2

Application Kit

Classes

ActionCell

Inherits From: Cell : Object

Configuring an ActionCell

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

- **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

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

Application

Inherits From: Responder : Object

Initializing the Class Object

+ **initialize** — Registers defaults for the application

Creating and Freeing Instances

+ **new** — Returns a new Application object
- **free** — Deallocates the Application object

Setting Up the Application

+ **workspace** — Returns the **id** of WorkspaceManager
- **loadNibFile:**(const char *)*filename* **owner:***anObject* — Loads objects built using Interface Builder
- **loadNibFile:**(const char *)*filename* **owner:***anObject* **withNames:**(BOOL)*flag* — Loads objects built using Interface Builder
- **loadNibFile:**(const char *)*filename* **owner:***anObject* **withNames:**(BOOL)*flag* **fromZone:**(NXZone *)*zone* — Loads objects built using Interface Builder
- **loadNibSection:**(const char *)*name* **owner:***anObject* — 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 *theWindow*
- **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 *theWindow*
- (int)**runModalSession:**(NXModalSession *)*session*
  Runs a modal session
- **endModalSession:**(NXModalSession *)*session* Finishes a modal session
- **delayedFree:***theObject* Frees *theObject* 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 *mask*
- (NXEvent *)**getNextEvent:**(int)*mask* Returns pointer to the next event matching *mask*
  waitFor:(double)*timeout*
  threshold:(int)*level*
- (NXEvent *)**peekAndGetNextEvent:**(int)*mask* Returns pointer to the next event matching *mask*
- (NXEvent *)**peekNextEvent:**(int)*mask* Returns pointer to the next event matching *mask*
  into:(NXEvent *)*eventPtr*

- (NXEvent *)**peekNextEvent:**(int)*mask*
  into:(NXEvent *)*eventPtr*
  waitFor:(float)*timeout*
  threshold:(int)*level*
  Returns pointer to the next event matching *mask*

Journaling

- (BOOL)**isJournalable** Returns whether the application can be journaled
- **setJournalable:**(BOOL)*flag* Sets whether the application can be journaled
- **masterJournaler** Returns the controlling NXJournaler object
- **slaveJournaler** Returns the controlled NXJournaler object

Handling User Actions and Events

- **applicationDefined:**(NXEvent *)*theEvent* Responds to an application-defined event
- **hide:***sender* Hides all the application's windows
- (BOOL)**isHidden** YES if windows are hidden
- (int)**unhide** Responds to message to unhide windows
- **unhide:***sender* Restores hidden windows to the screen
- **unhideWithoutActivation:***sender* Restores hidden windows without activating their owner
- **powerOff:**(NXEvent *)*theEvent* Responds to a power-off subevent
- (int)**powerOffIn:**(int)*ms* **andSave:**(int)*aFlag* Responds to message from Workspace Manager
- **rightMouseDown:**(NXEvent *)*theEvent* Causes main menu to pop up under the mouse
- (int)**unmounting:**(const char *)*fullPath*
  ok:(int *)*flag*
  Responds to message from Workspace Manager

Sending Action Messages

- (BOOL)**sendAction:**(SEL)*aSelector*
  to:*aTarget*
  from:*sender*
  Sends an action message to *aTarget* or up the responder chain
- (BOOL)**tryToPerform:**(SEL)*aSelector* Attempts to send a message to the application or the

with:*anObject*	delegate
- **calcTargetForAction:**(SEL)*theAction*	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

- **appIcon**	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* Adds a menu item for *aWindow*
  title:(const char *)*aString*
  filename:(BOOL)*isFilename*
- **removeWindowsItem:***aWindow* Removes the Window's menu item
- **changeWindowsItem:***aWindow* Changes the Window's menu item
  title:(const char *)*aString*
  filename:(BOOL)*isFilename*
- **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** — Returns the Application's DPS context
- **focusView** — Gets the currently lockFocus'ed View
- (const char *)**hostName** — 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* — Asks the delegate to open the *fullPath* file
- (int)**openTempFile:**(const char *)*fullPath* **ok:**(int *)*flag* — Asks the delegate to open a temporary file
- (int)**fileOperationCompleted:**(int)*operation* — Notification of completion of NXWorkspaceRequest; arbitrary *operation* and return

Responding to Devices

- (int)**mounted:**(const char *)*fullPath* — Notice that device at fullPath has been mounted; returns 0 (unless overrideen in a subclass)
- **unmounted:**(const char *)*fullPath* — Notice that device at fullPath has been unmounted; returns 0 (unless overrideen in a subclass)

Printing

- **setPrintInfo:***info* — Makes *info* the application's PrintInfo object
- **printInfo** — 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 launch[2]
  applicationWillLaunch:(const char *)*appName*
- **app:***sender* Notice that *appName* launched[2]
  applicationDidLaunch:(const char *)*appName*
- **appWillInit:***sender* Notifies delegate before initializing[2]
- **appDidInit:***sender* Notifies delegate before getting first event[2]
- **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 file[2]

- (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 interface[2]

- **app:***sender*
  fileOperationCompleted:(int)*operation*
  Notice that *operation* completed; your arbitrary return[2]

- **app:***sender* **mounted:**(const char *)*fullPath*
  Notification that device at fullPath was mounted[2]

- (int)**app:***sender*
  unmounting:(const char *)*fullPath*
  Facilitates unmounting a device[2]

- **app:***sender*
  unmounted:(const char *)*fullPath*
  Notification that device at fullPath was unmounted[2]

- **appWillTerminate:***sender*
  Notification that application will terminate[2]

- **app:***sender*
  willShowHelpPanel:*panel*
  Notification that *sender* will show *panel*

- **app:***sender*
  powerOffIn:(int)*ms*
  andSave:(int)*aFlag*
  Responds to message from Workspace Manager[2]

- **powerOff:**(NXEvent *)*theEvent*
  Responds to a power-off subevent

- **app:***sender*
  applicationDidTerminate:(const char *)*appName*
  Notification that application terminated[2]

[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.

Box

Inherits From: View : Responder : Object

Initializing and Freeing a Box

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

Modifying the Border and Title

- **setBorderType:**(int)*aType*	Sets the Box's border to *aType*
- (int)**borderType**	Returns the Box's border type
- **setTitlePosition:**(int)*aPosition*	Sets the position of the title to *aPosition*
- (int)**titlePosition**	Returns the position of the title
- **setTitle:**(const char *)*aString*	Sets the Box's title to *aString*
- (const char *)**title**	Returns the title of the Box
- **cell**	Returns the Cell used to draw the title
- **setFont:***fontObj*	Sets the Font of the title to *fontObj*
- **font**	Returns the Font used to draw the title

Setting and Placing the Content View

- **setContentView:***aView*	Replaces the **Box's content view** with *aView*
- **contentView**	Returns the **content view**
- **setOffsets:**(NXCoord)*w* **:**(NXCoord)*h*	Sets the distance between the border and the content view
- **getOffsets:**(NXSize *)*aSize*	Gets the distance between the border and the content view

Putting Views in the Box

- **addSubview:***aView*	Adds *aView* as a subview of the content view
- **replaceSubview:***aView* **with:***anotherView*	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 ªButtonº
- **initFrame:**(const NXRect *)*frameRect*
  Initializes a new Button within *frameRect*

- **initFrame:**(const NXRect *)*frameRect*
  icon:(const char *)*iconName*
  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*

- **initFrame:**(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 *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*

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

- (int)**state**

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

Setting the Repeat Interval

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

  Sets repeat parameters for continuous Buttons

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

  Gets repeat parameters for continuous Buttons

Setting the Titles

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

  Makes *aString* the Button's title

- **setTitleNoCopy:**(const char *)*aString* — Makes *aString* the Button's title without copying it
- (const char *)**title** — Returns the Button's title
- **setAltTitle:**(const char *)*aString* — Makes *aString* the Button's alternate title
- (const char *)**altTitle** — Returns the Button's alternate title

Setting the Icons

- **setIcon:**(const char *)*iconName* — Makes the NXImage named *iconName* the Button's icon
- **setIcon:**(const char *)*iconName* **position:**(int)*aPosition* — Sets the icon by name, and its position
- (const char *)**icon** — Returns the name of the Button's icon
- **setAltIcon:**(const char *)*iconName* — Makes the NXImage named *iconName* the alternate icon
- (const char *)**altIcon** — Returns the name of the Button's alternate icon
- **setImage:***image* — Makes the NXImage *image* the Button's icon
- **image** — Returns the Button's image
- **setAltImage:***altImage* — Makes the NXImage *image* the alternate icon
- **altImage** — Returns the Button's alternate image
- **setIconPosition:**(int)*aPosition* — Sets the position of the Button's icon
- (int)**iconPosition** — Returns the position of the Button's icon

Modifying Graphic Attributes

- **setTransparent:**(BOOL)*flag* — Sets whether the Button is transparent
- (BOOL)**isTransparent** — Returns whether the Button is transparent
- **setBordered:**(BOOL)*flag* — Sets whether the Button has a bezeled border
- (BOOL)**isBordered** — Returns whether the Button has a bezeled border

Displaying

- **display** — Displays the Button
- **highlight:**(BOOL)*flag* — 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 ªButtonº
- **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* — Calculates and returns the size of the ButtonCell
- **getDrawRect:**(NXRect *)*theRect* — Returns the rectangle the ButtonCell draws in
- **getTitleRect:**(NXRect *)*theRect* — Returns the rectangle the title is drawn in
- **getIconRect:**(NXRect *)*theRect* — Returns the rectangle the icon is drawn in

Setting the Titles

- **setTitle:**(const char *)*aString* — Makes a copy of *aString* the ButtonCell's title
- **setTitleNoCopy:**(const char *)*aString* — Makes *aString* the ButtonCell's title without copying it
- (const char *)**title** — Returns the ButtonCell's title
- **setAltTitle:**(const char *)*aString* — Makes a copy of *aString* the ButtonCell's alternate title
- (const char *)**altTitle** — Returns the ButtonCell's alternate title
- **setFont:***fontObject* — Sets the Font used to draw the title

Setting the Icons

- **setIcon:**(const char *)*iconName* — Makes the NXImage named *iconName* the ButtonCell's icon
- (const char *)**icon** — Returns the name of the ButtonCell's icon
- **setAltIcon:**(const char *)*iconName* — Makes the NXImage named *iconName* the ButtonCell's alternate icon
- (const char *)**altIcon** — Returns the name of the ButtonCell's alternate icon
- **setImage:***image* — Makes the NXImage object *image* the ButtonCell's icon
- **image** — Returns the ButtonCell's icon
- **setAltImage:***altImage* — Makes the NXImage object *image* the alternate icon
- **altImage** — Returns the ButtonCell's alternate icon
- **setIconPosition:**(int)*aPosition* — Sets the position of the ButtonCell's icon to *aPosition*
- (int)**iconPosition** — Returns the position of the ButtonCell's icon

Setting the Sound

- **setSound:***aSound*	Sets the Sound played by the ButtonCell on a mouse-down
- **sound**	Returns the Sound played by the ButtonCell

Setting the State

- **setDoubleValue:**(double)*aDouble*	Sets the ButtonCell's state (value) to *aDouble*
- (double)**doubleValue**	Returns the ButtonCell's state as a **double**
- **setFloatValue:**(float)*aFloat*	Sets the ButtonCell's state (value) to *aFloat*
- (float)**floatValue**	Returns the ButtonCell's state as a **float**
- **setIntValue:**(int)*anInt*	Sets the ButtonCell's state (value) to *anInt*
- (int)**intValue**	Returns the ButtonCell's state as an **int**
- **setStringValue:**(const char *)*aString*	Sets the ButtonCell's state (value) to a copy of *aString*
- **setStringValueNoCopy:**(const char *)*aString*	Sets the ButtonCell's state (value) to *aString*
- (const char *)**stringValue**	Returns the ButtonCell's state as a string

Setting the Repeat Interval

- **setPeriodicDelay:**(float)*delay* **andInterval:**(float)*interval*	Sets repeat parameters for continuous ButtonCells
- **getPeriodicDelay:**(float *)*delay* **andInterval:**(float *)*interval*	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 *)*aRect*
  inView:*controlView*
  Draws the inside of the ButtonCell
- **drawSelf:**(const NXRect *)*cellFrame*
  inView:*controlView*
  Draws the ButtonCell
- **highlight:**(const NXRect *)*cellFrame*
  inView:*controlView*
  lit:(BOOL)*flag*
  Highlights the ButtonCell

Archiving

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

Cell

Inherits From: Object

Initializing, Copying, and Freeing a Cell

- **init**
  Initializes a new Cell
- **initIconCell:**(const char *)*iconName*
  Initializes a new Cell with the NXImage named *iconName*
- **initTextCell:**(const char *)*aString*
  Initializes a new Cell with title *aString*
- **copyFromZone:**(NXZone *)*zone*
  Returns a copy of the receiving Cell from *zone*
- **free**
  Deallocates the Cell

Determining Component Sizes

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

- **calcCellSize:**(NXSize *)*theSize*
  inRect:(const NXRect *)*aRect*
  Returns the minimum size needed to display the Cell
- **calcDrawInfo:**(const NXRect *)*aRect*
  Implemented by subclasses to recalculate drawing sizes
- **getDrawRect:**(NXRect *)*theRect*
  Returns the rectangle the Cell draws in
- **getIconRect:**(NXRect *)*theRect*
  Returns the rectangle that an icon is drawn in
- **getTitleRect:**(NXRect *)*theRect*
  Returns the rectangle that a title is drawn in

Setting the Cell's Type

- **setType:**(int)*aType*
  Sets the Cell's type to *aType*
- (int)**type**
  Returns the Cell's type

Setting the Cell's State

- **setState:**(int)*value*
  Sets the state of the Cell to *value* (0 or 1)
- **incrementState**
  Increments the state of the Cell
- (int)**state**
  Returns the state of the Cell (0 or 1)

Enabling and Disabling the Cell

- **setEnabled:**(BOOL)*flag*
  Sets whether the Cell reacts to mouse events
- (BOOL)**isEnabled**
  Returns whether the Cell reacts to mouse events

Setting the Icon

- **setIcon:**(const char *)*iconName*
  Sets the Cell's icon to the NXImage named *iconName*
- (const char *)**icon**
  Returns the name of the Cell's icon

Setting the Cell's Value

- **setDoubleValue:**(double)*aDouble*
  Sets the Cell's value to *aDouble*
- (double)**doubleValue**
  Returns the Cell's value as a **double**

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

Interacting with Other Cells

- **takeDoubleValueFrom:***sender*	Sets the Cell's value to *sender*'s **doubleValue**
- **takeFloatValueFrom:***sender*	Sets the Cell's value to *sender*'s **floatValue**
- **takeIntValueFrom:***sender*	Sets the Cell's value to *sender*'s **intValue**
- **takeStringValueFrom:***sender*	Sets the Cell's value to *sender*'s **stringValue**

Modifying Text Attributes

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

Editing Text

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

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* — Sets the display format for floating point values
  left:(unsigned)*leftDigits*
  right:(unsigned)*rightDigits*

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* Draws the area within the Cell's border in *aView*
  inView:*aView*
- **drawSelf:**(const NXRect *)*cellFrame* Draws the Cell in *aView*
  inView:*aView*
- **highlight:**(const NXRect *)*cellFrame* Highlights the Cell according to *flag* in *aView*
  inView:*aView*
  lit:(BOOL)*flag*
- (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* — Implemented by subclasses to set an identifier tag
- (int)**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*
  mouseIsUp:(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* Reads the Cell from *stream*
- **write:**(NXTypedStream *)*stream* Writes the Cell to *stream*
- **awake** 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

- **initFrame:**(const NXRect *)*frameRect* Initializes a new ClipView instance
- **free** Releases the ClipView's storage

Modifying the Frame Rectangle

- **moveTo:**(NXCoord)*x* **:**(NXCoord)*y* Moves the origin of the frame rectangle
- **rotateTo:**(NXCoord)*angle* Overridden to disable rotation
- **sizeTo:**(NXCoord)*width* **:**(NXCoord)*height* Resizes the ClipView's frame

Modifying the Coordinate System

- **rotate:**(NXCoord)*angle* Overriden to disable rotation
- **scale:**(NXCoord)*x* **:**(NXCoord)*y* Rescales the coordinate system
- **setDrawOrigin:**(NXCoord)*x* **:**(NXCoord)*y* Sets the origin of the coordinate system
- **setDrawRotation:**(NXCoord)*angle* Disables rotation of the coordinate system

- **setDrawSize:**(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

- **initFrame:**(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*	Sets whether the Control reacts to mouse events
- (BOOL)**isEnabled**	Returns whether the Control reacts to mouse events

Identifying the Selected Cell

- **selectedCell**	Returns the Control's selected Cell
- (int)**selectedTag**	Returns the tag of the Control's selected Cell

Setting the Control's Value

- **setDoubleValue:**(double)*aDouble*	Sets the Control's value to *aDouble*
- (double)**doubleValue**	Returns the Control's value as a **double**
- **setFloatValue:**(float)*aFloat*	Sets the Control's value to *aFloate*
- (float)**floatValue**	Returns the Control's value as a **float**
- **setIntValue:**(int)*anInt*	Sets the Control's value to *anInt*
- (int)**intValue**	Returns the Control's value as an **int**
- **setStringValue:**(const char *)*aString*	Sets the Control's value to *aString*
- **setStringValueNoCopy:**(const char *)*aString*	Sets the Control's value to *aString*
- **setStringValueNoCopy:**(char *)*aString* **shouldFree:**(BOOL)*flag*	Sets the Control's value to *aString*
- (const char *)**stringValue**	Returns the Control's value as a string

Interacting with Other Controls

- **takeDoubleValueFrom:***sender*	Sets the Control's value to *sender*'s **doubleValue**
- **takeFloatValueFrom:***sender*	Sets the Control's value to *sender*'s **floatValue**
- **takeIntValueFrom:***sender*	Sets the Control's value to *sender*'s **intValue**
- **takeStringValueFrom:***sender*	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* **left:**(unsigned)*leftDigits* **right:**(unsigned)*rightDigits*	Sets the display format for floating point values

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* — Redisplays *aCell* or marks it for redisplay
- **updateCellInside:***aCell* — Redisplays the inside of *aCell* or marks it for redisplay

Target and Action

- **setAction:**(SEL)*aSelector* — Sets the Control's action method to *aSelector*
- (SEL)**action** — Returns the Control's action method
- **setTarget:***anObject* — Sets the Control's target object to *anObject*
- **target** — Returns the Control's target object
- **setContinuous:**(BOOL)*flag* — Sets whether the Control continuously sends its action
- (BOOL)**isContinuous** — Returns whether the Control continuously sends its action
- **sendAction:**(SEL)*theAction* **to:***theTarget* — Has the Application object send *theAction* to *theTarget*
- (int)**sendActionOn:**(int)*mask* — Determines when the action is sent while tracking

Assigning a Tag

- **setTag:**(int)*anInt* — Sets the Control's tag to *anInt*
- (int)**tag** — Returns the Control's tag

Tracking the Mouse

- **ignoreMultiClick:**(BOOL)*flag* — Sets whether multiple clicks are ignored
- **mouseDown:**(NXEvent *)*theEvent* — Handles a mouse-down event in the Control
- (int)**mouseDownFlags** — Returns flags in effect at beginning of tracking

Archiving

- **read:**(NXTypedStream *)*stream* — Reads the Control from *stream*
- **write:**(NXTypedStream *)*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*
+ **userFontOfSize:**(float)*fontSize* **matrix:**(const float *)*fontMatrix*	Returns the Font object representing the application's standard font of size *fontSize* and matrix *fontMatrix*
+ **systemFontOfSize:**(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**	Returns the full name of the font
- (const float *)**familyName**	Returns the name of the font's family
- (const char *)**name**	Returns the name of the font
- (int)**fontNum**	Returns the Window Server's font number
- (float)**getWidthOf:**(const char *)*string*	Returns the width of *string* in this font
- (BOOL)**hasMatrix**	Returns whether font differs from identity matrix
- (const float *)**matrix**	Returns a pointer to the font matrix
- (NXFontMetrics *)**metrics**	Returns pointer to a record of font information
- (float)**pointSize**	Returns the size of the font in points
- (NXFontMetrics *)**readMetrics:**(int)*flags*	Reads *flags* information into the font record
- **screenFont**	Returns the screen font for this font
- (int)**style**	Returns the font style

Setting the Font

- **set**	Makes this font the graphic state's current font
- **setStyle:**(int)*aStyle*	Sets the Font's style
+ **setUserFixedPitchFont:**(Font *)*aFont*	Sets the fixed-pitch font used by default in the application
+ **setUserFont:**(Font *)*aFont*	Sets the standard font used by default in the application

Archiving

- **awake**	Reinitializes the Font object
- **finishUnarchiving**	Checks whether the Font object already exists
- **read:**(NXTypedStream *)*stream*	Reads the Font object from the typed stream
- **write:**(NXTypedStream *)*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:***fontObj*	Converts the font in response to **changeFont:**
- **convertWeight:**(BOOL)*upFlag* **of:***fontObj*	Raises or lowers the weight of the font
- **convert:***fontObj* **toFace:**(const char *)*typeface*	Converts the font to the specified typeface
- **convert:***fontObj* **toFamily:**(const char *)*family*	Converts the font to the specified family
- **convert:***fontObj* **toSize:**(float)*size*	Converts the font to the specified point size
- **convert:***fontObj* **toHaveTrait:**(NXFontTraitMask)*trait*	Converts the font to have the specified trait
- **convert:***fontObj* **toNotHaveTrait:**(NXFontTraitMask)*trait*	Converts the font to remove the specified trait
- **findFont:**(const char *)*family* **traits:**(NXFontTraitMask)*traits* **weight:**(int)*weight* **size:**(float)*size*	Tries to find a font that matches the specified characteristics
- **getFamily:**(const char **)*family* **traits:**(NXFontTraitMask *)*traits* **weight:**(int *)*weight* **size:**(float *)*size* **ofFont:***fontObj*	Provides the characteristics of the given *fontObj*

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
- **setSelFont:***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
- **selFont**	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

- **initFrame:**(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 ªFieldº as its title
- **initTextCell:**(const char *)*aString* — Initializes a new FormCell with *aStrin*g 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 *)*aString*	Sets the FormCell's title to *aString*
- (const char *)**title**	Returns the FormCell's title
- **setTitleFont:***fontObject*	Sets the Font used to draw the title
- **titleFont**	Returns the Font used to draw the title
- **setTitleAlignment:**(int)*mode*	Sets the alignment of the title
- (int)**titleAlignment**	Returns the alignment of the title
- **setTitleWidth:**(NXCoord)*width*	Sets the width of the FormCell's title field to *width*
- (NXCoord)**titleWidth:**(const NXSize *)*aSize*	Returns the width of the title, constrained to aSize
- (NXCoord)**titleWidth**	Returns the width of the title

Modifying Graphic Attributes

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

Displaying

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

Managing Cursor Rectangles

- **resetCursorRect:**(const NXRect *)*cellFrame* **inView:***controlView*	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*)*event* **inRect:**(const NXRect*)*aRect* **ofView:***controlView*	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**	Returns registered name of the Listener's port
- **setPriority:**(int)*level*	Sets the priority for receiving messages to *level*
- (int)**priority**	Returns priority level for receiving messages
- **setTimeout:**(int)*ms*	Sets how long to wait on sending reply
- (int)**timeout**	Returns how long to wait on sending reply
+ **run**	Enables Listener in absence of an Application object

Providing for Program Control

- (int)**msgCalc:**(int *)*flag*	Receives message to update the current window
- (int)**msgCopyAsType:**(const char *)*aType* **ok:**(int *)*flag*	Receives message to copy the selection
- (int)**msgCutAsType:**(const char *)*aType* **ok:**(int *)*flag*	Receives message to cut selection as *aType* data
- (int)**msgDirectory:**(char *const *)*fullPath* **ok:**(int *)*flag*	Receives message asking for the current directory
- (int)**msgFile:**(char *const *)*fullPath* **ok:**(int *)*flag*	Receives message asking for the current document
- (int)**msgPaste:**(int *)*flag*	Receives message to paste data from pasteboard
- (int)**msgPosition:**(char *const *)*aString* **posType:**(int *)*anInt* **ok:**(int *)*flag*	Receives message requesting selection information
- (int)**msgPrint:**(const char *)*fullPath* **ok:**(int *)*flag*	Receives message to print the *fullPath* file
- (int)**msgQuit:**(int *)*flag*	Receives a remote message to quit
- (int)**msgSelection:**(char *const *)*bytes* **length:**(int *)*numBytes* **asType:**(const char *)*aType* **ok:**(int *)*flag*	Receives message requesting the current selection
- (int)**msgSetPosition:**(const char *)*aString* **posType:**(int)*anInt* **andSelect:**(int)*selectFlag*	Receives message to scroll so *aString* is visible

ok:(int *)*flag*

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

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** — Initializes the Matrix class

+ **setCellClass:***classId* — 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*
  mode:(int)*aMode*
  cellClass:*classId*
  numRows:(int)*numRows*
  numCols:(int)*numCols*
  — Initializes a new Matrix object in *frameRect*, with *aMode* as the selection mode, *classId* as the class used to make new Cells, and having *numRows* rows and *numCols* columns

- **initFrame:**(const NXRect *)*frameRect*
  mode:(int)*aMode*
  prototype:*aCell*
  numRows:(int)*numRows*
  numCols:(int)*numCols*
  — Initializes a new Matrix object with the given values with *aMode* as the selection mode, *aCell* as the prototype copied to make new Cells, and having *numRows* rows 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*	Sets the subclass of Cell used to make new Cells
- **setPrototype:***aCell*	Sets the prototype Cell copied to make new Cells
- **prototype**	Returns the prototype Cell copied to make new Cells

Laying Out the Matrix

- **addCol**	Adds a new column of Cells to the bottom of the Matrix
- **addRow**	Adds a new row of Cells to the right of the Matrix
- **insertColAt:**(int)*col*	Inserts a new column of Cells at *col*, creating as many as needed to make the Matrix *col* columns wide
- **insertRowAt:**(int)*row*	Inserts a new row of Cells at *row*, creating as many as needed to make the Matrix *row* rows wide
- **removeColAt:**(int)*col* **andFree:**(BOOL)*flag*	Removes the column at *col*, freeing the Cells if *flag* is YES
- **removeRowAt:**(int)*row* **andFree:**(BOOL)*flag*	Removes the row at *row*, freeing the Cells if *flag* is YES
- **makeCellAt:**(int)*row* **:**(int)*col*	Creates a new Cell at *row*, *col* in the Matrix and returns it
- **putCell:***newCell* **at:**(int)*row* **:**(int)*col*	Replaces Cell at *row* and *col* with *newCell*; returns old Cell
- **renewRows:**(int)*newRows* **cols:**(int)*newCols*	Changes the number of rows and columns in Matrix without freeing any Cells
- **setCellSize:**(const NXSize *)*aSize*	Sets the width and height of all Cells in the Matrix
- **getCellSize:**(NXSize *)*theSize*	Gets the width and height of Cells in the Matrix
- **getCellFrame:**(NXRect *)*theRect* **at:**(int)*row* **:**(int)*col*	Returns the frame of the Cell at *row* and *col*
- **setIntercell:**(const NXSize *)*aSize*	Sets the vertical and horizontal spacing between Cells
- **getIntercell:**(NXSize *)*theSize*	Gets the vertical and horizontal spacing between Cells
- (int)**cellCount**	Returns the number of Cells in the Matrix
- **getNumRows:**(int *)*rowCount* **numCols:**(int *)*colCount*	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 *)*aList* Puts the selected Cells into *aList* 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)*anInt* Returns the Cell with *anInt* as its tag
- **cellAt:**(int)*row* **:**(int)*col* Returns the Cell at row *row* and column *col*
- **cellList** Returns the Matrix's List of Cells

Modifying Graphic Attributes

- **setBackgroundColor:**(NXColor)*aColor* Sets the color of the background between Cells to *aColor*
- (NXColor)**backgroundColor** Returns the color of the background between Cells
- **setBackgroundGray:**(float)*value* Sets the gray of the background between Cells to *value*
- (float)**backgroundGray** Returns the gray of the background between Cells
- **setCellBackgroundColor:**(NXColor)*aColor* Sets the color of the background within Cells to *aColor*
- (NXColor)**cellBackgroundColor** Returns the color of the background within Cells
- **setCellBackgroundGray:**(float)*value* Sets the gray of the background within Cells to *value*
- (float)**cellBackgroundGray** Returns the gray of the background within Cells
- **setBackgroundTransparent:**(BOOL)*flag* Sets whether the background between Cells is transparent
- (BOOL)**isBackgroundTransparent** Returns whether the background between Cells is transparent
- **setCellBackgroundTransparent:**(BOOL)*flag* Sets whether the background within Cells is transparent
- (BOOL)**isCellBackgroundTransparent** Returns whether the background within Cells is transparent
- **setFont:***fontObject* 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* — Selects the text in the first or last editable Cell
- **selectTextAt:**(int)*row* **:**(int)*col* — Selects the text of the Cell at *row*, *col* in the Matrix

Setting Tab Key Behavior

- **setNextText:***anObject* — Sets the object selected when the user presses Tab while editing the last text Cell
- **setPreviousText:***anObject* — Sets the object selected when user presses Shift-Tab while editing the first text Cell

Assigning a Text Delegate

- **setTextDelegate:***anObject* — Sets the delegate for messages from the field editor
- **textDelegate** — Returns the delegate for messages from the field editor

Text Object Delegate Methods

- (BOOL)**textWillChange:***textObject* — Responds to a message from the field editor (see Text)
- **textDidChange:***textObject* — Responds to a message from the field editor (see Text)
- **textDidGetKeys:***textObject* **isEmpty:**(BOOL)*flag* — Responds to a message from the field editor (see Text)
- (BOOL)**textWillEnd:***textObject* — Responds to a message from the field editor (see Text)
- **textDidEnd:***textObject* **endChar:**(unsigned short)*whyEnd* — Responds to a message from the field editor (see Text)

Resizing the Matrix and Cells

- **setAutosizeCells:**(BOOL)flag — Sets whether the Matrix resizes its Cells automatically
- (BOOL)**doesAutosizeCells** — Returns whether the Matrix resizes its Cells automatically
- **calcSize** — Calculates Cell sizes
- **sizeTo:**(float)*width* **:**(float)*height* — Resizes the Matrix to *width* and *height*
- **sizeToCells** — 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

have one

- **sendAction:**(SEL)*theAction* **to:***theTarget* — Has the Application object send *theAction* to *anObject*
- **sendAction:**(SEL)*aSelector* **to:***anObject* **forAllCells:**(BOOL)*flag* — Sends *aSelector* to *anObject*, for all Cells if *flag* is YES
- **sendDoubleAction** — Sends the action corresponding to a double-click
- **setReaction:**(BOOL)*flag* — Sets whether sending an action clears the selection

Handling Event and Action Messages

- (BOOL)**acceptsFirstMouse** — Returns NO only if mode is NX_LISTMODE
- **mouseDown:**(NXEvent *)*theEvent* — Responds to a mouse-down event
- (int)**mouseDownFlags** — Returns the event flags in effect at start of tracking
- (BOOL)**performKeyEquivalent:**(NXEvent *)*theEvent* — 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* — Reads the Matrix from *stream*
- **write:**(NXTypedStream *)*stream* — Writes the Matrix to *stream*

Menu

Inherits From: Panel : Window : Responder : Object

Creating a Menu Zone

+ **setMenuZone:**(NXZone *)*zone*	Sets the zone from which Menus should be allocated
+ (NXZone *)**menuZone**	Returns the zone from which Menus should be allocated, creating one if necessary

Initializing a New Menu

- **init**	Initializes a new Menu with the title ªMenuº
- **initTitle:**(const char *)*aTitle*	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*	Adds a new item to the end of the Menu
- **setItemList:***aMatrix*	Replaces the current Matrix of items with *aMatrix*
- **itemList**	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*	Makes *aMenu* a submenu controlled by *aCell*
- **submenuAction:***sender*	Activates a submenu attached to *sender*'s Menu

Managing Menu Windows

- **moveTopLeftTo:**(NXCoord)*x* **:**(NXCoord)*y*	Moves the Menu's top left corner to *x*, *y*
- **windowMoved:**(NXEvent *)*theEvent*	Handles a submenu being torn off its supermenu
- **getLocation:**(NXPoint *)*theLocation*	Determines where to display an attached submenu

forSubmenu:*aSubmenu* when it's brought up

- **sizeToFit** Resizes the Menu to exactly fit the command items
- **close** Removes the Menu (and any submenus) from the screen

Displaying the Menu

- **display** Displays the Menu, resizing if needed
- **setAutoupdate:**(BOOL)*flag* Sets whether Menu reacts to **update** messages
- **update** Updates each MenuCell item

Handling Events

- **mouseDown:**(NXEvent *)*theEvent* Tracks the cursor in the Menu and submenus
- **rightMouseDown:**(NXEvent *)*theEvent* Pops the main menu up under the cursor

Archiving

- **read:**(NXTypedStream *)*stream* Reads the Menu from *stream*
- **write:**(NXTypedStream *)*stream* Writes the Menu to *stream*
- **awake** Reinitializes a Menu as it's unarchived

MenuCell

Inherits From: ButtonCell : ActionCell : Cell : Object

Initializing a New MenuCell

- **init** Initializes a new MenuCell with ªMenu Itemº as its title
- **initTextCell:**(const char *)*aString* Initializes a new MenuCell with *aString* as its title

Setting the Update Action

- **setUpdateAction:**(SEL)*aSelector* **forMenu:***aMenu* — Sets the update action for the MenuCell to *aSelector*, and sets *aMenu* to auto-update
- (SEL)**updateAction** — 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* — Sets the class to apply user-assigned key equivalents
- (unsigned short)**userKeyEquivalent** — Returns the user-assigned key equivalent for the MenuCell

Archiving

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

NXBitmapImageRep

Inherits From: NXImageRep : Object

Initializing a New NXBitmapImageRep object

- **initFromSection:**(const char *)*name* — Initializes the new object from TIFF data in the section
- **initFromFile:**(const char *)*filename* — Initializes the new object from TIFF data in *filename*
- **initFromStream:**(NXStream *)*stream* — Initializes the new object from TIFF data in *stream*
- **initData:**(unsigned char *)*data*
  fromRect:(const NXRect *)*rect* — Initializes the new object using data read from an image
- **initData:**(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
- **initDataPlanes:**(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*
pixelsWide:(int *)*width*
pixelsHigh:(int *)*height*
bitsPerSample:(int *)*bps*
samplesPerPixel:(int *)*spp*
hasAlpha:(BOOL *)*alpha*
isPlanar:(BOOL *)*config*
colorSpace:(NXColorSpace *)*space*
Provides information about the image bounded by the *rect* rectangle

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

- **initFrame:**(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* **andSendActionMessages:**(BOOL)*sendFlag* Enables arrow keys for scrolling and sending action messages
- **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*

	Sets whether buttons are used to scroll horizontally
- (BOOL)**areHorizontalScollButtonsEnabled**	Returns whether buttons are used to scroll horizontally
- **setHorizontalScollerEnabled:**(BOOL)*flag*	Sets whether Scrollers are used to scroll horizontally
- (BOOL)**areHorizontalScollerEnabled**	Returns whether Scrollers are used to scroll horizontally

Setting the NXBrowser's Appearance

- **setMinColumnWidth:**(int)*columnWidth*	Sets the minimum column width
- (int)**minColumnWidth**	Returns the minimum column width
- **setMaxVisibleColumns:**(int)*columnCount*	Sets the maximum number of columns displayed
- (int)**maxVisibleColumns**	Returns the maximum number of visible columns
- (int)**numVisibleColumns**	Returns the number of columns visible
- (int)**firstVisibleColumn**	Returns the index of the first visible column
- (int)**lastVisibleColumn**	Returns the index of the last visible column
- (int)**lastColumn**	Returns the index of the last column loaded
- **separateColumns:**(BOOL)*flag*	Sets whether to separate columns with bezeled borders
- (BOOL)**columnsAreSeparated**	Returns whether columns are separated by bezeled borders

Manipulating Columns

- **loadColumnZero**	Loads column zero; unloads previously loaded columns
- (BOOL)**isLoaded**	Returns whether column zero is loaded
- **addColumn**	Adds a column to the right of the last column
- **reloadColumn:**(int)*column*	Reloads *column* if it is loaded; sets it as the last column
- **displayColumn:**(int)*column*	Updates to display columns through index *column*
- **displayAllColumns**	Updates the NXBrowser to display all loaded columns
- **setLastColumn:**(int)*column*	Sets the last column to *column*
- **selectAll:***sender*	Selects all Cells in the last column of the NXBrowser
- (int)**selectedColumn**	Returns the index of the last column with a selected item
- (int)**columnOf:***matrix*	Returns the column number in which *matrix* is located

- **validateVisibleColumns** — Invokes delegate method **browser:columnIsValid:** 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 includingborders

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* **columnIsValid:**(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** — Initializes a new NXBrowserCell with ªBrowserItemº as its title
- **initTextCell:**(const char *)*aString* — 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** — Returns YES, since an NXBrowserCell is opaque
+ **branchIcon** — Returns the NXImage for branch NXBrowserCells
+ **branchIconH** — Returns the NXImage for highlighted branches

Displaying

- **drawInside:**(const NXRect *)*cellFrame* **inView:***aView* — Draws the inside of the NXBrowserCell in *aView*
- **drawSelf:**(const NXRect *)*cellFrame* **inView:***aView* — Draws the entire NXBrowserCell in *aView*
- **highlight:**(const NXRect *)*cellFrame* **inView:***aView* **lit:**(BOOL)*lit* — If *lit* is YES, highlights the NXBrowserCell in *aView*

Placing in Browser Hierarchy

- **setLeaf:**(BOOL)*flag* — Sets whether the NXBrowserCell is a leaf or a branch
- (BOOL)**isLeaf** — 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)**isLoaded** — 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

- **initFromWindow:**(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*
  style:(int)*aStyle*
  backing:(int)*bufferingType*
  buttonMask:(int)*mask*
  defer:(BOOL)*flag*
  — Returns the shared NXColorPanel

+ **newContent:**(const NXRect *)*contentRect*
  style:(int)*aStyle*
  backing:(int)*bufferingType*
  buttonMask:(int)*mask*
  defer:(BOOL)*flag*
  colorMask:(int)*colormask*
  — Returns the shared NXColorPanel

+ **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* — Sets the mode and returns the NXColorPanel
- **setAccessoryView:***aView* — Sets the accessory view to *aView*
- **setAction:**(SEL)*aSelector* — Sets the action message sent to the target
- **setShowAlpha:**(BOOL)*flag* — Sets the NXColorPanel to show alpha values
- **setTarget:***anObject* — Sets the target of the NXColorPanel

Setting Color

- **color:**(NXColor *)*color* — Returns the currently selected color
- **setColor:**(NXColor)*color* — Sets the *color* of the NXColorPanel
- \+ **dragColor:**(NXColor *)*color* **withEvent:**(NXEvent *)*event* **fromView:***sourceView* — Drags *color* into a destination view from *sourceView. event* is usually an NX_MOUSEUP

NXColorPicker

Inherits From: Object

Conforms To: NXColorPickingDefault

Initialization

- **initFromPickerMask:**(int)*theMask* **withColorPanel:***thePanel* — Initializes the receiver for the specified mask and color panel

Button Images

- **provideNewButtonImage** — Returns a new button image for the color picker
- **insertNewButtonImage:***newImage* **in:***newButtonCell* — Override to customize *newImag*e 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

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

Event Handling

- (BOOL)**acceptsFirstMouse**	Returns YES
- **mouseDown:**(NXEvent *)*theEvent*	Responds to mouse down in the NXColorWell

Drawing

- **drawSelf:**(const NXRect *)*rects* **:**(int)*rectCount*	Draws the entire NXColorWell, including borders
- **drawWellInside:**(const NXRect *)*insideRect*	Draws the colored area inside the NXColorWell,without drawing borders

Activating

- **deactivate**	Deactivates and returns the NXColorWell
+ **deactivateAllWells**	Deactivates all currently active NXColorWells
- **activate:**(int)*exclusive*	Activates and returns the NXColorWell
- (BOOL)**isActive**	Returns YES if the NXColorWell is active
- **setEnabled:**(BOOL)*enabled*	Enables the NXColorWell

Managing Color

- **activeWellsTakeColorFrom:***sender*	Changes color of all active wells to that of sender
- **activeWellsTakeColorFrom:***sender* **continuous:**(BOOL)*continuously*	Continuously changes color of all active, continuous wells to that of *sender*
- (NXColor)**color**	Returns the color of the NXColorPanel
- **takeColorFrom:***sender*	Changes color of the well to that of *sender*
- **acceptColor:**(NXColor)*color* **atPoint:**(NXPoint *)*aPoint*	Changes color of the well to *color* when *aPoint* is a point in the bounds of the NXColorWell
- **setColor:**(NXColor)*color*	Sets the color of the well to *color*
- **updateCustomColorList**	Saves the current color list in NX_COLORLISTMODE

Target and Action

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

Archiving

- **awake** Initializes the NXColorWell after unarchiving

NXCursor

Inherits From: Object

Initializing a New NXCursor Object

- **init** Initializes a new NXCursor, but doesn't set the image
- **initFromImage:***image* Initializes a new NXCursor object with *image*

Defining the Cursor

- **setImage:***newImage* Sets the NXImage object that supplies the cursor image
- **image** Returns the NXImage object that has the cursor image
- **setHotSpot:**(const NXPoint *)*spot* Sets the point on the cursor that's aligned with the mouse

Setting the Cursor

- **push** Makes the NXCursor the current cursor
- **pop** Restores the previous cursor
+ **pop** Restores the previous cursor

- **set**	Sets the NXCursor to be the current cursor
- **setOnMouseEntered:**(BOOL)*flag*	Determines whether **mouseEntered:** sets cursor
- **setOnMouseExited:**(BOOL)*flag*	Determines whether **mouseExited:** sets cursor
- **mouseEntered:**(NXEvent *)*theEvent*	Responds to a mouse-entered event
- **mouseExited:**(NXEvent *)*theEvent*	Responds to a mouse-exited event
+ **currentCursor**	Returns the current cursor

Archiving

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

NXCustomImageRep

Inherits From: NXImageRep : Object

Initializing a New NXCustomImageRep

- **initDrawMethod:**(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*	Reads the NXCustomImageRep from *stream*
- **write:**(NXTypedStream *)*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)*mode* Sets the link's update mode to *mode*
- (NXDataLinkUpdateMode)**updateMode** Returns the link's update mode
- **break** Breaks the link

NXDataLinkManager

Inherits From: Object

Initializing and Freeing a Link Manager

- **initWithDelegate:***anObject* Initializes and returns a newly allocated instance

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

Adding and Removing Links

- **addLink:**(NXDataLink *)*link* Adds the link *link* to the document
  at:(NXSelection *)*selection*
- **addLinkAsMarker:**(NXDataLink *)*link* Incorporates *link* into the document as a marker
  at:(NXSelection *)*selection*
- **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 is 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* Initializes the NXDataLinkPanel object
  style:(int)*aStyle*
  backing:(int)*bufferingType*
  buttonMask:(int)*mask*
  defer:(BOOL)*flag*

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*
Returns a List of NXEPSImageReps from EPS in *name*
zone:(NXZone *)*aZone*

+ (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* **inPath:**(const char *)*supplementPath*	Adds supplemental helpto the text displayed in the panel
- **free**	Frees the NXHelpPanel and its storage

Attaching Help to Objects

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

Setting Click-for-Help

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

Printing

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

Querying

-(NXAtom)**helpDirectory**	Returns the absolute path of the help directory
-(NXAtom) **helpFile**	Returns the path of the currently loaded help file

Showing Help

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

NXImage

Inherits From: Object

Initializing a New NXImage Instance

- **init**	Initializes the new NXImage without setting its size
- **initSize:**(const NXSize *)*aSize*	Initializes the new NXImage to the specified size

- **initFromSection:**(const char *)*name* — Initializes the new object from the data in *name* section
- **initFromFile:**(const char *)*filename* — Initializes the new NXImage from the data in *filename*
- **initFromPasteboard:**(Pasteboard *)*pasteboard* — Initializes the new NXImage from the data in *pasteboard*
- **initFromStream:**(NXStream *)*stream* — Initializes the new NXImage from the data in *stream*
- **initFromImage:**(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*

- Creates an empty representation to draw in
- (BOOL)**loadFromStream:**(NXStream *)*stream* 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* Composites the image to *aPoint*
  toPoint:(const NXPoint *)*aPoint*
- **composite:**(int)*op* Composites the *aRect* portion of the image to *aPoint*
  fromRect:(const NXRect *)*aRect*
  toPoint:(const NXPoint *)*aPoint*
- **dissolve:**(float)*delta* Composites the image using the **dissolve** operator
  toPoint:(const NXPoint *)*aPoint*
- **dissolve:**(float)*delta* Composites the image using the **dissolve** operator
  fromRect:(const NXRect *)*aRect*
  toPoint:(const NXPoint *)*aPoint*

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***e*
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

- **initFromPasteboard:**(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)*eventStatus* — Controls recording and playback
  soundStatus:(int)*soundStatus*
  eventStream:(NXStream *)*stream*
  soundfile:(const char *)*soundfile*
- **getEventStatus:**(int *)*eventStatusPtr* — Provides status information about the NXJournaler
  soundStatus:(int *)*soundStatusPtr*
  eventStream:(NXStream **)*streamPtr*
  soundfile:(char **)*soundfilePtr*
- **setRecordDevice:**(int)*device* — 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:***anObject* — Sets the NXJournaler's delegate
- **delegate** — Returns the NXJournaler's delegate

Implemented by the delegate

- **journalerDidEnd:***journaler* — Informs the delegate that the session terminated
- **journalerDidUserAbort:***journaler* — 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 *)*key*
  inTable:(const char *)*tableg*
  Returns whether key is a key to table

NXSpellChecker

Inherits From: Object

Making A Checker Available

+ **sharedInstance**
  Returns the NXSpellChecker to use

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

Managing The Spelling Panel

- **spellingPanel**
  Returns the NXSpellChecker's panel

- **accessoryView**
  Returns the spell panel's accessory view

- **setAccessoryView:***aView*
  Makes a view an accessory of the spell panel

Checking Spelling

- (BOOL) **checkSpelling:**(NXSpellCheckMode)*how*
  of:(id <NXReadOnlyTextStream, NXSelectRange>)*anObject*
  Starts the search for a misspelled word

- (BOOL) **checkSpelling:**(NXSpellCheckMode)*how*
  of:(id <NXReadOnlyTextStream, NXSelectRange>)*anObject*
  wordCount:(int *)*theCount*
  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* **caseSensitive:**(BOOL)*flag* — Returns YES if the word is in any open user dictionary

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

NXSplitView

Inherits From: View : Responder : Object

Initializing an NXSplitView

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

OpenPanel

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

Creating and Freeing an OpenPanel

+ **new** — Returns the shared OpenPanel object

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

- **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*
  endChar:(unsigned short)*theChar* — Updates the panel when the user finishes typing a page size
- (BOOL)**textWillChange:***textObject* — Updates the panel when a page size is typed
- **convertOldFactor:**(float *)*old*
  newFactor:(float *)*new* — Converts units for **pickedUnits:** method
- **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
- **initContent:**(const NXRect *)*contentRect* — Initializes the new Panel as specified
  style:(int)*aStyle*
  backing:(int)*bufferingType*
  buttonMask:(int)*mask*
  defer:(BOOL)*flag*

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 *)*name*	Returns the Pasteboard object named *name*
+ **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 *)*filename*	Creates a pasteboard with all types for *filename*
+ **newByFilteringData:**(NXData *)*data* **ofType:**(const char *)*type*	Creates a pasteboard with all types for *data*
+ **newByFilteringTypesInPasteboard:** (Pasteboard *)*pboard*	Creates a pasteboard with all types filterable from *pboard*
+ (NXAtom *)**typesFilterableTo:** (const char *)*type*	Returns all types *type* can be filtered to

Referring to a Pasteboard by Name

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

Writing Data

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

length:(int)*numBytes*

- **writeType:**(const char *)*dataType* — Writes stream data to the pasteboard server
  fromStream:(NXStream *)*stream*
- (BOOL)**writeFileContents:** — Writesdata from *filename* to the pasteboard server
  (const char *)*filename*

Discerning Types

- (const NXAtom *)**types** — Returns an array of the Pasteboard's data types
- (const char *)**findAvailableTypeFrom:** — Returns first type in *types* that matches a pasteboard type
  (const char *const *)*types*

Reading Data

- (int)**changeCount** — Returns the Pasteboard's change count
- **readType:**(const char *)*dataType* — Reads data from the pasteboard server
  data:(char **)*theData*
  length:(int *)*numBytes*
- (NXStream *)**readTypeToStream:** — Returns a stream to pasteboard data
  (const char *)dataType
- (char *)**readFileContentsType:** — Writes pasteboard data to a file
  (const char *)*type*
  toFile:(const char *)*filename*
- **deallocatePasteboardData:**(char *)*data* — Deallocates data received from the pasteboard
  length:(int)*numBytes*

Methods Implemented by the Owner

- **pasteboard:***sender* — Implemented to write promised data to *sender* as *type*
  provideData:(NXAtom)*type*
- **pasteboardChangedOwner:***sender* — 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* Sets the margins
  right:(NXCoord)*rightMargin*

top:(NXCoord)*topMargin*
bottom:(NXCoord)*bottomMargin*

- **getMarginLeft:**(NXCoord *)*leftMargin* — Returns the margins by reference
  right:(NXCoord *)*rightMargin*
  top:(NXCoord *)*topMargin*
  bottom:(NXCoord *)*bottomMargin*
- **setOrientation:**(char)*mode* — Sets the orientation as portrait or landscape
  andAdjust:(BOOL)*flag*
- (char)**orientation** — Returns the orientation is portrait or landscape
- **setPaperRect:**(const NXRect *)*aRect* — Sets the width and height of the paper
  andAdjust:(BOOL)*flag*
- (const NXRect *)**paperRect** — Returns the rectangle for the paper size
- **setPaperType:**(const char *)*type* — Sets the paper type
  andAdjust:(BOOL)*flag*
- (const char *)**paperType** — Returns the paper type

Page Range

- **setFirstPage:**(int)*anInt* — Sets the page number of first page to be printed
- (int)**firstPage** — Returns the page number of the first page to be printed
- **setLastPage:**(int)*anInt* — Sets the page number of last page to be printed
- (int)**lastPage** — Returns the page number of the last page to be printed
- **setAllPages:**(BOOL)*flag* — Sets whether all the pages are to be printed
- (BOOL)**isAllPages** — Returns whether all the pages are to be printed
- (int)**currentPage** — Returns the page number of the page being printed

Pagination and Scaling

- **setHorizPagination:**(int)*mode* — Sets the horizontal pagination mode
- (int)**horizPagination** — Returns the horizontal pagination mode
- **setVertPagination:**(int)*mode* — Sets the vertical pagination mode
- (int)**vertPagination** — Returns the vertical pagination mode

- **setScalingFactor:**(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* Sets the value of the given printing job feature
  toValue:(const char *)*value*
- (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* — Adds a View to the panel
- **accessoryView** — Returns the accessory View

Running the Panel

- (int)**runModal** — Displays the Print panel and begins its event loop
- **pickedButton:***sender* — Stops the event loop

Updating the Panel's Display

- **pickedAllPages:***sender* — Updates the panel when the user chooses all pages
- (BOOL)**textWillChange:***textObject* — Updates the panel when user types pages to print

Communicating with the PrintInfo Object

- **updateFromPrintInfo** — Reads PrintPanel's values from the PrintInfo object
- **finalWritePrintInfo** — Writes PrintPanel's values to the PrintInfo object

Responder

Inherits From: Object

Managing the NeXT Responder

- **setNextResponder:***aResponder* — Makes *aResponder* the receiver's next responder
- **nextResponder** — 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 *)*theEvent*	Returns NO to indicate *theEvent* isn't handled
- (BOOL)**tryToPerform:**(SEL)*anAction* **with:***anObject*	Aids in dispatching action messages

Forwarding Event Messages

- **mouseDown:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **rightMouseDown:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **mouseDragged:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **rightMouseDragged:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **mouseUp:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **rightMouseUp:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **mouseMoved:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **mouseEntered:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **mouseExited:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **keyDown:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **keyUp:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **flagsChanged:**(NXEvent *)*theEvent*	Passes the message to the receiver's next responder
- **noResponderFor:**(const char *)*eventType*	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* **file:**(const char *)name — Displays the SavePanel and begins its event loop
- (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* **endChar:**(unsigned short)*endChar* — Determines whether TAB or BACKTAB was pressed

- **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* Returns 1 if *filename1* precedes *filename2*, -1 in the
  compareFilenames:(const char *)*fileName1* opposite case, 0 if the two are equivalent
  :(const char *)*fileName2*
  checkCase:(BOOL)*flag*
- (BOOL)**panel:***sender* YES if *filename* can be saved in directory
  filterFile:(const char *)*filename*
  inDirectory:(const char *)*directory*
- (BOOL)**panelValidateFilenames:***sender* YES if the filename is acceptable to the delegate

Scroller

Inherits From: Control : View : Responder : Object

Initializing a Scroller

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

Laying out the Scroller

- (NXRect *)**calcRect:**(NXRect *)*aRect* Gets the rectangle that encloses *partCode*
  forPart:(int)*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* Responds to mouse-down events on the knob
- **trackScrollButtons:**(NXEvent *)*theEvent* Responds to mouse-down events on buttons

Archiving

- **awake** Ensures that Scroller's Bitmaps are created
- **read:**(NXTypedStream *)*stream* Reads the Scroller from the typed stream
- **write:**(NXTypedStream *)*stream* Writes the Scroller to the typed stream

ScrollView

Inherits From: View : Responder : Object

Initializing a ScrollView

- **initFrame:**(const NXRect *)*frameRect* Initializes a new ScrollView

Determining Component Sizes

- **getContentSize:**(NXSize *)*contentViewSize* Gets the content view's size
- **getDocVisibleRect:**(NXRect *)*aRect* Gets the visible portion of the document view

Laying Out the ScrollView

+ **getContentSize:**(NXSize *)*cSize* Gets the content view size for the given ScrollView size
  forFrameSize:(const NXSize *)*fSize*
  horizScroller:(BOOL)*hFlag*
  vertScroller:(BOOL)*vFlag*
  borderType:(int)*aType*
+ **getFrameSize:**(NXSize *)*fSize* 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* — Retiles the ScrollView after a **sizeTo::**
- **setHorizScrollerRequired:**(BOOL)*flag* — Makes space for a horizontal scroller
- **setVertScrollerRequired:**(BOOL)*flag* — Makes space for a vertical scroller
- **tile** — Retiles the scrollers and content view

Managing Component Views

- **setDocView:***aView* — Makes *aView* the ScrollView's document view
- **docView** — Returns the current document view
- **setHorizScroller:***anObject* — Sets the horizontal Scroller object
- **horizScroller** — Returns the horizontal Scroller
- **setVertScroller:***anObject* — Sets the vertical Scroller object
- **vertScroller** — Returns the vertical Scroller
- **reflectScroll:***cView* — Updates the Scrollers

Modifying Graphic Attributes

- **setBorderType:**(int)*aType* — Determines the border type of the ScrollView
- (int)**borderType** — Returns the border type
- **setBackgroundColor:**(NXColor)*color* — Sets the ScrollView's background color
- (NXColor) **backgroundColor** — Returns the ScrollView's background color
- **setBackgroundGray:**(float)*value* — Sets the ScrollView's background gray
- (float)**backgroundGray** — Returns the ScrollView's background gray

Setting Scrolling Behavior

- **setCopyOnScroll:**(BOOL)*flag* — Sets how newly exposed areas are redrawn

- **setDisplayOnScroll:**(BOOL)*flag* Sets how the doc view is displayed during scrolling
- **setDynamicScrolling:**(BOOL)*flag* Sets how the doc view is displayed during scrolling
- **setLineScroll:**(float)*value* Sets the amount to scroll when scrolling a line
- **setPageScroll:**(float)*value* 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 ªListItemº 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*	Sets whether SelectionCell is a leaf or a branch
- (BOOL)**isLeaf**	Returns whether the SelectionCell is a leaf or a branch
- (BOOL)**isOpaque**	Returns YES, since SelectionCells are opaque

Displaying

- **drawSelf:**(const NXRect *)*cellFrame* **inView:***aView*	Draws the SelectionCell in *cellFrame* within *aView*
- **drawInside:**(const NXRect *)*cellFrame* **inView:***aView*	Draws the inside of the SelectionCell in *aView*
- **highlight:**(const NXRect *)*cellFrame* **inView:***aView* **lit:**(BOOL)*flag*	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

- **initFrame:**(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*	Sets the SliderCell's value to *aDouble*
- (double)**doubleValue**	Returns the SliderCell's value as a **double**
- **setFloatValue:**(float)*aFloat*	Sets the SliderCell's value to *aFloat*
- (float)**floatValue**	Returns the SliderCell's value as a **float**
- **setIntValue:**(int)*anInt*	Sets the SliderCell's value to *anInt*
- (int)**intValue**	Returns SliderCell's value as an **int**
- **setStringValue:**(const char *)*aString*	Sets the SliderCell's value to a number represented by *aString*
- (const char *)**stringValue**	Returns the SliderCell's value as a string

Modifying Graphic Attributes

- **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
- (BOOL)**isOpaque**	Returns YES (SliderCells are always opaque)
- (int)**isVertical**	Returns 1 if vertical, 0 if horizontal, -1 if unknown

Displaying the SliderCell

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

Modifying Behavior

- **setEnabled:**(BOOL)*flag* — Sets whether the SliderCell reacts to events
- **setContinuous:**(BOOL)*flag* — Sets whether the Slider is continuous
- (BOOL)**isContinuous** — Returns whether the Slider is continuous
- **setAltIncrementValue:**(double)*incValue* — 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 *)*theEvent* **inRect:**(const NXRect *)*cellFrame* **ofView:***controlView* — Tracks the mouse
- (BOOL)**startTrackingAt:**(const NXPoint *)*startPoint* **inView:**controlView — Begins a tracking session
- (BOOL)**continueTracking:**(const NXPoint *)*lastPoint* **at:**(const NXPoint *)currentPoint **inView:***controlView* — Continues tracking the mouse
- **stopTracking:**(const NXPoint *)*lastPoint* — Ends the current tracking session

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

Archiving

- **read:**(NXTypedStream *)*stream* — Reads the SliderCell from *stream*
- **write:**(NXTypedStream *)*stream* — Writes the SliderCell to *stream*
- **awake** — 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* — Makes *aPort* the port messages will be sent to
- (port_t)**sendPort** — Returns the port the Speaker will send messages to
- **setReplyPort:**(port_t)*aPort* — Makes *aPort* the port where replies are received
- (port_t)**replyPort** — Returns the port where Speaker receives replies

Standard Remote Methods

- (int)**openFile:**(const char *)*fullPath* — Sends a remote message to open *fullPath* file
  ok:(int *)*flag*
- (int)**openTempFile:**(const char *)*fullPath* — Sends a remote message to open *fullPath* file
  ok:(int *)*flag*

Providing for Program Control

- (int)**msgCalc:**(int *)*flag* — Sends message to update the current window
- (int)**msgCopyAsType:**(const char *)*aType* — Sends message to copy selection as *aType* data
  ok:(int *)*flag*
- (int)**msgCutAsType:**(const char *)*aType* — Sends message to cut selection as *aType* data
  ok:(int *)*flag*
- (int)**msgDirectory:**(char *const *)*fullPath* — Sends message requesting the current directory
  ok:(int *)*flag*
- (int)**msgFile:**(char *const *)*fullPath* — Sends message requesting the current document
  ok:(int *)*flag*
- (int)**msgPaste:**(int *)*flag* — Sends message to paste data from pasteboard
- (int)**msgPosition:**(char *const *)*aString* — Sends message requesting selection information
  posType:(int *)*anInt*
  ok:(int *)*flag*
- (int)**msgPrint:**(const char *)*fullPath* — Sends message to print *fullPath* file
  ok:(int *)*flag*
- (int)**msgQuit:**(int *)*flag* — Sends remote message for application to quit
- (int)**msgSelection:**(char *const *)*bytes* — Sends message requesting the current selection
  length:(int *)*numBytes*

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*
ok:(int *)*flag* Sends an **openTempFile:ok:** remote message
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:***anObject* — Makes *anObject* the default Font object for Text
+ **getDefaultFont** — Returns the default Font object for Text
+ **excludeFromServicesMenu:**(BOOL)*flag* — Controls whether Text objects register for services
+ **registerDirective:**(const char *)*directive* **forClass:***class* — Associates an RTF control word with a class object
+ **initialize** — Performed automatically at startup

Initializing a New Text Object

- **initFrame:**(const NXRect *)*frameRect* — Initialize a new Text object
- **initFrame:**(const NXRect *)*frameRect* **text:**(const char *)*theText* **alignment:**(int)*mode* — Initialize a new Text object

Freeing a Text Object

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

Modifying the Frame Rectangle

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

Laying Out the Text

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

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

Setting, Reading, and Writing the Text

- **setText:**(const char *)*aString*	Replaces current text with *aString*

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

Allowing Multiple Fonts and Paragraph Styles

- **setMonoFont:**(BOOL)*flag* — Controls whether multiple fonts and parastyles are OK
- (BOOL)**isMonoFont** — Returns whether only one font and parastyle is permitted

Editing the Text

- **copy:***sender* — Copies selected text to the pasteboard
- **copyFont:***sender* — Copies selected text's font to the pasteboard
- **copyRuler:***sender* — Copies selected text's style to the pasteboard
- **paste:***sender* — Replaces selection with pasteboard's contents
- **pasteFont:***sender* — Replaces selection's font with pasteboard's contents
- **pasteRuler:***sender* — Replaces selection's style with pasteboard's contents
- **cut:***sender* — Deletes selected text; copies it to pasteboard
- **delete:***sender* — Deletes selected text
- **clear:***sender* — Deletes selected text
- **selectAll:***sender* — Makes receiver the first responder; selects all text
- **selectText:***sender* — Makes receiver the first responder; selects all text

Managing the Selection

- **subscript:***sender* — Subscripts the current selection
- **superscript:***sender* — Superscripts the current selection
- **unscript:***sender* — Removes sub/super script in the current selection
- **underline:***sender* — Toggles the underline attribute of text
- **showCaret** — Displays the previously hidden caret
- **hideCaret** — Removes the caret from the text display

- **setSelectable:**(BOOL)*flag*	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)*start* **:**(int)*end*	Selects text from *start* through *end*
- **getSel:**(NXSelPt *)*start* **:**(NXSelPt *)*end*	Gets *start* and *end* of the selection
- **replaceSel:**(const char *)*aString*	Replaces the selection with *aString*
- **replaceSel:**(const char *)*aString* **length:**(int)*length*	Replaces selection with *length* bytes of *aString*
- **replaceSel:**(const char *)*aString* **length:**(int)*length* **runs:**(NXRunArray *)*insertRuns*	Replaces selection with *length* bytes of *aString*
- **replaceSelWithRichText:**(NXStream *)*stream*	Replaces selection with RTF from *stream*
- **replaceSelWithRTFD:**(NXStream *)*stream*	Replaces selection with RTFD data from *stream*
- **scrollSelToVisible**	Brings the selection within the frame rectangle

Setting the Font

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

Checking Spelling

- **checkSpelling:***sender* — Searches for a misspelled word in the text
- **showGuessPanel:***sender* — Displays panel suggesting spelling corrections

Managing the Ruler

- **toggleRuler:***sender* — Controls the display of the ruler
- (NXColor)**isRulerVisible** — Returns whether the ruler is visible in the superview

Finding Text

- (BOOL)**findText:**(const char *)*string* — Searches for *string* in the text, starting at the insertion point
  ignoreCase:(BOOL)*ignoreCaseFlag*
  backwards:(BOOL)*backwardsFlag*
  wrap:(BOOL)*wrapFlag*

Modifying Graphic Attributes

- **setBackgroundGray:**(float)*value* — Sets the gray value of the text background
- (float)**backgroundGray** — Returns the gray value of the text background
- **setBackgroundColor:**(NXColor)*color* — Sets background color of the text
- (NXColor)**backgroundColor** — Returns the background color of the text
- **setSelGray:**(float)*value* — Sets the gray value of the selected text
- (float)**selGray** — Returns the gray value of the selected text
- (float)**runGray:**(NXRun *)*run* — Returns the gray value for the specified text run
- **setSelColor:**(NXColor)*color* — Sets the color of the selected text
- (NXColor)**selColor** — Returns the color of the selected text
- (NXColor)**runColor:**(NXRun *)*run* — Returns the color of the specified text run
- **setTextGray:**(float)*value* — Sets the gray value of the entire text
- (float)**textGray** — Returns the gray value of the entire text
- **setTextColor:**(NXColor)*color* — 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 *)*theEvent*	Responds to mouse-down events
- **keyDown:**(NXEvent *)*theEvent*	Responds to key-down events
- **moveCaret:**(unsigned short)*theKey*	Moves the caret in response to arrow keys

Displaying Graphics within the Text

+ **registerDirective:**(const char *)*directive* **forClass:***class*	Associates an RTF control word with a class object
- **replaceSelWithCell:***cell*	Replaces selection with image provided by *cell*
- **replaceSelWithView:***view*	Unimplemented
- **setLocation:**(NXPoint *)*origin* **ofCell:***cell*	Sets origin of *cell*
- **getLocation:**(NXPoint *)*origin* **ofCell:***cell*	Places coordinates of graphic object into *origin*
- **getLocation:**(NXPoint *)*origin* **ofView:***view*	Unimplemented
- **setGraphicsImportEnabled:**(BOOL)*flag*	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)*flag*	Controls whether Text objects use services menu
- **validRequestorForSendType:**(NXAtom)*sendType* **andReturnType:**(NXAtom)*returnType*	Determines which Service menu items are enabled
- **readSelectionFromPasteboard:***pboard*	Replaces selection with data from pboard
- (BOOL)**writeSelectionToPasteboard:***pboard*	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*	Reads the Text object from the typed stream
- **write:**(NXTypedStream *)*stream*	Writes the Text object to the typed stream

Assigning a Delegate

- **setDelegate:***anObject*	Makes *anObject* the Text object's delegate
- **delegate**	Returns the Text object's delegate

Implemented by the Delegate

- **textWillResize:***sender*	Informs delegate of impending size change
- **textDidResize:***sender* **oldBounds:**(const NXRect *)*oldBounds* **invalid:**(NXRect *)*invalidRect*	Reports size change to delegate
- (BOOL)**textWillChange:***sender*	Informs delegate of impending text change
- **textDidChange:***sender*	Alerts delegate to change in text
- (BOOL)**textWillEnd:***sender*	Warns of impending loss of first responder status
- **textDidEnd:***sender* **endChar:**(unsigned short)*whyEnd*	Reports to delegate loss of first responder status
- **textDidGetKeys:***sender* **isEmpty:**(BOOL)*flag*	Informs delegate of each text change
- **textWillSetSel:***sender* **toFont:***font*	Lets delegate intercede in the updating of the Font panel
- **textWillConvert:***sender* **fromFont:***from* **toFont:***to*	Lets delegate intercede in selection's font change
- **textWillStartReadingRichText:***sender*	Informs delegate that Text object will read RTF data
- **textWillFinishReadingRichText:***sender*	Informs delegate that Text finished reading RTF data
- **textWillWrite:***sender* **paperSize:**(NXSize *)*paperSize*	Lets the delegate specify paper size

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

Implemented by an Embedded Graphic Object

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

TextField

Inherits From: Control : View : Responder : Object

Initializing the TextField Class

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

Inializing a new TextField

- **initFrame:**(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)**isSelectable** — 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* **isEmpty:**(BOOL)*flag* — Responds to a message from the field editor

- **textDidChange:***textObject* Responds to a message from the field editor
- (BOOL)**textWillEnd:***textObject* Responds to a message from the field editor
- **textDidEnd:***textObject* Responds to a message from the field editor
  endChar:(unsigned short)*whyEnd*

Modifying Graphic Attributes

- **setTextColor:**(NXColor)*aColor* Sets the color of the TextField's text to *aColor*
- (NXColor)**textColor** Returns the color of the TextField's text
- **setTextGray:**(float)*value* Sets the gray of the TextField's text to *value*
- (float)**textGray** Returns the gray of the TextField's text
- **setBackgroundColor:**(NXColor)*aColor* Sets the color of the background to *aColor*
- (NXColor)**backgroundColor** Returns the color of the background
- **setBackgroundGray:**(float)*value* Sets the gray of the background to *value*
- (float)**backgroundGray** Returns the gray of the background
- **setBackgroundTransparent:**(BOOL)*flag* Sets whether the TextField background is transparent
- (BOOL)**isBackgroundTransparent** Returns whether the TextField background is transparent
- **setBezeled:**(BOOL)*flag* Sets whether the TextField has a bezeled border
- (BOOL)**isBezeled** Returns whether the TextField has a bezeled border
- **setBordered:**(BOOL)*flag* Sets whether the TextField has a plain border
- (BOOL)**isBordered** Returns whether the TextField has a plain border

Target and Action

- **setErrorAction:**(SEL)*aSelector* Sets the action method sent for an invalid value entered
- (SEL)**errorAction** 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** — Returns YES if text is editable or selectable
- **mouseDown:**(NXEvent *)*theEvent* — Responds to a mouse-down event

Archiving

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

TextFieldCell

Inherits From: ActionCell : Cell : Object

Initializing a new TextFieldCell

- **init** — Initializes a new TextFieldCell with text ªFieldº
- **initTextCell:**(const char *)*aString* — 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* — Sets the color of the text to *aColor*
- (NXColor)**textColor** — Returns the color of the text
- **setTextGray:**(float)*value* — Sets the gray of the text to *value*
- (float)**textGray** — Returns the gray of the text

- **setBackgroundColor:**(NXColor)*aColor* — Sets the color of the background to *aColor*
- (NXColor)**backgroundColor** — Returns the color of the background
- **setBackgroundGray:**(float)*value* — Sets the gray of the background to *value*
- (float)**backgroundGray** — Returns the gray of the background
- **setBackgroundTransparent:**(BOOL)*flag* — Sets whether the background is transparent
- (BOOL)**isBackgroundTransparent** — Returns whether the background is transparent
- **setTextAttributes:***textObject* — Sets the gray values of the background and text to those of *textObject*
- **setBezeled:**(BOOL)*flag* — Sets whether TextFieldCell has a bezeled border
- (BOOL)**isOpaque** — Returns whether the cell is opaque

Displaying

- **drawSelf:**(const NXRect *)*cellFrame*
  inView:*controlView* — Draws the TextFieldCell
- **drawInside:**(const NXRect *)*cellFrame*
  inView:*controlView* — 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* — Reads the TextFieldCell from *stream*
- **write:**(NXTypedStream *)*stream* — Writes the TextFieldCell to *stream*

View

Inherits From: Responder : Object

Initializing and Freeing View Objects

- **initFrame:**(const NXRect *)*frameRect*	Initializes a new View object
- **init**	Initializes a new View object
- **free**	Deallocates the View and its subviews

Managing the View Hierarchy

- **addSubview:***aView*	Makes *aView* a subview of the receiving View
- **addSubview:***aView* **:**(int)*place* **relativeTo:***otherView*	Makes *aView* a subview of the receiving View
- **findAncestorSharedWith:***aView*	Returns the ancestor shared by *aView* and the receiver
- (BOOL)**isDescendantOf:***aView*	Returns whether *aView* is an ancestor of the receiver
- **opaqueAncestor**	Returns the receiver's nearest opaque ancestor
- **removeFromSuperview**	Removes the receiver from the view hierarchy
- **replaceSubview:***oldView* **with:***newView*	Replaces *oldView* with *newView*
- **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:***newWindow*	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 *)*theRect*	Gets the View's frame rectangle
- **moveBy:**(NXCoord)*deltaX* **:**(NXCoord)*deltaY*	Moves the View by *deltaX* and *deltaHeight*
- **moveTo:**(NXCoord)*x* **:**(NXCoord)*y*	Moves the View to (*x*, *y*)

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

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 *)*theRect* 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)*angle* Rotates the View's coordinate system *by angle*
- **setDrawRotation:**(NXCoord)*angle* Rotates the View's coordinate system to *angle*
- **scale:**(NXCoord)*x* **:**(NXCoord)*y* Scales the View's coordinate system
- **setDrawSize:**(NXCoord)*width* **:**(NXCoord)*height* Resizes the View's coordinate system to *width* and *height*
- **translate:**(NXCoord)*x* **:**(NXCoord)*y* Shifts the View's coordinate system to (*x*, *y*)
- **setDrawOrigin:**(NXCoord)*x* **:**(NXCoord)*y* Sets the View's origin to (*x*, *y*)
- **setFlipped:**(BOOL)*flag* Sets whether polarity of y-axis is reversed

Converting Coordinates

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

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

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

Scrolling

- **adjustScroll:**(NXRect *)*newVisible* Lets the View adjust the visible rectangle
- **autoscroll:**(NXEvent *)*theEvent* Scrolls in response to a mouse-dragged event
- (BOOL)**calcUpdateRects:**(NXRect *)*rects*
  :(int *)*rectCount*
  :(NXRect *)*enclRect*
  :(NXRect *)*goodRect*
  Calculates the area to be redisplayed
- **invalidate:**(const NXRect *)*rects* **:**(int)*rectCount*
  Marks parts of the View as needing to be redrawn
- **scrollPoint:**(const NXPoint *)*aPoint* Aligns *aPoint* with the content view's origin
- **scrollRect:**(const NXRect *)*aRect*
  by:(const NXPoint *)*delta*
  Shifts the rectangle by *delta*
- **scrollRectToVisible:**(const NXRect *)*aRect* 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 *)*aRect*
  cursor:*anObj*
  Adds a cursor rectangle to the View
- **discardCursorRects** Removes all cursor rectangles in the View
- **removeCursorRect:**(const NXRect *)*aRect*
  cursor:*anObj*
  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* — Instigates an image-dragging session
  at:(NXPoint *)*location*
  offset:(NXPoint *)*initialOffset*
  event:(NXEvent *)*event*
  pasteboard:(Pasteboard *)*pboard*
  source:*sourceObject*
  slideBack:(BOOL)*slideFlag*
- **dragFile:**(const char *)*filename* — Instigates a file-dragging session
  fromRect:(NXRect *)*rect*

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* — Faxes the View and its subviews
  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*
- **copyPSCodeInside:**(const NXRect *)*rect* — Generates PostScript code for the rectangle
  to:(NXStream *)*stream*
- **writePSCodeInside:**(const NXRect *)*rect* — Places PostScript code for the rectangle on the pasteboard
  to:*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* — Returns whether the View paginates itself
  last:(int *)*lastPageNum*
- (BOOL)**getRect:**(NXRect *)*theRect* — Provides how much of the View will print on *page*
  forPage:(int)*page*
- **placePrintRect:**(const NXRect *)*aRect* — Locates the printing rectangle on the page
  offset:(NXPoint *)*location*
- (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 *)*boundingBox* **creationDate:**(const char *)dateCreated **createdBy:**(const char *)*anApplication* **fonts:**(const char *)*fontNames* **forWhom:**(const char *)*user* **pages:**(int)*numPages* **title:**(const char *)*aTitle*	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 *)*newRight* **left:**(float)*oldLeft* **right:**(float)*oldRight* **limit:**(float)*rightLimit*	Assists automatic pagination of the View
- **adjustPageHeightNew:**(float *)*newBottom* **top:**(float)*oldTop* **bottom:**(float)*oldBottom* **limit:**(float)*bottomLimit*	Assists automatic pagination of the View
- **beginPage:**(int)*ordinalNum* **label:**(const char *)*aString* **bBox:**(const NXRect *)*pageRect* **fonts:**(const char *)*fontNames*	Writes a page separator
- **beginPageSetupRect:**(const NXRect *)*aRect* **placement:**(const NXPoint *)*location*	Writes the beginning of a page setup section
- **drawSheetBorder:**(float)*width* **:**(float)*height*	Allows you to draw a sheet border
- **drawPageBorder:**(float)*width* **:**(float)*height*	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 *)*stream* Reads the View from the typed stream
- **write:**(NXTypedStream *)*stream* 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
- **initContent:**(const NXRect *)*contentRect* Initializes the new Window object as specified
  style:(int)*aStyle*
  backing:(int)*bufferingType*
  buttonMask:(int)*mask*
  defer:(BOOL)*flag*
- **initContent:**(const NXRect *)*contentRect* Initializes the new Window object for *screen* as specified
  style:(int)*aStyle*
  backing:(int)*bufferingType*
  buttonMask:(int)*mask*
  defer:(BOOL)*flag*
  screen:(const NXScreen *)*aScreen*

Freeing a Window Object

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

Computing Frame and Content Rectangles

+ **getFrameRect:**(NXRect *)*frame* **forContentRect:**(const NXRect *)*content* **style:**(int)*aStyle*	Gets frame rectangle for given content rectangle
+ **getContentRect:**(NXRect *)*content* **forFrameRect:**(const NXRect *)*frame* **style:**(int)*aStyle*	Gets content rectangle for given frame rectangle
+ (NXCoord)**minFrameWidth:**(const char *)*aTitle* **forStyle:**(int)aStyle **buttonMask:**(int)*aMask*	Returns minimum frame width needed for *aTitle*

Accessing the Frame Rectangle

- **getFrame:**(NXRect *)*theRect*	Gets the Window's frame rectangle
- **getFrame:**(NXRect *)*theRect* **andScreen:**(const NXScreen **)*theScreen*	Gets the Window's frame rectangle and screen
- (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* Moves the Window to (*x*, *y*) relative to *aScreen*
  :(NXCoord)*y*
  screen:(const NXScreen *)*aScreen*
- **moveTopLeftTo:**(NXCoord)*x* **:**(NXCoord)*y* Moves top left corner of the Window to (*x*, *y*)
- **moveTopLeftTo:**(NXCoord)*x* Moves top left corner of the Window relative to *aScreen*
  :(NXCoord)*y*
  screen:(const NXScreen *)*aScreen*
- **dragFrom:**(float)*x* **:**(float)*y* **eventNum:**(int)*num* Lets user drag the Window from (*x*, *y*)
- **constrainFrameRect:**(NXRect *)*frameRect* Constrains the Window to *screen*
  toScreen:(const NXScreen *)*screen*
- **placeWindow:**(const NXRect *)*frameRect* Resizes the Window to new frame rectangle
- **placeWindow:**(const NXRect *)*frameRect* Resizes the Window relative to *aScreen*
  screen:(const NXScreen *)*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:***sender* Puts the Window at the front of its tier
- **orderBack:***sender* Puts the Window at the back of its tier
- **orderOut:***sender* Removes the Window from the screen list
- **orderWindow:**(int)*place* **relativeTo:**(int)*otherWin*
  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)*flag* Sets whether deactivization hides the Window
- (BOOL)**doesHideOnDeactivate** Returns whether deactivization hides the Window

Converting Coordinates

- **convertBaseToScreen:**(NXPoint *)*aPoint* Converts point from base to screen coordinates
- **convertScreenToBase:**(NXPoint *)*aPoint* 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)*flag* Sets whether Window's Views should optimize drawing
- **update** Implemented by subclasses (see Menu)

Screens and Window Depths

- (const NXScreen *)**screen**	Returns the screen that (most of) the Window is on
- (const NXScreen *)**bestScreen**	Returns the deepest screen that the Window is on
+ (NXWindowDepth)**defaultDepthLimit**	Returns the maximum depth for the current context
- **setDepthLimit:**(NXWindowDepth)***limit***	Sets the Window's depth limit to *limit*
- (NXWindowDepth)**depthLimit**	Returns the Window's depth limit
- **setDynamicDepthLimit:**(BOOL)***flag***	Sets whether the depth limit will depend on the screen
- (BOOL)**hasDynamicDepthLimit**	Returns whether the depth limit depends on the screen
- (BOOL)**canStoreColor**	Returns whether Window is deep enough to store colors

Cursor Management

- **addCursorRect:**(const NXRect *)*aRect* **cursor:***anObject* **forView:***aView*	Adds a new cursor rectangle to the Window
- **removeCursorRect:**(const NXRect *)*aRect* **cursor:***anObject* **forView:***aView*	Removes a cursor rectangle from the Window
- **invalidateCursorRectsForView:***aView*	Marks cursor rectangles invalid for *aView*
- **disableCursorRects**	Disables all cursor rectangles in the Window
- **enableCursorRects**	Enables cursor rectangles in the Window
- **discardCursorRects**	Removes all cursor rectangles in the Window
- **resetCursorRects**	Resets cursor rectangles for the Window

Handling User Actions and Events

- **close**	Closes the Window
- **performClose:***sender*	Simulates user clicking the close button
- **miniaturize:***sender*	Hides the Window and displays its miniwindow

- **performMiniaturize:***sender* — Simulates user clicking the miniaturize button
- **deminiaturize:***sender* — Hides the miniwindow and redisplays the Window
- **setDocEdited:**(BOOL)*flag* — Sets whether the Window's document has been edited
- (BOOL)**isDocEdited** — Returns whether Window's document has been edited
- **windowExposed:**(NXEvent *)*theEvent* — Redisplays exposed part of the Window
- **windowMoved:**(NXEvent *)*theEvent* — Updates the frame rectangle
- **screenChanged:**(NXEvent *)*theEvent* — Adjusts depth limit of Windows with a dynamic limit

Setting the Event Mask

- (int)**setEventMask:**(int)*newMask* — Sets the Window's event mask to *newMask*
- (int)**addToEventMask:**(int)*newEvents* — Adds *newEvents* to the event mask
- (int)**removeFromEventMask:**(int)*oldEvents* — Removes *oldEvents* from the event mask
- (int)**eventMask** — Returns the Window's event mask

Aiding Event Handling

- **getMouseLocation:**(NXPoint *)*thePoint* — Provides current location of the cursor
- **setTrackingRect:**(const NXRect *)*aRect* — Sets a tracking rectangle within the Window
  inside:(BOOL)*insideFlag*
  owner:*anObject*
  tag:(int)*trackNum*
  left:(BOOL)*leftDown*
  right:(BOOL)*rightDown*
- **discardTrackingRect:**(int)*trackNum* — Clears tracking rectangle within the Window
- **makeFirstResponder:***aResponder* — Makes *aResponder* the first responder
- **firstResponder** — Returns the first responder
- **sendEvent:**(NXEvent *)*theEvent* — Dispatches mouse and keyboard events
- **rightMouseDown:**(NXEvent *)*theEvent* — Handles right mouse-down events
- (BOOL)**commandKey:**(NXEvent *)*theEvent* — Handles Command key-down events
- (BOOL)**tryToPerform:**(SEL)*anAction* — 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 *)*filename*	Spools *filename* to the printer
- **copyPSCodeInside:**(const NXRect *)*rect* **to:**(NXStream *)*stream*	Writes PostScript code for the rectangle to *stream*
- (BOOL)**knowsPagesFirst:**(int *)*firstPageNum* **last:**(int *)*lastPageNum*	Returns whether the Window paginates itself
- (BOOL)**getRect:**(NXRect *)*theRect* **forPage:**(int)*page*	Provides how much of the Window fits 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 next page
- (float)**widthAdjustLimit**	Returns how much of a page can go on next page
- **beginPSOutput**	Initializes the printing environment
- **beginPrologueBBox:**(const NXRect *)*boundingBox* **creationDate:**(const char *)dateCreated **createdBy:**(const char *)*anApplication* **fonts:**(const char *)*fontNames* **forWhom:**(const char *)*user* **pages:**(int)*numPages* **title:**(const char *)*aTitle*	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)*ordinalNum* **label:**(const char *)*aString* **bBox:**(const NXRect *)*pageRect* **fonts:**(const char *)*fontNames*	Writes a page separator
- **beginPageSetupRect:**(const NXRect *)*aRect* **placement:**(const NXPoint *)*location*	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** — Writes the end of the trailer
- **endPSOutput** — Finishes the printing job

Archiving

- **read:**(NXTypedStream *)*stream* — Reads the Window from the typed stream
- **write:**(NXTypedStream *)*stream* — Writes the Window to the typed stream
- **awake** — Redisplays and reinitializes the Window

Assigning a Delegate

- **setDelegate:***anObject* — Makes *anObject* the Window's delegate
- **delegate** — Returns the Window's delegate

Implemented by the Delegate

- **windowWillClose:***sender* — Notifies delegate that the Window is about to close
- **windowWillReturnFieldEditor:***sender* **toObject:***client* — Lets delegate provide another Text object
- **windowWillResize:***sender* **toSize:**(NXSize *)*frameSize* — Lets delegate constrain resizing
- **windowDidResize:***sender* — Notifies delegate that the Window was resized
- **windowDidExpose:***sender* — Notifies delegate that the Window was exposed
- **windowWillMove:***sender* — Notifies delegate that the Window will move
- **windowDidMove:***sender* — Notifies delegate that the Window did move
- **windowDidChangeScreen:***sender* — Notifies delegate that the Window changed screens
- **windowDidBecomeKey:***sender* — Notifies delegate that the Window is the key window
- **windowDidResignKey:***sender* — Notifies delegate that the Window isn't the key window
- **windowDidBecomeMain:***sender* — Notifies delegate that the Window is the main window
- **windowDidResignMain:***sender* — Notifies delegate that the Window isn't the main window
- **windowWillMiniaturize:***sender* — Notifies delegate that the Window will miniaturized

toMiniwindow:*miniwindow*

- windowDidMiniaturize:*sender*	Notifies delegate that	the Window was miniaturized
- windowDidDeminiaturize:*sender*	Notifies delegate that	the Window was restored to screen
- windowDidUpdate:*sender*	Notifies delegate that	the Window was updated