Application Kit Functions

Rectangle Drawing Functions

Optimize Drawing

void **NSEraseRect(**NSRect *aRect***)**

Erases the rectangle by filling it with white. (This does not alter the current drawing color.)

void **NSHighlightRect(**NSRect *aRect***)**

Highlights or unhighlights a rectangle by switching light gray for white and vice versa, when drawing on the screen. If not drawing to the screen, the rectangle is filled with light gray.

void **NSRectClip(**NSRect *aRect***)**

Intersects the current clipping path with the rectangle *aRect*, to determine a new clipping path.

void **NSRectClipList(**const NSRect **rects*, int *count***)**

Takes an array of *count* number of rectangles and intersects the current clipping path with each of them. Thus, the new clipping path is the graphic intersection of all the rectangles and the original clipping path.

void **NSRectFill(**NSRect *aRect***)**

Fills the rectangle referred to by *aRect* with the current color.

void **NSRectFillList(**const NSRect **rects*, int *count***)**

Fills an array of *count* rectangles with the current color.

void **NSRectFillListWithGrays(**const NSRect **rects*, const float **grays*, int *count***)**

Fills each rectangle in the array *rects* with the gray whose value is stored at the corresponding location in the array *grays*. Both arrays must be count elements long. Avoid rectangles that overlap, because the order in which they'll be filled can't be guaranteed.

Draw a Bordered Rectangle

void **NSDrawButton(**NSRect *aRect*, NSRect *clipRect***)**

Draws the bordered light gray rectangle whose appearance signifies a button in the OpenStep user interface. *aRect* is the bounds for the button, but only the area where *aRect* intersects *clipRect* is drawn.

void **NSDrawGrayBezel(**NSRect *aRect*, NSRect *clipRect***)**

Draws a bordered light gray rectangle with the appearance of a pushed-in button, clipped by intersecting with *clipRect*.

void **NSDrawGroove(**NSRect *aRect*, NSRect *clipRect***)**

Draws a light gray rectangle whose border is a groove, giving the appearance of a typical box in the OpenStep user interface.

NSRect **NSDrawTiledRects(**NSRect *boundsRect*, NSRect *clipRect*, const NSRectEdge **sides*, const float **grays*,

Draws an unfilled rectangle, clipped by *clipRect*, whose border is defined by the parallel arrays *sides* and *grays*, both of length *count*. Each element of *sides* specifies an edge of the rectangle, which is drawn with a width of

int *count*)

1.0 using the corresponding gray level from *grays*. If the *edges* array contains recurrences of the same edge, each is inset within the previous edge.

void **NSDrawWhiteBezel(**NSRect *aRect*, NSRect *clipRect*)

Draws a white rectangle with a bezeled border. Only the area that intersects *clipRect* is drawn.

void **NSFrameRect(**NSRect *aRect*)

Draws a frame of width 1.0 around the inside of a rectangle, using the current color.

void **NSFrameRectWithWidth(**NSRect *aRect*, float *frameWidth*)

Draws a frame of width *frameWidth* around the inside of a rectangle, using the current color.

Color Functions

Get Information About Color Space and Window Depth

const NSWindowDepth ***NSAvailableWindowDepths(**void**)**

Returns a zero-terminated list of available window depths.

NSWindowDepth **NSBestDepth(**NSString **colorSpace*, int *bitsPerSample*, int *bitsPerPixel*, BOOL *planar*, BOOL **exactMatch*)

Returns a window depth deep enough for the given number of colors, bits per sample, bits per pixel, and if planar. Upon return, the variable pointed to by *exactMatch* is YES if the window depth can accommodate all of the values given for all of the parameters, NO if not.

int **NSBitsPerPixelFromDepth(**NSWindowDepth *depth*)

Returns the number of bits per pixel for the given window depth.

int **NSBitsPerSampleFromDepth(**NSWindowDepth *depth*)

Returns the number of bits per sample (bits per pixel in each color component) for the given window depth.

NSString ***NSColorSpaceFromDepth(**NSWindowDepth *depth*)

Returns the name of the color space that matches the given window depth.

int **NSNumberOfColorComponents(**NSString **colorSpaceName*)

Returns the number of color components in the named color space.

BOOL **NSPlanarFromDepth(**NSWindowDepth *depth*)

Returns YES if the given window depth is planar, NO if not.

Read the Color at a Screen Position

NSColor ***NSReadPixel(**NSPoint *location*)

Returns the color of the pixel at the given location, which must be specified in the current view's coordinate system.

Text Functions

Filter Characters Entered into a Text Object

unsigned short **NSEditorFilter(**unsigned short *theChar*, int *flags,* NSStringEncoding *theEncoding*)

Identical to **NSFieldFilter()** except that it passes on values corresponding to Return, Tab, and Shift-Tab directly to

the NSText object.

unsigned short **NSFieldFilter(**unsigned short *theChar*,
int *flags*,
NSStringEncoding *theEncoding***)**

Checks each character the user types into an NSText object's text, allowing the user to move the selection among text fields by pressing Return, Tab, or Shift-Tab. Alphanumeric characters are passed to the NSText object for display. The function returns either the ASCII value of the character typed, 0 (for illegal characters or ones entered while a Command key is held down), or a constant that the Text object interprets as a movement command.

Calculate or Draw a Line of Text (in Text Object)

int **NSDrawALine(**id *self*,
NSLayInfo **layInfo***)*

Draws a line of text, using the global variables set by **NSScanALine()**. The return value has no significance.

int **NSScanALine(**id *self*,
NSLayInfo **layInfo***)*

Determines the placement of characters in a line of text. *self* refers to the NSText object calling the function, and **layInfo* is an NSLayInfo struct. The function returns 1 if a word's length exceeds the width of a line and the NSText's charWrap instance variable is NO. Otherwise, it returns 0.

Calculate Font Ascender, Descender, and Line Height (in Text Object)

void **NSTextFontInfo(**id *fid*,
float **ascender*, float **descender*,
float **lineHeight***)*

Calculates, and returns by reference, the ascender, descender, and line height values for the NSFont given by *font*.

Access Text Object's Word Tables

NSData * **NSDataWithWordTable(**const unsigned char **smartLeft*,
const unsigned char **smartRight*,
const unsigned char **charClasses*,
const NSFSM **wrapBreaks*,
int *wrapBreaksCount*,
const NSFSM **clickBreaks*,
int *clickBreaksCount*,
BOOL *charWrap***)**

Given pointers to word table structures, records the structures in the returned NSData object. The arguments are similar to those of **NSReadWordTable()**.

void **NSReadWordTable(**NSZone **zone*,
NSData **data*,
unsigned char ***smartLeft*,
unsigned char ***smartRight*,
unsigned char ***charClasses*,
NSFSM ***wrapBreaks*,
int **wrapBreaksCount*,
NSFSM ***clickBreaks*,
int **clickBreaksCount*,
BOOL **charWrap***)**

Given *data*, creates word tables in the memory zone specified by *zone*, returning (in the subsequent arguments) pointers to the various tables. The integer pointer arguments return the length of the preceding array, and *charWrap* indicates whether words whose length exceeds the NSText object's line length should be wrapped on a character-by-character basis.

Array Allocation Functions for Use by the NSText Class

NSTextChunk ***NSChunkCopy(**NSTextChunk **pc*,

Copies the array *pc* to the array *dpc* and returns a pointer to

NSTextChunk ***dpc**)

the copy.

NSTextChunk ***NSChunkGrow(**NSTextChunk ***pc**, int *newUsed*)

Increases the array identified by the pointer *pc* to a size of *newUsed* bytes.

NSTextChunk ***NSChunkMalloc(**int *growBy*, int *initUsed*)

Allocates initial memory for a structure whose first field is an **NSTextChunk** structure and whose subsequent field is a variable-sized array. The amount of memory allocated is equal to *initUsed*. If *initUsed* is 0, *growBy* bytes are allocated. *growBy* specifies how much memory should be allocated when the chunk grows.

NSTextChunk ***NSChunkRealloc(**NSTextChunk ***pc**)

Increases the amount of memory available for the array identified by the pointer *pc*, as determined by the array's NSTextChunk.

NSTextChunk ***NSChunkZoneCopy(**NSTextChunk ***pc**, NSTextChunk ***dpc**, NSZone ***zone**)

Like **NSChunkCopy()**, but uses the specified zone of memory.

NSTextChunk ***NSChunkZoneGrow(**NSTextChunk ***pc**, int *newUsed*, NSZone ***zone**)

Like **NSChunkGrow()**, but uses the specified zone of memory.

NSTextChunk ***NSChunkZoneMalloc(**int *growBy*, int *initUsed*, NSZone ***zone**)

Like **NSChunkMalloc()**, but uses the specified zone of memory.

NSTextChunk ***NSChunkZoneRealloc(**NSTextChunk ***pc**, NSZone ***zone**)

Like **NSChunkRealloc()**, but uses the specified zone of memory.

Imaging Functions

Copy an image

void **NSCopyBitmapFromGState(**int *srcGstate*, NSRect *srcRect*, NSRect *destRect*)

Copies the pixels in the rectangle *srcRect* to the rectangle *destRect*. The source rectangle is defined in the graphics state designated by *srcGstate*, and the destination is defined in the current graphics state.

void **NSCopyBits(**int *srcGstate*, NSRect *srcRect*, NSPoint *destPoint*)

Copies the pixels in the rectangle *srcRect* to the location *destPoint*. The source rectangle is defined in the current graphics state if *srcGstate* is NSNullObject; otherwise, in the graphics state designated by *srcGstate*. The *destPoint* destination is defined in the current graphics state.

Render Bitmap Images

void **NSDrawBitmap**(NSRect *rect*, int *pixelsWide*, int *pixelsHigh*, int *bitsPerSample*, int *samplesPerPixel*, int *bitsPerPixel*, int *bytesPerRow*, BOOL *isPlanar*,

Renders an image from a bitmap. *rect* is the rectangle in which the image is drawn, and *data* is the bitmap data, stored in up to 5 channels unless *isPlanar* is NO (in which case the channels are interleaved in a single array).

BOOL *hasAlpha*,
NSString **colorSpaceName*,
const unsigned char *const *data*[5]**)**

Attention Panel Functions

Create an Attention Panel without Running It Yet

id **NSGetAlertPanel(**NSString **title*,
NSString **msg*,
NSString **defaultButton*,
NSString **alternateButton*,
NSString **otherButton, ...***)**

Returns an NSPanel object that you can use in a modal session. Unlike **NSRunAlertPanel()**, no button is displayed if *defaultButton* is NULL.

Create and Run an Attention Panel

int **NSRunAlertPanel(**NSString **title*,
NSString **msg,*
NSString **defaultButton,*
NSString **alternateButton,*
NSString **otherButton, ...***)**

Creates an attention panel that alerts the user to some consequence of a requested action, and runs the panel in a modal event loop. *title* is the panel's title (by default, ªAlertº); *msg* is the **printf()**-style message that's displayed in the panel; *defaultButton* (by default, ªOKº) is the title for the main button, also activated by Return; *alternateButton* and *otherButton* give two more choices, which are displayed only if the corresponding argument isn't NULL. The trailing arguments are a variable number of **printf()**-style arguments to msg.

int **NSRunLocalizedAlertPanel(**NSString **table*,
NSString **title*,
NSString **msg,*
NSString **defaultButton,*
NSString **alternateButton,*
NSString **otherButton, ...***)**

Similar to **NSRunAlertPanel()**, but preferred, as it makes use of OpenStep's localization feature for languages of different countries.

Release an Attention Panel

void **NSReleaseAlertPanel(**id *panel***)**

Releases the specified alert panel.

Services Menu Functions

Determine Whether an Item Is Included in Services Menus

int **NSSetShowsServicesMenuItem(**NSString **item*,
BOOL *showService***)**

Determines (based on the value of *showService*) whether the *item* command will be included in other applications' Services menus. *item* describes a service provided by this application, and should be the same string entered in the ªMenu Item:º field of the services file. The function returns 0 upon success.

BOOL **NSShowsServicesMenuItem(**NSString **item***)**

Returns YES if item is currently shown in Services menus.

Programmatically Invoke a Service

BOOL **NSPerformService(**NSString ***item***, NSPasteboard ***pboard*****)**

Invokes a service found in the application's Services menu. *item* is the name of a Services menu item, in any language; a slash in this name represents a submenu. *pboard* must contain the data required by the service, and when the function returns, *pboard* will contain the data supplied by the service provider.

Force Services Menu to Update Based on New Services

void **NSUpdateDynamicServices(**void**)**

Re-registers the services the application is willing to provide, by reading the file with the extension ª.serviceº in the application path or in the standard path for services.

Other Application Kit Functions

Play the System Beep

void **NSBeep(**void**)**

Plays the system beep.

Return File-related Pasteboard Types

NSString ***NSCreateFileContentsPboardType(**NSString ****fileType*****)**

Returns a string naming a pasteboard type that represents a file's contents, based on the supplied string *fileType*. *fileType* should generally be the extension part of a file name. The conversion from a named file type to a pasteboard type is simple; no mapping to standard pasteboard types is attempted.

NSString ***NSCreateFilenamePboardType(**NSString ****filename*****)**

Returns a string naming a pasteboard type that represents a a file name, based on the supplied string *filename*.

NSString ***NSGetFileType(**NSString ****pboardType*****)**

Returns the extension or file name from which the pasteboard type *pboardType* was derived. **nil** is returned if *pboardType* isn't a pasteboard type created by **NSCreateFileContentsPboardType()** or **NSCreateFilenamePboardType()**.

NSArray ***NSGetFileTypes(**NSArray ****pboardTypes*****)**

Accepts an array of pasteboard types and returns an array of the unique extensions and file names from the file-content and file-name types found in the input array. It returns **nil** if the input array contains no file-content or file-name types.

Draw a Distinctive Outline around Linked Data

void **NSFrameLinkRect(**NSRect *aRect*, BOOL *isDestination*)

Draws a distinctive link outline just outside the rectangle *aRect*. To draw an outline around a destination link, *isDestination* should be YES, otherwise NO.

float **NSLinkFrameThickness(**void**)**

Returns the thickness of the link outline so that the outline can be properly erased by the application, or for other

purposes.

Convert an Event Mask Type to a Mask

unsigned int **NSEventMaskFromType**(NSEventType *type*)

Returns the event mask corresponding to *type* (which is an enumeration constant). The returned mask equals 1 left-shifted by *type* bits.