **controlView**

Returns the View in which the ActionCell was most recently drawn (usually a Control)

Target and Action

- **setAction:**(SEL)*aSelector*

Sets the ActionCell's action method to *aSelector*

- (SEL)**action**

Returns the ActionCell's action method

- **setTarget:***anObject*

Sets the ActionCell's target object to *anObject*

- **target**

Returns the ActionCell's target object

Assigning a Tag

- **setTag:**(int)*anInt*

Sets the ActionCell's tag to *anInt*

- (int)**tag**

Returns the ActionCell's tag

Archiving

Reads the `ActionCell` from *stream*

Writes the `ActionCell` to *stream*

Application

Inherits From: Responder : Object

Initializing the Class Object

Registers defaults for the application

Creating and Freeing Instances

Returns a new Application object

Deallocates the Application object

Setting Up the Application

Returns the **id** of WorkspaceManager

Loads objects built using Interface Builder

Loads objects built using Interface Builder

Loads objects built using Interface Builder

Loads objects built using Interface Builder

- **loadNibSection:**(const char *)*name* Loads objects built using Interface Builder
 owner:*anObject*
 withNames:(BOOL)*flag*
- **loadNibSection:**(const char *)*name* Loads objects built using Interface Builder
 owner:*anObject*
 withNames:(BOOL)*flag*
 fromHeader:(const struct mach_header *)*header*
- **loadNibSection:**(const char *)*name* Loads objects built using Interface Builder
 owner:*anObject*
 withNames:(BOOL)*flag*
 fromZone:(NXZone *)*zone*
- **loadNibSection:**(const char *)*name* Loads objects built using Interface Builder
 owner:*anObject*
 withNames:(BOOL)*flag*
 fromHeader:(const struct mach_header *)*header*
 fromZone:(NXZone *)*zone*
- (const char *)**appName** Returns the application's name
- **setMainMenu:***aMenu* Makes *aMenu* the application's main menu
- **mainMenu** Returns the **id** of the application's main menu

Responding to Notification)

- **applicationDidLaunch:***appName* Notice that *appName* launched; your arbitrary return
- **applicationDidTerminate:***appName* Notice that *appName* ended; your arbitrary return
- **applicationWillLaunch:***appName* Notice that *appName* will launch; your arbitrary return

Changing the Active Application

- (int)**activate:**(int)*contextNumber* Makes *contextNumber* the active application
- (int)**activateSelf:**(BOOL)*flag* Makes this the active application
- (int)**activeApp** Returns context number of the active application
- **becomeActiveApp** Responds to activating the application

- deactivateSelf	Deactivates the application
- (BOOL) isActive	Returns whether this is the active application
- resignActiveApp	Responds to deactivating the application

Running the Event Loop

- run	Starts the main event loop
- stop:sender	Stops the main event loop
- (int) runModalFor:theWindow	Starts a modal event loop for <i>theWindow</i>
- stopModal	Stops the modal event loop
- stopModal:(int)returnCode	Stops the event loop started by runModalFor:
- (void) abortModal	Aborts the event loop started by runModalFor:
- (NXModalSession *) beginModalSession:(NXModalSession *)session for:theWindow	Sets up a modal session with <i>theWindow</i>
- (int) runModalSession:(NXModalSession *)session	Runs a modal session
- endModalSession:(NXModalSession *)session	Finishes a modal session
- delayedFree:theObject	Frees <i>theObject</i> after finishing the current event
- (BOOL) isRunning	Returns whether the main event loop is running
- sendEvent:(NXEvent *)theEvent	Dispatches events to other objects

Getting and Peeking at Events

- (NXEvent *) currentEvent	Returns pointer to the current event
- (NXEvent *) getNextEvent:(int)mask	Returns pointer to the next event matching <i>mask</i>
- (NXEvent *) getNextEvent:(int)mask waitFor:(double)timeout threshold:(int)level	Returns pointer to the next event matching <i>mask</i>
- (NXEvent *) peekAndGetNextEvent:(int)mask	Returns pointer to the next event matching <i>mask</i>
- (NXEvent *) peekNextEvent:(int)mask into:(NXEvent *)eventPtr	Returns pointer to the next event matching <i>mask</i>

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

Returns pointer to the next event matching *mask*

Journaling

- (BOOL)**isJournalable**
- **setJournalable**:(BOOL)*flag*
- **masterJournaler**
- **slaveJournaler**

Returns whether the application can be journaled

Sets whether the application can be journaled

Returns the controlling NXJournaler object

Returns the controlled NXJournaler object

Handling User Actions and Events

- **applicationDefined**:(NXEvent *)*theEvent*
- **hide**:*sender*
- (BOOL)**isHidden**
- (int)**unhide**
- **unhide**:*sender*
- **unhideWithoutActivation**:*sender*
- **powerOff**:(NXEvent *)*theEvent*
- (int)**powerOffIn**:(int)*ms* **andSave**:(int)*aFlag*
- **rightMouseDown**:(NXEvent *)*theEvent*
- (int)**unmounting**:(const char *)*fullPath*
ok:(int *)*flag*

Responds to an application-defined event

Hides all the application's windows

YES if windows are hidden

Responds to message to unhide windows

Restores hidden windows to the screen

Restores hidden windows without activating their owner

Responds to a power-off subevent

Responds to message from Workspace Manager

Causes main menu to pop up under the mouse

Responds to message from Workspace Manager

Sending Action Messages

- (BOOL)**sendAction**:(SEL)*aSelector*
to:*aTarget*
from:*sender*
- (BOOL)**tryToPerform**:(SEL)*aSelector*

Sends an action message to *aTarget* or up the responder chain

Attempts to send a message to the application or the

with:*anObject*

- **calcTargetForAction:**(SEL)*theAction*

delegate

Looks up receiver for *theAction* message

Remote Messaging

- **setAppListener:***aListener*

Makes *aListener* the application's Listener

- **appListener**

Returns the Listener for this application

- **setAppSpeaker:***aSpeaker*

Makes *aSpeaker* the application's Speaker

- **appSpeaker**

Returns the Speaker for this application

- (const char *)**appListenerPortName**

Returns name used to register Listener with name server

- (port_t)**replyPort**

Returns port for synchronous return messages

Managing Windows

- **applcon**

Returns the Window with the application's icon

- **findWindow:**(int)*windowNum*

Returns Window object corresponding to *windowNum*

- **getWindowNumbers:**(int **)*list*
 count:(int *)*winCount*

Gets window numbers for the application's windows

- **keyWindow**

Returns the the key window

- **mainWindow**

Returns the main window

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

Sends *aSelector* message to the Windows

- **setAutoupdate:**(BOOL)*flag*

Sets whether to send Windows **update** messages

- **updateWindows**

Sends **update** message to all on-screen Windows

- **windowList**

Returns a List of the application's Windows

- **miniaturizeAll:**

Miniaturizes all the receiver's application windows

- **preventWindowOrdering**

Suppresses usual window ordering entirely

Managing the Windows Menu

- **setWindowsMenu:***aMenu*

Sets the Windows menu

- **windowsMenu** Returns the Windows menu
- **arrangeInFront:** Orders all registered Windows to the front
- **addWindowsItem:***aWindow*
 title:(const char *)*aString*
 filename:(BOOL)*isFilename*
 Adds a menu item for *aWindow*
- **removeWindowsItem:***aWindow* Removes the Window's menu item
- **changeWindowsItem:***aWindow*
 title:(const char *)*aString*
 filename:(BOOL)*isFilename*
 Changes the Window's menu item
- **updateWindowsItem:***aWindow* Updates the Window's menu item

Managing Panels

- **showHelpPanel:***sender* Shows the application's help panel or default
- **orderFrontDataLinkPanel:** Shows the shared instance; creates if need be

Managing the Services menu

- **setServicesMenu:***aMenu* Sets the Services menu
- **servicesMenu** Returns the Services menu
- **registerServicesMenuSendTypes:**(const char *const *)*sendTypes*
 andReturnTypes:(const char *const *)*returnTypes*
 Registers pasteboard types the application can send/receive
- **validRequestorForSendType:**
 (NXAtom)**sendType**
 andReturnType:(NXAtom)*returnType*
 Indicates whether the Application can send and receive the specified types

Managing Screens

- (const NXScreen *)**mainScreen** Returns the main screen
- (const NXScreen *)**colorScreen** Returns the best screen for color
- **getScreens:**(const NXScreen **)*list* Gets information about every connected screen

- count:**(int *)*numScreens*
- **getScreenSize:**(NXSize *)*theSize*

Provides the size of the screen in pixels

Querying the Application

- (DPSContext)**context**
- **focusView**
- (const char *)**hostName**

Returns the Application's DPS context

Gets the currently lockFocus'ed View

Returns machine running the Window Server

Reporting Current Languages

- (const char *const *)**systemLanguages**

Gets a list of the user's preferred languages

Opening Files

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

Asks the delegate to open the *fullPath* file

Asks the delegate to open a temporary file

Notification of completion of NXWorkspaceRequest; arbitrary *operation* and return

Responding to Devices

- (int)**mounted:**(const char *)*fullPath*
- **unmounted:**(const char *)*fullPath*

Notice that device at *fullPath* has been mounted; returns 0 (unless overrideen in a subclass)

Notice that device at *fullPath* has been unmounted; returns 0 (unless overrideen in a subclass)

Printing

- **setPrintInfo:***info*
- **printInfo**

Makes *info* the application's PrintInfo object

Returns the application's PrintInfo object

- **runPageLayout:***sender*

Runs the application's PageLayout panel

Color

- **orderFrontColorPanel:***sender*

Brings up the color panel

- **doesImportAlpha**

YES if application responds to opacity in imported colors

- **setImportAlpha:**

Enable/disable response to opacity in imported colors

Terminating the Application

- **terminate:***sender*

Frees the Application object and exits the application

Assigning a Delegate

- **setDelegate:***anObject*

Makes *anObject* the Application's delegate

- **delegate**

Returns the Application's delegate

Implemented by the Delegate

- **app:***sender*

Notice that *appName* will launch²

applicationWillLaunch:(const char *)*appName*

- **app:***sender*

Notice that *appName* launched²

applicationDidLaunch:(const char *)*appName*

- **appWillInit:***sender*

Notifies delegate before initializing²

- **appDidInit:***sender*

Notifies delegate before getting first event²

- **appDidBecomeActive:***sender*

Notifies delegate on activating the application

- **appDidResignActive:***sender*

Notifies delegate on deactivating the application

- **appDidHide:***sender*

Notifies delegate application has been hidden

- **appDidUnhide:***sender*

Notifies delegate application has been unhidden

- **appWillUpdate:***sender*

Notifies delegate application's windows will be updated

- **appDidUpdate:***sender*

Notifies delegate on updating the application's windows

- (BOOL)**appAcceptsAnotherFile:***sender*

YES if it's okay to open another file²

- (int)**app:sender**
openFile:(const char *)*filename*
type:(const char *)*aType*
Opens *filename*²
- (int)**app:sender**
openTempFile:(const char *)*filename*
type:(const char *)*aType*
Opens temporary file *filename*²
- (NXDataLinkManager *)**app:sender**
openFileWithoutUI:(const char *)*filename*
type:(const char *)*aType*
Open application to run without user interface²
- **app:sender**
fileOperationCompleted:(int)*operation*
Notice that *operation* completed; your arbitrary return²
- **app:sender** **mounted:**(const char *)*fullPath*
Notification that device at *fullPath* was mounted²
- (int)**app:sender**
unmounting:(const char *)*fullPath*
Facilitates unmounting a device²
- **app:sender**
unmounted:(const char *)*fullPath*
Notification that device at *fullPath* was unmounted²
- **appWillTerminate:***sender*
Notification that application will terminate²
- **app:sender**
willShowHelpPanel:*panel*
Notification that *sender* will show *panel*
- **app:sender**
powerOffIn:(int)*ms*
andSave:(int)*aFlag*
Responds to message from Workspace Manager²
- **powerOff:**(NXEvent *)*theEvent*
Responds to a power-off subevent
- **app:sender**
applicationDidTerminate:(const char *)*appName*
Notification that application terminated²

² These methods may be defined in the delegate and/or in a subclass. If a method is implemented both in a subclass and in the delegate, the message is sent to the delegate.

Inherits From: View : Responder : Object

Initializing and Freeing a Box

- **initWithFrame:**(const NXRect *)*frameRect*
 - **free**
- Initializes a new Box object with the given *frameRect*

Deallocates the Box

Modifying the Border and Title

- **setBorderType:**(int)*aType*
 - (int)**borderType**
 - **setTitlePosition:**(int)*aPosition*
 - (int)**titlePosition**
 - **setTitle:**(const char *)*aString*
 - (const char *)**title**
 - **cell**
 - **setFont:***fontObj*
 - **font**
- Sets the Box's border to *aType*

Returns the Box's border type

Sets the position of the title to *aPosition*

Returns the position of the title

Sets the Box's title to *aString*

Returns the title of the Box

Returns the Cell used to draw the title

Sets the Font of the title to *fontObj*

Returns the Font used to draw the title

Setting and Placing the Content View

- **setContentView:***aView*
 - **contentView**
 - **setOffsets:**(NXCoord)*w* :(NXCoord)*h*
 - **getOffsets:**(NXSize *)*aSize*
- Replaces the **Box's content view** with *aView*

Returns the **content view**

Sets the distance between the border and the content view

Gets the distance between the border and the content view

Putting Views in the Box

- **addSubview:***aView*
 - **replaceSubview:***aView with:anotherView*
- Adds *aView* as a subview of the content view

Replaces *aView* with *anotherView* within the content view

Resizing the Box

- setFrameFromContentFrame:(const NXRect *)*contentFrame*
Resizes the Box to accommodate *contentFrame*
- sizeTo:(NXCoord)*width* :(NXCoord)*height*
Resizes the Box to *width* and *height*
- sizeToFit
Resizes the Box to exactly enclose its subviews

Drawing the Box

- drawSelf:(const NXRect *)*rects* :(int)*rectCount* Draws the Box

Archiving

- awake
Lays out the graphic elements of the Box
- read:(NXTypedStream *)*stream*
Reads the Box object from the typed stream
- write:(NXTypedStream *)*stream*
Writes the Box object to the typed stream

Button

Inherits From: Control : View : Responder : Object

Initializing the Button Factory

- + setCellClass:*classId*
Sets the subclass of ButtonCell used by Button

Initializing a Button

- init
Initializes a new Button with title ^aButton^o
- initWithFrame:(const NXRect *)*frameRect*
Initializes a new Button within *frameRect*

- **initWithFrame:**(const NXRect *)*frameRect*
icon:(const char *)*iconName*
tag:(int)*anInt*
target:*anObject*
action:(SEL)*aSelector*
key:(unsigned short)*charCode*
enabled:(BOOL)*flag*
- **initWithFrame:**(const NXRect *)*frameRect*
title:(const char *)*aString*
tag:(int)*anInt*
target:*anObject*
action:(SEL)*aSelector*
key:(unsigned short)*charCode*
enabled:(BOOL)*flag*

Initializes a new Button within *frameRect*, with the NXImage named *iconName* as its icon, *anInt* as its tag, *anObject* as its target object, *aSelector* as its action message, a key equivalent of *charCode*, and enabled according to *flag*

Initializes a new Button within *frameRect*, with *aString* as its title, *anInt* as its tag, *anObject* as its target object, *aSelector* as its action message, a key equivalent of *charCode*, and enabled according to *flag*

Setting the Button Type

- **setType:**(int)*aType*

Sets how the Button highlights and shows its state

Setting the State

- **setState:**(int)*value*
- (int)**state**

Sets the Button's state to *value* (0 or 1)

Returns the Button's current state (0 or 1)

Setting the Repeat Interval

- **setPeriodicDelay:**(float)*delay*
andInterval:(float)*interval*
- **getPeriodicDelay:**(float *)*delay*
andInterval:(float *)*interval*

Sets repeat parameters for continuous Buttons

Gets repeat parameters for continuous Buttons

Setting the Titles

- **setTitle:**(const char *)*aString*

Makes *aString* the Button's title

- **setTitleNoCopy:**(const char *)*aString*
- (const char *)**title**
- **setAltTitle:**(const char *)*aString*
- (const char *)**altTitle**

Makes *aString* the Button's title without copying it

Returns the Button's title

Makes *aString* the Button's alternate title

Returns the Button's alternate title

Setting the Icons

- **setIcon:**(const char *)*iconName*
- **setIcon:**(const char *)*iconName*
position:(int)*aPosition*
- (const char *)**icon**
- **setAltIcon:**(const char *)*iconName*
- (const char *)**altIcon**
- **setImage:***image*
- **image**
- **setAltImage:***altImage*
- **altImage**
- **setIconPosition:**(int)*aPosition*
- (int)**iconPosition**

Makes the NXImage named *iconName* the Button's icon

Sets the icon by name, and its position

Returns the name of the Button's icon

Makes the NXImage named *iconName* the alternate icon

Returns the name of the Button's alternate icon

Makes the NXImage *image* the Button's icon

Returns the Button's image

Makes the NXImage *image* the alternate icon

Returns the Button's alternate image

Sets the position of the Button's icon

Returns the position of the Button's icon

Modifying Graphic Attributes

- **setTransparent:**(BOOL)*flag*
- (BOOL)**isTransparent**
- **setBordered:**(BOOL)*flag*
- (BOOL)**isBordered**

Sets whether the Button is transparent

Returns whether the Button is transparent

Sets whether the Button has a beveled border

Returns whether the Button has a beveled border

Displaying

- **display**
- **highlight:**(BOOL)*flag*

Displays the Button

Highlights (or unhighlights) the Button according to *flag*

Setting the Key Equivalent

- **setKeyEquivalent:**(unsigned short)*charCode* Makes *charCode* the Button's key equivalent
- (unsigned short)**keyEquivalent** Returns the Button's key equivalent

Handling Events and Action Messages

- (BOOL)**acceptsFirstMouse** Ensures that the Button accepts first mouse-down
- **performClick:***sender* Simulates the user clicking the Button
- (BOOL)**performKeyEquivalent:**(NXEvent *)*theEvent* Simulates a mouse click, if the key is right

Setting the Sound

- **setSound:***soundObject* Sets the Sound played when the Button is pressed
- **sound** Returns the Sound played when the Button is pressed

ButtonCell

Inherits From: ActionCell : Cell : Object

Initializing, Copying, and Freeing a ButtonCell

- **init** Initializes a new ButtonCell with title ^aButton^o
- **initTextCell:**(const char *)*aString* Initializes a new ButtonCell with title *aString*
- **initIconCell:**(const char *)*iconName* Initializes a new ButtonCell with an NXImage named *iconName* as its icon
- **copyFromZone:**(NXZone *)*zone* Returns a copy of the ButtonCell allocated from *zone*
- **free** Deallocates the ButtonCell

Determining Component Sizes

- **calcCellSize:**(NXSize *)*theSize*
 inRect:(const NXRect *)*aRect*
- **getDrawRect:**(NXRect *)*theRect*
- **getTitleRect:**(NXRect *)*theRect*
- **getIconRect:**(NXRect *)*theRect*

Calculates and returns the size of the ButtonCell

Returns the rectangle the ButtonCell draws in

Returns the rectangle the title is drawn in

Returns the rectangle the icon is drawn in

Setting the Titles

- **setTitle:**(const char *)*aString*
- **setTitleNoCopy:**(const char *)*aString*
- (const char *)**title**
- **setAltTitle:**(const char *)*aString*
- (const char *)**altTitle**
- **setFont:***fontObject*

Makes a copy of *aString* the ButtonCell's title

Makes *aString* the ButtonCell's title without copying it

Returns the ButtonCell's title

Makes a copy of *aString* the ButtonCell's alternate title

Returns the ButtonCell's alternate title

Sets the Font used to draw the title

Setting the Icons

- **setIcon:**(const char *)*iconName*
- (const char *)**icon**
- **setAltIcon:**(const char *)*iconName*
- (const char *)**altIcon**
- **setImage:***image*
- **image**
- **setAltImage:***altImage*
- **altImage**
- **setIconPosition:**(int)*aPosition*
- (int)**iconPosition**

Makes the NXImage named *iconName* the ButtonCell's icon

Returns the name of the ButtonCell's icon

Makes the NXImage named *iconName* the ButtonCell's alternate icon

Returns the name of the ButtonCell's alternate icon

Makes the NXImage object *image* the ButtonCell's icon

Returns the ButtonCell's icon

Makes the NXImage object *image* the alternate icon

Returns the ButtonCell's alternate icon

Sets the position of the ButtonCell's icon to *aPosition*

Returns the position of the ButtonCell's icon

Setting the Sound

- **setSound:***aSound*
- **sound**

Sets the Sound played by the ButtonCell on a mouse-down

Returns the Sound played by the ButtonCell

Setting the State

- **setDoubleValue:**(double)*aDouble*
- (double)**doubleValue**
- **setFloatValue:**(float)*aFloat*
- (float)**floatValue**
- **setIntValue:**(int)*anInt*
- (int)**intValue**
- **setStringValue:**(const char *)*aString*
- **setStringValueNoCopy:**(const char *)*aString*
- (const char *)**stringValue**

Sets the ButtonCell's state (value) to *aDouble*

Returns the ButtonCell's state as a **double**

Sets the ButtonCell's state (value) to *aFloat*

Returns the ButtonCell's state as a **float**

Sets the ButtonCell's state (value) to *anInt*

Returns the ButtonCell's state as an **int**

Sets the ButtonCell's state (value) to a copy of *aString*

Sets the ButtonCell's state (value) to *aString*

Returns the ButtonCell's state as a string

Setting the Repeat Interval

- **setPeriodicDelay:**(float)*delay*
 andInterval:(float)*interval*
- **getPeriodicDelay:**(float *)*delay*
 andInterval:(float *)*interval*

Sets repeat parameters for continuous ButtonCells

Gets repeat parameters for continuous ButtonCells

Tracking the Mouse

- (BOOL)**trackMouse:**(NXEvent *)*theEvent*
 inRect:(const NXRect *)*cellFrame*
 ofView:*controlView*

Plays the Sound, then tracks the mouse

Setting the Key Equivalent

- **setKeyEquivalent:**(unsigned short)*charCode* Sets the ButtonCell's key equivalent
- **setKeyEquivalentFont:***fontObj* Sets the Font used to draw the key equivalent
- **setKeyEquivalentFont:**(const char *)*fontName* **size:**(float)*fontSize* Sets the Font and size used to draw the key equivalent
- (unsigned short)**keyEquivalent** Returns the ButtonCell's key equivalent

Setting Parameters

- **setParameter:**(int)*aParameter to:*(int)*value* Sets various flag values
- (int)**getParameter:**(int)*aParameter* Returns various flag values

Modifying Graphic Attributes

- **setBordered:**(BOOL)*flag* Sets whether the ButtonCell has a bezeled border
- (BOOL)**isBordered** Returns whether the ButtonCell has a bezeled border
- **setTransparent:**(BOOL)*flag* Sets whether the ButtonCell is transparent
- (BOOL)**isTransparent** Returns whether the ButtonCell is transparent
- (BOOL)**isOpaque** Returns whether receiver is opaque

Modifying Graphic Attributes

- **setType:**(int)*aType* Sets the ButtonCell's display behavior
- **setHighlightsBy:**(int)*aType* Sets how the ButtonCell highlights
- (int)**highlightsBy** Returns how the ButtonCell highlights
- **setShowsStateBy:**(int)*aType* Sets how the ButtonCell shows its alternate state
- (int)**showsStateBy** Returns how ButtonCell shows its alternate state

Simulating a Click

- **performClick:***sender* Simulates clicking the ButtonCell

Displaying

- drawInside: (const NXRect *) <i>aRect</i> inView: <i>controlView</i>	Draws the inside of the ButtonCell
- drawSelf: (const NXRect *) <i>cellFrame</i> inView: <i>controlView</i>	Draws the ButtonCell
- highlight: (const NXRect *) <i>cellFrame</i> inView: <i>controlView</i> lit: (BOOL) <i>flag</i>	Highlights the ButtonCell

Archiving

- read: (NXTypedStream *) <i>stream</i>	Reads the ButtonCell from <i>stream</i>
- write: (NXTypedStream *) <i>stream</i>	Writes the ButtonCell to <i>stream</i>

Cell

Inherits From: Object

Initializing, Copying, and Freeing a Cell

- init	Initializes a new Cell
- initWithIconCell: (const char *) <i>iconName</i>	Initializes a new Cell with the NXImage named <i>iconName</i>
- initWithTextCell: (const char *) <i>aString</i>	Initializes a new Cell with title <i>aString</i>
- copyFromZone: (NXZone *) <i>zone</i>	Returns a copy of the receiving Cell from <i>zone</i>
- free	Deallocates the Cell

Determining Component Sizes

- calcCellSize: (NXSize *) <i>theSize</i>	Returns the minimum size needed to display the Cell
--	---

- **calcCellSize:**(NXSize *)*theSize*
 inRect:(const NXRect *)*aRect*
- **calcDrawInfo:**(const NXRect *)*aRect*
- **getDrawRect:**(NXRect *)*theRect*
- **getIconRect:**(NXRect *)*theRect*
- **getTitleRect:**(NXRect *)*theRect*

Returns the minimum size needed to display the Cell

Implemented by subclasses to recalculate drawing sizes

Returns the rectangle the Cell draws in

Returns the rectangle that an icon is drawn in

Returns the rectangle that a title is drawn in

Setting the Cell's Type

- **setType:**(int)*aType*
- (int)**type**

Sets the Cell's type to *aType*

Returns the Cell's type

Setting the Cell's State

- **setState:**(int)*value*
- **incrementState**
- (int)**state**

Sets the state of the Cell to *value* (0 or 1)

Increments the state of the Cell

Returns the state of the Cell (0 or 1)

Enabling and Disabling the Cell

- **setEnabled:**(BOOL)*flag*
- (BOOL)**isEnabled**

Sets whether the Cell reacts to mouse events

Returns whether the Cell reacts to mouse events

Setting the Icon

- **setIcon:**(const char *)*iconName*
- (const char *)**icon**

Sets the Cell's icon to the NXImage named *iconName*

Returns the name of the Cell's icon

Setting the Cell's Value

- **setDoubleValue:**(double)*aDouble*
- (double)**doubleValue**

Sets the Cell's value to *aDouble*

Returns the Cell's value as a **double**

- setFloatValue: (float) <i>aFloat</i>	Sets the Cell's value to <i>aFloat</i>
- (float) floatValue	Returns the Cell's value as a float
- setIntValue: (int) <i>anInt</i>	Sets the Cell's value to <i>anInt</i>
- (int) intValue	Returns the Cell's value as an int
- setStringValue: (const char *) <i>aString</i>	Sets the Cell's value to a copy of <i>aString</i>
- setStringValueNoCopy: (const char *) <i>aString</i>	Sets the Cell's value to <i>aString</i>
- setStringValueNoCopy: (char *) <i>aString</i> shouldFree: (BOOL) <i>flag</i>	Sets the Cell's value to <i>aString</i> ; will free the string when freed if <i>flag</i> is YES
- (const char *) stringValue	Returns the Cell's value as a string

Interacting with Other Cells

- takeDoubleValueFrom: <i>sender</i>	Sets the Cell's value to <i>sender</i> 's doubleValue
- takeFloatValueFrom: <i>sender</i>	Sets the Cell's value to <i>sender</i> 's floatValue
- takeIntValueFrom: <i>sender</i>	Sets the Cell's value to <i>sender</i> 's intValue
- takeStringValueFrom: <i>sender</i>	Sets the Cell's value to <i>sender</i> 's stringValue

Modifying Text Attributes

- setAlignment: (int) <i>mode</i>	Sets the alignment of text in the Cell to <i>mode</i>
- (int) alignment	Returns the alignment of text in the Cell
- setFont: <i>fontObject</i>	Sets the Font used to display text in the Cell to <i>fontObject</i>
- font	Returns the Font used to display text in the Cell
- setEditable: (BOOL) <i>flag</i>	Sets whether the Cell's text is editable
- (BOOL) isEditable	Returns whether the Cell's text is editable
- setSelectable: (BOOL) <i>flag</i>	Sets whether the Cell's text is selectable
- (BOOL) isSelectable	Returns whether the Cell's text is selectable
- setScrollable: (BOOL) <i>flag</i>	Sets whether the Cell scrolls to follow typing
- (BOOL) isScrollable	Returns whether the Cell scrolls to follow typing
- setTextAttributes: <i>textObject</i>	Sets Text parameters for drawing or editing
- setWrap: (BOOL) <i>flag</i>	Sets whether the Cell's text is word-wrapped

Editing Text

- **edit:**(const NXRect *)*aRect*
 inView:*aView*
 editor:*textObject*
 delegate:*anObject*
 event:(NXEvent *)*theEvent*
Allows text editing in response to a mouse-down event
- **endEditing:***textObject*
Ends any text editing occurring in the Cell
- **select:**(const NXRect *)*aRect*
 inView:*aView*
 editor:*textObject*
 delegate:*anObject*
 start:(int)*selStart*
 length:(int)*selLength*
Allows text selection in response to a mouse-down event

Validating Input

- **setEntryType:**(int)*aType*
Sets the type of data the user can type into the Cell
- (int)**entryType**
Returns the type of data the user can type into the Cell
- (BOOL)**isEntryAcceptable:**(const char *)*aString*
Returns whether *aString* is acceptable for the entry type

Formatting Data

- **setFloatingPointFormat:**(BOOL)*autoRange*
 left:(unsigned)*leftDigits*
 right:(unsigned)*rightDigits*
Sets the display format for floating point values

Modifying Graphic Attributes

- **setBezeled:**(BOOL)*flag*
Sets whether the Cell has a bezeled border
- (BOOL)**isBezeled**
Returns whether the Cell has a bezeled border
- **setBordered:**(BOOL)*flag*
Sets whether the Cell has a plain border

- (BOOL)**isBordered** Returns whether Cell has a plain border
- (BOOL)**isOpaque** Returns whether the Cell is opaque

Setting Parameters

- **setParameter:(int)aParameter to:(int)value** Sets various Cell flags
- (int)**getParameter:(int)aParameter** Returns various Cell flag values

Displaying

- **controlView** Implemented by subclasses to return the View last drawn in
- **drawInside:(const NXRect *)cellFrame**
inView:aView Draws the area within the Cell's border in *aView*
- **drawSelf:(const NXRect *)cellFrame**
inView:aView Draws the Cell in *aView*
- **highlight:(const NXRect *)cellFrame**
inView:aView
lit:(BOOL)flag Highlights the Cell according to *flag* in *aView*
- (BOOL)**isHighlighted** Returns whether the Cell is highlighted

Target and Action

- **setAction:(SEL)aSelector** Implemented by subclasses to set the action method
- (SEL)**action** Implemented by subclasses to return the action method
- **setTarget:anObject** Implemented by subclasses to set the target object
- **target** Implemented by subclasses to return the target object
- **setContinuous:(BOOL)flag** Sets whether the Cell continuously sends action
- (BOOL)**isContinuous** Returns whether the Cell continuously sends action
- (int)**sendActionOn:(int)mask** Determines when the action is sent while tracking

Assigning a Tag

- **setTag:**(int)*anInt*
- (int)**tag**

Implemented by subclasses to set an identifier tag

Implemented by subclasses to return the identifier tag

Handling Keyboard Alternatives

- (unsigned short)**keyEquivalent**

Implemented by subclasses to return a key equivalent

Tracking the Mouse

- + (BOOL)**prefersTrackingUntilMouseUp**

Returns NO, so tracking stops when the mouse leaves the Cell; subclasses may override

- (int)**mouseDownFlags**

Returns the event flags set at the start of mouse tracking

- **getPeriodicDelay:**(float*)*delay*
andInterval:(float*)*interval*

Returns repeat values for continuous sending of the action

- (BOOL)**trackMouse:**(NXEvent *)*theEvent*
inRect:(const NXRect *)*cellFrame*
ofView:*aView*

Controls tracking behavior of the Cell

- (BOOL)**startTrackingAt:**(const NXPoint *)*startPoint*
inView:*aView*

Determines whether tracking should begin based on *startPoint* within *aView*

- (BOOL)**continueTracking:**(const NXPoint *)*lastPoint*
at:(const NXPoint *)*currentPoint*
inView:*aView*

Returns whether tracking should continue based on *lastPoint* and *currentPoint* within *aView*

- **stopTracking:**(const NXPoint *)*lastPoint*
at:(const NXPoint *)*stopPoint*
inView:*aView*
mouseUp:(BOOL)*flag*

Allows the Cell to update itself to end tracking, based on *lastPoint*, *stopPoint*, within *aView*; *flag* is YES if the this method was invoked because mouse went up

Managing the Cursor

- **resetCursorRect:**(const NXRect *)*cellFrame*
inView:*aView*

Sets text Cells to show I-beam cursor

Archiving

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

Reads the Cell from *stream*

Writes the Cell to *stream*

Reinitializes the Cell after being read

ClipView

Inherits From: View : Responder : Object

Initializing the Class

- + **initialize**

Initializes the ClipView class

Initializing and Freeing a ClipView

- **initWithFrame:**(const NXRect *)*frameRect*
- **free**

Initializes a new ClipView instance

Releases the ClipView's storage

Modifying the Frame Rectangle

- **moveTo:**(NXCoord)x :(NXCoord)y
- **rotateTo:**(NXCoord)*angle*
- **sizeTo:**(NXCoord)*width* :(NXCoord)*height*

Moves the origin of the frame rectangle

Overridden to disable rotation

Resizes the ClipView's frame

Modifying the Coordinate System

- **rotate:**(NXCoord)*angle*
- **scale:**(NXCoord)x :(NXCoord)y
- **setDrawOrigin:**(NXCoord)x :(NXCoord)y
- **setDrawRotation:**(NXCoord)*angle*

Overridden to disable rotation

Rescales the coordinate system

Sets the origin of the coordinate system

Disables rotation of the coordinate system

- **setSize:(NXCoord)width :(NXCoord)height**Scales the coordinate system
- **translate:(NXCoord)x :(NXCoord)y** Shifts the coordinate system

Managing Component Views

- **docView** Returns the ClipView's document view
- **setDocView:aView** Makes *aView* the ClipView's document view
- **getDocRect:(NXRect *)aRect** Returns the document rectangle
- **getDocVisibleRect:(NXRect *)aRect** Gets the visible portion of the document view
- **resetCursorRects** Resets the cursor rectangle for the document view
- **setDocCursor:anObj** Sets the cursor for the document view

Modifying Graphic Attributes and Displaying

- (float)**backgroundGray** Returns the ClipView's background gray
- **setBackgroundGray:(float)value** Sets the ClipView's background gray
- (NXColor)**backgroundColor** Returns the ClipView's background color
- **setBackgroundColor:(NXColor)color** Sets the ClipView's background color
- **drawSelf:(const NXRect *)rects :(int)rectCount** Fills the background gray where needed

Scrolling

- **autoscroll:(NXEvent *)theEvent** Scrolls in response to mouse-dragged events
- **constrainScroll:(NXPoint *)newOrigin** Prevents scrolling to an undesirable position
- **rawScroll:(const NXPoint *)newOrigin** Lowest-level unconstrained scrolling routine
- **setCopyOnScroll:(BOOL)flag** Sets how the visible areas are redrawn
- **setDisplayOnScroll:(BOOL)flag** Sets how the document view is displayed during scrolling

Coordinating with Other Views

- **descendantFlipped:sender** Notification that the document's orientation has changed

- **descendantFrameChanged:***sender*

Notification that the document's frame has changed

Archiving

- **awake**

Initializes the ClipView after unarchiving

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

Reads the ClipView from the typed stream

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

Writes the ClipView to the typed stream

Implemented by ClipView's Superview

- **reflectScroll:***aClipView*

Notifies the superview to update indicators

- **scrollClip:***aClipView to:(const NXPoint *)aPoint*

Notifies the superview of a scroll

Control

Inherits From: View : Responder : Object

Initializing and Freeing a Control

- **initWithFrame:**(const NXRect *)*frameRect*

Initializes a new Control

- **free**

Deallocates the Control

Setting the Control's Cell

+ **setCellClass:***classId*

Implemented by subclasses to set the Cell class used

- **setCell:***aCell*

Sets the Control's Cell to *aCell*

- **cell**

Returns the Control's Cell

Enabling and Disabling the Control

- **setEnabled:**(BOOL)*flag*
- (BOOL)**isEnabled**

Sets whether the Control reacts to mouse events

Returns whether the Control reacts to mouse events

Identifying the Selected Cell

- **selectedCell**
- (int)**selectedTag**

Returns the Control's selected Cell

Returns the tag of the Control's selected Cell

Setting the Control's Value

- **setDoubleValue:**(double)*aDouble*
- (double)**doubleValue**
- **setFloatValue:**(float)*aFloat*
- (float)**floatValue**
- **setIntValue:**(int)*anInt*
- (int)**intValue**
- **setStringValue:**(const char *)*aString*
- **setStringValueNoCopy:**(const char *)*aString*
- **setStringValueNoCopy:**(char *)*aString*
 shouldFree:(BOOL)*flag*
- (const char *)**stringValue**

Sets the Control's value to *aDouble*

Returns the Control's value as a **double**

Sets the Control's value to *aFloat*

Returns the Control's value as a **float**

Sets the Control's value to *anInt*

Returns the Control's value as an **int**

Sets the Control's value to *aString*

Sets the Control's value to *aString*

Sets the Control's value to *aString*

Returns the Control's value as a string

Interacting with Other Controls

- **takeDoubleValueFrom:***sender*
- **takeFloatValueFrom:***sender*
- **takeIntValueFrom:***sender*
- **takeStringValueFrom:***sender*

Sets the Control's value to *sender*'s **doubleValue**

Sets the Control's value to *sender*'s **floatValue**

Sets the Control's value to *sender*'s **intValue**

Sets the Control's value to *sender*'s **stringValue**

Formatting Text

- **setAlignment:**(int)*mode* Sets the alignment of text in the Control to *mode*
- (int)**alignment** Returns the alignment of text in the Control
- **setFont:***fontObject* Sets the Font used to draw text in the Control to *fontObject*
- **font** Returns the Font used to draw text in the Control
- **setFloatingPointFormat:**(BOOL)*autoRange* Sets the display format for floating point values
- left:**(unsigned)*leftDigits*
- right:**(unsigned)*rightDigits*

Managing the Field Editor

- **abortEditing** Aborts editing of text displayed by the Control
- **currentEditor** Returns the object used to edit text in the Control
- **validateEditing** Validates the user's changes to editable text

Managing the Cursor

- **resetCursorRects** Sets text-bearing Controls to show an I-beam cursor

Resizing the Control

- **calcSize** Recalculates internal size information
- **sizeTo:**(NXCoord)*width* :(NXCoord)*height* Resizes the Control to *width* and *height*
- **sizeToFit** Resizes the Control to fit its Cell

Displaying the Control and Cell

- **drawCell:***aCell* Redraws *aCell* if it's the Control's Cell
- **drawCellInside:***aCell* Redraws *aCell*'s inside if it's the Control's Cell
- **drawSelf:**(const NXRect *)*rects* :(int)*rectCount* Draws the Control
- **selectCell:***aCell* Selects *aCell* if it's the Control's cell
- **update** Redisplays the Control or marks it for later redisplay

- **updateCell:***aCell*
- **updateCellInside:***aCell*

Redisplays *aCell* or marks it for redisplay

Redisplays the inside of *aCell* or marks it for redisplay

Target and Action

- **setAction:**(SEL)*aSelector*
- (SEL)**action**
- **setTarget:***anObject*
- **target**
- **setContinuous:**(BOOL)*flag*
- (BOOL)**isContinuous**
- **sendAction:**(SEL)*theAction* **to:***theTarget*
- (int)**sendActionOn:**(int)*mask*

Sets the Control's action method to *aSelector*

Returns the Control's action method

Sets the Control's target object to *anObject*

Returns the Control's target object

Sets whether the Control continuously sends its action

Returns whether the Control continuously sends its action

Has the Application object send *theAction* to *theTarget*

Determines when the action is sent while tracking

Assigning a Tag

- **setTag:**(int)*anInt*
- (int)**tag**

Sets the Control's tag to *anInt*

Returns the Control's tag

Tracking the Mouse

- **ignoreMultiClick:**(BOOL)*flag*
- **mouseDown:**(NXEvent *)*theEvent*
- (int)**mouseDownFlags**

Sets whether multiple clicks are ignored

Handles a mouse-down event in the Control

Returns flags in effect at beginning of tracking

Archiving

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

Reads the Control from *stream*

Writes the Control to *stream*

Font

Inherits From: Object

Initializing the Class Object

+ initialize

Performed automatically at start-up

+ **useFont:**(const char *)*fontName*

Registers that *fontName* is used in the document

Creating and Freeing a Font Object

```
+ newFont:(const char *)fontName  
  size:(float)fontSize
```

Returns the specified Font object

```
+ newFont:(const char *)fontName  
  size:(float)fontSize  
  matrix:(const float *)fontMatrix
```

Returns the specified Font object

```
+ newFont:(const char *)fontName  
  size:(float)fontSize  
  style:(int)fontStyle  
  matrix:(const float *)fontMatrix
```

Returns the specified Font object

```
+ boldSystemFontOfSize:(float)fontSize  
  matrix:(const float *)fontMatrix
```

Returns the Font object representing the bold system font of size *fontSize* and matrix *fontMatrix*

```
+ userFixedPitchFontOfSize:(float)fontSize  
  matrix:(const float *)fontMatrix
```

Returns the Font object representing the application's fixed-pitch font of size *fontSize* and matrix *fontMatrix*

```
+ userFontSize:(float)fontSize  
  matrix:(const float *)fontMatrix
```

Returns the Font object representing the application's standard font of size *fontSize* and matrix *fontMatrix*

```
+ systemFontSize:(float)fontSize  
  matrix:(const float *)fontMatrix
```

Returns the Font object representing the system font of size *fontSize* and matrix *fontMatrix*

- free

Has no effect

Querying the Font Object

- (const float *)**displayName**
- (const float *)**familyName**
- (const char *)**name**
- (int)**fontNum**
- (float)**getWidthOf:**(const char *)*string*
- (BOOL)**hasMatrix**
- (const float *)**matrix**
- (NXFontMetrics *)**metrics**
- (float)**pointSize**
- (NXFontMetrics *)**readMetrics:**(int)*flags*
- **screenFont**
- (int)**style**

Returns the full name of the font

Returns the name of the font's family

Returns the name of the font

Returns the Window Server's font number

Returns the width of *string* in this font

Returns whether font differs from identity matrix

Returns a pointer to the font matrix

Returns pointer to a record of font information

Returns the size of the font in points

Reads *flags* information into the font record

Returns the screen font for this font

Returns the font style

Setting the Font

- **set**
- **setStyle:**(int)*aStyle*
- + **setUserFixedPitchFont:**(Font *)*aFont*
- + **setUserFont:**(Font *)*aFont*

Makes this font the graphic state's current font

Sets the Font's style

Sets the fixed-pitch font used by default in the application

Sets the standard font used by default in the application

Archiving

- **awake**
- **finishUnarchiving**
- **read:**(NXTypedStream *)*stream*
- **write:**(NXTypedStream *)*stream*

Reinitializes the Font object

Checks whether the Font object already exists

Reads the Font object from the typed stream

Writes the Font object to the typed stream

FontManager

Inherits From: Object

Creating a FontManager

+ new	Returns the application-wide FontManager
--------------	--

Converting Fonts

- | | |
|--|---|
| - convertFont: <i>fontObj</i> | Converts the font in response to changeFont: |
| - convertWeight: (BOOL) <i>upFlag of:fontObj</i> | Raises or lowers the weight of the font |
| - convert: <i>fontObj</i> toFace: (const char *) <i>typeface</i> | Converts the font to the specified typeface |
| - convert: <i>fontObj</i> toFamily: (const char *) <i>family</i> | Converts the font to the specified family |
| - convert: <i>fontObj</i> toSize: (float) <i>size</i> | Converts the font to the specified point size |
| - convert: <i>fontObj</i>
toHaveTrait: (NXFontTraitMask) <i>trait</i> | Converts the font to have the specified trait |
| - convert: <i>fontObj</i>
toNotHaveTrait: (NXFontTraitMask) <i>trait</i> | Converts the font to remove the specified trait |
| - findFont: (const char *) <i>family</i>
traits: (NXFontTraitMask) <i>traits</i>
weight: (int) <i>weight</i>
size: (float) <i>size</i> | Tries to find a font that matches the specified characteristics |
| - getFamily: (const char **) <i>family</i>
traits: (NXFontTraitMask *) <i>traits</i>
weight: (int *) <i>weight</i>
size: (float *) <i>size</i>
ofFont: <i>fontObj</i> | Provides the characteristics of the given <i>fontObj</i> |

Setting Parameters

- **setAction:***(SEL)aSelector* Sets the action sent by the FontManager
- + **setFontPanelFactory:***classId* Sets the class used to create the Font panel
- + **setFontManagerFactory:***classId* Sets the class used to create the font manager
- **setSelfFont:***fontObj* **isMultiple:***(BOOL)flag* Notifies FontManager of selection's current font
- **setEnabled:***(BOOL)flag* Enables and disables the Font panel and menu

Querying Parameters

- **(SEL)action** Gets the action sent by the FontManager
- **(char **)availableFonts** Provides a list of all available fonts
- **getFontMenu:***(BOOL)create* Returns the Font menu
- **getFontPanel:***(BOOL)create* Returns the Font panel
- **(BOOL)isMultiple** Returns whether selection contains multiple fonts
- **selfFont** Returns the first font in the current selection
- **(BOOL)isEnabled** Returns whether the Font panel and menu are enabled

Target and Action Methods

- **modifyFont:***sender* Converts current selection's font
- **addFontTrait:***sender* Causes trait to be added to font in current selection
- **removeFontTrait:***sender* Causes trait to be removed from font in current selection
- **modifyFontViaPanel:***sender* Converts font according to Font panel settings
- **orderFrontFontPanel:***sender* Orders the FontPanel front
- **sendAction** Dispatches *action* message up responder chain

Assigning a Delegate

- **setDelegate:***anObject* Sets the FontManager's delegate

- **delegate:** Returns the FontManager's delegate

Archiving the FontManager

- **finishUnarchiving** Finishes unarchiving by creating the FontManager

FontPanel

Inherits From: Panel : Window : Responder : Object

Creating a FontPanel

+ **new** Returns a FontPanel object

+ **newContent:(const NXRect *)contentRect** Returns a FontPanel object
 style:(int)aStyle
 backing:(int)bufferingType
 buttonMask:(int)mask
 defer:(BOOL)flag

Setting the Font

- **setPanelFont:fontObj isMultiple:(BOOL)flag** Sets the Font panel's current font
- **panelConvertFont:fontObj** Converts *fontObj* to the user's choice from the panel

Configuring the FontPanel

- **accessoryView** Returns the application-customized view
- **setAccessoryView:aView** Adds application-customized View to the FontPanel
- **setEnabled:(BOOL)flag** Enables and disables the FontPanel's Set button
- **(BOOL)isEnabled** Returns whether the FontPanel's Set button is enabled

- (BOOL)**worksWhenModal** Returns whether FontPanel works when another window is modal

Editing the FontPanel's Fields

- **textDidEnd:***textObject* Detects completion of size field editing
 endChar:(unsigned short)*endChar*
- **textDidGetKeys:***textObject* **isEmpty:**(BOOL)*flag* Detects empty size field

Displaying the FontPanel

- **orderWindow:**(int)*place* **relativeTo:**(int)*otherWin* Repositions panel and updates it if necessary

Resizing the FontPanel

- **windowWillResize:***sender* Constrains FontPanel resizing
 toSize:(NXSize *)*frameSize*

Form

Inherits From: Matrix : Control : View : Responder : Object

Setting Form's Cell Class

+ **setCellClass:***classId* Sets the subclass of Cell used by Form

Initializing a Form

- **initWithFrame:**(const NXRect *)*frameRect* Initializes a new Form in *frameRect*

Laying Out the Form

- **addEntry:**(const char *)*title*
Adds a new entry with *title* as its title at the end of the Form
- **addEntry:**(const char *)*title*
 tag:(int)*anInt*
 target:*anObject*
 action:(SEL)*aSelector*
Adds a new entry with *title* as its title at the end of the Form and sets its tag, target, and action
- **insertEntry:**(const char *)*title* **at:**(int)*index*
Inserts a new entry with *title* as its title at *index*
- **insertEntry:**(const char *)*title*
 at:(int)*index*
 tag:(int)*anInt*
 target:*anObject*
 action:(SEL)*aSelector*
Inserts a new entry with *title* as its title at *index* and sets its tag, target, and action
- **removeEntryAt:**(int)*index*
Removes the entry at *index*
- **setInterline:**(NXCoord)*spacing*
Sets the spacing between entries

Assigning a Tag

- **setTag:**(int)*anInt* **at:**(int)*index*
Sets the tag of the entry at *index* to *anInt*

Finding Indices

- (int)**findIndexWithTag:**(int)*aTag*
Returns the index for the entry with tag *aTag*
- (int)**selectedIndex**
Returns the index of the currently selected entry

Modifying Graphic Attributes

- **setBezeled:**(BOOL)*flag*
Sets whether entries have a bezeled border
- **setBordered:**(BOOL)*flag*
Sets whether the entries have a plain border
- **setFont:***fontObject*
Sets the Font used to draw both titles and text
- **setTitleFont:***fontObject*
Sets the Font used to draw entry titles
- **setTextFont:***fontObject*
Sets the Font used to draw entry text

- **setTitleAlignment:**(int)*mode* Sets how titles are aligned
- **setTextAlignment:**(int)*mode* Sets how text is aligned within the entries

Setting Item Titles

- **setTitle:**(const char *)*aString* **at:**(int)*index* Sets the title of the entry at *index* to *aString*
- (const char *)**titleAt:**(int)*index* Returns the title of the entry at *index*

Setting Item Values

- **setDoubleValue:**(double)*aDouble* **at:**(int)*index* Sets the value of the entry at *index* to *aDouble*
- (double)**doubleValueAt:**(int)*index* Returns the value of the entry at *index* as a **double**
- **setFloatValue:**(float)*aFloat* **at:**(int)*index* Sets the value of the entry at *index* to *aFloat*
- (float)**floatValueAt:**(int)*index* Returns the value of the entry at *index* as a **float**
- **setIntValue:**(int)*anInt* **at:**(int)*index* Sets the value of the entry at *index* to *anInt*
- (int)**intValueAt:**(int)*index* Returns the value of the entry at *index* as an **int**
- **setStringValue:**(const char *)*aString* **at:**(int)*index* Sets the value of the entry at *index* to a *aString*
- (const char *)**stringValueAt:**(int)*index* Returns the value of the entry at *index* as a string

Editing Text

- **selectTextAt:**(int)*index* Selects the text in the entry at *index*

Resizing the Form

- **calcSize** Recalculates title positions in the Form
- **setEntryWidth:**(NXCoord)*width* Sets the width of all the entries
- **sizeTo:**(NXCoord)*width* :(NXCoord)*height* Resizes the Form and updates the widths of its entries
- **sizeToFit** Modifies the Form's frame to fit its entries

Displaying

- **drawCellAt:**(int)*index* Displays the Cell at the specified *index*

Target and Action

- **setAction:**(SEL)*aSelector* **at:**(int)*index* Sets the action method of the entry at *index* to *aSelector*
- **setTarget:***anObject* **at:**(int)*index* Sets the target object of the entry at *index* to *anObject*

FormCell

Inherits From: ActionCell : Cell : Object

Initializing, Copying, and Freeing a FormCell

- **init** Initializes a new FormCell with ^aField^o as its title
- **initTextCell:**(const char *)*aString* Initializes a new FormCell with *aString* as its title
- **copyFromZone:**(NXZone *)*zone* Returns a copy of the FormCell allocated from *zone*
- **free** Deallocates the FormCell

Determining a FormCell's Size

- **calcCellSize:**(NXSize *)*theSize*
 inRect:(const NXRect *)*aRect* Calculates the FormCell's size within *aRect*

Enabling the FormCell

- **setEnabled:**(BOOL)*flag* Sets whether the FormCell reacts to events

Modifying the Title

- setTitle: (const char *) <i>aString</i>	Sets the FormCell's title to <i>aString</i>
- (const char *) title	Returns the FormCell's title
- setTitleFont: <i>fontObject</i>	Sets the Font used to draw the title
- titleFont	Returns the Font used to draw the title
- setTitleAlignment: (int) <i>mode</i>	Sets the alignment of the title
- (int) titleAlignment	Returns the alignment of the title
- setTitleWidth: (NXCoord) <i>width</i>	Sets the width of the FormCell's title field to <i>width</i>
- (NXCoord) titleWidth: (const NXSize *) <i>aSize</i>	Returns the width of the title, constrained to <i>aSize</i>
- (NXCoord) titleWidth	Returns the width of the title

Modifying Graphic Attributes

- (BOOL) isOpaque	Returns whether the FormCell is opaque
--------------------------	--

Displaying

- drawInside: (const NXRect *) <i>cellFrame</i> inView: <i>controlView</i>	Draws the editable text portion of the cell
- drawSelf: (const NXRect *) <i>cellFrame</i> inView: <i>controlView</i>	Draws the entire FormCell

Managing Cursor Rectangles

- resetCursorRect: (const NXRect *) <i>cellFrame</i> inView: <i>controlView</i>	Resets the cursor rectangle so that the cursor becomes an I-beam when over the editable portion of the FormCell
--	---

Tracking the Mouse

- (BOOL) trackMouse: (NXEvent*) <i>event</i> inRect: (const NXRect*) <i>aRect</i> ofView: <i>controlView</i>	Overrides Cell's method to allow editing
---	--

Archiving

- **read:**(NXTypedStream *)*stream* Reads the FormCell from *stream*
- **write:**(NXTypedStream *)*stream* Writes the FormCell to *stream*

Listener

Inherits From: Object

Initializing the Class

- + **initialize** Sets up a table of understood messages

Initializing a New Listener Instance

- **init** Initializes the Listener after it's allocated

Freeing a Listener

- **free** Deallocates the Listener and its ports

Setting Up a Listener

- **addPort** Sets procedure to receive messages at port
- **removePort** Removes procedure that receives messages
- (int)**checkInAs:**(const char *)*name* Allocates a port and registers it as *name*
- (int)**usePrivatePort** Allocates a port but doesn't register it
- (int)**checkOut** Unregisters the port, making it private
- (port_t)**listenPort** Returns the Listener's port
- (port_t)**signaturePort** Returns the port used to validate the Listener

- (const char *)**portName**
- **setPriority:**(int)*level*
- (int)**priority**
- **setTimeout:**(int)*ms*
- (int)**timeout**
- + **run**

Returns registered name of the Listener's port
 Sets the priority for receiving messages to *level*
 Returns priority level for receiving messages
 Sets how long to wait on sending reply
 Returns how long to wait on sending reply
 Enables Listener in absence of an Application object

Providing for Program Control

- (int)**msgCalc:**(int *)*flag*
- (int)**msgCopyAsType:**(const char *)*aType*
ok:(int *)*flag*
- (int)**msgCutAsType:**(const char *)*aType*
ok:(int *)*flag*
- (int)**msgDirectory:**(char *const *)*fullPath*
ok:(int *)*flag*
- (int)**msgFile:**(char *const *)*fullPath*
ok:(int *)*flag*
- (int)**msgPaste:**(int *)*flag*
- (int)**msgPosition:**(char *const *)*aString*
posType:(int *)*anInt*
ok:(int *)*flag*
- (int)**msgPrint:**(const char *)*fullPath*
ok:(int *)*flag*
- (int)**msgQuit:**(int *)*flag*
- (int)**msgSelection:**(char *const *)*bytes*
length:(int *)*numBytes*
asType:(const char *)*aType*
ok:(int *)*flag*
- (int)**msgSetPosition:**(const char *)*aString*
posType:(int *)*anInt*
andSelect:(int)*selectFlag*

Receives message to update the current window
 Receives message to copy the selection
 Receives message to cut selection as *aType* data
 Receives message asking for the current directory
 Receives message asking for the current document
 Receives message to paste data from pasteboard
 Receives message requesting selection information
 Receives message to print the *fullPath* file
 Receives a remote message to quit
 Receives message requesting the current selection
 Receives message to scroll so *aString* is visible

- **ok:**(int *)*flag*
- (int)**msgVersion:**(char *const *)*aString*
ok:(int *)*flag*
 - Receives message requesting version information

Receiving Remote Messages

- **messageReceived:**(NXMessage *)*msg*
 - Receives messages at the Listener's port
- (int)**performRemoteMethod:**(NXRemoteMethod *)*method*
paramList:(NXParamValue *)*params*
 - Performs Listener's remote *method*
- (NXRemoteMethod *)**remoteMethodFor:**(SEL)*aSelector*
 - Looks up remote method for *aSelector*

Assigning a Delegate

- **setDelegate:***anObject*
 - Makes *anObject* the Listener's delegate
- **delegate**
 - Returns the Listener's delegate
- **setServicesDelegate:***anObject*
 - Makes *anObject* the receiver of service requests
- **servicesDelegate**
 - Returns the object that receives service requests

Archiving

- **read:**(NXTypedStream *)*stream*
 - Reads the Listener from *stream*
- **write:**(NXTypedStream *)*stream*
 - Writes the Listener to *stream*

Matrix

Inherits From: Control : View : Responder : Object

Initializing the Matrix Class

+ **initialize**

+ **setCellClass:***classId*

Initializes the Matrix class

Sets the default class used to make Cells

Initializing and Freeing a Matrix

- **initFrame:**(const NXRect *)*frameRect*

Initializes a new Matrix object in *frameRect*

- **initFrame:**(const NXRect *)*frameRect*

Initializes a new Matrix object in *frameRect*,

mode:(int)*aMode*

with *aMode* as the selection mode,

cellClass:*classId*

classId as the class used to make new Cells,

numRows:(int)*numRows*

and having *numRows* rows

numCols:(int)*numCols*

and *numCols* columns

- **initFrame:**(const NXRect *)*frameRect*

Initializes a new Matrix object with the given values

mode:(int)*aMode*

with *aMode* as the selection mode,

prototype:*aCell*

aCell as the prototype copied to make new Cells,

numRows:(int)*numRows*

and having *numRows* rows

numCols:(int)*numCols*

and *numCols* columns

- **free**

Deallocates the Matrix and all its Cells

Setting the Selection Mode

- **setMode:**(int)*aMode*

Sets the selection mode of the Matrix

- (int)**mode**

Returns the selection mode of the Matrix

Configuring the Matrix

- **setEnabled:**(BOOL)*flag*

Sets whether the Matrix reacts to events

- **setEmptySelectionEnabled:**(BOOL)*flag*

Sets whether there may be no Cells selected

- (BOOL)**isEmptySelectionEnabled**

Returns whether there may be no Cells selected

- **setSelectionByRect:**(BOOL)*flag*

Sets whether a user can drag a rectangular selection
(the default is YES)

- (BOOL)**isSelectionByRect**

Returns whether a user can drag a rectangular selection

Setting the Cell class

- **setCellClass:***classId*
- **setPrototype:***aCell*
- **prototype**

Sets the subclass of Cell used to make new Cells

Sets the prototype Cell copied to make new Cells

Returns the prototype Cell copied to make new Cells

Laying Out the Matrix

- **addCol**
- **addRow**
- **insertColAt:**(int)*col*
- **insertRowAt:**(int)*row*
- **removeColAt:**(int)*col* **andFree:**(BOOL)*flag*
- **removeRowAt:**(int)*row* **andFree:**(BOOL)*flag*
- **makeCellAt:**(int)*row* **:**(int)*col*
- **putCell:***newCell* **at:**(int)*row* **:**(int)*col*
- **renewRows:**(int)*newRows* **cols:**(int)*newCols*
- **setCellSize:**(const NXSize *)*aSize*
- **getCellSize:**(NXSize *)*theSize*
- **getCellFrame:**(NXRect *)*theRect*
at:(int)*row*
:(int)*col*
- **setInterCell:**(const NXSize *)*aSize*
- **getInterCell:**(NXSize *)*theSize*
- (int)**cellCount**
- **getNumRows:**(int *)*rowCount*
numCols:(int *)*colCount*

Adds a new column of Cells to the bottom of the Matrix

Adds a new row of Cells to the right of the Matrix

Inserts a new column of Cells at *col*, creating as many as needed to make the Matrix *col* columns wide

Inserts a new row of Cells at *row*, creating as many as needed to make the Matrix *row* rows wide

Removes the column at *col*, freeing the Cells if *flag* is YES

Removes the row at *row*, freeing the Cells if *flag* is YES

Creates a new Cell at *row*, *col* in the Matrix and returns it

Replaces Cell at *row* and *col* with *newCell*; returns old Cell

Changes the number of rows and columns in Matrix without freeing any Cells

Sets the width and height of all Cells in the Matrix

Gets the width and height of Cells in the Matrix

Returns the frame of the Cell at *row* and *col*

Sets the vertical and horizontal spacing between Cells

Gets the vertical and horizontal spacing between Cells

Returns the number of Cells in the Matrix

Gets the number of rows and columns in the Matrix

Finding Matrix Coordinates

- **getRow:(int *)row
andCol:(int *)col
ofCell:aCell** Gets the *row* and *col* position of *aCell*
- **getRow:(int *)row
andCol:(int *)col
forPoint:(const NXPoint *)aPoint** Gets the *row* and *col* position corresponding to *aPoint*, and returns the Cell at that point

Modifying Individual Cells

- **setIcon:(const char *)iconName
at:(int)row
:(int)col** Sets the icon for the Cell at *row* and *col* to the NXImage named *iconName*
- **setState:(int)value at:(int)row :(int)col** Sets the state of the Cell at *row* and *col* to *value*
- **setTitle:(const char *)aString at:(int)row :(int)col** Assigns Cell at *row* and *col* the title *aString*
- **setTag:(int)anInt at:(int)row :(int)col** Assigns the Cell at *row* and *col* the tag *anInt*
- **setTag:(int)anInt
target:anObject
action:(SEL)aSelector
at:(int)row
:(int)col** Assigns a tag, target, and action to the specified Cell

Selecting Cells

- **selectCell:aCell** Selects the Cell *aCell* if it is in the Matrix
- **selectCellAt:(int)row :(int)col** Selects the Cell at *row* and *col*
- **selectCellWithTag:(int)anInt** Selects the Cell with the tag *anInt*
- **setSelectionFrom:(int)startPos
to:(int)endPos
anchor:(int)anchorPos
lit:(BOOL)flag** Selects the Cells in the Matrix from *startPos* to *endPos*, counting in row order from the upper left, as though *anchorPos* were the number of the last Cell selected, and highlighting the Cells according to *flag*
- **selectAll:sender** Selects all the Cells in the Matrix

- selectedCell	Returns the last (lowest and rightmost) selected Cell
- getSelectedCells: (List *) <i>aList</i>	Puts the selected Cells into <i>aList</i> and returns the List
- (int) selectedCol	Returns the column of the selected Cell
- (int) selectedRow	Returns the row of the selected Cell
- clearSelectedCell	Deselects the selected Cell

Finding Cells

- findCellWithTag: (int) <i>anInt</i>	Returns the Cell with <i>anInt</i> as its tag
- cellAt: (int) <i>row</i> :(int) <i>col</i>	Returns the Cell at row <i>row</i> and column <i>col</i>
- cellList	Returns the Matrix's List of Cells

Modifying Graphic Attributes

- setBackgroundColor: (NXColor) <i>aColor</i>	Sets the color of the background between Cells to <i>aColor</i>
- (NXColor) backgroundColor	Returns the color of the background between Cells
- setBackgroundGray: (float) <i>value</i>	Sets the gray of the background between Cells to <i>value</i>
- (float) backgroundGray	Returns the gray of the background between Cells
- setCellBackgroundColor: (NXColor) <i>aColor</i>	Sets the color of the background within Cells to <i>aColor</i>
- (NXColor) cellBackgroundColor	Returns the color of the background within Cells
- setCellBackgroundGray: (float) <i>value</i>	Sets the gray of the background within Cells to <i>value</i>
- (float) cellBackgroundGray	Returns the gray of the background within Cells
- setBackgroundTransparent: (BOOL) <i>flag</i>	Sets whether the background between Cells is transparent
- (BOOL) isBackgroundTransparent	Returns whether the background between Cells is transparent
- setCellBackgroundTransparent: (BOOL) <i>flag</i>	Sets whether the background within Cells is transparent
- (BOOL) isCellBackgroundTransparent	Returns whether the background within Cells is transparent
- setFont: <i>fontObject</i>	Sets the Font used to display text in the Cells
- font	Returns the Font used to display text in the Cells

Editing Text in Cells

- **selectText:***sender*
- **selectTextAt:***(int)row :(int)col*

Selects the text in the first or last editable Cell

Selects the text of the Cell at *row*, *col* in the Matrix

Setting Tab Key Behavior

- **setNextText:***anObject*
- **setPreviousText:***anObject*

Sets the object selected when the user presses Tab while editing the last text Cell

Sets the object selected when user presses Shift-Tab while editing the first text Cell

Assigning a Text Delegate

- **setTextDelegate:***anObject*
- **textDelegate**

Sets the delegate for messages from the field editor

Returns the delegate for messages from the field editor

Text Object Delegate Methods

- (BOOL)**textWillChange:***textObject*
- **textDidChange:***textObject*
- **textDidGetKeys:***textObject*
isEmpty:*(BOOL)flag*
- (BOOL)**textWillEnd:***textObject*
- **textDidEnd:***textObject*
endChar:*(unsigned short)whyEnd*

Responds to a message from the field editor (see Text)

Responds to a message from the field editor (see Text)

Responds to a message from the field editor (see Text)

Responds to a message from the field editor (see Text)

Responds to a message from the field editor (see Text)

Resizing the Matrix and Cells

- **setAutosizeCells:***(BOOL)flag*
- (BOOL)**doesAutosizeCells**
- **calcSize**
- **sizeTo:***(float)width :(float)height*
- **sizeToCells**

Sets whether the Matrix resizes its Cells automatically

Returns whether the Matrix resizes its Cells automatically

Calculates Cell sizes

Resizes the Matrix to *width* and *height*

Resizes the Matrix to fit its Cells exactly

- **sizeToFit** Resizes the Cells and Matrix to fit the Cell contents
- **validateSize:(BOOL)flag** Sets whether the Cell size needs to be recalculated

Scrolling

- **setAutoScroll:(BOOL)flag** Sets whether the Matrix automatically scrolls when dragged in
- **setScrollable:(BOOL)flag** Makes all the Cells scrollable
- **scrollCellToVisible:(int)row :(int)col** Scrolls Matrix so the Cell at *row* and *col* is visible

Displaying

- **display** Draws the Matrix and its Cells
- **drawSelf:(const NXRect *)rects :(int)rectCount** Draws the Matrix and its Cells
- **drawCell:aCell** Draws *aCell* if it's in the Matrix
- **drawCellAt:(int)row :(int)col** Displays the Cell at *row* and *col*
- **drawCellInside:aCell** Draws the inside of *aCell* if it's in the Matrix
- **highlightCellAt:(int)row :(int)col lit:(BOOL)flag** Highlights (or unhighlights) the Cell at *row*, *col*

Target and Action

- **setTarget:anObject** Sets the target of the Matrix to *anObject*
- **target** Returns the target of the Matrix
- **setAction:(SEL)aSelector** Sets the action of the Matrix to *aSelector*
- **(SEL)action** Returns the action of the Matrix
- **setDoubleAction:(SEL)aSelector** Sets the action method used on double-clicks to *aSelector*
- **(SEL)doubleAction** Returns the action method for double clicks
- **setErrorAction:(SEL)aSelector** Sets the action method for editing errors to *aSelector*
- **(SEL)errorAction** Returns the action method for editing errors
- **setTarget:anObject at:(int)row :(int)col** Assigns *anObject* as the target of the Cell at *row*, *col*
- **setAction:(SEL)aSelector at:(int)row :(int)col** Assigns *aSelector* as the action method of the Cell at *row*, *col*
- **sendAction** Sends the selected Cell's action, or the Matrix's action if the Cell doesn't

- **sendAction:**(SEL)*theAction* **to:***theTarget*
- **sendAction:**(SEL)*aSelector*
 to:*anObject*
 forAllCells:(BOOL)*flag*
- **sendDoubleAction**
- **setReaction:**(BOOL)*flag*

have one

Has the Application object send *theAction* to *anObject*

Sends *aSelector* to *anObject*, for all Cells if *flag* is YES

Sends the action corresponding to a double-click

Sets whether sending an action clears the selection

Handling Event and Action Messages

- (BOOL)**acceptsFirstMouse**
- **mouseDown:**(NXEvent *)*theEvent*
- (int)**mouseDownFlags**
- (BOOL)**performKeyEquivalent:**(NXEvent *)*theEvent*

Returns NO only if mode is NX_LISTMODE

Responds to a mouse-down event

Returns the event flags in effect at start of tracking

Simulates mouse click in the appropriate Cell

Managing the Cursor

- **resetCursorRects**

Resets cursor rectangles so that the cursor becomes an I-beam over text Cells

Archiving

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

Reads the Matrix from *stream*

Writes the Matrix to *stream*

Menu

Inherits From: Panel : Window : Responder : Object

Creating a Menu Zone

- + **setMenuZone:**(NXZone *)*zone*
- + (NXZone *)**menuZone**

Sets the zone from which Menus should be allocated

Returns the zone from which Menus should be allocated, creating one if necessary

Initializing a New Menu

- **init**
- **initWithTitle:**(const char *)*aTitle*

Initializes a new Menu with the title *Menu*

Initializes a new Menu with *aTitle* as its title

Setting Up the Menu Commands

- **addItem:**(const char *)*aString*
 action:(SEL)*aSelector*
 keyEquivalent:(unsigned short)*charCode*
- **setItemList:***aMatrix*
- **itemList**

Adds a new item to the end of the Menu

Replaces the current Matrix of items with *aMatrix*

Returns the Menu's Matrix of MenuCell items

Finding Menu Items

- **findCellWithTag:**(int)*aTag*

Returns the MenuCell that has *aTag* as its tag

Building Submenus

- **setSubmenu:***aMenu* **forItem:***aCell*
- **submenuAction:***sender*

Makes *aMenu* a submenu controlled by *aCell*

Activates a submenu attached to *sender*'s Menu

Managing Menu Windows

- **moveTopLeftTo:**(NXCoord)x :(NXCoord)y
- **windowMoved:**(NXEvent *)*theEvent*
- **getLocation:**(NXPoint *)*theLocation*

Moves the Menu's top left corner to x, y

Handles a submenu being torn off its supermenu

Determines where to display an attached submenu

forSubMenu:*aSubmenu*

- **sizeToFit**
- **close**

when it's brought up

Resizes the Menu to exactly fit the command items

Removes the Menu (and any submenus) from the screen

Displaying the Menu

- **display**
- **setAutoupdate:**(BOOL)*flag*
- **update**

Displays the Menu, resizing if needed

Sets whether Menu reacts to **update** messages

Updates each MenuCell item

Handling Events

- **mouseDown:**(NXEvent *)*theEvent*
- **rightMouseDown:**(NXEvent *)*theEvent*

Tracks the cursor in the Menu and submenus

Pops the main menu up under the cursor

Archiving

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

Reads the Menu from *stream*

Writes the Menu to *stream*

Reinitializes a Menu as it's unarchived

MenuCell

Inherits From: ButtonCell : ActionCell : Cell : Object

Initializing a New MenuCell

- **init**
- **initWithCell:**(const char *)*aString*

Initializes a new MenuCell with ^aMenu Item^o as its title

Initializes a new MenuCell with *aString* as its title

Setting the Update Action

- **setUpdateAction:**(SEL)*aSelector*
forMenu:*aMenu*
- (SEL)**updateAction**

Sets the update action for the MenuCell to *aSelector*,
and sets *aMenu* to auto-update
Returns the update action for the MenuCell

Checking for a Submenu

- (BOOL)**hasSubmenu**

Returns whether the MenuCell has a submenu

Tracking the Mouse

- (BOOL)**trackMouse:**(NXEvent *)*theEvent*
inRect:(const NXRect *)*cellFrame*
ofView:*controlView*

Refers mouse tracking to the MenuCell's Menu

Setting User Key Equivalents

- + **useUserKeyEquivalents:**(BOOL)*flag*
- (unsigned short)**userKeyEquivalent**

Sets the class to apply user-assigned key equivalents
Returns the user-assigned key equivalent for the MenuCell

Archiving

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

Reads the MenuCell from *stream*
Writes the MenuCell to *stream*

NXBitmapImageRep

Inherits From: NXImageRep : Object

Initializing a New NXBitmapImageRep object

- **initWithSection:**(const char *)*name* Initializes the new object from TIFF data in the section
- **initWithFile:**(const char *)*filename* Initializes the new object from TIFF data in *filename*
- **initWithStream:**(NXStream *)*stream* Initializes the new object from TIFF data in *stream*
- **initWithData:**(unsigned char *)*data*
 fromRect:(const NXRect *)*rect* Initializes the new object using data read from an image
- **initWithData:**(unsigned char *)*data*
 pixelsWide:(int)*width*
 pixelsHigh:(int)*height*
 bitsPerSample:(int)*bps*
 samplesPerPixel:(int)*spp*
 hasAlpha:(BOOL)*alpha*
 isPlanar:(BOOL)*config*
 colorSpace:(NXColorSpace)*space*
 bytesPerRow:(int)*rowBytes*
 bitsPerPixel:(int)*pixelBits* Initializes the new object from raw bitmap data
- **initWithDataPlanes:**(unsigned char **)*planes*
 pixelsWide:(int)*width*
 pixelsHigh:(int)*height*
 bitsPerSample:(int)*bps*
 samplesPerPixel:(int)*spp*
 hasAlpha:(BOOL)*alpha*
 isPlanar:(BOOL)*config*
 colorSpace:(NXColorSpace)*space*
 bytesPerRow:(int)*rowBytes*
 bitsPerPixel:(int)*pixelBits* Initializes the new object from raw bitmap data in the
 planes data buffers

Creating a List of NXBitmapImageReps

- + (List *)**newListFromSection:**(const char *)*name*
Returns a List of NXBitmapImageReps from *name* data
- + (List *)**newListFromSection:**(const char *)*name*
 zone:(NXZone *)*aZone* Returns a List of NXBitmapImageReps from *name* data

- + (List *)**newListFromFile:(const char *)filename** Returns a List of NXBitmapImageReps from *filename*
- + (List *)**newListFromFile:(const char *)filename** Returns a List of NXBitmapImageReps from *filename*
zone:(NXZone *)aZone
- + (List *)**newListFromStream:(NXStream *)stream**
Returns a List of NXBitmapImageReps from *stream* data
- + (List *)**newListFromStream:(NXStream *)stream**
zone:(NXZone *)aZone Returns a List of NXBitmapImageReps from *stream* data

Reading Information from a Rendered Image

- + (int)**sizeImage:(const NXRect *)rect** Returns the number of bytes in bitmap for the *rect* image
- + (int)**sizeImage:(const NXRect *)rect** Provides information about the image bounded by the *rect*
pixelsWide:(int *)width rectangle
pixelsHigh:(int *)height
bitsPerSample:(int *)bps
samplesPerPixel:(int *)spp
hasAlpha:(BOOL *)alpha
isPlanar:(BOOL *)config
colorSpace:(NXColorSpace *)space

Copying and Freeing an NXBitmapImageRep

- **copyFromZone:(NXZone *)zone** Returns a copy of the NXBitmapImageRep
- **free** Deallocates the NXBitmapImageRep

Getting Information about the Image

- (int)**bitsPerPixel** Returns how many bits are needed to specify one pixel
- (int)**samplesPerPixel** Returns the number of samples (components) in the data
- (BOOL)**isPlanar** Returns YES if in planar configuration, NO if meshed
- (int)**numPlanes** Returns the number of data planes
- (int)**bytesPerPlane** Returns the number of bytes in each data plane
- (int)**bytesPerRow** Returns the number of bytes in a scan line

- (NXColorSpace)**colorSpace** Returns how bitmap data is to be interpreted

Getting Image Data

- (unsigned char *)**data** Returns a pointer to the bitmap data
- **getDataPlanes:**(unsigned char *)**planes** Provides pointers to each plane of bitmap data

Drawing the Image

- (BOOL)**draw** Draws the image at (0.0, 0.0) in current coordinates
- (BOOL)**drawIn:**(const NXRect *)**rect** Modifies coordinates so image is drawn in *rect* rectangle

Producing a TIFF Representation of the Image

- **writeTIFF:**(NXStream *)**stream** Writes a TIFF representation of the image to *stream*
- **writeTIFF:**(NXStream *)**stream**
 usingCompression:(int)**compression** Writes a TIFF representation of the image to *stream*
- **writeTIFF:**(NXStream *)**stream**
 usingCompression:(int)**compression**
 andFactor:(float)**factor** Writes a TIFF representation of the image to *stream*

Setting and Checking Compression Types

+ (void)**getTIFFCompressionTypes:**(const int **)**list**
 count:(int *)**numTypes** Returns all available compression types
+ (const char *)**localizedNameForTIFFCompressionType:**(int)**compression**
 Returns the localized name for the compression type
- (BOOL)**canBeCompressedUsing:**(int)**compression**
 YES if the image can be compressed using *compression*
- (void)**getCompression:**(int *)**compression**
 andFactor:(float *)**factor** Returns the compression type and compression factor
- (void)**setCompression:**(int)**compression**
 andFactor:(float)**factor** Sets the compression type and compression factor

Archiving

- **read:**(NXTypedStream *)*stream* Reads the NXBitmapImageRep from *stream*
- **write:**(NXTypedStream *)*stream* Writes the NXBitmapImageRep to *stream*

NXBrowser

Inherits From: Control : View : Responder : Object

Initializing and Freeing an NXBrowser

- **initWithFrame:**(const NXRect *)*frameRect* Initializes a new NXBrowser within *frameRect*
- **free** Frees the NXBrowser and its Matrices, NXBrowserCells and other objects (but not the delegate)

Setting the Delegate

- **setDelegate:***anObject* Sets the NXBrowser's delegate to *anObject*
- **delegate** Returns the NXBrowser's delegate

Target and Action

- **setAction:**(SEL)*aSelector* Sets the NXBrowser's action method to *aSelector*
- (SEL)**action** Returns the NXBrowser's action method
- **setTarget:***anObject* Sets the NXBrowser's target object to *anObject*
- **target** Returns the NXBrowser's target object
- **setDoubleAction:**(SEL)*aSelector* Sets the NXBrowser's double-click action to *aSelector*
- (SEL)**doubleAction** Returns the NXBrowser's double-click action method

Setting Component Classes

- **setMatrixClass:***classId* Sets the class of Matrix used in the NXBrowser's columns
- **setCellClass:***classId* Sets the class of Cell used in the columns of NXBrowser
- **setCellPrototype:***aCell* Sets the Cell instance copied to display items in the columns of NXBrowser
- **cellPrototype** Returns the NXBrowser's prototype Cell

Setting NXBrowser Behavior

- **setMultipleSelectionEnabled:**(BOOL)*flag* Sets whether the user can select multiple items
- (BOOL)**isMultipleSelectionEnabled** Returns whether the user can select multiple items
- **setBranchSelectionEnabled:**(BOOL)*flag* Sets whether the user can select branch items when multiple selection is enabled
- (BOOL)**isBranchSelectionEnabled** Returns whether the user can select branch items when multiple selection is enabled
- **setEmptySelectionEnabled:**(BOOL)*flag* Sets whether there can be nothing selected
- (BOOL)**isEmptySelectionEnabled** Returns whether there can be nothing selected
- **reuseColumns:**(BOOL)*flag* Prevents Matrices from being freed when their columns are unloaded, so they can be reused
- **setEnabled:**(BOOL)*flag* Sets whether the NXBrowser reacts to events
- (BOOL)**acceptsFirstResponder**
- **acceptArrowKeys:**(BOOL)*acceptFlag* Enables arrow keys for scrolling and sending action messages
- **andSendActionMessages:**(BOOL)*sendFlag*
- **getTitleFromPreviousColumn:**(BOOL)*flag* Sets whether the title of a column is set to the title of the selected Cell in the previous column

Configuring Controls

- **useScrollBars:**(BOOL)*flag* Sets whether Scrollers are used to scroll columns
- **useScrollButtons:**(BOOL)*flag* Sets whether buttons are used to scroll columns
- **setHorizontalScollButtonsEnabled:**(BOOL)*flag*

- (BOOL)**areHorizontalScollButtonsEnabled**
- **setHorizontalScollerEnabled:**(BOOL)*flag*
- (BOOL)**areHorizontalScollerEnabled**

Sets whether buttons are used to scroll horizontally
Returns whether buttons are used to scroll horizontally
Sets whether Scrollers are used to scroll horizontally
Returns whether Scrollers are used to scroll horizontally

Setting the NXBrowser's Appearance

- **setMinColumnWidth:**(int)*columnWidth*
- (int)**minColumnWidth**
- **setMaxVisibleColumns:**(int)*columnCount*
- (int)**maxVisibleColumns**
- (int)**numVisibleColumns**
- (int)**firstVisibleColumn**
- (int)**lastVisibleColumn**
- (int)**lastColumn**
- **separateColumns:**(BOOL)*flag*
- (BOOL)**columnsAreSeparated**

Sets the minimum column width
Returns the minimum column width
Sets the maximum number of columns displayed
Returns the maximum number of visible columns
Returns the number of columns visible
Returns the index of the first visible column
Returns the index of the last visible column
Returns the index of the last column loaded
Sets whether to separate columns with bezeled borders
Returns whether columns are separated by bezeled borders

Manipulating Columns

- **loadColumnZero**
- (BOOL)**isLoading**
- **addColumn**
- **reloadColumn:**(int)*column*
- **displayColumn:**(int)*column*
- **displayAllColumns**
- **setLastColumn:**(int)*column*
- **selectAll:***sender*
- (int)**selectedColumn**
- (int)**columnOf:***matrix*

Loads column zero; unloads previously loaded columns
Returns whether column zero is loaded
Adds a column to the right of the last column
Reloads *column* if it is loaded; sets it as the last column
Updates to display columns through index *column*
Updates the NXBrowser to display all loaded columns
Sets the last column to *column*
Selects all Cells in the last column of the NXBrowser
Returns the index of the last column with a selected item
Returns the column number in which *matrix* is located

- validateVisibleColumns

Invokes delegate method **browser:columnsIsValid:** for visible columns

Manipulating Column Titles

- **setTitled:**(BOOL)*flag*

Sets whether columns display titles

- (BOOL)**isTitled**

Returns whether columns display titles

- **setTitle:**(const char *)*aString*
 ofColumn:(int)*column*

Sets the title of the column at index *column* to *aString*

- (const char *)**titleOfColumn:**(int)*column*

Returns the title displayed for the column at index *column*

- (NXRect *)**getTitleFrame:**(NXRect *)*theRect*
 ofColumn:(int)*column*

Returns the bounds of the title frame for the column at index *column*

- (NXCoord)**titleHeight**

Returns the height of column titles

- **drawTitle:**(const char *)*title*
 inRect:(const NXRect *)*aRect*
 ofColumn:(int)*column*

Draws the title for the column at index *column*

- **clearTitleInRect:**(const NXRect *)*aRect*
 ofColumn:(int)*column*

Clears the title for the column at index *column*

Scrolling an NXBrowser

- **scrollColumnsLeftBy:**(int)*shiftAmount*

Scrolls columns left by *shiftAmount* columns

- **scrollColumnsRightBy:**(int)*shiftAmount*

Scrolls columns right by *shiftAmount* columns

- **scrollColumnToVisible:**(int)*column*

Scrolls to make the column at index *column* visible

- **scrollUpOrDown:***sender*

Scrolls a column up or down

- **scrollViaScroller:***sender*

Scrolls columns left or right based on a Scroller

- **reflectScroll:***clipView*

Updates scroll buttons to reflect column contents

- **updateScroller**

Updates the horizontal Scroller to reflect column positions

Event Handling

- **mouseDown:**(NXEvent *)*theEvent*

Handles mouse-down events in the NXBrowser

- **keyDown:**(NXEvent *)*theEvent*

Handles key-down events

- **doClick:***sender* Responds to mouse clicks in a column of NXBrowser
- **doDoubleClick:***sender* Responds to double-clicks in a column of NXBrowser

Getting Matrices and Cells

- **getLoadedCellAtRow:**(int)*row*
inColumn:(int)*column* Loads if necessary and returns the Cell at *row* in *column*
- **matrixInColumn:**(int)*column* Returns the matrix located in *column*
- **selectedCell** Returns the last selected Cell (rightmost and lowest)
- **getSelectedCells:***aList* Returns in *aList* all the rightmost selected Cells

Getting Column Frames

- (NXRect *)**getFrame:**(NXRect *)*theRect*
ofColumn:(int)*column* Returns the rectangle containing the column at index *column*
- (NXRect *)**getFrame:**(NXRect *)*theRect*
ofInsideOfColumn:(int)*column* Returns the rectangle containing the column at index *column*, not including borders

Manipulating Paths

- **setPathSeparator:**(unsigned short)*charCode* Sets the path separator to *charCode*
- **setPath:**(const char *)*path* Parses *path* and selects corresponding items in columns
- (char *)**getPath:**(char *)*thePath*
toColumn:(int)*column* Returns string representing path from the first column to the column at index *column*

Drawing

- **drawSelf:**(const NXRect *)*rects* :(int)*rectCount* Draws the NXBrowser

Resizing the NXBrowser

- **sizeTo:**(NXCoord)*width* :(NXCoord)*height* Resizes the NXBrowser to *width* and *height*
- **sizeToFit** Resizes the NXBrowser to fit all its contents

Arranging an NXBrowser's Components

- **tile** Adjusts the NXBrowser's components

Methods Implemented by the Delegate

- (BOOL)**browser:sender**
 columnsValid:(int)column Returns whether the contents of the column are valid
- **browserDidScroll:sender** Notifies the delegate when the NXBrowser has scrolled
- (int)**browser:sender**
 fillMatrix:matrix
 inColumn:(int)column Returns the number of rows in a column and loads
NXBrowserCells in *matrix*
- (int)**browser:sender**
 getNumRowsInColumn:(int)column Returns the number of rows of data in the column at index
 column
- **browser:sender**
 loadCell:cell
 atRow:(int)row
 inColumn:(int)column Requests the delegate to load Cell at *row* in the column at
 index *column*
- (BOOL)**browser:sender**
 selectCell:(const char *)title
 inColumn:(int)column Requests the delegate to select the Cell with title *title* in the
 column at index *column*
- (const char *)**browser:sender**
 titleOfColumn:(int)column Queries the delegate for the title to display above the
 column at index *column*
- **browserWillScroll:sender** Notifies the delegate when the NXBrowser will scroll

NXBrowserCell

Inherits From: Cell : Object

Initializing a NXBrowserCell

- **init**
- **initWithCell:**(const char *)*aString*

Initializes a new NXBrowserCell with ^aBrowserItem^o as its title
Initializes a newNXBrowserCell with *aString* as its title

Determining Component Sizes

- **calcCellSize:**(NXSize *)*theSize*
inRect:(const NXRect *)*aRect*

Calculates the size of the NXBrowserCell within *aRect*

Accessing Graphic Attributes

- (BOOL)**isOpaque**
- + **branchIcon**
- + **branchIconH**

Returns YES, since an NXBrowserCell is opaque
Returns the NXImage for branch NXBrowserCells
Returns the NXImage for highlighted branches

Displaying

- **drawInside:**(const NXRect *)*cellFrame*
inView:*aView*
- **drawSelf:**(const NXRect *)*cellFrame*
inView:*aView*
- **highlight:**(const NXRect *)*cellFrame*
inView:*aView* **lit:**(BOOL)*lit*

Draws the inside of the NXBrowserCell in *aView*

Draws the entire NXBrowserCell in *aView*

If *lit* is YES, highlights the NXBrowserCell in *aView*

Placing in Browser Hierarchy

- **setLeaf:**(BOOL)*flag*
- (BOOL)**isLeaf**

Sets whether the NXBrowserCell is a leaf or a branch
Returns whether the NXBrowserCell is a leaf or a branch

Determining Loaded Status

- **setLoaded:**(BOOL)*flag*

Sets whether the NXBrowserCell is loaded and displayble

- (BOOL)**isLoading**

Returns whether the NXBrowserCell is loaded

Setting State

- **set**

Highlights the NXBrowserCell and sets its state to 1

- **reset**

Unhighlights the NXBrowserCell and sets its state to 0

NXCachedImageRep

Inherits From: NXImageRep : Object

Initializing a New NXCachedImageRep

- **initWithWindow:**(Window *)*aWindow*
 rect:(const NXRect *)*aRect*

Initializes the new NXCachedImageRep for an image to be drawn in *aWindow*

- **copyFromZone:**(NXZone *)*theZone*

Creates and returns a copy of the receiver

Freeing an NXCachedImageRep

- **free**

Deallocates the NXCachedImageRep

Getting the Representation

- **getWindow:**(Window **)*theWindow*
 andRect:(NXRect *)*theRect*

Provides the Window and rectangle where the image is cached

Drawing the Image

- (BOOL)**draw**

Reads the cached image and renders it

Archiving

- **read:**(NXTypedStream *)*stream* Reads the NXCachedImageRep from *stream*
- **write:**(NXTypedStream *)*stream* Writes the NXCachedImageRep to *stream*

NXColorPanel

Inherits From: Panel : Window : Responder : Object

Creating a New NXColorPanel

- + **newColorMask:**(int)*colormask* Returns the shared NXColorPanel
- + **newContent:**(const NXRect *)*contentRect* Returns the shared NXColorPanel
 - style:**(int)*aStyle*
 - backing:**(int)*bufferingType*
 - buttonMask:**(int)*mask*
 - defer:**(BOOL)*flag*
- + **newContent:**(const NXRect *)*contentRect* Returns the shared NXColorPanel
 - style:**(int)*aStyle*
 - backing:**(int)*bufferingType*
 - buttonMask:**(int)*mask*
 - defer:**(BOOL)*flag*
 - colorMask:**(int)*colormask*
- + **sharedInstance:**(BOOL)*create* If YES, creates if necessary and returns the shared NXColorPanel

Setting the NXColorPanel

- (int)**colorMask** Returns the color mask of the NXColorPanel
- **setColorMask:**(int)*colormask* Sets the color mask of the NXColorPanel
- **setContinuous:**(BOOL)*flag* Sets the NXColorPanel to continuously send the action message to the target

- **setMode:**(int)*mode*
- **setAccessoryView:***aView*
- **setAction:**(SEL)*aSelector*
- **setShowAlpha:**(BOOL)*flag*
- **setTarget:***anObject*

Sets the mode and returns the NXColorPanel

Sets the accessory view to *aView*

Sets the action message sent to the target

Sets the NXColorPanel to show alpha values

Sets the target of the NXColorPanel

Setting Color

- **color:**(NXColor *)*color*
- **setColor:**(NXColor)*color*
- + **dragColor:**(NXColor *)*color*
 withEvent:(NXEvent *)*event*
 fromView:*sourceView*

Returns the currently selected color

Sets the *color* of the NXColorPanel

Drags *color* into a destination view from *sourceView*.
event is usually an NX_MOUSEUP

NXColorPicker

Inherits From: Object

Conforms To: NXColorPickingDefault

Initialization

- **initWithPickerMask:**(int)*theMask*
 withColorPanel:*thePanel*

Initializes the receiver for the specified mask and color panel

Button Images

- **provideNewButtonImage**
- **insertNewButtonImage:***newImage*
 in:*newButtonCell*

Returns a new button image for the color picker

Override to customize *newImage* before insertion
in *newButtonCell*

View Management

- **viewSizeChanged:***sender* Does nothing. Override to respond to size change.

Alpha Control Check

- **alphaControlAddedOrRemoved:***sender* Responds to change in color panel alpha control status

Order of Button Appearance

- (float)**insertionOrder** Returns the color picker button's insertion order

Using Color Lists

- **attachColorList:***colorList* Override to attach a color list to a color picker

- **detachColorList:***colorList* Override to detach a color list from a color picker

Mode

- **setMode:**(int)*mode* Override to set the color picker's mode

NXColorWell

Inherits From: Control: View: Responder: Object

New

- **initWithFrame:**(const NXRect *)*theFrame* Initializes and returns a new instance of NXColorWell

Event Handling

- (BOOL)**acceptsFirstMouse**
- **mouseDown:**(NXEvent *)*theEvent*

Returns YES

Responds to mouse down in the NXColorWell

Drawing

- **drawSelf:**(const NXRect *)*rects*
:(int)*rectCount*
- **drawWellInside:**(const NXRect *)*insideRect*

Draws the entire NXColorWell, including borders

Draws the colored area inside the NXColorWell, without drawing borders

Activating

- **deactivate**
- + **deactivateAllWells**
- **activate:**(int)*exclusive*
- (BOOL)**isActive**
- **setEnabled:**(BOOL)*enabled*

Deactivates and returns the NXColorWell

Deactivates all currently active NXColorWells

Activates and returns the NXColorWell

Returns YES if the NXColorWell is active

Enables the NXColorWell

Managing Color

- **activeWellsTakeColorFrom:***sender*
- **activeWellsTakeColorFrom:***sender*
continuous:(BOOL)*continuously*
- (NXColor)**color**
- **takeColorFrom:***sender*
- **acceptColor:**(NXColor)*color*
atPoint:(NXPoint *)*aPoint*
- **setColor:**(NXColor)*color*
- **updateCustomColorList**

Changes color of all active wells to that of sender

Continuously changes color of all active, continuous wells to that of *sender*

Returns the color of the NXColorPanel

Changes color of the well to that of *sender*

Changes color of the well to *color* when *aPoint* is a point in the bounds of the NXColorWell

Sets the color of the well to *color*

Saves the current color list in NX_COLORLISTMODE

Target and Action

- (SEL)**action**
- **setAction:**(SEL) *aSelector*
- **setTarget:***anObject*
- **target**

Returns the NXColorWell's action message

Sets the NXColorWell's action message

Sets the NXColorWell's target

Returns the NXColorWell's target

Archiving

- **awake**

Initializes the NXColorWell after unarchiving

NXCursor

Inherits From: Object

Initializing a New NXCursor Object

- **init**
- **initWithImage:***image*

Initializes a new NXCursor, but doesn't set the image

Initializes a new NXCursor object with *image*

Defining the Cursor

- **setImage:***newImage*
- **image**
- **setHotSpot:**(const NXPoint *)*spot*

Sets the NXImage object that supplies the cursor image

Returns the NXImage object that has the cursor image

Sets the point on the cursor that's aligned with the mouse

Setting the Cursor

- **push**
- **pop**
- + **pop**

Makes the NXCursor the current cursor

Restores the previous cursor

Restores the previous cursor

- **set**
- **setOnMouseEntered:**(BOOL)*flag*
- **setOnMouseExited:**(BOOL)*flag*
- **mouseEntered:**(NXEvent *)*theEvent*
- **mouseExited:**(NXEvent *)*theEvent*
- + **currentCursor**

Sets the NXCursor to be the current cursor

Determines whether **mouseEntered:** sets cursor

Determines whether **mouseExited:** sets cursor

Responds to a mouse-entered event

Responds to a mouse-exited event

Returns the current cursor

Archiving

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

Reads the NXCursor from the typed stream *stream*

Writes the NXCursor to the typed steam *stream*

NXCustomImageRep

Inherits From: NXImageRep : Object

Initializing a New NXCustomImageRep

- **initWithDrawMethod:**(SEL)*aSelector*
inObject:*anObject*

Initializes the new object so that *anObject*'s *aSelector* method will draw the image

Drawing the Image

- (BOOL)**draw**

Sends a message to draw the image

Archiving

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

Reads the NXCustomImageRep from *stream*

Writes the NXCustomImageRep to *stream*

NXDataLink

Inherits From: Object

Initializing a Link

- **initFromFile:**(const char *)*filename* Initializes a new instance from *filename*
- **initFromPasteboard:**(Pasteboard *)*pasteboard* Initializes a new instance from *pasteboard*
- **initLinkedToFile:**(const char *)*filename* Initializes a new instance corresponding to *filename*
- **initLinkedToSourceSelection:**(NXSelection *)*selection*
 managedBy:*linkManager* Initializes a new instance as specified
 supportingTypes:(const char * const*)*newTypes*
 count:(int)*numTypes*
- **copyFromZone:**(NXZone *)*zone* Returns a copy of the receiver, allocated from *zone*

Exporting a Link

- **writeToPasteboard:**(Pasteboard *)*pasteboard* Writes the link onto the pasteboard *pasteboard*
- **saveLinkIn:**(const char *)*directoryName* Saves the link with a file name provided by the user
- **writeToFile:**(const char *)*filename* Writes the link into the file *filename*

Information about the Link

- (NXDataLinkManager *)**manager** Returns the link's manager
- (NXDataLinkDisposition)**disposition** Identifies the link's type
- (NXDataLinkNumber)**linkNumber** Returns the link's number

Information about the Link's Source

- (const char *) sourceAppName	Returns the name of the application containing the source
- (const char *) sourceFilename	Returns the file name of the source document
- (NXSelection *) sourceSelection	Returns the source selection
- openSource	Opens the document corresponding to source selection
- (time_t) lastUpdateTime	Returns the last time the link was updated
- (const NXAtom *) types	Returns the types that the source document can provide

Information about the Link's Destination

- (const char *) destinationAppName	Returns the name of the application containing the destination link
- (const char *) destinationFilename	Returns the file name of the destination document
- (NXSelection *) destinationSelection	Returns the destination selection

Information about the Link's Data

- sourceEdited	Sent to a source link to inform it that the data referred to by its source selection has changed
- updateDestination	Updates the data referred to by the link's destination selection
- setUpdateMode: (NXDataLinkUpdateMode) <i>mode</i>	Sets the link's update mode to <i>mode</i>
- (NXDataLinkUpdateMode) updateMode	Returns the link's update mode
- break	Breaks the link

NXDataLinkManager

Inherits From: Object

Initializing and Freeing a Link Manager

- initWithDelegate: <i>anObject</i>	Initializes and returns a newly allocated instance
--	--

- **initWithDelegate:***anObject*
fromFile:(const char *)*path* Initializes and returns a newly allocated instance
- **free** Frees the objects and storage held by the link manager

Adding and Removing Links

- **addLink:**(NXDataLink *)*link*
at:(NXSelection *)*selection* Adds the link *link* to the document
- **addLinkAsMarker:**(NXDataLink *)*link*
at:(NXSelection *)*selection* Incorporates *link* into the document as a marker
- **writeLinksToPasteboard:**(Pasteboard *)*pboard* Writes all the link manager's links to the pasteboard
- (NXDataLink *)**addLinkPreviouslyAt:**(NXSelection *)*oldSelection*
fromPasteboard:(Pasteboard *)*pasteboard* Creates and adds a new destination link corresponding to
at:(NXSelection *)*selection* *oldSelection*
- **breakAllLinks** Breaks all the destination links in the document

Informing the Link Manager of Document Status

- **documentClosed** Informs link manager that document has been closed
- **documentEdited** Informs link manager that document has been edited
- **documentReverted** Informs link manager that changes have been reverted
- **documentSaved** Informs link manager that document has been saved
- **documentSavedAs:**(const char *)*path* Informs link manager that document has been saved
- **documentSavedTo:**(const char *)*path* Informs link manager that document has been saved

Getting and Setting Information about the Link Manager

- (const char *)**filename** Returns the filename for the link manager's document
- (BOOL)**isEdited** Returns YES if the document was edited since the last save
- **setLinksVerifiedByDelegate:**(BOOL)*flag* Sets whether the delegate is asked to verify updates
- (BOOL)**areLinksVerifiedByDelegate** Returns YES if delegate is asked to verify updates
- **delegate** Returns the data link manager's delegate

- **setInteractsWithUser:**(BOOL)*flag* Sets whether manager displays panels if link errors occur
- (BOOL)**interactsWithUser** Tells whether manager displays panels if link errors occur

Getting and Setting Information about the Manager's Links

- **setLinkOutlinesVisible:** Sets whether outlines are visible
- (BOOL)**areOutlinesVisible** Returns YES if outlines are visible
- (NXDataLink *)**findDestinationLinkWithSelection:**(NXSelection *)*destSel*
Returns the destination link for the selection *destSel*
- **prepareEnumerationState:**(NXLinkEnumerationState *)*state*
forLinksOfType:(NXDataLinkDisposition *)*srcOrDest*
Prepares manager to enumerate links
- (NXDataLink *)**nextLinkUsing:**(NXLinkEnumerationState *)*state*
Returns the link manager's next link based on *state*

NXDataLinkPanel

Inherits From: Panel : Window : Responder : Object

Returning the Panel

- + **new** Returns the application's sole NXDataLinkPanel object
- + **newContents:**(const NXRect *)*contentRect*
style:(int)*aStyle*
backing:(int)*bufferingType*
buttonMask:(int)*mask*
defer:(BOOL)*flag*
Initializes the NXDataLinkPanel object

Keeping the Panel Up to Date

- + **setLink:**(NXDataLink *)*link* Informs the receiver of the current document and selection
 andManager:(NXDataLinkManager *)*linkManager*
 isMultiple:(BOOL)*flag*
- **setLink:**(NXDataLink *)*link* Informs the receiver of the current document and selection
 andManager:(NXDataLinkManager *)*linkManager*
 isMultiple:(BOOL)*flag*
- + **getLink:**(NXDataLink **)*link* Gets information about the currently selected link
 andManager:(NXDataLinkManager **)*linkManager*
 isMultiple:(BOOL *)*flag*
- **getLink:**(NXDataLink **)*link* Gets information about the currently selected link
 andManager:(NXDataLinkManager **)*linkManager*
 isMultiple:(BOOL *)*flag*

Customizing the Panel

- **setAccessoryView:***aView* Adds *aView* to the NXDataLinkPanel's view hierarchy
- **accessoryView** Returns the NXDataLinkPanel's custom accessory view

Responding to User Input

- **pickedBreakAllLinks:***sender* Invoked when the user clicks the Break All Links button
- **pickedBreakLink:***sender* Invoked when the user clicks the Break Link button
- **pickedOpenSource:***sender* Invoked when the user clicks the Open Source button
- **pickedUpdateDestination:***sender* Invoked when the user clicks Update from Source button
- **pickedUpdateMode:***sender* Invoked when the user selects the update mode

NXEPSImageRep

Inherits From: NXImageRep : Object

Initializing a New NXEPSImageRep Instance

- **initFromSection:**(const char *)*name* Initializes the new object from EPS code in the section
- **initFromFile:**(const char *)*filename* Initializes the new object from EPS code in *filename*
- **initFromStream:**(NXStream *)*stream* Initializes the new object from EPS code in *stream*

Creating a List of NXEPSImageReps

- + (List *)**newListFromSection:**(const char *)*name*
Returns a List of NXEPSImageReps from EPS in *name*
- + (List *)**newListFromSection:**(const char *)*name*
zone:(NXZone *)*aZone*
Returns a List of NXEPSImageReps from EPS in *name*
- + (List *)**newListFromFile:**(const char *)*filename* Returns a List of NXEPSImageReps from *filename* data
- + (List *)**newListFromFile:**(const char *)*filename* Returns a List of NXEPSImageReps from *filename* data
zone:(NXZone *)*aZone*
- + (List *)**newListFromStream:**(NXStream *)*stream*
Returns a List of NXEPSImageReps from EPS in *stream*
- + (List *)**newListFromStream:**(NXStream *)*stream*
zone:(NXZone *)*aZone* Returns a List of NXEPSImageReps from EPS in *stream*

Copying and Freeing an NXEPSImageRep

- **copyFromZone:**(NXZone *)*zone* Returns a copy of the NXEPSImageRep
- **free** Deallocates the NXEPSImageRep

Getting the Rectangle that Bounds the Image

- **getBoundingBox:**(NXRect *)*rect* Copies the EPS bounding box into the *rect* rectangle

Getting Image Data

- **getEPS:**(char **)*theEPS* **length:**(int *)*numBytes* Provides a pointer to the EPS code

Drawing the Image

- **prepareGState** Implemented by subclasses to prepare the graphics state
- (BOOL)**draw** Draws the image at (0.0, 0.0) in current coordinates
- (BOOL)**drawIn:(const NXRect *)rect** Draws the image so it fits within the *rect* rectangle

Archiving

- **read:(NXTypedStream *)stream** Reads the NXEPSImageRep from *stream*
- **write:(NXTypedStream *)stream** Writes the NXEPSImageRep to *stream*

NXHelpPanel

Inherits From: Panel : Window : Responder : Object

Initializing and Freeing

- + **new** Creates, if necessary, and returns the NXHelpPanel object
- + **newForDirectory:(const char *)helpDirectory** Creates, if necessary, and returns the NXHelpPanel object
- **addSupplement:(const char *)helpDirectory** Adds supplemental help to the text displayed in the panel
- inPath:(const char *)supplementPath**
- **free** Frees the NXHelpPanel and its storage

Attaching Help to Objects

- + **attachHelpFile:(const char *)filename** Associates the help file at *markerName* with *anObject*
- markerName:(const char *)markerName**
- to:anObject**
- + **detachHelpFrom:anObject** Removes any help information associated with *anObject*

Setting Click-for-Help

- + (BOOL)**isClickForHelpEnabled**

+ **setClickForHelpEnabled:**(BOOL)*enabled*
- Returns whether the click-for-help feature is enabled

Sets whether the click-for-help feature is enabled

Printing

- **print:***sender*

- **printPanel:***sender*
- Prints the currently displayed help text

Prints the currently displayed help text

Querying

- (NXAtom)**helpDirectory**

-(NXAtom) **helpFile**
- Returns the absolute path of the help directory

Returns the path of the currently loaded help file

Showing Help

- **showFile:**(const char *)*filename*

atMarker:(const char *)*markerName*

- (BOOL)**showHelpAttachedTo:***anObject*
- Causes the Help panel to display the help contained in
 filename at *markerName*

Causes the Help panel to display help attached to *anObject*

NXImage

Inherits From: Object

Initializing a New NXImage Instance

- **init**

- **initWithSize:**(const NXSize *)*aSize*
- Initializes the new NXImage without setting its size

Initializes the new NXImage to the specified size

- **initWithSection:**(const char *)*name* Initializes the new object from the data in *name* section
- **initWithFile:**(const char *)*filename* Initializes the new NXImage from the data in *filename*
- **initWithPasteboard:**(Pasteboard *)*pasteboard* Initializes the new NXImage from the data in *pasteboard*
- **initWithStream:**(NXStream *)*stream* Initializes the new NXImage from the data in *stream*
- **initWithImage:**(NXImage *)*image*
rect:(const NXRect *)*rect* Initializes the new NXImage *to be a subimage of image*
- **copyFromZone:**(NXZone *)*zone* Creates and returns a copy of the NXImage in *zone*

Freeing an NXImage object

- **free** Frees the NXImage and its representations

Setting the Size of the Image

- **setSize:**(const NXSize *)*aSize* Sets the size of the image in base coordinates
- **getSize:**(NXSize *)*theSize* Provides the size of the image

Referring to Images by Name

- (BOOL)**setName:**(const char *)*string* Assigns *string* as the name of the NXImage object
- (const char *)**name** Returns the name of the NXImage object
- + **findImageNamed:**(const char *)*name* Returns the NXImage object with *name*

Specifying the Image

- (BOOL)**useDrawMethod:**(SEL)*aSelector*
inObject:*anObject* Creates a representation that will use a delegated method to draw the image
- (BOOL)**useFromSection:**(const char *)*name* Creates representations for the data in the *name* section
- (BOOL)**useFromFile:**(const char *)*filename* Creates representations for the data in *filename* file
- (BOOL)**useRepresentation:**(NXImageRep *)*imageRep*
Adds imageRep to the List of representations
- (BOOL)**useCacheWithDepth:**(NXWindowDepth)*depth*

- (BOOL)**loadFromStream:**(NXStream *)*stream* Creates an empty representation to draw in
Creates representation for the data read from *stream*
- (BOOL)**loadFromFile:**(const char *)*fileName* Creates representation for the data read from *filename*
- (BOOL)**lockFocus** Prepares for drawing in the best representation
- (BOOL)**lockFocusOn:**(NXImageRep *)*imageRep* Prepares for drawing in *imageRep*
- **unlockFocus** Balances a previous **lockFocus** or **lockFocusOn:**

Using the Image

- **composite:**(int)*op*
 toPoint:(const NXPoint *)*aPoint* Composites the image to *aPoint*
- **composite:**(int)*op*
 fromRect:(const NXRect *)*aRect*
 toPoint:(const NXPoint *)*aPoint* Composites the *aRect* portion of the image to *aPoint*
- **dissolve:**(float)*delta*
 toPoint:(const NXPoint *)*aPoint* Composites the image using the **dissolve** operator
- **dissolve:**(float)*delta*
 fromRect:(const NXRect *)*aRect*
 toPoint:(const NXPoint *)*aPoint* Composites the image using the **dissolve** operator

Choosing Which Image Representation to Use

- **setColorMatchPreferred:**(BOOL)*flag* Determines whether color matches are preferred
- (BOOL)**isColorMatchPreferred** Returns whether color matches are preferred
- **setEPSUsedOnResolutionMismatch:**(BOOL)*flag*
Sets whether to use EPS representations on mismatch
- (BOOL)**isEPSUsedOnResolutionMismatch** Returns whether to use EPS representations on mismatch
- **setMatchedOnMultipleResolution:**(BOOL)*flag* Sets whether resolution multiples match
- (BOOL)**isMatchedOnMultipleResolution** Returns whether resolution multiples match

Getting the Representations

- (NXImageRep *)**lastRepresentation** Returns the last representation added to the NXImage
- (NXImageRep *)**bestRepresentation** Returns the best representation for the deepest screen
- (List *)**representationList** Returns the List of all the representations
- **removeRepresentation:(NXImageRep *)imageRep**
Removes *imageRep* from the List of representations

Determining How the Image is Stored

- **setUnique:(BOOL)flag** Sets whether representations are cached alone
- (BOOL)**isUnique** Returns whether representations are cached alone
- **setDataRetained:(BOOL)flag** Sets whether image data is retained by the object
- (BOOL)**isDataRetained** Returns whether image data is retained
- **setCacheDepthBounded:(BOOL)flag** Sets whether the default depth limit applies to caches
- (BOOL)**isCacheDepthBounded** Returns whether the default depth limit applies to caches
- **getImage:(NXImage **)image**
rect:(NXRect *)rect Gets the image that the receiver is a subimage of

Determining How the Image is Drawn

- **setFlipped:(BOOL)flag** Inverts the polarity of the y-axis for drawing the image
- (BOOL)**isFlipped** Returns whether the polarity of the y-axis is inverted
- **setScalable:(BOOL)flag** Determines whether representations are scaled to fit
- (BOOL)**isScalable** Returns whether representations are scaled to fit
- **setBackgroundColor:(NXColor)aColor** Sets the background color of the image
- (NXColor)**backgroundColor** Returns the background color of the image
- (BOOL)**drawRepresentation:(NXImageRep *)imageRep**
inRect:(const NXRect *)rect Has *imageRep* draw the representation
- **recache** Invalidates caches of all representations, so they will be redrawn

Assigning a Delegate

- **setDelegate:***anObject* Makes *anObject* the delegate of the NXImage
- **delegate** Returns the delegate of the NXImage

Producing TIFF Data for the Image

- **writeTIFF:**(NXTypedStream *)*stream* Writes TIFF for the best representation to *stream*
- **writeTIFF:**(NXTypedStream *)*stream*
allRepresentations:(BOOL)*flag* Writes TIFF for all the representations to *stream*

Managing NXImageRep subclasses

- + (void)**registerImageRep:***imageRepClass* Registers a new class for managing image data
- + (void)**unregisterImageRep:***imageRepClass* Unregisters a class for managing image data
- + (Class)**imageRepForFileType:**(const char *)*type*
Returns image rep that handles data of *type*
- + (Class)**imageRepForPasteboardType:**(NXAtom)*type*
Returns image rep that handles data of *type*
- + (Class)**imageRepForStream:**(NXStream *)*stream*
Returns image rep that handles data on *stream*
- + (const char *const *)**imageUnfilteredFileTypes**
Returns a list of filetypes handled by the NXImage
- + (const NXAtom *)**imageUnfilteredPasteboardTypes**
Returns a list of pasteboard types handled by the NXImage

Testing Image Data Sources

- + (BOOL)**canInitFromPasteboard:**(Pasteboard *)*pasteboard*
YES if NXImage can create a representation from *pasteboard*
- + (const char *const *)**imageFileTypes** Returns an array of supported image data file types
- + (const NXAtom *)**imagePasteboardTypes** Returns an array of supported pasteboard types

Archiving

- **read:**(NXTypedStream *)*stream* Reads the NXImage and its representations from *stream*
- **write:**(NXTypedStream *)*stream* Writes the NXImage and its representations to *stream*
- **finishUnarchiving** Replaces the NXImage with one having the same name

Methods Implemented by the Delegate

- (NXImage *)**imageDidNotDraw:***sender*
inRect:(NXRect *)*aRect* Responds to message that image couldn't be composited

NXImageRep

Inherits From: Object

Initializing

- **initWithPasteboard:**(Pasteboard *)*pasteboard* Initializes the receiver from *pasteboard*

Checking data types

- + (BOOL)**canInitFromPasteboard:**(Pasteboard *)*pasteboard*
YES if NXImageRep can initialize itself from *pasteboard*
- + (BOOL)**canLoadFromStream:**(NXStream *)*stream*
YES if NXImageRep can initialize itself from *stream*
- + (const char *const *)**imageFileTypes** Returns an array of strings representing all file types
- + (const NXAtom *)**imagePasteboardTypes** Returns an array representing all pasteboard types
- + (const char *const *)**imageUnfilteredFileTypes** Returns an array representing directly supported file types
- + (const NXAtom *)**imageUnfilteredPasteboardTypes**
Returns an array representing directly supported pasteboards

Setting the Size of the Image

- **setSize:**(const NXSize *)*aSize* Sets the size of the image
- **getSize:**(NXSize *)*theSize* Copies the size of the image into the *theSize* structure

Specifying Information about the Representation

- **setNumColors:**(int)*anInt* Informs the object that there are *anInt* color components
- (int)**numColors** Returns the number of color components
- **setAlpha:**(BOOL)*flag* Informs object whether there is a coverage component
- (BOOL)**hasAlpha** Returns whether there is a coverage component
- **setBitsPerSample:**(int)*anInt* Informs object there are *anInt* bits/pixel in a component
- (int)**bitsPerSample** Returns the number of bits per pixel in each component
- **setPixelsHigh:**(int)*anInt* Informs object that data is for an image *anInt* pixels high
- (int)**pixelsHigh** Returns the height specified in the image data
- **setPixelsWide:**(int)*anInt* Informs object that data is for an image *anInt* pixels wide
- (int)**pixelsWide** Returns the width specified in the image data

Drawing the Image

- (BOOL)**draw** Implemented by subclasses to draw the image
- (BOOL)**drawAt:**(const NXPoint *)*point* Modifies current coordinates so image is drawn at *point*
- (BOOL)**drawIn:**(const NXRect *)*rect* Modifies current coordinates so image is drawn in *rect*

Archiving

- **read:**(NXTypedStream *)*stream* Reads the NXImageRep from *stream*
- **write:**(NXTypedStream *)*stream* Writes the NXImageRep to *stream*

NXJournaler

Inherits From: Object

Initializing and Freeing a Journaler

- | | |
|---------------|-------------------------------|
| - init | Initializes a new NXJournaler |
| - free | Deallocates the NXJournaler |

Controlling Journaling

- | | |
|--|---|
| - setEventStatus: (int) <i>eventStatus</i>
soundStatus: (int) <i>soundStatus</i>
eventStream: (NXStream *) <i>stream</i>
soundfile: (const char *) <i>soundfile</i> | Controls recording and playback |
| - getEventStatus: (int *) <i>eventStatusPtr</i>
soundStatus: (int *) <i>soundStatusPtr</i>
eventStream: (NXStream **) <i>streamPtr</i>
soundfile: (char **) <i>soundfilePtr</i> | Provides status information about the NXJournaler |
| - setRecordDevice: (int) <i>device</i> | Sets whether CODEC or DSP is used for sound input |
| - (int) recordDevice | Returns NX_CODEC or NX_DSP |

Identifying Associated Objects

- | | |
|---------------------------------------|---|
| - speaker | Returns the NXJournaler's Speaker object |
| - listener | Returns the NXJournaler's Listener object |
| - setDelegate: <i>anObject</i> | Sets the NXJournaler's delegate |
| - delegate | Returns the NXJournaler's delegate |

Implemented by the delegate

- | | |
|--|--|
| - journalerDidEnd: <i>journaler</i> | Informs the delegate that the session terminated |
| - journalerDidUserAbort: <i>journaler</i> | Informs the delegate that the user aborted the session |

NXPrinter

Inherits From: Object

Finding an NXPrinter

- + (NXPrinter *)**newForName:**(const char *)*name*
Returns the NXPrinter with the given name
- + (NXPrinter *)**newForName:**(const char *)*name* **host:**(const char *)*hostName*
Returns the NXPrinter with the given *name* and *host*
- + (NXPrinter *)**newForName:**(const char *)*name* **host:**(const char *)*hostName* **domain:**(const char *)*domain* **includeUnavailable:**(BOOL)*includeFlag*
Returns the NXPrinter with the given name, *host*, and *domain*
- + (NXPrinter *)**newForType:**(const char *)*type*
Returns an NXPrinter object for a given printer type
- + (char **)**printerTypes:**(BOOL)*normalFlag* **custom:**(BOOL)*customFlag*
Returns the names of the recognized printer types

Printer Attributes

- (const char *)**domain**
Returns the name of the printer's domain
- (const char *)**host**
Returns the name of the printer's host computer
- (const char *)**name**
Returns the printer's name
- (const char *)**note**
Returns the note associated with the printer
- (const char *)**type**
Returns the name of the printer's type
- (BOOL)**isReallyAPrinter**
Returns whether the object corresponds to an actual printer

Retrieving Specific Information

- (BOOL)**acceptsBinary** Returns YES if the printer accepts binary PostScript
- (NXRect)**imageRectForPaper:(const char *)paperType**
Returns the printing rectangle for the named paper type
- (NXSize)**pageSizeForPaper:(const char *)paperType**
Returns the size of the page for the named paper type
- (BOOL)**isColor** Returns whether the printer can print color
- (BOOL)**isFontAvailable:(const char *)name** Returns whether the named font is available to the printer
- (BOOL)**isValid** Returns whether the NXPrinter is valid
- (int)**languageLevel** Returns the PostScript Language Level recognized by the printer
- (BOOL)**isOutputStackInReverseOrder** Returns whether the printer outputs pages in reverse page order

Querying the NXPrinter Tables

- (BOOL)**booleanForKey:(const char *)key
inTable:(const char *)table** Returns a boolean value for the given key in the given table
- (void *)**dataForKey:(const char *)key
inTable:(const char *)table
length:(int *)bytes** Returns untyped data for the key in the table
- (float)**floatForKey:(const char *)key
inTable:(const char *)table** Returns a float value for the key in the table
- (int)**intForKey:(const char *)key
inTable:(const char *)table** Returns an integer value for the key in the table
- (NXRect)**rectForKey:(const char *)key
inTable:(const char *)table** Returns an NXRect for the key in the table
- (NXSize)**sizeForKey:(const char *)key
inTable:(const char *)table** Returns an NXSize for the key in the table
- (const char *)**stringForKey:(const char *)key
inTable:(const char *)table** Returns a string for the key in the table
- (const char **)**stringListForKey:(const char *)key
inTable:(const char *)table** Returns an array of strings for the key in the table
- (int)**statusForTable:(const char *)table** Returns the status of the given table

- (BOOL) isKey: (const char *) <i>key</i> inTable: (const char *) <i>table</i> <i>g</i>	Returns whether key is a key to table
--	---------------------------------------

NXSpellChecker

Inherits From: Object

Making A Checker Available

+ sharedInstance	Returns the NXSpellChecker to use
+ sharedInstance: (BOOL) <i>flag</i>	Returns the NXSpellChecker to use but creates a new one only when <i>flag</i> is YES

Managing The Spelling Panel

- spellingPanel	Returns the NXSpellChecker's panel
- accessoryView	Returns the spell panel's accessory view
- setAccessoryView: <i>aView</i>	Makes a view an accessory of the spell panel

Checking Spelling

- (BOOL) checkSpelling: (NXSpellCheckMode) <i>how</i> of: (id <NXReadOnlyTextStream, NXSelectRange>) <i>anObject</i>	Starts the search for a misspelled word
- (BOOL) checkSpelling: (NXSpellCheckMode) <i>how</i> of: (id <NXReadOnlyTextStream, NXSelectRange>) <i>anObject</i> wordCount: (int *) <i>theCount</i>	Starts the search for a misspelled word and the count of words

Managing the Language Being Checked

- (const char*) **language** Returns the current spelling language
- **setLanguage:**(const char *)*aLanguage* Sets the current spelling language

Managing Ignored Words

- **closeSpellClient:***aClient* Notifies the NXSpellChecker that a document has closed
- (char **) **ignoredWordsForSpellClient:***aClient* Returns the list of ignored words for a document
- **setIgnoredWords:**(const char *const *)*someWords*
forSpellClient:(int)*tag* Initializes the list of ignored words for a document

NXSpellServer

Inherits From: Object

Checking in Your Service

- (BOOL) **registerLanguage:**(const char *)*language*
byVendor:(const char *)*vendor*

Assigning a Delegate

- **delegate** Returns the NXSpellServer's delegate
- **setDelegate:***anObject* Makes the spelling service program the delegate of the NXSpellServer object

Running the Service

- **run** Starts the event loop in the NXSpellServer's delegate

Checking User Dictionaries

- (BOOL)**isInUserDictionary**:(const char *)*word* Returns YES if the word is in any open user dictionary
caseSensitive:(BOOL)*flag*

Seeking alternative spellings

- **addGuess**:(const char *)*guess* Called by the delegate to append the guesses it has found

Methods Implemented by the Delegate

- (BOOL)**spellServer**:(NXSpellServer *)*sender* Searches for a misspelled word; return YES if one is found
findMisspelledWord:(int *)*start*
length:(int *)*length*
inLanguage:(const char *)*language*
inTextStream:(id <NXReadOnlyTextStream>)*textStream*
startingAt:(int)*startPosition*
wordCount:(int *)*number*
countOnly:(BOOL)*flag*
- (void)**spellServer**:(NXSpellServer *)*sender* Searches for alternatives to the misspelled word; returns
suggestGuessesForWord:(const char *)*word* guesses as a side effect, using **addGuess**:
inLanguage:(const char *)*language*
- (void)**spellServer**:(NXSpellServer *)*sender* Notifies the delagte of a word added to the user's hidden
didLearnWord:(const char *)*word* wordlist
inLanguage:(const char *)*language*
- (void)**spellServer**:(NXSpellServer *)*sender* Notifies the delagte of a word removed from the user's
didForgetWord:(const char *)*word* hidden wordlist
inLanguage:(const char *)*language*;

NXSplitView

Inherits From: View : Responder : Object

Initializing an `NXSplitView`

- **`initWithFrame:(const NXRect *)frameRect`** Initializes a new `NXSplitView`

Handling Events

- **`mouseDown:(NXEvent *)theEvent`** Handles mouse-down events
- **`acceptsFirstMouse`** Allows the `NXSplitView` to respond to the mouse event that makes its
Window the key window

Managing Component Views

- **`adjustSubviews`** Adjusts the heights of the subviews
- **`resizeSubviews:`** Forces adjustment of the subviews
- **`(NXCoord)dividerHeight`** Returns the height of the divider
- **`drawSelf:(const NXRect *) rects :(int)rectCount`** Draws the `NXSplitView`
- **`drawDivider:(const NXRect *)aRect`** Draws the divider
- **`setAutoresizeSubviews:(BOOL)flag`** Ensures that the subviews are automatically resized

Assigning a Delegate

- **`setDelegate:anObject`** Sets the `NXSplitView`'s delegate
- **`delegate`** Returns the `NXSplitView`'s delegate

Implemented by the Delegate

- **`splitViewDidResizeSubviews:sender`** Informs the delegate that subviews were resized
- **`splitView:sender`** Limits divider travel
 `getMinY:(NXCoord *)minY`
 `maxY:(NXCoord *)maxY`
 `ofSubviewAt:(int)offset`
- **`splitView:sender`** Allows custom resizing behavior
 `resizeSubviews:(const NXSize *)oldSize`

Object Additions

This method is declared in the Application Kit as an addition to the root Object class.

Sending Messages Determined at Run Time

- **perform:(SEL)*aSelector***
 with:*anObject*
 afterDelay:(int)*ms*
 cancelPrevious:(BOOL)*flag*
Sends an *aSelector* message to the receiver after *ms* delay

OpenPanel

Inherits From: SavePanel : Panel : Window : Responder : Object

Creating and Freeing an OpenPanel

- + **new**
Returns the shared OpenPanel object
- + **newContent:(const NXRect *)*contentRect***
 style:(int)*aStyle*
 backing:(int)*bufferingType*
 buttonMask:(int)*mask*
 defer:(BOOL)*flag*
Returns the shared OpenPanel object
- **free**
Deallocates the OpenPanel object

Setting the OpenPanel Class

- + **setOpenPanelFactory:*class***
Sets class for initializing an OpenPanel

Filtering Files

- **allowMultipleFiles:**(BOOL)*flag* Sets whether the user can open multiple files

Querying the Chosen Files

- (const char *const *)**filenames** Gets the names of the selected files

Running the OpenPanel

- (int)**runModalForDirectory:**(const char *)*path* Displays the panel and begins its event loop
 file:(const char *)*name*
- (int)**runModalForDirectory:**(const char *)*path* Displays the panel and begins its event loop
 file:(const char *)*name*
 types:(const char *const *)*fileTypes*
- (int)**runModalForTypes:**(const char *const *)*fileTypes* Displays the panel and begins its event loop

PageLayout

Inherits From: Panel : Window : Responder : Object

Creating and Freeing a PAgeLayout Instance

- + **new** Returns a default PageLayout object
- + **newContent:**(const NXRect *)*contentRect* Used in PageLayout instantiation
 style:(int)*aStyle*
 backing:(int)*bufferingType*
 buttonMask:(int)*mask*
 defer:(BOOL)*flag*

- **free**

Deallocates the PageLayout panel

Running the PageLayout Panel

- (int)**runModal**

Displays the panel and begins its event loop

Customizing the PageLayout Panel

- **setAccessoryView:***aView*

Adds a View to the panel

- **accessoryView**

Returns the PageLayout's accessory View

Updating the Panel's Display

- **pickedLayout:***sender*

Updates the panel when a new layout is selected

- **pickedOrientation:***sender*

Updates the panel with the selected orientation

- **pickedPaperSize:***sender*

Updates the panel when a paper size is selected

- **pickedUnits:***sender*

Updates the panel when a new unit is selected

- **textDidEnd:***textObject*

Updates the panel when the user finishes typing a page size

endChar:(unsigned short)*theChar*

- (BOOL)**textWillChange:***textObject*

Updates the panel when a page size is typed

- **convertOldFactor:**(float *)*old*

Converts units for **pickedUnits:** method

newFactor:(float *)*new*

- **pickedButton:***sender*

Stops the event loop

Communicating with the PrintInfo Object

- **readPrintInfo**

Reads the PageLayout's values from the PrintInfo object

- **writePrintInfo**

Writes the PageLayout's values to the PrintInfo object

Panel

Inherits From: Window : Responder : Object

Initializing a New Panel

- **init** Initializes the new Panel with default values
- **initWithContent:(const NXRect *)contentRect style:(int)aStyle backing:(int)bufferingType buttonMask:(int)mask defer:(BOOL)flag** Initializes the new Panel as specified

Handling Events

- (BOOL)**commandKey:(NXEvent *)theEvent** Initiates **performKeyEquivalent:** messages
- **keyDown:(NXEvent *)theEvent** Convert key-down event to a **commandKey:** message

Determining the Panel Interface

- **setBecomeKeyOnlyIfNeeded:(BOOL)flag** Sets whether Panel waits to become key window
- (BOOL)**doesBecomeKeyOnlyIfNeeded** Returns whether Panel waits to become key window
- **setFloatingPanel:(BOOL)flag** Sets whether the Panel floats above other windows
- (BOOL)**isFloatingPanel** Returns whether the Panel floats above other windows
- **setWorksWhenModal:(BOOL)flag** Sets whether the Panel can operate on an attention panel
- (BOOL)**worksWhenModal** Returns whether Panel can operate on an attention panel

Pasteboard

Inherits From: Object

Creating and Freeing a Pasteboard

+ new	Returns the selection Pasteboard object
+ newName: (const char *) <i>name</i>	Returns the Pasteboard object named <i>name</i>
+ newUnique	Creates a uniquely named Pasteboard
- free	Releases the Pasteboard object's storage
- freeGlobally	Frees the object and the domain for its name

Getting Data in Different Formats

+ newByFilteringFile: (const char *) <i>filename</i>	Creates a pasteboard with all types for <i>filename</i>
+ newByFilteringData: (NXData *) <i>data</i> ofType: (const char *) <i>type</i>	Creates a pasteboard with all types for <i>data</i>
+ newByFilteringTypesInPasteboard: (Pasteboard *) <i>pboard</i>	Creates a pasteboard with all types filterable from <i>pboard</i>
+ (NXAtom *) typesFilterableTo: (const char *) <i>type</i>	Returns all types <i>type</i> can be filtered to

Referring to a Pasteboard by Name

+ newName: (const char *) <i>name</i>	Returns the Pasteboard object named <i>name</i>
- (const char *) name	Returns the Pasteboard object's name

Writing Data

- declareTypes: (const char *const *) <i>newTypes</i> num: (int) <i>numTypes</i> owner: <i>newOwner</i>	Sets data types and owner of the Pasteboard
- (int) addTypes: (const char *const *) <i>newTypes</i> num: (int) <i>numTypes</i> owner: <i>newOwner</i>	Adds data types to the pasteboard
- writeType: (const char *) <i>dataType</i> data: (const char *) <i>theData</i>	Writes <i>theData</i> to the pasteboard server

- **length:**(int)*numBytes*
- **writeType:**(const char *)*dataType*
fromStream:(NXStream *)*stream*
- (BOOL)**writeFileContents:**
(const char *)*filename*

Writes stream data to the pasteboard server

Writes data from *filename* to the pasteboard server

Discerning Types

- (const NXAtom *)**types**
- (const char *)**findAvailableTypeFrom:**
(const char *const *)*types*

Returns an array of the Pasteboard's data types

Returns first type in *types* that matches a pasteboard type

Reading Data

- (int)**changeCount**
- **readType:**(const char *)*dataType*
data:(char **)*theData*
length:(int *)*numBytes*
- (NXStream *)**readTypeToStream:**
(const char *)*dataType*
- (char *)**readFileContentsType:**
(const char *)*type*
toFile:(const char *)*filename*
- **deallocatePasteboardData:**(char *)*data*
length:(int)*numBytes*

Returns the Pasteboard's change count

Reads data from the pasteboard server

Returns a stream to pasteboard data

Writes pasteboard data to a file

Deallocates data received from the pasteboard

Methods Implemented by the Owner

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

Implemented to write promised data to *sender* as *type*

Notifies prior owner that ownership changed

PopUpList

Inherits From: Menu : Panel : Window : Responder : Object

Initializing a PopUpList

- **init** Initializes a new PopUpList

Setting Up the Items

- **addItem:(const char *)title** Adds an item with *title* as its title to the end of the list
- **insertItem:(const char *)title
at:(unsigned int)index** Inserts an item with *title* as its title at position *index*
- **removeItem:(const char *)title** Removes the item matching *title*
- **removeItemAt:(unsigned int)index** Removes the item at the specified *index*
- **(int)indexOfItem:(const char *)title** Returns the index of the item matching *title*
- **(unsigned int)count** Returns the number of items in the list

Interacting with the Trigger Button

- **changeButtonTitle:(BOOL)flag** Sets whether the PopUpList is a pop-up or a pull-down list
- **getButtonFrame:(NXRect *)bFrame** Gets the size needed for the Button that pops up the list

Activating the PopUpList

- **popUp:trigger** Pops the list up over *trigger*

Returning the User's Selection

- (const char *)**selectedItem**

Returns the title of selected item

Modifying the Items

- **setFont:***fontObject*

Sets the Font used to draw the items

- **font**

Returns the Font used to draw the items

Target and Action

- **setAction:**(SEL)*aSelector*

Sets the PopUpList's action method to *aSelector*

- (SEL)**action**

Returns the PopUpList's action method

- **setTarget:***anObject*

Sets the PopUpList's target object to *anObject*

- **target**

Returns the PopUpList's target object

Resizing the PopUpList

- **sizeWindow:**(NXCoord)*width* :(NXCoord)*height* Resizes the PopUpList to *width*, *height*

PrintInfo

Inherits From: Object

Initializing and Freeing a PrintInfo Instance

- **init**

Initializes the PrintInfo instance after it's allocated

- **free**

Deallocates the PrintInfo object

Defining the Printing Rectangle

- **setMarginLeft:**(NXCoord)*leftMargin*
 right:(NXCoord)*rightMargin*

Sets the margins

- top:**(NXCoord)*topMargin*
- bottom:**(NXCoord)*bottomMargin*
- **getMarginLeft:**(NXCoord *)*leftMargin*
- right:**(NXCoord *)*rightMargin*
- top:**(NXCoord *)*topMargin*
- bottom:**(NXCoord *)*bottomMargin*
- **setOrientation:**(char)*mode*
- andAdjust:**(BOOL)*flag*
- (char)**orientation**
- **setPaperRect:**(const NXRect *)*aRect*
- andAdjust:**(BOOL)*flag*
- (const NXRect *)**paperRect**
- **setPaperType:**(const char *)*type*
- andAdjust:**(BOOL)*flag*
- (const char *)**paperType**

Returns the margins by reference

Sets the orientation as portrait or landscape

Returns the orientation is portrait or landscape

Sets the width and height of the paper

Returns the rectangle for the paper size

Sets the paper type

Returns the paper type

Page Range

- **setFirstPage:**(int)*anInt*
- (int)**firstPage**
- **setLastPage:**(int)*anInt*
- (int)**lastPage**
- **setAllPages:**(BOOL)*flag*
- (BOOL)**isAllPages**
- (int)**currentPage**

Sets the page number of first page to be printed

Returns the page number of the first page to be printed

Sets the page number of last page to be printed

Returns the page number of the last page to be printed

Sets whether all the pages are to be printed

Returns whether all the pages are to be printed

Returns the page number of the page being printed

Pagination and Scaling

- **setHorizPagination:**(int)*mode*
- (int)**horizPagination**
- **setVertPagination:**(int)*mode*
- (int)**vertPagination**

Sets the horizontal pagination mode

Returns the horizontal pagination mode

Sets the vertical pagination mode

Returns the vertical pagination mode

- **setScaleFactor:**(float)*aFloat* Sets the scaling factor
- (float)**scalingFactor** Returns the scaling factor

Positioning the Image on the Page

- **setHorizCentered:**(BOOL)*flag* Sets whether the image is centered horizontally
- (BOOL)**isHorizCentered** Returns whether the image is centered horizontally
- **setVertCentered:**(BOOL)*flag* Sets whether the image is centered vertically
- (BOOL)**isVertCentered** Returns whether the image is centered vertically
- **setPagesPerSheet:**(short)*aShort* Sets the number of pages printed per sheet of paper
- (short)**pagesPerSheet** Returns the number of pages printed per sheet of paper

Print Job Attributes

- **initializeJobDefaults** Invoked automatically to initialize printing defaults
- **setJobFeature:**(const char *)*feature*
 toValue:(const char *)*value* Sets the value of the given printing job feature
- (const char *)**valueForJobFeature:**(const char *)*feature* Returns the value for the given printing job feature
- **removeJobFeature:**(const char *)*key* Removes the given printing job feature
- (const char **)**jobFeatures** Returns the keys to the job features table
- **setPageOrder:**(char)*mode* Sets the order in which pages will be printed
- (char)**pageOrder** Returns the order in which pages will be printed
- **setReversePageOrder:**(BOOL)*flag* Sets whether the page order is reversed
- (BOOL)**reversePageOrder** Returns whether the page order is reversed
- **setCopies:**(int)*anInt* Sets the number of copies to be printed
- (int)**copies** Returns the number of copies to be printed
- **setPaperFeed:**(const char *)*paperFeedSlot* Sets the paper feed slot used during printing
- (const char *)**paperFeed** Returns the paper feed slot used during printing

Specifying the Printer

- + **setDefaultPrinter:**(NXPrinter *)*printer* *Sets the user's default printer*
- + (NXPrinter *)**getDefaultPrinter** *Returns the user's default printer*
- **setPrinter:**(NXPrinter *)*aPrinter* Sets the printer that's used in subsequent printing jobs
- (NXPrinter *)**printer** Returns the NXPrinter that's used for printing

Spooling

- **setOutputFile:**(const char *)*aString* Sets the output file for printing
- (const char *)**outputFile** Returns the output file for printing
- **setContext:**(DPSContext)*aContext* Sets the DPS context used for printing
- (DPSContext)**context** Returns the DPS context used for printing

Archiving

- **read:**(NXTypedStream *)*stream* Reads the PrintInfo from the typed stream
- **write:**(NXTypedStream *)*stream* Writes the PrintInfo to the typed stream

PrintPanel

Inherits From: Panel : Window : Responder : Object

Creating and Freeing a PrintPanel

- + **new** Returns a default PrintPanel object
- + **newContent:**(const NXRect *)*contentRect* Returns a PrintPanel object
- style:**(int)*aStyle*
- backing:**(int)*bufferingType*
- buttonMask:**(int)*mask*
- defer:**(BOOL)*flag*
- **free** Deallocates the PrintPanel

Customizing the PrintPanel

- **setAccessoryView:***aView*
- **accessoryView**

Adds a View to the panel

Returns the accessory View

Running the Panel

- **(int)runModal**
- **pickedButton:***sender*

Displays the Print panel and begins its event loop

Stops the event loop

Updating the Panel's Display

- **pickedAllPages:***sender*
- **(BOOL)textViewWillChange:***textObject*

Updates the panel when the user chooses all pages

Updates the panel when user types pages to print

Communicating with the PrintInfo Object

- **updateFromPrintInfo**
- **finalWritePrintInfo**

Reads PrintPanel's values from the PrintInfo object

Writes PrintPanel's values to the PrintInfo object

Responder

Inherits From: Object

Managing the NeXT Responder

- **setNextResponder:***aResponder*
- **nextResponder**

Makes *aResponder* the receiver's next responder

Returns the receiver's next responder

Determining the First Responder

- | | |
|---------------------------------------|--|
| - (BOOL) acceptsFirstResponder | Returns NO to refuse first responder status |
| - becomeFirstResponder | Notifies the receiver it's the first responder |
| - resignFirstResponder | Notifies the receiver it's not the first responder |

Aiding Event Processing

- | | |
|---|--|
| - (BOOL) performKeyEquivalent: (NXEvent *) <i>theEvent</i> | Returns NO to indicate <i>theEvent</i> isn't handled |
| - (BOOL) tryToPerform: (SEL) <i>anAction</i>
with: <i>anObject</i> | Aids in dispatching action messages |

Forwarding Event Messages

- | | |
|--|---|
| - mouseDown: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - rightMouseDown: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - mouseDragged: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - rightMouseDragged: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - mouseUp: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - rightMouseUp: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - mouseMoved: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - mouseEntered: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - mouseExited: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - keyDown: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - keyUp: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - flagsChanged: (NXEvent *) <i>theEvent</i> | Passes the message to the receiver's next responder |
| - noResponderFor: (const char *) <i>eventType</i> | Prints warning message to syslog if debugging |

Services Menu Support

- **validRequestorForSendType:**(NXAtom)*typeSent*
andReturnType:(NXAtom)*typeReturned* Implemented by subclasses to determine available services

Archiving

- **read:**(NXTypedStream *)*stream* Reads the Responder from the typed stream *stream*
- **write:**(NXTypedStream *)*stream* Writes the Responder to the typed stream *stream*

SavePanel

Inherits From: Panel : Window : Responder : Object

Creating and Freeing a SavePanel

- + **newContent:**(const NXRect *)*contentRect* Creates and returns a SavePanel object
style:(int)*aStyle*
backing:(int)*bufferingType*
buttonMask:(int)*mask*
defer:(BOOL)*flag*
- **free** Deallocates the SavePanel

Setting the SavePanel Class

- + **setSavePanelFactory:***class* Sets class for initializing an SavePanel

Customizing the SavePanel

- **setAccessoryView:***aView* Adds application-customized view to the panel
- **accessoryView** Returns the application-customized view
- **setTitle:**(const char *)*title* Sets the title of the SavePanel to *title*

- **setPrompt:**(const char *)*prompt* Sets the title of the file name form field

Setting Directory and File Type

- **setDirectory:**(const char *)*path* Sets the current directory of the SavePanel
- **setRequiredFileType:**(const char *)*type* Sets the required file type (if any)
- (const char *)**requiredFileType** Gets the required file type (if any)

Running the SavePanel

- (int)**runModalForDirectory:**(const char *)*path* Displays the SavePanel and begins its event loop
 file:(const char *)*name*
- (int)**runModal** Displays the SavePanel and begins its event loop

Reading Save Information

- (const char *)**directory** Returns directory chosen file resides in
- (const char *)**filename** Returns full name of file to be saved

Completing a Partial Filename

- (BOOL)**commandKey:**(NXEvent *)*theEvent* Enables command-space to do filename completion

Target and Action Methods

- **ok:***sender* Method invoked by the OK button
- **cancel:***sender* Method invoked by the Cancel button

Responding to User Input

- **selectText:***sender* Called when TAB is pressed in the form
- **textDidEnd:***textObject* Determines whether TAB or BACKTAB was pressed
 endChar:(unsigned short)*endChar*

- **textDidGetKeys:***textObj* **isEmpty:**(BOOL)*flag* Determines whether there's any text in the form

Setting the Delegate

- **setDelegate:***anObject* Makes *anObject* the SavePanel's delegate

Methods implemented by the Delegate

- (int)**panel:***sender* **compareFileNames:**(const char *)*fileName1* **:**(const char *)*fileName2* **checkCase:**(BOOL)*flag* Returns 1 if *fileName1* precedes *fileName2*, -1 in the opposite case, 0 if the two are equivalent
- (BOOL)**panel:***sender* **filterFile:**(const char *)*filename* **inDirectory:**(const char *)*directory* YES if *filename* can be saved in directory
- (BOOL)**panelValidateFileNames:***sender* YES if the filename is acceptable to the delegate

Scroller

Inherits From: Control : View : Responder : Object

Initializing a Scroller

- **initWithFrame:**(const NXRect *)*frameRect* Initializes a new Scroller

Laying out the Scroller

- (NXRect *)**calcRect:**(NXRect *)*aRect* **forPart:**(int)*partCode* Gets the rectangle that encloses *partCode*
- **checkSpaceForParts** Checks for room for knob and scroll buttons
- **setArrowsPosition:**(int)*where* Sets position of scroll buttons in Scroller

Setting Scroller values

- (float)**floatValue** Returns Scroller's float value
- **setFloatValue:**(float)*aFloat* Sets value; positions knob
- **setFloatValue:**(float)*aFloat* :(float)*percent* Sets value; positions and sizes knob

Resizing the Scroller

- **sizeTo:**(NXCoord)*width* :(NXCoord)*height* Sizes the Scroller

Displaying

- **drawArrow:**(BOOL)*whichButton* :(BOOL)*flag* Draws highlighted and unhighlighted arrows
- **drawKnob** Draws the knob
- **drawParts** Caches Bitmaps for knob and scroll arrows
- **drawSelf:**(const NXRect *)*rects* :(int)*rectCount* Draws the Scroller
- **highlight:**(BOOL)*flag* Highlights scroll button that's under mouse

Target and Action

- **setAction:**(SEL)*aSelector* Sets the Scroller's action to *aSelector*
- (SEL)**action** Returns the Scroller's action
- **setTarget:***anObject* Sets the Scroller's target to *anObject*
- **target** Returns the Scroller's target

Handling Events

- (BOOL)**acceptsFirstMouse** Makes the Scroller respond to the first mouse event
- (int)**hitPart** Returns Scroller part that received mouse-down
- **mouseDown:**(NXEvent *)*theEvent* Responds to mouse-down events
- (int)**testPart:**(const NXPoint *)*thePoint* Returns Scroller part that's under *thePoint*

- **trackKnob:**(NXEvent *)*theEvent*
- **trackScrollButtons:**(NXEvent *)*theEvent*

Responds to mouse-down events on the knob

Responds to mouse-down events on buttons

Archiving

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

Ensures that Scroller's Bitmaps are created

Reads the Scroller from the typed stream

Writes the Scroller to the typed stream

ScrollView

Inherits From: View : Responder : Object

Initializing a ScrollView

- **initWithFrame:**(const NXRect *)*frameRect*

Initializes a new ScrollView

Determining Component Sizes

- **getContentSize:**(NXSize *)*contentViewSize*
- **getDocVisibleRect:**(NXRect *)*aRect*

Gets the content view's size

Gets the visible portion of the document view

Laying Out the ScrollView

- + **getContentSize:**(NXSize *)*cSize*
forFrameSize:(const NXSize *)*fSize*
horizScroller:(BOOL)*hFlag*
vertScroller:(BOOL)*vFlag*
borderType:(int)*aType*
- + **getFrameSize:**(NXSize *)*fSize*

Gets the content view size for the given ScrollView size

Gets the ScrollView size for the given content view size

forContentSize:(const NXSize *)*cSize*
horizScroller:(BOOL)*hFlag*
vertScroller:(BOOL)*vFlag*
borderType:(int)*aType*

- **resizeSubviews:**(const NXSize *)*oldSize*
- **setHorizScrollerRequired:**(BOOL)*flag*
- **setVertScrollerRequired:**(BOOL)*flag*
- **tile**

Retiles the ScrollView after a **sizeTo::**
Makes space for a horizontal scroller
Makes space for a vertical scroller
Retiles the scrollers and content view

Managing Component Views

- **setDocView:***aView*
- **docView**
- **setHorizScroller:***anObject*
- **horizScroller**
- **setVertScroller:***anObject*
- **vertScroller**
- **reflectScroll:***cView*

Makes *aView* the ScrollView's document view
Returns the current document view
Sets the horizontal Scroller object
Returns the horizontal Scroller
Sets the vertical Scroller object
Returns the vertical Scroller
Updates the Scrollers

Modifying Graphic Attributes

- **setBorderType:**(int)*aType*
- (int)**borderType**
- **setBackgroundColor:**(NXColor)*color*
- (NXColor) **backgroundColor**
- **setBackgroundGray:**(float)*value*
- (float)**backgroundGray**

Determines the border type of the ScrollView
Returns the border type
Sets the ScrollView's background color
Returns the ScrollView's background color
Sets the ScrollView's background gray
Returns the ScrollView's background gray

Setting Scrolling Behavior

- **setCopyOnScroll:**(BOOL)*flag*

Sets how newly exposed areas are redrawn

- | | |
|--|---|
| - setDisplayOnScroll: (BOOL) <i>flag</i> | Sets how the doc view is displayed during scrolling |
| - setDynamicScrolling: (BOOL) <i>flag</i> | Sets how the doc view is displayed during scrolling |
| - setLineScroll: (float) <i>value</i> | Sets the amount to scroll when scrolling a line |
| - setPageScroll: (float) <i>value</i> | Sets the amount of overlap for a page scroll |

Displaying

- **drawSelf:**(const NXRect *)*rects* :(int)*rectCount* Draws the ScrollView

Managing the Cursor

- **setDocCursor:***anObj* Sets the cursor for the document view

Archiving

- **read:**(NXTypedStream *)*stream* Reads the ScrollView from the typed stream
- **write:**(NXTypedStream *)*stream* Writes the ScrollView to the typed stream

SelectionCell

Inherits From: Cell : Object

Creating a SelectionCell

- **init** Initializes a new SelectionCell with ^aListIt^em^o as its title
- **initTextCell:**(const char *)*aString* Initializes a new SelectionCell with *aString* as its title

Determining Component Sizes

- **calcCellSize:**(NXSize *)*theSize* Calculates the size of the SelectionCell within *aRect*

inRect:(const NXRect *)*aRect*

Accessing Graphic Attributes

- **setLeaf:**(BOOL)*flag*
- (BOOL)**isLeaf**
- (BOOL)**isOpaque**

Sets whether SelectionCell is a leaf or a branch

Returns whether the SelectionCell is a leaf or a branch

Returns YES, since SelectionCells are opaque

Displaying

- **drawSelf:**(const NXRect *)*cellFrame*
inView:*aView*
- **drawInside:**(const NXRect *)*cellFrame*
inView:*aView*
- **highlight:**(const NXRect *)*cellFrame*
inView:*aView*
lit:(BOOL)*flag*

Draws the SelectionCell in *cellFrame* within *aView*

Draws the inside of the SelectionCell in *aView*

Highlights the SelectionCell within *cellFrame* in
controlView

Archiving

- **awake**

Reinitializes the SelectionCell when it's unarchived

Slider

Inherits From: Control : View : Responder : Object

Setting Slider's Cell Class

- + **setCellClass:***classId*

Sets the subclass of SliderCell used by Slider

Initializing a new Slider

- **initWithFrame:**(const NXRect *)*frameRect*

Initializes a new Slider in *frameRect*

Modifying a Slider's appearance

- **setKnobThickness:**(NXCoord)*aFloat*

Sets the knob's thickness to *aFloat*

- (NXCoord)**knobThickness**

Returns the knob's thickness

- **setImage:***image*

Sets the background image to *image*

- **image**

Returns the background image

- **setTitle:**(const char *)*aString*

Sets the background title to a copy of *aString*

- **setTitleNoCopy:**(const char *)*aString*

Sets the background title to *aString*

- (const char *)**title**

Returns the background title

- **setTitleCell:***aCell*

Sets the Cell used to draw the background title

- **titleCell**

Returns the Cell used to draw the background title

- **setTitleFont:***fontObject*

Sets the Font used to draw the background title

- **titleFont**

Returns the Font used to draw the background title

- **setTitleColor:**(NXColor)*aColor*

Sets the color of text in the background title to *aColor*

- (NXColor)**titleColor**

Returns the color of text in the background title

- **setTitleGray:**(float)*aFloat*

Sets the gray of text in the background title to *aFloat*

- (float)**titleGray**

Returns the gray of text in the background title

- (int)**isVertical**

Returns 1 if vertical, 0 if horizontal, -1 if unknown

Setting Value Limits

- **setMinValue:**(double)*aDouble*

Sets the Slider's minimum value to *aDouble*

- (double)**minValue**

Returns the Slider's minimum value

- **setMaxValue:**(double)*aDouble*

Sets the Slider's maximum value to *aDouble*

- (double)**maxValue**

Returns the Slider's maximum value

Resizing the Slider

- **sizeToFit**

Modifies the Slider's size to fit its Cell

Handling Events

- (BOOL)**acceptsFirstMouse**

Returns YES, since Sliders always accept first mouse

- **setEnabled:**(BOOL)*flag*

Sets whether the Slider reacts to events

- **mouseDown:**(NXEvent *)*theEvent*

Responds to mouse-down by initiating tracking

SliderCell

Inherits From: ActionCell : Cell : Object

Initializing a new SliderCell

- **init**

Initializes a new SliderCell

Determining Component Sizes

- **calcCellSize:**(NXSize *)*theSize*

inRect:(const NXRect *)*aRect*

Returns the size of the SliderCell

- **getKnobRect:**(NXRect*)*knobRect*

flipped:(BOOL)*flipped*

Gets the rectangle the knob will be drawn in

Setting Value Limits

- **setMinValue:**(double)*aDouble*

Sets the SliderCell's minimum value to *aDouble*

- (double)**minValue**

Returns the SliderCell's minimum value

- **setMaxValue:**(double)*aDouble*

Sets the maximum value of the SliderCell to *aDouble*

- (double)**maxValue**

Returns the SliderCell's maximum value

Setting Values

- **setDoubleValue:**(double)*aDouble*
- (double)**doubleValue**
- **setFloatValue:**(float)*aFloat*
- (float)**floatValue**
- **setIntValue:**(int)*anInt*
- (int)**intValue**
- **setStringValue:**(const char *)*aString*
- (const char *)**stringValue**

Sets the SliderCell's value to *aDouble*

Returns the SliderCell's value as a **double**

Sets the SliderCell's value to *aFloat*

Returns the SliderCell's value as a **float**

Sets the SliderCell's value to *anInt*

Returns SliderCell's value as an **int**

Sets the SliderCell's value to a number represented by *aString*

Returns the SliderCell's value as a string

Modifying Graphic Attributes

- **setKnobThickness:**(NXCoord)*aFloat*
- (NXCoord)**knobThickness**
- **setImage:***image*
- **image**
- **setTitle:**(const char *)*aString*
- **setTitleNoCopy:**(const char *)*aString*
- (const char *)**title**
- **setTitleCell:***aCell*
- **titleCell**
- **setTitleFont:***fontObject*
- **titleFont**
- **setTitleColor:**(NXColor)*aColor*
- (NXColor)**titleColor**
- **setTitleGray:**(float)*aFloat*
- (float)**titleGray**
- (BOOL)**isOpaque**
- (int)**isVertical**

Sets the knob's thickness to *aFloat*

Returns the knob's thickness

Sets the background image to *image*

Returns the background image

Sets the background title to a copy of *aString*

Sets the background title to *aString*

Returns the background title

Sets the Cell used to draw the background title

Returns the Cell used to draw the background title

Sets the Font used to draw the background title

Returns the Font used to draw the background title

Sets the color of text in the background title to *aColor*

Returns the color of text in the background title

Sets the gray of text in the background title to *aFloat*

Returns the gray of text in the background title

Returns YES (SliderCells are always opaque)

Returns 1 if vertical, 0 if horizontal, -1 if unknown

Displaying the SliderCell

- drawSelf: (const NXRect *) <i>cellFrame</i> inView: <i>controlView</i>	Draws the SliderCell's bar and knob in <i>controlView</i>
- drawInside: (const NXRect *) <i>cellFrame</i> inView: <i>controlView</i>	Draws the inside of the SliderCell in <i>controlView</i>
- drawBarInside: (const NXRect *) <i>aRect</i> flipped: (BOOL) <i>flipped</i>	Draws the SliderCell's bar
- drawKnob	Draws the SliderCell's knob
- drawKnob: (const NXRect*) <i>knobRect</i>	Draws the SliderCell's knob in <i>knobRect</i>

Modifying Behavior

- setEnabled: (BOOL) <i>flag</i>	Sets whether the SliderCell reacts to events
- setContinuous: (BOOL) <i>flag</i>	Sets whether the Slider is continuous
- (BOOL) isContinuous	Returns whether the Slider is continuous
- setAltIncrementValue: (double) <i>incValue</i>	Sets how far the SliderCell moves when the knob is dragged one pixel with the Alternate key held down
- (double) altIncrementValue	Returns how far the SliderCell moves when alt-dragged

Tracking the Mouse

+ (BOOL) prefersTrackingUntilMouseUp	Returns YES, since SliderCells must track even when the mouse leaves their bounds
- (BOOL) trackMouse: (NXEvent *) <i>theEvent</i> inRect: (const NXRect *) <i>cellFrame</i> ofView: <i>controlView</i>	Tracks the mouse
- (BOOL) startTrackingAt: (const NXPoint *) <i>startPoint</i> inView: <i>controlView</i>	Begins a tracking session
- (BOOL) continueTracking: (const NXPoint *) <i>lastPoint</i> at: (const NXPoint *) <i>currentPoint</i> inView: <i>controlView</i>	Continues tracking the mouse
- stopTracking: (const NXPoint *) <i>lastPoint</i>	Ends the current tracking session

at:(const NXPoint *)*stopPoint*
inView:*controlView*
mouselsUp:(BOOL)*flag*

Archiving

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

Reads the SliderCell from *stream*

Writes the SliderCell to *stream*

Caches knob icons when the SliderCell is unarchived

Speaker

Inherits From: Object

Initializing a New Speaker Instance

- **init** Initializes the Speaker after it has been allocated

Freeing a Speaker

- **free** Deallocates the Speaker (but not its ports)

Setting Up a Speaker

- **setSendTimeout:**(int)*ms* Sets how long to wait for messages to be delivered
- (int)**sendTimeout** Returns how long to wait for messages to be delivered
- **setReplyTimeout:**(int)*ms* Sets how long Speaker will wait for a reply
- (int)**replyTimeout** Returns how long Speaker will wait for a reply

Managing the Ports

- **setSendPort:**(port_t)*aPort*
- (port_t)**sendPort**
- **setReplyPort:**(port_t)*aPort*
- (port_t)**replyPort**

Makes *aPort* the port messages will be sent to
 Returns the port the Speaker will send messages to
 Makes *aPort* the port where replies are received
 Returns the port where Speaker receives replies

Standard Remote Methods

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

Sends a remote message to open *fullPath* file

Sends a remote message to open *fullPath* file

Providing for Program Control

- (int)**msgCalc:**(int *)*flag*
- (int)**msgCopyAsType:**(const char *)*aType*
ok:(int *)*flag*
- (int)**msgCutAsType:**(const char *)*aType*
ok:(int *)*flag*
- (int)**msgDirectory:**(char *const *)*fullPath*
ok:(int *)*flag*
- (int)**msgFile:**(char *const *)*fullPath*
ok:(int *)*flag*
- (int)**msgPaste:**(int *)*flag*
- (int)**msgPosition:**(char *const *)*aString*
posType:(int *)*anInt*
ok:(int *)*flag*
- (int)**msgPrint:**(const char *)*fullPath*
ok:(int *)*flag*
- (int)**msgQuit:**(int *)*flag*
- (int)**msgSelection:**(char *const *)*bytes*
length:(int *)*numBytes*

Sends message to update the current window

Sends message to copy selection as *aType* data

Sends message to cut selection as *aType* data

Sends message requesting the current directory

Sends message requesting the current document

Sends message to paste data from pasteboard

Sends message requesting selection information

Sends message to print *fullPath* file

Sends remote message for application to quit

Sends message requesting the current selection

- asType:**(const char *)*aType*
ok:(int *)*flag*
- (int)**msgSetPosition:**(const char *)*aString* Sends message to scroll so *aString* is visible
posType:(int)*anInt*
andSelect:(int)*sflag*
ok:(int *)*flag*
- (int)**msgVersion:**(char *const *)*aString* Sends message requesting version information
ok:(int *)*flag*

Sending Remote Messages

- (int)**performRemoteMethod:**(const char *)*methodName* Sends remote *methodName* message
- (int)**performRemoteMethod:**(const char *)*methodName*
with:(const char *)*data* Sends remote message with *numBytes* of *data*
length:(int)*numBytes*
- (int)**selectorRPC:**(const char *)*methodName* Sends remote message with variable arguments
paramTypes:(char *)*params*,
- (int)**sendOpenFileMsg:**(const char *)*fullPath* Sends an **openFile:ok:** remote message
ok:(int *)*flag*
andDeactivateSelf:(BOOL)*deactivateFirst*
- (int)**sendOpenTempFileMsg:**(const char *)*fullPath* Sends an **openTempFile:ok:** remote message
ok:(int *)*flag*
andDeactivateSelf:(BOOL)*deactivateFirst*

Assigning a Delegate

- **setDelegate:***anObject* Makes *anObject* the Speaker's delegate
- **delegate** Returns the Speaker's delegate

Archiving

- **read:**(NXTypedStream *)*stream* Reads the Speaker from *stream*

- **write:**(NXTypedStream *)*stream* Writes the Speaker to *stream*

Text

Inherits From: View : Responder : Object

Conforms To: NXChangeSpelling
NXIgnoreMisspelledWords
NXReadOnlyTextStream
NXSelectText

Initializing the Class Object

+ setDefaultFont: <i>anObject</i>	Makes <i>anObject</i> the default Font object for Text
+ getDefaultFont	Returns the default Font object for Text
+ excludeFromServicesMenu: (BOOL) <i>flag</i>	Controls whether Text objects register for services
+ registerDirective: (const char *) <i>directive</i> forClass: <i>class</i>	Associates an RTF control word with a class object
+ initialize	Performed automatically at startup

Initializing a New Text Object

- initWithFrame: (const NXRect *) <i>frameRect</i>	Initialize a new Text object
- initWithFrame: (const NXRect *) <i>frameRect</i> text: (const char *) <i>theText</i> alignment: (int) <i>mode</i>	Initialize a new Text object

Freeing a Text Object

- free	Frees the Text object and its storage
---------------	---------------------------------------

Modifying the Frame Rectangle

- setMaxSize: (const NXSize *) <i>newMaxSize</i>	Sets maximum size of the Text object
- getMaxSize: (NXSize *) <i>theSize</i>	Gets maximum size of the Text object
- setMinSize: (const NXSize *) <i>newMinSize</i>	Sets minimum size of the Text object
- getMinSize: (NXSize *) <i>theSize</i>	Gets minimum size of the Text object
- setVertResizable: (BOOL) <i>flag</i>	Sets whether frame height can change
- (BOOL) isVertResizable	Returns whether frame height can change
- setHorizResizable: (BOOL) <i>flag</i>	Sets whether frame width can change
- (BOOL) isHorizResizable	Returns whether frame width can change
- sizeTo: (NXCoord) <i>width</i> :(NXCoord) <i>height</i>	Resizes the Text object to <i>width</i> and <i>height</i>
- sizeToFit	Resizes the frame to accommodate the text
- resizeText: (const NXRect *) <i>oldBounds</i> :(const NXRect *) <i>maxRect</i>	Used by Text object to resize and redisplay itself
- moveTo: (NXCoord) <i>x</i> :(NXCoord) <i>y</i>	Moves the Text object to (x, y)

Laying Out the Text

- setMarginLeft: (NXCoord) <i>leftMargin</i> right: (NXCoord) <i>rightMargin</i> top: (NXCoord) <i>topMargin</i> bottom: (NXCoord) <i>bottomMargin</i>	Adjusts margins around the text
- getMarginLeft: (NXCoord *) <i>leftMargin</i> right: (NXCoord *) <i>rightMargin</i> top: (NXCoord *) <i>topMargin</i> bottom: (NXCoord *) <i>bottomMargin</i>	Gets dimensions of margins around the text
- getMinWidth: (NXCoord *) <i>width</i> minHeight: (NXCoord *) <i>height</i> maxWidth: (NXCoord) <i>widthMax</i> maxHeight: (NXCoord) <i>heightMax</i>	Calculates area needed to display the text
- setAlignment: (int) <i>mode</i>	Sets how text is aligned at margins

- (int) alignment	Returns how text is aligned at margins
- alignSelLeft: <i>sender</i>	Aligns the text to the left margin
- alignSelCenter: <i>sender</i>	Aligns the text between the margins
- alignSelRight: <i>sender</i>	Aligns the text to the right margin
- setSelProp: (NXParagraphProp) <i>prop</i> to: (NXCoord) <i>val</i>	Sets the paragraph style for one or more paragraphs
- changeTabStopAt: (NXCoord) <i>oldX</i> to: (NXCoord) <i>newX</i>	Resets the position of the specified tab stop
- (int) calcLine	Calculates line breaks
- setCharWrap: (BOOL) <i>flag</i>	Returns whether extra long words are wrapped
- (BOOL) charWrap	Sets whether extra long words are wrapped
- setNoWrap	Disables word wrap
- setParaStyle: (void *) <i>paraStyle</i>	Sets paragraph style for the entire text
- (void *) defaultParaStyle	Returns the default paragraph style
- (void *) calcParagraphStyle: <i>fontId</i> :(int) <i>alignment</i>	Recalculates paragraph style
- setLineHeight: (NXCoord) <i>value</i>	Sets height of a line of text
- (NXCoord) lineHeight	Returns height of a line of text
- setDescentLine: (NXCoord) <i>value</i>	Sets distance from base line to bottom of line
- (NXCoord) descentLine	Returns distance from base line to bottom of line

Reporting Line and Position

- (int) lineFromPosition: (int) <i>position</i>	Converts character position to line number
- (int) positionFromLine: (int) <i>line</i>	Converts line number to character position
- (int) offsetFromPosition: (int) <i>position</i>	Returns the byte offset corresponding to <i>position</i>
- (int) positionFromOffset: (int) <i>offset</i>	Returns the position corresponding to the byte offset

Setting, Reading, and Writing the Text

- setText: (const char *) <i>aString</i>	Replaces current text with <i>aString</i>
---	---

- readText: (NXStream *) <i>stream</i>	Replaces current text with text from <i>stream</i>
- startReadingRichText	Sent before Text object begins reading RTF data
- readRichText: (NXStream *) <i>stream</i>	Replaces text with RTF data from <i>stream</i>
- readRichText: (NXStream *) <i>stream</i> atPosition: (int) <i>position</i>	Lets you add RTF data to <i>stream</i>
- finishReadingRichText	Sent after Text object reads RTF data
- (NXRTFDError) openRTFDFrom: (const char *) <i>path</i>	Opens the RTFD file package specified by <i>path</i>
- (NXRTFDError) saveRTFDTo: (const char *) <i>path</i> removeBackup: (BOOL) <i>removeBackup</i> errorHandler: <i>errorHandler</i>	Saves the contents (text and images) of the Text object to the file package specified by <i>path</i>
- writeText: (NXStream *) <i>stream</i>	Writes all the text to <i>stream</i>
- writeRichText: (NXStream *) <i>stream</i>	Writes all the text to <i>stream</i> using RTF
- writeRichText: (NXStream *) <i>stream</i> from: (int) <i>start</i> to: (int) <i>end</i>	Writes text to <i>stream</i> using RTF
- writeRTFDSelectionTo: (NXStream *) <i>stream</i>	Writes the selection's text and images to <i>stream</i>
- writeRTFDTo: (NXStream *) <i>stream</i>	Writes all the text and images to <i>stream</i>
- (NXStream *) stream	Returns stream access to Text object's text
- (NXTextBlock *) firstTextBlock	Returns pointer to first text block
- getParagraph: (int) <i>prNumber</i> start: (int *) <i>startPos</i> end: (int *) <i>endPos</i> rect: (NXRect *) <i>paragraphRect</i>	Gets position, length, and size of a paragraph
- (int) getSubstring: (char *) <i>buf</i> start: (int) <i>startPos</i> length: (int) <i>numChars</i>	Copies <i>numChars</i> at <i>startPos</i> to <i>buf</i>
- (int) byteLength	Returns length of the Text object's contents in bytes
- (int) charLength	Returns number of characters in the text
- (int) textLength	Returns number of characters in the text

Setting Editability

- **setEditable:**(BOOL)*flag*
- (BOOL)**isEditable**

Sets whether the text can be edited

Returns whether the text can be edited

Allowing Multiple Fonts and Paragraph Styles

- **setMonoFont:**(BOOL)*flag*
- (BOOL)**isMonoFont**

Controls whether multiple fonts and parastyles are OK

Returns whether only one font and parastyle is permitted

Editing the Text

- **copy:***sender*
- **copyFont:***sender*
- **copyRuler:***sender*
- **paste:***sender*
- **pasteFont:***sender*
- **pasteRuler:***sender*
- **cut:***sender*
- **delete:***sender*
- **clear:***sender*
- **selectAll:***sender*
- **selectText:***sender*

Copies selected text to the pasteboard

Copies selected text's font to the pasteboard

Copies selected text's style to the pasteboard

Replaces selection with pasteboard's contents

Replaces selection's font with pasteboard's contents

Replaces selection's style with pasteboard's contents

Deletes selected text; copies it to pasteboard

Deletes selected text

Deletes selected text

Makes receiver the first responder; selects all text

Makes receiver the first responder; selects all text

Managing the Selection

- **subscript:***sender*
- **superscript:***sender*
- **unscript:***sender*
- **underline:***sender*
- **showCaret**
- **hideCaret**

Subscripts the current selection

Superscripts the current selection

Removes sub/super script in the current selection

Toggles the underline attribute of text

Displays the previously hidden caret

Removes the caret from the text display

- setSelectable: (BOOL) <i>flag</i>	Sets whether the text can be selected
- (BOOL) isSelectable	Returns whether the text can be selected
- selectError	Selects all the text
- selectNull	Deselects the current selection
- setSel: (int) <i>start</i> :(int) <i>end</i>	Selects text from <i>start</i> through <i>end</i>
- getSel: (NXSelPt *) <i>start</i> :(NXSelPt *) <i>end</i>	Gets <i>start</i> and <i>end</i> of the selection
- replaceSel: (const char *) <i>aString</i>	Replaces the selection with <i>aString</i>
- replaceSel: (const char *) <i>aString</i> length:(int) <i>length</i>	Replaces selection with <i>length</i> bytes of <i>aString</i>
- replaceSel: (const char *) <i>aString</i> length:(int) <i>length</i> runs:(NXRunArray *) <i>insertRuns</i>	Replaces selection with <i>length</i> bytes of <i>aString</i>
- replaceSelWithRichText: (NXStream *) <i>stream</i>	Replaces selection with RTF from <i>stream</i>
- replaceSelWithRTFD: (NXStream *) <i>stream</i>	Replaces selection with RTFD data from <i>stream</i>
- scrollSelToVisible	Brings the selection within the frame rectangle

Setting the Font

- setFontPanelEnabled: (BOOL) <i>flag</i>	Sets whether the Font panel can affect text
- (BOOL) isFontPanelEnabled	Sets whether the Font panel can affect text
- changeFont: <i>sender</i>	Changes font of selection
- setFont: <i>fontObj</i>	Sets Font object for the entire text
- font	Returns a monofont Text object's font
- setFont: <i>fontObj</i> paraStyle: (void *) <i>paraStyle</i>	Sets Font and paragraph style for all text
- setSelFont: <i>fontId</i>	Sets Font object for the selection
- setSelFontFamily: (const char *) <i>fontName</i>	Sets font family for the selection
- setSelFontSize: (float) <i>size</i>	Sets font size for the selection
- setSelFontStyle: (NXFontTraitMask) <i>traits</i>	Sets font style for the selection
- setSelFont: <i>fontId</i> paraStyle: (void *) <i>paraStyle</i>	Sets font and paragraph style for the selection

Checking Spelling

- **checkSpelling:***sender*
- **showGuessPanel:***sender*

Searches for a misspelled word in the text

Displays panel suggesting spelling corrections

Managing the Ruler

- **toggleRuler:***sender*
- (NXColor)**isRulerVisible**

Controls the display of the ruler

Returns whether the ruler is visible in the superview

Finding Text

- (BOOL)**findText:**(const char *)*string*
 ignoreCase:(BOOL)*ignoreCaseFlag*
 backwards:(BOOL)*backwardsFlag*
 wrap:(BOOL)*wrapFlag*

Searches for *string* in the text, starting at the insertion point

Modifying Graphic Attributes

- **setBackgroundGray:**(float)*value*
- (float)**backgroundGray**
- **setBackgroundColor:**(NXColor)*color*
- (NXColor)**backgroundColor**
- **setSelGray:**(float)*value*
- (float)**selGray**
- (float)**runGray:**(NXRun *)*run*
- **setSelColor:**(NXColor)*color*
- (NXColor)**selColor**
- (NXColor)**runColor:**(NXRun *)*run*
- **setTextGray:**(float)*value*
- (float)**textGray**
- **setTextColor:**(NXColor)*color*

Sets the gray value of the text background

Returns the gray value of the text background

Sets background color of the text

Returns the background color of the text

Sets the gray value of the selected text

Returns the gray value of the selected text

Returns the gray value for the specified text run

Sets the color of the selected text

Returns the color of the selected text

Returns the color of the specified text run

Sets the gray value of the entire text

Returns the gray value of the entire text

Sets the text color of the entire text

- (NXColor)**textColor** Returns the text color of the draw entire text

Reusing a Text Object

- **renewFont:***newFontId*
 text:(const char *)*newText*
 frame:(const NXRect *)*newFrame*
 tag:(int)*newTag* Resets Text object to draw different text

- **renewFont:**(const char *)*newFontName*
 size:(float)*newFontSize*
 style:(int)*newFontStyle*
 text:(const char *)*newText*
 frame:(const NXRect *)*newFrame*
 tag:(int)*newTag* Resets Text object to draw different text

- **renewRuns:**(NXRunArray *)*newRuns*
 text:(const char *)*newText*
 frame:(const NXRect *)*newFrame*
 tag:(int)*newTag* Resets Text object to draw different text

- **windowChanged:***newWindow* Hides caret whenever the Text's window changes

Displaying

- **drawSelf:**(const NXRect *)*rects* :(int)*rectCount* Draws the Text object

- **setRetainedWhileDrawing:**(BOOL)*flag* Allows use of retained window when drawing

- (BOOL)**isRetainedWhileDrawing** Returns whether retained window is used for drawing

Assigning a Tag

- **setTag:**(int)*anInt* Makes *anInt* the Text object's tag

- (int)**tag** Returns the Text object's tag

Handling Event Messages

- (BOOL) acceptsFirstResponder	Returns whether receiver can be the first responder
- becomeFirstResponder	Informs Text object that it's becoming first responder
- resignFirstResponder	Stops being the first responder, if delegate agrees
- becomeKeyWindow	Activates caret if selection has width of 0
- resignKeyWindow	Deactivates the caret
- mouseDown: (NXEvent *) <i>theEvent</i>	Responds to mouse-down events
- keyDown: (NXEvent *) <i>theEvent</i>	Responds to key-down events
- moveCaret: (unsigned short) <i>theKey</i>	Moves the caret in response to arrow keys

Displaying Graphics within the Text

+ registerDirective: (const char *) <i>directive</i> forClass: <i>class</i>	Associates an RTF control word with a class object
- replaceSelWithCell: <i>cell</i>	Replaces selection with image provided by <i>cell</i>
- replaceSelWithView: <i>view</i>	Unimplemented
- setLocation: (NXPoint *) <i>origin</i> ofCell: <i>cell</i>	Sets origin of <i>cell</i>
- getLocation: (NXPoint *) <i>origin</i> ofCell: <i>cell</i>	Places coordinates of graphic object into <i>origin</i>
- getLocation: (NXPoint *) <i>origin</i> ofView: <i>view</i>	Unimplemented
- setGraphicsImportEnabled: (BOOL) <i>flag</i>	Sets whether a Text object imports TIFF and EPS images
- (BOOL) isGraphicsImportEnabled	Returns YES if the object imports TIFF and EPS images

Using the Services Menu

+ excludeFromServicesMenu: (BOOL) <i>flag</i>	Controls whether Text objects use services menu
- validRequestorForSendType: (NXAtom) <i>sendType</i> andReturnType: (NXAtom) <i>returnType</i>	Determines which Service menu items are enabled
- readSelectionFromPasteboard: <i>pboard</i>	Replaces selection with data from pboard
- (BOOL) writeSelectionToPasteboard: <i>pboard</i>	Copies selection to pboard

types:(NXAtom *)*types*

Setting Tables and Functions

- **setCharFilter:**(NXCharFilterFunc)*aFunc* Makes *aFunc* the character filter function
- (NXCharFilterFunc)**charFilter** Returns the current character filter function
- **setTextFilter:**(NXTextFilterFunc)*aFunc* Makes *aFunc* the text filter function
- (NXTextFilterFunc)**textFilter** Returns the current text filter function
- **setBreakTable:**(const NXFSM *)*aTable* Sets table defining word boundaries
- (const NXFSM *)**breakTable** Gets table defining word boundaries
- **setPreSelSmartTable:**(const unsigned char *)*aTable* Sets cut and paste table for left word boundary
- (const unsigned char *)**preSelSmartTable** Gets cut and paste table for left word boundary
- **setPostSelSmartTable:**(const unsigned char *)*aTable* Sets cut and paste table for right word boundary
- (const unsigned char *)**postSelSmartTable** Gets cut and paste table for right word boundary
- **setCharCategoryTable:**(const unsigned char *)*aTable* Sets table defining character categories
- (const unsigned char *)**charCategoryTable** Returns table defining character categories
- **setClickTable:**(const NXFSM *)*aTable* Sets table defining double-click selection
- (const NXFSM *)**clickTable** Gets table defining double-click selection
- **setScanFunc:**(NXTextFunc)*aFunc* Makes *aFunc* the scan function
- (NXTextFunc)**scanFunc** Returns the current scan function
- **setDrawFunc:**(NXTextFunc)*aFunc* Makes *aFunc* the function that draws the text
- (NXTextFunc)**drawFunc** Returns the current draw function

Printing

- **adjustPageHeightNew:**(float *)*newBottom* Assists automatic pagination of text
top:(float)*oldTop*
bottom:(float)*oldBottom*

limit:(float)*bottomLimit*

Archiving

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

Reads the Text object from the typed stream

Writes the Text object to the typed stream

Assigning a Delegate

- **setDelegate:***anObject*
- **delegate**

Makes *anObject* the Text object's delegate

Returns the Text object's delegate

Implemented by the Delegate

- **textWillResize:***sender*
- **textDidResize:***sender*
oldBounds:(const NXRect *)*oldBounds*
invalid:(NXRect *)*invalidRect*
- (BOOL)**textWillChange:***sender*
- **textDidChange:***sender*
- (BOOL)**textWillEnd:***sender*
- **textDidEnd:***sender*
endChar:(unsigned short)*whyEnd*
- **textDidGetKeys:***sender* **isEmpty:**(BOOL)*flag*
- **textWillSetSel:***sender* **toFont:***font*
- **textWillConvert:***sender*
fromFont:*from*
toFont:*to*
- **textWillStartReadingRichText:***sender*
- **textWillFinishReadingRichText:***sender*
- **textWillWrite:***sender*
paperSize:(NXSize *)*paperSize*

Informs delegate of impending size change

Reports size change to delegate

Informs delegate of impending text change

Alerts delegate to change in text

Warns of impending loss of first responder status

Reports to delegate loss of first responder status

Informs delegate of each text change

Lets delegate intercede in the updating of the Font panel

Lets delegate intercede in selection's font change

Informs delegate that Text object will read RTF data

Informs delegate that Text finished reading RTF data

Lets the delegate specify paper size

- **textDidLoad:***sender*
paperSize:(NXSize *)*paperSize* Lets the delegate review paper size

Implemented by an Embedded Graphic Object

- **calcCellSize:**(NXSize *)*theSize* Provides the size of the object
- **drawSelf:**(const NXRect *)*rect*
inView:*view* Draws the object
- **highlight:**(const NXRect *)*rect*
inView:*view*
lit:(BOOL)*flag* Highlights or unhighlights the object
- **readRichText:**(NXStream *)*stream*
forView:*view* Reads representation from RTF data
- **writeRichText:**(NXStream *)*stream*
forView:*view* Writes RTF representation to stream
- (BOOL)**trackMouse:**(NXEvent *)*theEvent*
inRect:(const NXRect *)*rect*
ofView:*view* Controls tracking of the mouse

TextField

Inherits From: Control : View : Responder : Object

Initializing the TextField Class

- + **setCellClass:***classId* Sets the Cell class used by TextField

Iniializing a new TextField

- **initWithFrame:**(const NXRect *)*frameRect* Initializes a new TextField object with no text

Enabling the TextField

- **setEnabled:***(BOOL)flag*

Sets whether the TextField reacts to events

Setting User Access to Text

- **setSelectable:***(BOOL)flag*

Sets whether the TextField's text is selectable

- **(BOOL)isSelected**

Returns whether the TextField's text is selectable

- **setEditable:***(BOOL)flag*

Sets whether the TextField's text is editable

- **(BOOL)isEditable**

Returns whether the TextField's text is editable

Editing Text

- **selectText:***sender*

Selects all of the text if it's selectable or editable

Setting Tab Key Behavior

- **setNextText:***anObject*

Sets the object selected when the user presses Tab

- **nextText**

Sets the object selected when the user presses Tab

- **setPreviousText:***anObject*

Sets the object selected when the user types Shift-Tab

- **previousText**

Sets the object selected when the user types Shift-Tab

Assigning a Text Delegate

- **setTextDelegate:***anObject*

Sets the delegate for messages from the field editor

- **textDelegate**

Returns the delegate for messages from field editor

Text Object Delegate Methods

- **(BOOL)textWillChange:***textObject*

Responds to a message from the field editor

- **textDidGetKeys:***textObject*

Responds to a message from the field editor

- isEmpty:***(BOOL)flag*

- **textDidChange:***textObject*
- (BOOL)**textWillEnd:***textObject*
- **textDidEnd:***textObject*
endChar:(unsigned short)*whyEnd*

Responds to a message from the field editor

Responds to a message from the field editor

Responds to a message from the field editor

Modifying Graphic Attributes

- **setTextColor:**(NXColor)*aColor*
- (NXColor)**textColor**
- **setTextGray:**(float)*value*
- (float)**textGray**
- **setBackgroundColor:**(NXColor)*aColor*
- (NXColor)**backgroundColor**
- **setBackgroundGray:**(float)*value*
- (float)**backgroundGray**
- **setBackgroundTransparent:**(BOOL)*flag*
- (BOOL)**isBackgroundTransparent**
- **setBezeled:**(BOOL)*flag*
- (BOOL)**isBezeled**
- **setBordered:**(BOOL)*flag*
- (BOOL)**isBordered**

Sets the color of the TextField's text to *aColor*

Returns the color of the TextField's text

Sets the gray of the TextField's text to *value*

Returns the gray of the TextField's text

Sets the color of the background to *aColor*

Returns the color of the background

Sets the gray of the background to *value*

Returns the gray of the background

Sets whether the TextField background is transparent

Returns whether the TextField background is transparent

Sets whether the TextField has a bezeled border

Returns whether the TextField has a bezeled border

Sets whether the TextField has a plain border

Returns whether the TextField has a plain border

Target and Action

- **setErrorAction:**(SEL)*aSelector*
- (SEL)**errorAction**

Sets the action method sent for an invalid value entered

Returns the action method sent for an invalid value

Resizing a TextField

- **sizeTo:**(float)*width* :(float)*height*

Resizes the TextField to *width* and *height*

Handling Events

- (BOOL)**acceptsFirstResponder**
- **mouseDown:**(NXEvent *)*theEvent*

Returns YES if text is editable or selectable

Responds to a mouse-down event

Archiving

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

Reads the TextField from *stream*

Writes the TextField to *stream*

TextFieldCell

Inherits From: ActionCell : Cell : Object

Initializing a new TextFieldCell

- **init**
- **initWithCell:**(const char *)*aString*

Initializes a new TextFieldCell with text ^{aField}_o

Initializes a new TextFieldCell with text *aString*

Copying a TextFieldCell

- **copyFromZone:**(NXZone *)*zone*

Returns a copy of the TextFieldCell allocated from *zone*

Modifying Graphic Attributes

- **setTextColor:**(NXColor)*aColor*
- (NXColor)**textColor**
- **setTextGray:**(float)*value*
- (float)**textGray**

Sets the color of the text to *aColor*

Returns the color of the text

Sets the gray of the text to *value*

Returns the gray of the text

- **setBackgroundColor:**(NXColor)*aColor*
- (NXColor)**backgroundColor**
- **setBackgroundGray:**(float)*value*
- (float)**backgroundGray**
- **setBackgroundTransparent:**(BOOL)*flag*
- (BOOL)**isBackgroundTransparent**
- **setTextAttributes:***textObject*
- **setBezeled:**(BOOL)*flag*
- (BOOL)**isOpaque**

Sets the color of the background to *aColor*

Returns the color of the background

Sets the gray of the background to *value*

Returns the gray of the background

Sets whether the background is transparent

Returns whether the background is transparent

Sets the gray values of the background and text to those of *textObject*

Sets whether TextFieldCell has a bezeled border

Returns whether the cell is opaque

Displaying

- **drawSelf:**(const NXRect *)*cellFrame*
inView:*controlView*
- **drawInside:**(const NXRect *)*cellFrame*
inView:*controlView*

Draws the TextFieldCell

Draws the inside of the TextFieldCell

Tracking the Mouse

- (BOOL)**trackMouse:**(NXEvent *)*theEvent*
inRect:(const NXRect *)*aRect*
ofView:*controlView*

Starts editing if possible

Archiving

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

Reads the TextFieldCell from *stream*

Writes the TextFieldCell to *stream*

View

Inherits From: Responder : Object

Initializing and Freeing View Objects

- | | |
|---|---------------------------------------|
| - initWithFrame: (const NXRect *) <i>frameRect</i> | Initializes a new View object |
| - init | Initializes a new View object |
| - free | Deallocates the View and its subviews |

Managing the View Hierarchy

- | | |
|---|--|
| - addSubview: <i>aView</i> | Makes <i>aView</i> a subview of the receiving View |
| - addSubview: <i>aView</i>
:(int) <i>place</i>
relativeTo: <i>otherView</i> | Makes <i>aView</i> a subview of the receiving View |
| - findAncestorSharedWith: <i>aView</i> | Returns the ancestor shared by <i>aView</i> and the receiver |
| - (BOOL) isDescendantOf: <i>aView</i> | Returns whether <i>aView</i> is an ancestor of the receiver |
| - opaqueAncestor | Returns the receiver's nearest opaque ancestor |
| - removeFromSuperview | Removes the receiver from the view hierarchy |
| - replaceSubview: <i>oldView</i> with: <i>newView</i> | Replaces <i>oldView</i> with <i>newView</i> |
| - subviews | Returns a List of the View's subviews |
| - superview | Returns the receiving View's superview |
| - window | Returns the Window in which the View is displayed |
| - windowChanged: <i>newWindow</i> | Notifies the View that the Window it's in is changing |

Modifying the Frame Rectangle

- | | |
|---|--|
| - (float) frameAngle | Returns the angle of frame rectangle rotation |
| - getFrame: (NXRect *) <i>theRect</i> | Gets the View's frame rectangle |
| - moveBy: (NXCoord) <i>deltaX</i> :(NXCoord) <i>deltaY</i> | Moves the View by <i>deltaX</i> and <i>deltaHeight</i> |
| - moveTo: (NXCoord) <i>x</i> :(NXCoord) <i>y</i> | Moves the View to (x, y) |

- rotateBy: (NXCoord) <i>deltaAngle</i>	Rotates the View's frame rectangle by <i>deltaAngle</i>
- rotateTo: (NXCoord) <i>angle</i>	Rotates the View's frame rectangle to <i>angle</i>
- setFrame: (const NXRect *) <i>frameRect</i>	Assigns the View a new frame rectangle
- sizeBy: (NXCoord) <i>deltaWidth</i> :(NXCoord) <i>deltaHeight</i>	Resizes the View by <i>deltaWidth</i> and <i>deltaHeight</i>
- sizeTo: (NXCoord) <i>width</i> :(NXCoord) <i>height</i>	Resizes the View to <i>width</i> and <i>height</i>

Modifying the Coordinate System

- (float) boundsAngle	Returns the rotation of the View's coordinate system
- drawInSuperview	Makes the View use its superview's coordinate system
- getBounds: (NXRect *) <i>theRect</i>	Gets the View's bounds rectangle
- (BOOL) isFlipped	Returns whether the View is flipped
- (BOOL) isRotatedFromBase	Returns whether the View is rotated
- (BOOL) isRotatedOrScaledFromBase	Returns whether the View is rotated or scaled
- rotate: (NXCoord) <i>angle</i>	Rotates the View's coordinate system <i>by angle</i>
- setDrawRotation: (NXCoord) <i>angle</i>	Rotates the View's coordinate system to <i>angle</i>
- scale: (NXCoord) <i>x</i> :(NXCoord) <i>y</i>	Scales the View's coordinate system
- setDrawSize: (NXCoord) <i>width</i> :(NXCoord) <i>height</i>	Resizes the View's coordinate system to <i>width</i> and <i>height</i>
- translate: (NXCoord) <i>x</i> :(NXCoord) <i>y</i>	Shifts the View's coordinate system to (x, y)
- setDrawOrigin: (NXCoord) <i>x</i> :(NXCoord) <i>y</i>	Sets the View's origin to (x, y)
- setFlipped: (BOOL) <i>flag</i>	Sets whether polarity of y-axis is reversed

Converting Coordinates

- centerScanRect: (NXRect *) <i>aRect</i>	Converts the rectangle to lie on center of pixels
- convertPoint: (NXPoint *) <i>aPoint</i> fromView: <i>aView</i>	Converts <i>the</i> point to the receiver's coordinates
- convertPoint: (NXPoint *) <i>aPoint</i> toView: <i>aView</i>	Converts <i>the</i> point to <i>aView</i> 's coordinates
- convertPointFromSuperview: (NXPoint *) <i>aPoint</i>	

- **convertPointToSuperview:**(NXPoint *)*aPoint* Converts the point to the receiver's coordinates
- **convertRect:**(NXRect *)*aRect* **fromView:***aView* Converts the point to the superview's coordinates
- **convertRect:**(NXRect *)*aRect* **toView:***aView* Converts the rectangle to the receiver's coordinates
- **convertRectFromSuperview:**(NXRect *)*aRect* Converts the rectangle to *aView*'s coordinates
- **convertRectToSuperview:**(NXRect *)*aRect* Converts the rectangle to the receiver's coordinates
- **convertSize:**(NXSize *)*aSize* **fromView:***aView* Converts the rectangle to the superview's coordinates
- **convertSize:**(NXSize *)*aSize* **toView:***aView* Converts the size to the receiver's coordinates
- **convertSize:**(NXSize *)*aSize* **toView:***aView* Converts the size to *aView*'s coordinates

Notifying Ancestor Views

- **descendantFlipped:***sender* Notifies that *sender*'s y-axis has flipped
- **descendantFrameChanged:***sender* Notifies that *sender*'s frame rectangle changed
- **notifyAncestorWhenFrameChanged:**(BOOL)*flag* Sets whether to notify ancestors of frame change
- **notifyWhenFlipped:**(BOOL)*flag* Sets whether to notify ancestors of flipped y-axis
- **suspendNotifyAncestorWhenFrameChanged:**(BOOL)*flag* Suspends ancestor notification

Resizing Subviews

- **resizeSubviews:**(const NXSize *)*oldSize* Initiates **superviewSizeChanged:** messages
- **setAutoresizeSubviews:**(BOOL)*flag* Sets whether to notify subviews of resizing
- **setAutosizing:**(unsigned int)*mask* Determines automatic resizing behavior
- (unsigned int)**autosizing** Returns the View's autosizing mask
- **superviewSizeChanged:**(const NXSize *)*oldSize* Notifies subviews that superview changed size

Graphics State Objects

- **allocateGState** Allocates a graphics state object (when next focused upon)

- freeGState	Frees the View's graphics state object
- (int) gState	Returns the View's graphics state object
- initGState	Initializes the View's graphics state object
- renewGState	Reinitializes the View's graphics state object
- notifyToInitGState:(BOOL)flag	Determines whether initGState message is sent

Focusing

- clipToFrame:(const NXRect *)frameRect	Clips to the frame rectangle during focusing
- setClipping:(BOOL)flag	Sets whether the View is clipped to its frame rectangle
- (BOOL) doesClip	Returns whether the View clips to its frame rectangle
- (BOOL) isFocusView	Returns whether the View is currently in focus
- (BOOL) lockFocus	Brings the View into focus
- unlockFocus	Unfocuses the View

Displaying

- (BOOL) canDraw	Returns whether the View can draw
- display	Displays the View and its subviews
- display:(const NXRect *)rects :(int)rectCount	Displays the View and its subviews
- display:(const NXRect *)rects :(int)rectCount :(BOOL)clipFlag	Displays the View and its subviews
- displayFromOpaqueAncestor:(const NXRect *)rects :(int)rectCount :(BOOL)clipFlag	Displays underlying ancestors and the View
- displayIfNeeded	Conditionally displays the View and its subviews
- drawSelf:(const NXRect *)rects :(int)rectCount	Implemented by subclasses to supply drawing instructions
- (BOOL) getVisibleRect:(NXRect *)theRect	Gets the View's visible portion
- (BOOL) isAutodisplay	Returns whether the View automatically updates
- setAutodisplay:(BOOL)flag	Determines whether update <i>redisplays the View</i>

- (BOOL) isOpaque	Returns whether the View is registered as opaque
- setOpaque: (BOOL) <i>flag</i>	Registers the View as opaque
- (BOOL) needsDisplay	Returns whether the View needs to be redisplayed
- setNeedsDisplay: (BOOL) <i>flag</i>	Marks the View as changed, needing redisplay
- (BOOL) shouldDrawColor	Returns whether the View should be drawn in color
- update	Conditionally redisplay the View

Scrolling

- adjustScroll: (NXRect *) <i>newVisible</i>	Lets the View adjust the visible rectangle
- autoscroll: (NXEvent *) <i>theEvent</i>	Scrolls in response to a mouse-dragged event
- (BOOL) calcUpdateRects: (NXRect *) <i>rects</i> :(int *) <i>rectCount</i> :(NXRect *) <i>enclRect</i> :(NXRect *) <i>goodRect</i>	Calculates the area to be redisplayed
- invalidate: (const NXRect *) <i>rects</i> :(int) <i>rectCount</i>	Marks parts of the View as needing to be redrawn
- scrollPoint: (const NXPoint *) <i>aPoint</i>	Aligns <i>aPoint</i> with the content view's origin
- scrollRect: (const NXRect *) <i>aRect</i> by: (const NXPoint *) <i>delta</i>	Shifts the rectangle by <i>delta</i>
- scrollRectToVisible: (const NXRect *) <i>aRect</i>	Scrolls the View so the rectangle is visible
- (float) backgroundGray	Returns the View's background gray (used by a ScrollView's document view only)

Managing the Cursor

- addCursorRect: (const NXRect *) <i>aRect</i> cursor: <i>anObj</i>	Adds a cursor rectangle to the View
- discardCursorRects	Removes all cursor rectangles in the View
- removeCursorRect: (const NXRect *) <i>aRect</i> cursor: <i>anObj</i>	Removes a cursor rectangle from the View
- resetCursorRects	Resets the View's cursor rectangles

Assigning a Tag

- **findViewWithTag:**(int)*aTag* Returns the subview with *aTag* as its tag
- (int)**tag** Returns the View's tag

Aiding Event Handling

- (BOOL)**acceptsFirstMouse** Returns NO to refuse first mouse-down event
- **hitTest:**(NXPoint *)*aPoint* Returns the lowest subview containing the point
- (BOOL)**mouse:**(NXPoint *)*aPoint*
 inRect:(NXRect *)*aRect* Returns whether the point lies inside the rectangle
- (BOOL)**performKeyEquivalent:**(NXEvent *)*theEvent*
Returns whether a subview handled *theEvent*
- (BOOL)**shouldDelayWindowOrderingForEvent:**(NXEvent *)*anEvent*
Returns whether the View's Window is brought forward normally (mouse-down) or delayed (mouse-up)

Dragging

- **registerForDraggedTypes:**(const char *const *)*pbTypes* **count:**(int)*count*
Registers the Pasteboard types that the Window will accept in an image-dragging session
- **unregisterDraggedTypes** Unregisters the Window as a recipient of dragged images
- **dragImage:***anImage*
 at:(NXPoint *)*location*
 offset:(NXPoint *)*initialOffset*
 event:(NXEvent *)*event*
 pasteboard:(Pasteboard *)*pboard*
 source:*sourceObject*
 slideBack:(BOOL)*slideFlag*
Instigates an image-dragging session
- **dragFile:**(const char *)*filename*
 fromRect:(NXRect *)*rect*
Instigates a file-dragging session

slideBack:(BOOL) *aFlag*
event:(NXEvent *)*event*

Printing

- **printPSCode:***sender* Prints the View and its subviews
- **faxPSCode:***sender* Faxes the View and its subviews
- **faxPSCode:***sender*
 toList:(const char *const *)*names*
 numberList:(const char *const *)*numbers*
 sendAt:(time_t)*time*
 wantsCover:(BOOL)*coverFlag*
 wantsNotify:(BOOL)*notifyFlag*
 wantsHires:(BOOL)*hiresFlag*
 faxName:(const char *)*string* Faxes the View and its subviews
- **copyPSCodeInside:**(const NXRect *)*rect*
 to:(NXStream *)*stream* Generates PostScript code for the rectangle
- **writePSCodeInside:**(const NXRect *)*rect*
 to:*pasteboard* Places PostScript code for the rectangle on the pasteboard
- **openSpoolFile:**(char *)*filename* Opens *filename* for print spooling
- **spoolFile:**(const char *)*filename* Spools *filename* to the printer
- (BOOL)**canPrintRIB** Indicates whether the View can print RIB files

Setting Up Pages

- (BOOL)**knowsPagesFirst:**(int *)*firstPageNum*
 last:(int *)*lastPageNum* Returns whether the View paginates itself
- (BOOL)**getRect:**(NXRect *)*theRect*
 forPage:(int)*page* Provides how much of the View will print on *page*
- **placePrintRect:**(const NXRect *)*aRect*
 offset:(NXPoint *)*location* Locates the printing rectangle on the page
- (float)**heightAdjustLimit** Returns how much of a page can go on the next page
- (float)**widthAdjustLimit** Returns how much of a page can go on the next page

Writing Conforming PostScript

- beginPSOutput	Initializes the printing environment
- beginPrologueBBox: (const NXRect *) <i>boundingBox</i> creationDate: (const char *) <i>dateCreated</i> createdBy: (const char *) <i>anApplication</i> fonts: (const char *) <i>fontNames</i> forWhom: (const char *) <i>user</i> pages: (int) <i>numPages</i> title: (const char *) <i>aTitle</i>	Writes the beginning of the prologue for a print job
- endHeaderComments	Writes the end of the header
- endPrologue	Writes the end of the prologue
- beginSetup	Writes the beginning of the document setup section
- endSetup	Writes the end of the document setup section
- adjustPageWidthNew: (float *) <i>newRight</i> left: (float) <i>oldLeft</i> right: (float) <i>oldRight</i> limit: (float) <i>rightLimit</i>	Assists automatic pagination of the View
- adjustPageHeightNew: (float *) <i>newBottom</i> top: (float) <i>oldTop</i> bottom: (float) <i>oldBottom</i> limit: (float) <i>bottomLimit</i>	Assists automatic pagination of the View
- beginPage: (int) <i>ordinalNum</i> label: (const char *) <i>aString</i> bBox: (const NXRect *) <i>pageRect</i> fonts: (const char *) <i>fontNames</i>	Writes a page separator
- beginPageSetupRect: (const NXRect *) <i>aRect</i> placement: (const NXPoint *) <i>location</i>	Writes the beginning of a page setup section
- drawSheetBorder: (float) <i>width</i> :(float) <i>height</i>	Allows you to draw a sheet border
- drawPageBorder: (float) <i>width</i> :(float) <i>height</i>	Allows you to draw a page border
- addToPageSetup	Allows you to add scaling to PostScript code
- endPageSetup	Writes the end of a page setup section

- | | |
|-----------------------|---|
| - endPage | Writes the end of a page |
| - beginTrailer | Writes the beginning of the trailer for the print job |
| - endTrailer | Writes the end of the trailer |
| - endPSOutput | Finishes the printing job |

Archiving

- | | |
|---|--------------------------------------|
| - awake | Initializes the View after reading |
| - read: (NXTypedStream *) <i>stream</i> | Reads the View from the typed stream |
| - write: (NXTypedStream *) <i>stream</i> | Writes the View to the typed stream |

Window

Inherits From: Responder : Object

Initializing a New Window Object

- | | |
|--|--|
| - init | Initializes the new Window with default parameters |
| - initWithContent: (const NXRect *) <i>contentRect</i>
style: (int) <i>aStyle</i>
backing: (int) <i>bufferingType</i>
buttonMask: (int) <i>mask</i>
defer: (BOOL) <i>flag</i> | Initializes the new Window object as specified |
| - initWithContent: (const NXRect *) <i>contentRect</i>
style: (int) <i>aStyle</i>
backing: (int) <i>bufferingType</i>
buttonMask: (int) <i>mask</i>
defer: (BOOL) <i>flag</i>
screen: (const NXScreen *) <i>aScreen</i> | Initializes the new Window object for <i>screen</i> as specified |

Freeing a Window Object

- **free** Frees the Window object and its Views

Computing Frame and Content Rectangles

+ **getFrameRect:**(NXRect *)*frame* Gets frame rectangle for given content rectangle
 forContentRect:(const NXRect *)*content*
 style:(int)*aStyle*

+ **getContentRect:**(NXRect *)*content* Gets content rectangle for given frame rectangle
 forFrameRect:(const NXRect *)*frame*
 style:(int)*aStyle*

+ (NXCoord)**minFrameWidth:**(const char *)*aTitle* Returns minimum frame width needed for *aTitle*
 forStyle:(int)*aStyle*
 buttonMask:(int)*aMask*

Accessing the Frame Rectangle

- **getFrame:**(NXRect *)*theRect* Gets the Window's frame rectangle

- **getFrame:**(NXRect *)*theRect* Gets the Window's frame rectangle and screen
 andScreen:(const NXScreen **)*theScreen*

- (BOOL)**setFrameUsingName:**(const char *)*name* Sets the frame rectangle from the named default

- (void)**saveFrameUsingName:**(const char *)*name* Saves the frame rectangle as a system default

+ (void)**removeFrameUsingName:**(const char *)*name* Removes the named frame data from the system defaults

- (BOOL)**setFrameAutosaveName:**(const char *)*name* Sets the name that's used to autosave the frame rectangle as a system default

- (const char *)**frameAutosaveName** Returns the name that's used to autosave the frame rectangle as a system default

- (void)**setFrameFromString:**(const char *)*string* Sets the frame rectangle from *string*

- **saveFrameToString:**(const char *)*string*

Saves the frame rectangle data to *string*

Accessing the Content View

- **setContentView:***aView*

Makes *aView* the Window's content view

- **contentView**

Returns the Window's content view

Querying Window Attributes

- (int)**windowNum**

Returns the window number

- (int)**buttonMask**

Returns the mask identifying Window controls

- (int)**style**

Returns the Window's border and title bar style

- (BOOL)**worksWhenModal**

Returns NO

Window Graphics

- **setTitle:**(const char *)*aString*

Makes *aString* the Window's title

- **setTitleAsFilename:**(const char *)*aString*

Formats *aString* and makes it the Window's title

- (const char *)**title**

Returns the Window's title string

- **setBackgroundColor:**(NXColor)*color*

Sets *the* Window's background color to *color*

- (NXColor)**backgroundColor**

Returns the Window's background color

- **setBackgroundGray:**(float)*value*

Sets the gray *value* for Window's background gray

- (float)**backgroundGray**

Returns the Window's background gray

Window Device Attributes

- **setBackingType:**(int)*backing*

Sets the type of window device backing

- (int)**backingType**

Returns the window device backing type

- **setOneShot:**(BOOL)*flag*

Sets whether memory for the window should be freed

- (BOOL)**isOneShot**

Returns whether memory for the window is freed

- **setFreeWhenClosed:**(BOOL)*flag*

Sets whether closing the Window also frees it

Graphics State Objects

- (int)**gState**

Returns the graphics state object for the Window

The Miniwindow

- **counterpart**

Returns the receiver's miniwindow/Window companion

- **setMiniwindowIcon:**(const char *)*name*

Sets the icon that's displayed in the miniwindow

- (const char *)**miniwindowIcon**

Returns the icon that's displayed in the miniwindow

- **setMiniwindowImage:***image*

Sets the image that's displayed in the miniwindow

- (NXImage *)**miniwindowImage**

Returns the image that's displayed in the miniwindow

- **setMiniwindowTitle:**(const char *)*title*

Sets the title that's displayed in the miniwindow

- (const char *)**miniwindowTitle**

Sets the title that's displayed in the miniwindow

The Field Editor

- **getFieldEditor:**(BOOL)*flag* **for:***anObject*

Returns the Window's field editor

- **endEditingFor:***anObject*

Ends the field editor's editing assignment

Window Status

- **makeKeyWindow**

Makes the receiver the key window

- **makeKeyAndOrderFront:***sender*

Makes Window the key window and brings it forward

- **becomeKeyWindow**

Records Window's new status as the key window

- (BOOL)**isKeyWindow**

Returns whether Window is the key window

- **resignKeyWindow**

Records that Window no longer is the key window

- (BOOL)**canBecomeKeyWindow**

Returns whether Window can be the key window

- **becomeMainWindow**

Records Window's new status as the main window

- (BOOL)**isMainWindow**

Returns whether Window is the main window

- **resignMainWindow**

Records that Window no longer is the main window

- (BOOL)**canBecomeMainWindow** Returns whether Window can be the main window

Moving and Resizing the Window

- **moveTo:**(NXCoord)x :(NXCoord)y Moves the Window to (x, y)
- **moveTo:**(NXCoord)x
:(NXCoord)y
screen:(const NXScreen *)aScreen Moves the Window to (x, y) relative to aScreen
- **moveTopLeftTo:**(NXCoord)x :(NXCoord)y Moves top left corner of the Window to (x, y)
- **moveTopLeftTo:**(NXCoord)x
:(NXCoord)y
screen:(const NXScreen *)aScreen Moves top left corner of the Window relative to aScreen
- **dragFrom:**(float)x :(float)y **eventNum:**(int)num Lets user drag the Window from (x, y)
- **constrainFrameRect:**(NXRect *)frameRect
toScreen:(const NXScreen *)screen Constrains the Window to screen
- **placeWindow:**(const NXRect *)frameRect Resizes the Window to new frame rectangle
- **placeWindow:**(const NXRect *)frameRect
screen:(const NXScreen *)aScreen Resizes the Window relative to aScreen
- **placeWindowAndDisplay:**(const NXRect *)frameRect
Resizes the Window while redisplaying its Views
- **sizeWindow:**(NXCoord)width :(NXCoord)height
Resizes the Window's content area
- **center** Centers the Window on the screen
- **setMinSize:**(const NXSize *)aSize Sets the Window's minimum size
- **getMinSize:**(NXSize *)aSize Returns the Window's minimum size
- **setMaxSize:**(const NXSize *)aSize Sets the Window's maximum size
- **getMaxSize:**(NXSize *)aSize Returns the Window's maximum size
- (int)**resizeFlags** Returns the event flags during resizing

Reordering the Window

- **makeKeyAndOrderFront:**sender Makes the Window the key window and brings it forward

- orderFront: <i>sender</i>	Puts the Window at the front of its tier
- orderBack: <i>sender</i>	Puts the Window at the back of its tier
- orderOut: <i>sender</i>	Removes the Window from the screen list
- orderWindow:(int) <i>place</i> relativeTo:(int) <i>otherWin</i>	Repositions the Window in screen list
- orderFrontRegardless	Puts the Window at the front even if the application is inactive
- (BOOL) isVisible	Returns whether the Window is in the screen list
- setHideOnDeactivate:(BOOL) <i>flag</i>	Sets whether deactivization hides the Window
- (BOOL) doesHideOnDeactivate	Returns whether deactivization hides the Window

Converting Coordinates

- convertBaseToScreen:(NXPoint *) <i>aPoint</i>	Converts point from base to screen coordinates
- convertScreenToBase:(NXPoint *) <i>aPoint</i>	Converts point from screen to base coordinates

Managing the Display

- display	Displays all the Window's Views
- displayIfNeeded	Displays all the Window's Views that need it
- disableDisplay	Inhibits Views from drawing in the Window
- (BOOL) isDisplayEnabled	Returns whether Views can draw in the Window
- reenableDisplay	Reenables drawing by Views in the Window
- flushWindow	Flushes the Window's buffer to the screen
- flushWindowIfNeeded	Conditionally flushes the Window's buffer to the screen
- disableFlushWindow	Disables flushing for a buffered Window
- reenableFlushWindow	Reenables flushing for a buffered Window
- (BOOL) isFlushWindowDisabled	Returns whether flushing is disabled
- displayBorder	Displays the border and title bar
- useOptimizedDrawing:(BOOL) <i>flag</i>	Sets whether Window's Views should optimize drawing
- update	Implemented by subclasses (see Menu)

Screens and Window Depths

- (const NXScreen *)**screen**
- (const NXScreen *)**bestScreen**
- + (NXWindowDepth)**defaultDepthLimit**
- **setDepthLimit:**(NXWindowDepth)*limit*
- (NXWindowDepth)**depthLimit**
- **setDynamicDepthLimit:**(BOOL)*flag*
- (BOOL)**hasDynamicDepthLimit**
- (BOOL)**canStoreColor**

Returns the screen that (most of) the Window is on

Returns the deepest screen that the Window is on

Returns the maximum depth for the current context

Sets the Window's depth limit to *limit*

Returns the Window's depth limit

Sets whether the depth limit will depend on the screen

Returns whether the depth limit depends on the screen

Returns whether Window is deep enough to store colors

Cursor Management

- **addCursorRect:**(const NXRect *)*aRect*
 cursor:anObject
 forView:aView
- **removeCursorRect:**(const NXRect *)*aRect*
 cursor:anObject
 forView:aView
- **invalidateCursorRectsForView:***aView*
- **disableCursorRects**
- **enableCursorRects**
- **discardCursorRects**
- **resetCursorRects**

Adds a new cursor rectangle to the Window

Removes a cursor rectangle from the Window

Marks cursor rectangles invalid for *aView*

Disables all cursor rectangles in the Window

Enables cursor rectangles in the Window

Removes all cursor rectangles in the Window

Resets cursor rectangles for the Window

Handling User Actions and Events

- **close**
- **performClose:***sender*
- **miniaturize:***sender*

Closes the Window

Simulates user clicking the close button

Hides the Window and displays its miniwindow

- **performMiniaturize:***sender*
- **deminiaturize:***sender*
- **setDocEdited:**(BOOL)*flag*
- (BOOL)**isDocEdited**
- **windowExposed:**(NXEvent *)*theEvent*
- **windowMoved:**(NXEvent *)*theEvent*
- **screenChanged:**(NXEvent *)*theEvent*

Simulates user clicking the miniaturize button
 Hides the miniwindow and redisplays the Window
 Sets whether the Window's document has been edited
 Returns whether Window's document has been edited
 Redisplays exposed part of the Window
 Updates the frame rectangle
 Adjusts depth limit of Windows with a dynamic limit

Setting the Event Mask

- (int)**setEventMask:**(int)*newMask*
- (int)**addToEventMask:**(int)*newEvents*
- (int)**removeFromEventMask:**(int)*oldEvents*
- (int)**eventMask**

Sets the Window's event mask to *newMask*
 Adds *newEvents* to the event mask
 Removes *oldEvents* from the event mask
 Returns the Window's event mask

Aiding Event Handling

- **getMouseLocation:**(NXPoint *)*thePoint*
- **setTrackingRect:**(const NXRect *)*aRect*
 inside:(BOOL)*insideFlag*
 owner:*anObject*
 tag:(int)*trackNum*
 left:(BOOL)*leftDown*
 right:(BOOL)*rightDown*
- **discardTrackingRect:**(int)*trackNum*
- **makeFirstResponder:***aResponder*
- **firstResponder**
- **sendEvent:**(NXEvent *)*theEvent*
- **rightMouseDown:**(NXEvent *)*theEvent*
- (BOOL)**commandKey:**(NXEvent *)*theEvent*
- (BOOL)**tryToPerform:**(SEL)*anAction*

Provides current location of the cursor
 Sets a tracking rectangle within the Window

 Clears tracking rectangle within the Window
 Makes *aResponder* the first responder
 Returns the first responder
 Dispatches mouse and keyboard events
 Handles right mouse-down events
 Handles Command key-down events
 Aids in dispatching action messages

with:*anObject*

- **setAvoidsActivation:**(BOOL)*flag*

Establishes whether the application will become active when the user clicks in the Window

- (BOOL)**avoidsActivation**

Returns the value set by **setAvoidsActivation:**

Dragging

- **registerForDraggedTypes:**(const char *const *)*pbTypes* **count:**(int)*count*

Registers the Pasteboard types that the Window will accept in an image-dragging session

- **unregisterDraggedTypes**

Unregisters the Window as a recipient of dragged images

- **dragImage:***anImage*

Instigates an image-dragging session

at:(NXPoint *)*location*

offset:(NXPoint *)*initialOffset*

event:(NXEvent *)*event*

pasteboard:(Pasteboard *)*pboard*

source:*sourceObject*

slideBack:(BOOL)*slideFlag*

Services and Windows Menu Support

- **validRequestorForSendType:**(NXAtom)*typeSent* Invoked by clicking a Services menu item
andReturnType:(NXAtom)*typeReturned*

- **setExcludedFromWindowsMenu:**(BOOL)*flag* Sets whether Window is left out of the Windows menu

- (BOOL)**isExcludedFromWindowsMenu** Returns whether Window is left out of Windows menu

Printing

- **printPSCode:***sender*

Prints all the Window's Views

- **smartPrintPSCode:***sender*

Prints all the Window's Views

- **faxPSCode:***sender*

Faxes all the Window's Views

- **smartFaxPSCode:***sender*

Faxes all the Window's Views

- **openSpoolFile:**(char *)*filename*

Opens *filename* for print spooling

- spoolFile: (const char *) <i>filename</i>	Spools <i>filename</i> to the printer
- copyPSCodeInside: (const NXRect *) <i>rect</i> to: (NXStream *) <i>stream</i>	Writes PostScript code for the rectangle to <i>stream</i>
- (BOOL) knowsPagesFirst: (int *) <i>firstPageNum</i> last: (int *) <i>lastPageNum</i>	Returns whether the Window paginates itself
- (BOOL) getRect: (NXRect *) <i>theRect</i> forPage: (int) <i>page</i>	Provides how much of the Window fits on <i>page</i>
- placePrintRect: (const NXRect *) <i>aRect</i> offset: (NXPoint *) <i>location</i>	Locates the printing rectangle on the page
- (float) heightAdjustLimit	Returns how much of a page can go on next page
- (float) widthAdjustLimit	Returns how much of a page can go on next page
- beginPSOutput	Initializes the printing environment
- beginPrologueBBox: (const NXRect *) <i>boundingBox</i> creationDate: (const char *) <i>dateCreated</i> createdBy: (const char *) <i>anApplication</i> fonts: (const char *) <i>fontNames</i> forWhom: (const char *) <i>user</i> pages: (int) <i>numPages</i> title: (const char *) <i>aTitle</i>	Writes the beginning of the prologue for a print job
- endHeaderComments	Writes the Application Kit print package
- endPrologue	Writes the end of the prologue
- beginSetup	Writes the beginning of the document setup section
- endSetup	Writes the end of the document setup section
- beginPage: (int) <i>ordinalNum</i> label: (const char *) <i>aString</i> bBox: (const NXRect *) <i>pageRect</i> fonts: (const char *) <i>fontNames</i>	Writes a page separator
- beginPageSetupRect: (const NXRect *) <i>aRect</i> placement: (const NXPoint *) <i>location</i>	Writes the beginning of a page setup section
- endPageSetup	Writes the end of a page setup section
- endPage	Writes the end of a page description
- beginTrailer	Writes the beginning of trailer for the print job

- **endTrailer**
- **endPSOutput**

Writes the end of the trailer
 Finishes the printing job

Archiving

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

Reads the Window from the typed stream
 Writes the Window to the typed stream
 Redisplays and reinitializes the Window

Assigning a Delegate

- **setDelegate:***anObject*
- **delegate**

Makes *anObject* the Window's delegate
 Returns the Window's delegate

Implemented by the Delegate

- **windowWillClose:***sender*
- **windowWillReturnFieldEditor:***sender*
toObject:*client*
- **windowWillResize:***sender*
toSize:(NXSize *)*frameSize*
- **windowDidResize:***sender*
- **windowDidExpose:***sender*
- **windowWillMove:***sender*
- **windowDidMove:***sender*
- **windowDidChangeScreen:***sender*
- **windowDidBecomeKey:***sender*
- **windowDidResignKey:***sender*
- **windowDidBecomeMain:***sender*
- **windowDidResignMain:***sender*
- **windowWillMiniaturize:***sender*

Notifies delegate that the Window is about to close
 Lets delegate provide another Text object

Lets delegate constrain resizing

Notifies delegate that the Window was resized
 Notifies delegate that the Window was exposed
 Notifies delegate that the Window will move
 Notifies delegate that the Window did move
 Notifies delegate that the Window changed screens
 Notifies delegate that the Window is the key window
 Notifies delegate that the Window isn't the key window
 Notifies delegate that the Window is the main window
 Notifies delegate that the Window isn't the main window
 Notifies delegate that the Window will miniaturized

toMiniwindow:*miniwindow*

- **windowDidMiniaturize:***sender*

- **windowDidDeminiaturize:***sender*

- **windowDidUpdate:***sender*

Notifies delegate that the Window was miniaturized

Notifies delegate that the Window was restored to screen

Notifies delegate that the Window was updated