Title: OPENSTEP 4.1 Release Notes: Application Kit
Entry Number: xxxx
Last Updated: September 5, 1996

OPENSTEP 4.1 Release Notes: Application Kit

New Features in 4.1

ComboBox

The NSComboBox class has been added to the Application Kit. It offers similar functionality to the Combo Box control defined in the Microsoft° Windows® user interface.

Splash Screen

OPENSTEP for Windows now provides support for a "splash" screen in applicationsÐbasically a panel that comes up with a static image as the application is being launched. To use this feature, simply provide an 8-bit uncompressed bmp image named **Splash.bmp** as a resource in your application. You can make it localizable if you wish. (Thus the image

will be either in *appname*.app/Resources or *appname*.app/Resources/*language*.lproj.) At launch time, the image will be loaded and displayed before any other initialization in the application, and will be removed from the screen before the first window is displayed.

For other changes in 4.1 please refer to the "Changes since 4.0" section below.

New Features in 4.0

OpenStep

OpenStep brings numerous API changes to the AppKit relative to Release 3.3. These changes and the tools provided for applications to convert to OpenStep are described in **/NextLibrary/Documentation/NextDev/Conversion/ConversionGuide**, available on OPENSTEP for Mach.

New Text System

4.0 includes a new text system composed of several different classes: NSTextView (front-end UI), NSTextStorage and NSAttributedString (back-end text storage), NSLayoutManager (management of text layout process and info), and NSTextContainer (description of text flow areas). These classes provide an open, powerful interface and allow text editing in multiple languages, using the Unicode standard. Please refer to the documentation for detailed info. An overview can be found in **/NextLibrary/Documentation/NextDev/TasksAndConcepts/ProgrammingTopics/TextOverview.rtfd** (Mach) or in the **/NextLibrary/Documentation/NextDev/TasksAndConcepts/Topics.hlp** (Windows). There is also a separate release note which describes some of the defaults that can be used to customize the new text system and how to remap or add to the standard supported keybindings. In addition to these notes, the source code to several text-specific examples, including the full source to TextEdit, can be found in the **/NextDeveloper/Examples/AppKit** directory.

TableView

TableView has been completely rewritten and moved into the AppKit. All four classes making up the new TableView are public and fully subclassable.

Keyboard UI

Keyboard access is now provided to most of the controls in the AppKit. For more information, see the separate Keyboard UI Release Notes.

Formatting and Validation

Cells may now be assigned arbitrary object values, which are converted into presentation strings by associated formatter objects. This allows the developer to directly set an NSDate, for instance, as the value of a cell. The cell's associated date formatter will present a localized string representation of the date to the user. The formatter objects, along with control delegates, can also perform validation on user-entered data, thereby restricting entries to valid ranges or quantities, for instance only allowing dates between January 1, 1995 and June 30, 1995.

Rich Text in Cells

NSCell and subclasses can now display and edit rich text. The rich text is specified via instances of NSAttributedString. The new formatting/validation API also includes support for attributed strings.

RulerView

NSRulerView is designed as a general-purpose ruler that can be associated with any scroll view and used by any view

that's in the scroll view. It supports both horizontal and vertical rulers, allows arbitrary markers along the rule, and can accept an accessory view.

System Colors

New API has been added to access system-defined colors, such as the color of buttons, controls, text and text selection colors. On Windows, where the user can change the system colors at any point, these colors will change at runtime to match the user's selection.

Help System

The AppKit's Help API has changed significantly. NSHelpPanel has been obsoleted in favor of a new class, NSHelpManager, which provides a more platform-independent approach to presenting help.

Known Problems in This Release

NSComboBox

Reference: 71826

Problem: NSComboBox should support keyboard interface

Description: NSComboBox does not support keyboard accelerators to open, dismiss, or navigate its popup list.

Workaround: None.

Reference: 60438

Problem: NSComboBox uses a private cell class.

Description: For the OPENSTEP 4.1 release, the cell class for NSComboBox is private.

Workaround: None.

NSTableView

TableView does not yet have printing support.

Reference: 51731

Problem: TableView does not fully support keyboard ui

Description: Only simple tabbing between editable cells is implemted in TableView. Arrow keys do not affect selection.

Workaround: Override keyDown: in a subclass or in the owning window and change the selection accordingly.

Reference: 63967

Problem: Header and row height do not change relative to font size

Description: When the developer changes the font size of the header view cells or data cells in a TableView, the heights of the header view and rows do not change appropriately.

Workaround: Manually change the height of the header view and row height.

NSCell

Reference: 52511

Problem: **setUpFieldEditorAttributes:** called only when editing

Description: **setUpFieldEditorAttributes:** is not called when non-editable text is displayed in cells. This makes it difficult to change colors in custom subclasses of NSCell.

Workaround: To display colored text in cells, text fields or attributed string values may be used.

NSImage

Reference: 60413

Problem: JPEG TIFFs don't work on Windows.

Description: Because the imageserver process is not yet available under Windows, JPEG TIFFs cannot be loaded in.

Workaround: None. The images can be compressed using another compression scheme using tiffutil on OpenStep for Mach.

New Text System

Reference: 58479

Problem: Rotated text views don't so selection very well

Description: The selection drawing (at least) in a rotated NSTextView needs work. The selection is generally drawn too large and leaves pixels around.

Workaround: None. Avoid editable rotated NSTextViews for now.

Reference: 50123

Problem: Some features are currently unimplemented

Description: The following features are currently unimplemented: Full justification, tabs other than left, certain paragraph

style attributes, scaled images.

Workaround: None.

Reference: 54951

Problem: Screen fragments in containers with holes

Description: If you attempt to create an NSTextContainer with holes such that one physical line can have multiple line fragments, there are still problems with erasing old drawing below the hole when text is deleted.

Workaround: None.

Reference: 49744

Problem: Images are not scaled in scaled text

Description: Using the standard NSTextAttachment class, the images appear at 100%.

Workaround: This problem can be fixed by using a custom attachment cell.

Reference: 68768

Problem: Sometimes two lines will appear on top of each other

Description: The backing store is fine, but the text is laid out with two lines right on top of each other. This is fairly rare but somewhat disconcerting.

Workaround: Typing a character in the overlapped lines and then deleting it will fix the layout.

NSWindow

Reference: 71671

Problem: "Flash active caption bar" visual warning option might cause apps to hang

Description: If you have enabled SoundSentry (under Accessibility Options in the Control Panel), and turned on "Flash active caption bar" visual warning option, there is a good chance apps will hang when they call NSBeep(). This function is called by the Application Kit in a few casesÐwhen you perform an unrecognized command key, for instance.

Workaround: None. Use one of the other visual warning options.

Reference: 72040

Problem: Resizing a modal panel causes it stop processing mouse down events

Description: If you resize a modal panel it will stop responding to mouse down events until it is dismissed and brought back on the screen again.

Workaround: Because keyboard events still work, you can dismiss the panel by navigating to the appropriate button via keyboard UI.

Reference: 71648

Problem: Closing a modal window or panel does not always end the modal session

Description: Closing a modal window or panel by clicking on the close box will not always end the modal session, leaving the

application in a state where mouse clicks and key strokes are ignored. On Windows, modal panels often have a close box. This acts the same as a cancel button and, by default, clicking the close box while running modal will stop the modal session with the NSCancelButton return code. However, if the Window's delegate decides to control when the window should close by implementing windowShouldClose:, then it is also up to the delegate to stop the modal session in windowWillClose:. On Mach, modal windows and panels rarely have a close box; if they do they must stop modal in windowWillClose:.

Workaround: On Windows, a window delegate implementing windowShouldClose should stop the modal session in windowWillClose:. On Mach, any modal window or panel with a close box should stop the modal session in windowWillClose:

Reference: 64988

Problem: NSBackingStoreNonretained don't work under Windows

Description: Under Windows, nonretained windows aren't implemented.

Workaround: Use retained or buffered windows instead.

Reference: 69973

Problem: Miniaturizing and then restoring a window under Windows causes first responder to be lost

Description: When a window is miniaturized it loses track of its current first responder. When it is restored again, the first responder is not restored.

Workaround: You can use makeFirstResponder to set a first responder yourself or just make the user click something.

Reference: 69327

Problem: Clicking in the contents a window which is key won't bring it to the front

Description: Under Windows, if a non-key panel is in front of the key window, clicking in the content view of the key window will not bring it to the front.

Workaround: Clicking the title bar will bring it to the front.

Drag/Drop

Reference: 60829

Problem: On Windows, **ignoreModifierKeysWhileDragging** is never invoked on drag source

Workaround: None.

Reference: 67620

Problem: On Windows **draggedImage** always returns nil.

Description: During a dragging session, the DraggingInfo will always return nil when queried for its draggedImage.

Workaround: None.

Reference: 63466

Problem: On Windows, the drag source's image offset is ignored

Description: Currently the dragged image will appear to the lower right of the cursor's hot spot.

Workaround: None.

NSFont

Reference: 61824

Problem: Microsoft's *WingDings* font is not supported well under OPENSTEP for Windows

Description: There isn't currently a Unicode mapping for any of the glyphs in it, even into the User Zone, so it works as any other "unknown" font (i.e., characters 0-255 are mapped directly to glyphs 0-255).

Workaround: None.

Keyboard UI

Reference: 51182

Problem: AccessoryViews are not added to keyView loop

Description: When a panel supporting accessory views, such as the SavePanel, is brought on screen, the accessory views are not automatically added to the keyView loop.

Workaround: Add the accessory view to the keyView loop by hand using setNextKeyView: before brining the panel on screen.

NSOpenPanel

Reference: 63490

Problem: On Windows, one can open file packages but not directories without extensions.

Description: The OpenPanel on Windows does not provide a way to select and open plain directories. Clicking on directories without extensions will expand the directory rather than opening it. Directories with extensions are correctly dealt with as file packages.

Workaround: Either require selectable directories to have extensions or provide a well defined file inside the directory for the user to select.

Accessory Views

On Windows, the PageLayout panel does not yet support application supplied accessory views. Also on Windows, accessory views in the Open, Save and Print panels work only on NT 4.0, not NT 3.51.

Changes since 4.0

The following list all the API changes and some of the bug fixes that were done in the Application Kit between 4.0 and 4.1.

Note that in your application if you use any API which was added between 4.0 and 4.1, your application might not run under 4.0. (If you are not deploying under 4.0, then this is not an issue.) You can catch some of the potential problems by defining **STRICT_40** at compile time. This #define is used in the Application and Foundation Kits to mark new API.

Similarly, if you rely on a bug fix done in 4.1, your application may not function correctly under 4.0. Make sure you test on your intended deployment platform.

NSApplication

IBOutlet and **IBAction** are now defined in the kit and automatically imported if you import **AppKit.h**. You can remove your own custom definitions of these (if you had any).

The **hide:** method has been changed to do nothing on Windows.

The default value of the **NSUseRunningCopy** command line option has been changed to YES. This means that anytime you cause a second copy of the application to be launched, it will connect you to the already running copy. You should also If you wish to run multiple copies of an application, you should use "-NSUseRunningCopy NO". You might need this if you need to debug a new instance of the application while it's already running.

Another change is that all command line options which are not defaults options (meaning a pair of arguments where the first one starts with a "-") are now treated as file names to be opened, as if they were prefixed with -NSOpen.

As a result of these two changes, you can now associate documents with OPENSTEP apps on Windows simply by providing the location of the app. The default command line provided by the Explorer will be enough to open documents and also connect to a running copy of the application, if there is one.

On Windows, **[NSApplication init]** (called by **sharedApplication**) changes the current directory to the root of the drive from which the application is launched. This is done in order to avoid problems associated with the application starting off in random places (such as the directory of a file wrapper, when launched by double-clicking on a file in a file wrapper). One consequence of this is that any directory changes which take place in the **main()** function, before **sharedApplication**, will fail to stick.

NSWindow

In OPENSTEP for Windows, document edited state is now indicated in the title bar by prefixing the title with an asterisk. OPENSTEP for Mach still uses a broken "X".

Tracking rects are now implemented on Windows.

The following are not actually changes since 4.1, but are a couple things to be aware of concerning NSWindows in OPENSTEP for Windows.

· Under Windows, the frameRect and contentRect are currently the same size. The frameRect of an NSWindow under OPENSTEP for WIndows NT does NOT include the title bar, menu bar, resize border, etc...

· Under Windows, the contentView while a window is miniaturized is a NSImageView, not the original contentView. The contentView is restored when the window is deminiaturized. This is necessary because under some versions of Windows NT, the same actual window is used when the window is full size or miniaturized. It is just moved and resized when miniaturing or restoring. Since most content views are not prepared to size themselves that small, the content view is removed from the window while it miniaturizes and is restored when the window is restored.

NSTableView

A new notification, **NSTableViewSelectionDidChangeNotification**, and a corresponding delegate method, **tableViewSelectionIsChanging:**, have been added to NSTableView. The existing **NSTableViewSelectionDidChangeNotification** is only sent after the user has moused up when changing a selection. The new notification is sent while the selection is changing.

NSSavePanel

The save panel will now put up an alert letting you confirm before you replace an existing file.

NSInputServer

The following methods were added to NSInputServer. They are sent by current input manager when the application changes state so that the server can update its concept of who's current. The actually "active" sender is the last one to have sent a **senderDidBecomeActive:** message. These methods may not arrive in the expected order.

```
- (void)senderDidBecomeActive:(id)sender;
- (void)senderDidResignActive:(id)sender;
```

This method is sent by input manager when the conversation within a particular sender changes:

```
- (void)activeConversationWillChange:(id)sender oldConversation:(long)conv newConversation:(long)new;
```

NSTextView

To allow entering a character without an appropriate keyboard, you can now use the Alt key in conjunction with the digit keys from the keypad. While holding down the Alt key, type the index of the desired character in the encoding of the current font (in most cases this will be the NEXTSTEP encoding). The appropriate character will be inserted into the text when the Alt key is released.

NSWorkspace

The **openFile:withApplication:** will now correctly open the file with the specified application. In 4.0 this method always

returned NO when an application name was supplied.

OPENSTEP for Window now contains an "open" command. Like on Mach, "open filename" will open the specified file in its registered application. You can also use the "-e" flag to open the file in TextEdit or the "-a" flag to launch an application.

NSAttributedString

If supplied a plain text file, the **initWithPath:documentAttributes:** method will now go ahead and still create an attributed string from the plain text contents. Note that if the file looks like an RTF file (has the appropriate magic header), but fails to open, the method will still return nil.

NSColor

Unarchived NSColors and NSColors created with **colorWithCalibratedRed:green:blue:alpha:** and **colorWithCalibratedWhite:alpha:** will be replaced by the cached and uniqued colors returned by methods such as [NSColor whiteColor] if they are exactly the same. In some cases this will greatly reduce the number of colors created at runtime and speed up certain operations. However, in the unlikely event that this causes problems in your application, or prevents your 4.0 application from running correctly under 4.1, you can set the **NSCachedColorConversion** default to NO.

System colors (such as those returned by [NSColor controlColor]) have been improved to generate colors which match those used by Windows. Under 8-bit color this should result in a visible improvement in certain color schemes.

Changes before 4.0

There are tops scripts to assist PR1 to PR2 and PR2 to 4.0 migration. Many of the API changes that have happened between prereleases can be automatically applied to your code with these commands:

```
tops -semiverbose -scriptfile /NextDeveloper/OpenStepConversion/ConversionScripts/PR1toPR2.tops *.[mchCM] *.psw *.pswm

tops -semiverbose -scriptfile /NextDeveloper/OpenStepConversion/ConversionScripts/PR2to40.tops *.[mchCM] *.psw *.pswm
```

Formatting and Validation

NSFormatter and NSDateFormatter have moved into the Foundation Kit. FoundationKit now also includes a number formatter. Imports of <AppKit/NSFormatter.h> or <AppKit/NSDateFormatter.h> will need to be changed to import the corresponding file from Foundation; imports of <AppKit/AppKit.h> don't need to be changed. The tops script mentioned above takes care of this. Also, a new NSNumberFormatter has been added.

The control delegate message **control:validateObject:** has been changed to **control:isValidObject:**.

NSApplication

NSApplication defines a new delegate method similar to **-application:openFile:** called

```
-(void)application:(NSApplication *)app printFile:(NSString *)path
```

Upon launching, an application scans its command line arguments for -NSOpen and, now, -NSPrint arguments. These arguments are followed by the path to a file to open or print respectively. NSApplication sends the above messages to

its delegate if it responds in this way for the files specified on the command line. These messages are also invoked in response to Speaker/Listener interaction, but this aspect is perhaps less interesting to developers.

This new message (and the new command-line argument -NSPrint) is supported on both Mach and Windows.

Under Windows NT, when you install OPENSTEP, Explorer associations are automatically created for each file extension that the OPENSTEP applications claim. The file-type's icon is associated with its extension and Open and Print commands are registered as well. These commands work through the new command-line options, and can be accessed through Explorer's right-mouse context menu. So, for example, right clicking an RTF file in Explorer will give you a menu that includes both Open and Print as well as some other stuff. Choosing Open or Print will cause TextEdit to open or print the file. Open is registered as the default command for these file extensions, so double-clicking an RTF file will open it in TextEdit.

A new notification, **NSApplicationWillTerminateNotification,** has been added to OpenStep. It is sent before the Application's **terminate** method after **applicationShouldTerminate** is queried.

A Windows-specific method has been added to allow access to the Windows application handle. This will only be of interest to developers calling the Win32 API directly. (Arguments that varioius types of Windows handles are declared as void currently to avoid polluting the namespace with the vast amount of un-prefixed names in the Windows system headers.)

```
- (void * /*HINSTANCE*/)applicationHandle;
```

A Windows-specific method has been added to allow developers to look up an NSWindow object given a Windows HWND handle. This will only be of interest to developers calling the Win32 API directly.

```
- (NSWindow *)windowWithWindowHandle:(void * /*HWND*/)hWnd;
```

A Windows-specific method has been added for developers who choose to implement their own WinMain() function If you do so, call the method below before doing any other AppKit calls. The arguments are those passed to WinMain() from the system.

```
+ (void)setApplicationHandle:(void * /*HINSTANCE*/)hInstance
      previousHandle:(void * /*HINSTANCE*/)PrevInstance
      commandLine:(NSString *)cmdLine
      show:(int)cmdShow;
```

NSTableView

The table view has a new mode where it resizes all the columns, not just the last one, to fill the scroll view. In this mode, if the table view fits exactly into its scroll view, and is then grown, it will grow all of its resizable columns to keep itself the size of the scroll view. Here are the new methods to enable this:

```
- (void)setAutoresizesAllColumnsToFit:(BOOL)flag;
- (BOOL)autoresizesAllColumnsToFit;
```

While tracking a cell or performing the table view's action, one may discover what cell was clicked on to cause the action to perform with these additional methods:

```
- (int)clickedColumn;
- (int)clickedRow;
```

NSAttributedString

Parts of NSAttributedString and NSMutableAttributedString have moved into the Foundation Kit. Just using Foundation, without the Application Kit, you can create and manipulate attributed strings; however you do not have access to the

standard attribute types such as color, font, and paragraph style. The rest of the attributed string functionality is provided by categories in Application Kit.

The following names have changed in NSAttributedString:

```
NSAttributedStringEditedAttributes -> NSTextStorageEditedAttributes
NSAttributedStringEditedCharacters -> NSTextStorageEditedCharacters
```

The following methods to read RTF:

```
+ (NSAttributedString *)attributedStringFromRTF:(NSData *)rtfData;
+ (NSAttributedString *)attributedStringFromRTFD:(NSData *)rtfData;
+ (NSAttributedString *)attributedStringFromRTFDFile:(NSString *)rtfdPath;
```

have been replaced by:

```
- (id)initWithRTF:(NSData *)data documentAttributes:(NSDictionary **)dict;
- (id)initWithRTFD:(NSData *)data documentAttributes:(NSDictionary **)dict;
- (id)initWithPath:(NSString *)path documentAttributes:(NSDictionary **)dict;
- (id)initWithRTFDFileWrapper:(NSFileWrapper *)wrapper documentAttributes:(NSDictionary **)dict;
```

The optional dict argument is for returning document-wide attributes, which currently include "PaperSize", "LeftMargin", "RightMargin", "TopMargin", and "BottomMargin". Pass NULL for dict if you don't care about these.

The following methods to write RTF:

```
- (NSData *)RTFFromRange:(NSRange)range;
- (NSData *)RTFDFromRange:(NSRange)range;
```

```
- (BOOL)writeRTFDToFile:(NSString *)path atomically:(BOOL)flag;
```

have been replaced by:

```
- (NSData *)RTFFromRange:(NSRange)range documentAttributes:(NSDictionary *)dict;
- (NSData *)RTFDFromRange:(NSRange)range documentAttributes:(NSDictionary *)dict;
- (NSFileWrapper *)RTFDFileWrapperFromRange:(NSRange)range documentAttributes:(NSDictionary *)dict;
```

Pass nil for dict if you don't want to specify any of the document attributes.

To write RTFD to file you will need to send **writeToFile:atomically:updateFilenames:** to the NSFileWrapper you get back from **RTFDFileWrapperFromRange:documentAttributes:**. The updateFilenames: flag should usually be YES (unless you are doing a "Save To").

The following method names are changed in NSMutableAttributedString:

Before:

```
- (void)fixAttributesAfterEditingRange:(NSRange)range;
- (void)fixFontAttributeAfterEditingRange:(NSRange)range;
- (void)fixParagraphStyleAfterEditingRange:(NSRange)range;
- (void)fixAttachmentAfterEditingRange:(NSRange)range;
```

After:

```
- (void)fixAttributesInRange:(NSRange)range;
- (void)fixFontAttributeInRange:(NSRange)range;
- (void)fixParagraphStyleAttributeInRange:(NSRange)range;
- (void)fixAttachmentAttributeInRange:(NSRange)range;
```

The value of **NSAttachmentCharacter** has been changed from `0xF6FF` to `0xFFFC` to match the one proposed in the

Unicode standard. Code which uses this value will need to be recompiled; places where this value is archived explicitly might need to check for the old value. (RTF files did archive this value, so no need for changes there.) If the old value ever appears in an attributed string at runtime, the new value will be used, and a warning will be generated (single warning per app session).

The method **updateAttachmentsFromPath:** has been added to NSMutableAttributedString to allow updating the file name information in all the attachments in the attributed string.

NSImage

NSImage now understands the bmp, ico, and cur image formats. ico and cur files with multiple images will be loaded as images with multiple representations. Typically these representations will have different sizes (unlike multiple-representation tiffs, which have different depths); by default, NSImage will choose the largest image when compositing.

Windows-specific methods have been added to NSImage and NSBitmapImageRep which allow creation of these objects from a Windows icon handle or bitmap handle. These will only be of interest to developers calling the Win32 API directly.

```
- (id)initWithIconHandle:(void * /* HICON */)icon;
- (id)initWithBitmapHandle:(void * /* HBITMAP */)bitmap;
```

setFlipped: and **flipped** have been brought back into the API.

Compositing Operation name changes

Some of the constants declared by DPSClient for various compositing operations have had their names corrected:

Old Name	New Name

```
NSCompositeDataOver                     NSCompositeDestinationOver
NSCompositeDataInreplace                NSCompositeDestinationIn
NSCompositeDataOut                      NSCompositeDestinationOut
NSCompositeDataAtop                     NSCompositeDestinationAtop
```

NSCursor

In order to make NSCursor immutable, the following methods:

```
- (id)initWithImage:(NSImage *)newImage;
- (void)setImage:(NSImage *)newImage;
- (void)setHotSpot:(NSPoint)spot;
```

have been replaced by:

```
- (id)initWithImage:(NSImage *)newImage hotSpot:(NSPoint)aPoint;
- (id)initWithImage:(NSImage *)newImage
      foregroundColorHint:(NSColor *)fg
      backgroundColorHint:(NSColor *)bg
      hotSpot:(NSPoint)hotSpot;
```

The color hints in the latter method are ignored in OPENSTEP for Windows and for Mach.

The **initWithImage:** and **setImage:** methods are somewhat generic and hard to detect, so the tops script might fail to catch all your uses.

Another change in NSCursor since PR2 is the support for custom cursor selections on Windows. OPENSTEP applications will use the correct arrow and I-beam cursors where applicable. If you ask for the image of one of these standard cursors on OPENSTEP for Windows, you will get back the correct image used by the system. You should note

that the size of this image may be different than the traditional 16x16 used on Mach. For animated cursors you will simply get back the first frame.

NSText

The following method names are changed in the NSText API:

Before:

```
- (NSString *)text;
- (void)setText:(NSString *)string;
- (void)setText:(NSString *)string range:(NSRange)range;
- (void)replaceRange:(NSRange)range withRTF:(NSData *)rtfData;
- (void)replaceRange:(NSRange)range withRTFD:(NSData *)rtfdData;
- (void)setColor:(NSColor *)color ofRange:(NSRange)range;
- (void)setFont:(NSFont *)font ofRange:(NSRange)range;
```

After:

```
- (NSString *)string;
- (void)setString:(NSString *)string;
- (void)replaceCharactersInRange:(NSRange)range withString:(NSString *)aString;
- (void)replaceCharactersInRange:(NSRange)range withRTF:(NSData *)rtfData;
- (void)replaceCharactersInRange:(NSRange)range withRTFD:(NSData *)rtfdData;
- (void)setTextColor:(NSColor *)color range:(NSRange)range;
- (void)setFont:(NSFont *)font range:(NSRange)range;
```

NSTextView

The following methods are added to NSTextView:

```
- (void)pasteAsPlainText:(id)sender;
- (void)pasteAsRichText:(id)sender;

- (NSPoint)textContainerOrigin;
- (void)invalidateTextContainerOrigin;

- (NSRange)rangeForUserTextChange;
- (NSRange)rangeForUserCharacterAttributeChange;
- (NSRange)rangeForUserParagraphAttributeChange;

- (BOOL)smartInsertDeleteEnabled;
- (void)setSmartInsertDeleteEnabled:(BOOL)flag;
- (NSRange)smartDeleteRangeForProposedRange:(NSRange)proposedCharRange;
- (void)smartInsertForString:(NSString *)pasteString replacingRange:(NSRange)charRangeToReplace
                beforeString:(NSString **)beforeString afterString:(NSString **)afterString;
```

The following method names are changed in the NSTextView API to allow richer specification of selected text attributes:

Before:

```
- (void)setSelectionColor:(NSColor *)color;
- (NSColor *)selectionColor;
```

After:

```
- (void)setSelectedTextAttributes:(NSDictionary *)attributeDictionary;
- (NSDictionary *)selectedTextAttributes;
```

Underlining is now supported in the new text system; **NSSingleUnderlineStyle** has been added as a possible value for the **NSUnderlineStyleAttributeName** attribute.

The delegate method **textView:clickedOnCell:inRect:** has been added to indicate single-clicks on text attachments.

The method **attachmentForDraggedFilename:** has been removed from the API.

NSTextStorage

NSTextStorage now sends out notification messages after edits:

```
NSTextStorageWillProcessEditingNotification
NSTextStorageDidProcessEditingNotification
```

It also has a delegate who gets the following messages:

```
- (void)textStorageWillProcessEditing:(NSNotification *)notification;
- (void)textStorageDidProcessEditing:(NSNotification *)notification;
```

These methods are called during post-edit processing (basically after **endEditing** has been called, or, if endEditing was not in effect, after each edit that causes **edited:range:changeInLength:**). The first notification is sent before attributes are fixed, the second is sent after the attributes are fixed, but before the layout managers are notified. During the execution of **textStorageWillProcessEditing:** the delegate can change the attributed string all it wants; during **textStorageDidProcessEditing:** it should refrain from changing the character contents. The changes done by the delegate will not cause further notifications to be sent. However, during the execution of these methods and notifications the delegate and the observers can obtain information about what's changed via the following methods:

```
- (unsigned)editedMask;
- (NSRange)editedRange;
- (int)changeInLength;
```

NSTextStorage now also has a clear funnel point for all post-edit processing:

```
- (void)processEditing;
```

This method sends the above notification methods, calls **fixAttributesInRange:**, and informs the layout managers of the changes.

The following methods, now much less useful, have been removed from the API:

```
- (void)verifyEdited:(unsigned)mask range:(NSRange)range changeInLength:(int)delta;
- (void)notifyEdited:(unsigned)mask range:(NSRange)range changeInLength:(int)delta
    invalidatedRange:(NSRange)invRange;
```

Subclassers, note: NSTextStorage is now a semi-abstract class. It implements change management (**beginEditing**/**endEditing**), verification of attributes, and layout management notification. The one aspect it does not implement is the actual attributed string storage--this is left up to the subclassers, which need to override the two NSMutableAttributedString primitives **replaceCharactersInRange:withString:** and **setAttributes:range:**. These primitives should perform the change then call **edited:range:changeInLength:** to get everything else to happen.

Here is a sample NSTextStorage subclasser:

```
@implementation SimpleTextStorage

/* Provide your own init methods, whatever you wish... Here we provide one (initWithAttributedString:)
and override the DI from super (init). */

- (id)initWithAttributedString:(NSAttributedString *)attrStr {
    if (self = [super init]) {
        contents = attrStr ? [attrStr mutableCopyWithZone:[self zone]] :
                             [[NSMutableAttributedString allocWithZone:[self zone]] init];
```

```
    }
    return self;
}

- init {
    return [self initWithAttributedString:nil];
}

- (void)dealloc {
    [contents release];
    [super dealloc];
}

/* The next set of methods are the primitives for attributed and mutable attributed string... */

- (NSString *)string {
    return [contents string];
}

- (NSDictionary *)attributesAtIndex:(unsigned)location effectiveRange:(NSRange *)range {
    return [contents attributesAtIndex:location effectiveRange:range];
}

/* In the mutable primitives (and any non-primitives which are overridden and which don't go through the
primitives), we need to call edited:range:changeInLength: to allow the text storage change management to
do its job. */

- (void)replaceCharactersInRange:(NSRange)range withString:(NSString *)str {
    [contents replaceCharactersInRange:range withString:str];
    [self edited:NSAttributedStringEditedCharacters
           range:range
```

```
  changeInLength:(int)[str length] - (int)range.length];
}

- (void)setAttributes:(NSDictionary *)attrs range:(NSRange)range {
    [contents setAttributes:attrs range:range];
    [self edited:NSAttributedStringEditedAttributes range:range changeInLength:0];
}

@end
```

NSFileWrapper

A new public class has been added to the AppKit. NSFileWrapper provides support for the concept of a document wrapper (like a .rtfd, .nib, or many other NEXTSTEP file types). The new text system uses this class to handle RTFD documents, but it is designed to be generally useful. It handles reading and writing file packages in the file system as well as serializing them for use with the Pasteboard. See the General Reference for details.

NSTextAttachment

The following changes in NSTextAttachment take advantage of NSFileWrapper:

Before:

```
- (id)initWithContentsOfFile:(NSString *)path;
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)flag;
```

After:

```
- (id)initWithFileWrapper:(NSFileWrapper *)fileWrapper;  /* Designated initializer */
- (void)setFileWrapper:(NSFileWrapper *)fileWrapper;
- (NSFileWrapper *)fileWrapper;
```

The **name** and **setName:** methods have also been removed from the API.

NSParagraphStyle

Subclassers, note: **NSParagraphStyle** and **NSMutableParagraphStyle** used to be abstract; they are now concrete.

setParagraphStyle: has been added to NSMutableParagraphStyle to allow setting all the values with one call.

NSRulerView/NSRulerMarker

The delegate method **rulerView:willSetClientView:** has been added to NSRulerView to give the existing client a chance to clean up.

The following method names are changed in NSRulerMarker:

Before:

```
- (void)setLocation:(float)location;
- (float)location;
- (id)initWithRulerView:(NSRulerView *)ruler location:(float)location image:(NSImage *)image
imageOrigin:(NSPoint)imageOrigin;
```

After:

```
- (void)setMarkerLocation:(float)location;
- (float)markerLocation;
- (id)initWithRulerView:(NSRulerView *)ruler location:(float)location image:(NSImage *)image
imageOrigin:(NSPoint)imageOrigin;
```

NSWorkspace

The following NSWorkspace methods will raise under OPENSTEP for Windows to indicate that they are not implemented (and the return value, being wrong, shouldn't be relied upon):

```
- (BOOL)fileSystemChanged;
- (BOOL)userDefaultsChanged;
- (NSArray *)mountedRemovableMedia;
- (NSArray *)mountNewRemovableMedia;
```

The following methods are no-ops:

```
- (void)checkForRemovableMedia;
- (void)findApplications;
- (void)hideOtherApplications;
- (void)noteUserDefaultsChanged;
- (void)noteFileSystemChanged;
- (BOOL)getFileSystemInfoForPath:isRemovable:isWritable:isUnmountable:description:type:;
- (int)extendPowerOffBy;
- (void)slideImage:(NSImage *)image from:(NSPoint)fromPoint to:(NSPoint)toPoint;
```

In addition, note that the **NSWorkspaceRecycleOperation** will currently remove the files in OPENSTEP for Windows.

NSBrowser

The width of columns are now dealt with as floats rather than ints to be consistent with the rest of the AppKit:

Before:

```
- (void)setMinColumnWidth:(int)columnWidth;
- (int)minColumnWidth;
```

After:

```
- (void)setMinColumnWidth:(float)columnWidth;
- (float)minColumnWidth;
```

A couple other api items changed names:

Before:

```
- (void)drawTitle:(NSString *)title inRect:(NSRect)aRect ofColumn:(int)column;
- (BOOL)browser:(NSBrowser *)sender selectCell:(NSString *)title inColumn:(int)column;
```

After:

```
- (void)drawTitleOfColumn:(int)column inRect:(NSRect)aRect;
- (BOOL)browser:(NSBrowser *)sender selectCellWithString:(NSString *)title inColumn:(int)column;
```

Selection may now be done by index as well as path. Previously the developer was forced to resort to NSMatrix apis when trying to select a row in a browser column by index. Now she may use:

```
- (void)selectRow:(int)row inColumn:(int)column;
- (int)selectedRowInColumn:(int)column;
```

And the following delegate method was added in parallel to allow the delegate to perform the selection:

```
- (BOOL)browser:(NSBrowser *)sender selectRow:(int)row inColumn:(int)column;
```

Alerts

Two pairs of new functions have been added to support the alert styles found in the Windows UI. **NSRunInformationalAlertPanel()** and **NSRunCriticalAlertPanel()** take the same paramenters as **NSRunAlertPanel()**, but on Windows they present an alert with the "informational" or "critcal" icons, respectively. On Windows **NSRunAlertPanel()** presents an alert with the "warning" icon. On Mach calling any of the three functions produces the same UI. Also note that on Windows alerts have no title, so the title argument to these functions is not displayed in the UI. In addition, **NSGetInformationalAlertPanel()** and **NSGetCriticalAlertPanel()** have been added, which have the same relationship to the existing **NSGetAlertPanel()**.

The function **NSRunLocalizedAlertPanel()** is removed from the API; use **NSRunAlertPanel()** with explicitly localized arguments instead.

Services

The API involved in vending services has changed in PR2. The need for two names to resolve a service provider has been removed. In PR1, services were advertised in the Info.plist using a combination of NSExecutable, and NSProviderName. The provider name is now obsolete, so you simply advertise a single name under the entry NSPortName. This entry has the same meaning as in 3.3 services. Usually NSPortName will be the name of your application as NSExecutable was (e.g. "Webster"). The following API changes have also been made:

- **registerServiceProvider:withName:** has been obsoleted. Use **setServicesProvider:**.
- **unregisterServiceProviderNamed:** has been obsolete. Use **setServicesProvider:nil**.

Two new functions, **NSRegisterServiceProvider()** and **NSUnregisterServicesProvider()** can be used to register

service providers under a given name. These functions should only be used by service providers not creating an application object (filter services often fall into this category). The name argument should be the same as that specified by the NSPortName entry of the Info.plist. See the Services documentation for more details.

NSWindow

The following new methods control the resizing behavior of the window. The first pair are used to make a window resize in fixed increments. The second pair are used to make a window resize such that it maintains an aspect ratio.

```
- (void)setResizeIncrements:(NSSize)increments;
- (NSSize)resizeIncrements;
- (void)setAspectRatio:(NSSize)ratio;
- (NSSize)aspectRatio;
```

The following new methods provide an OpenStep alternative to the NEXTSTEP-specific feature of instance drawing. The first method saves away a part of the window's image. Typically after this step, some temporary drawing is done. The second method restores the previously saved image, undoing any drawing that has happened since it was saved. The third method releases the memory used to hold the saved image.

```
- (void)cacheImageInRect:(NSRect)aRect;
- (void)restoreCachedImage;
- (void)discardCachedImage;
```

The **fax:** method has been removed from the OpenStep specification, and is now specific to the Mach platform.

A Windows-specific method has been added to allow access to the Windows window handle of an NSWindow object. This will only be of interest to developers calling the Win32 API directly.

```
- (void * /*HWND*/)windowHandle;
```

NSView

The following new method removes a view from its view hierarchy, without marking that the now exposed region in its superview needs redisplay. This can be useful when a view is temporaily swapped in so as to draw something on behalf of its superview.

```
- (void)removeFromSuperviewWithoutNeedingDisplay;
```

The following method was added in order to preserve the utility of panels that "become key only if needed" (the Font Panel, for example). This feature was defeated by the advent of increased keyboard accessibility to the UI, as nearly all widgets now accept becoming the first responder, which meant the panel would become key almost no matter where the user clicked. The following method is implemented by a widget to return YES if clicking on it should cause one of these panels to become key (which is the same role **acceptsFirstResponder** used to play in these panels).

```
- (BOOL)needsPanelToBecomeKey;
```

The **fax:** method has been removed from the OpenStep spec, and is now specific to the Mach platform.

The following API was removed because it was unused in any of the OPENSTEP products.

```
@interface NSView(NSRIBPrinting)
- (BOOL)canPrintRIB;
@end

typedef enum _NSPosition {
    NSNoPosition = 0,
```

```
        NSLeft = 1,
        NSRight = 2,
        NSTop = 3,
        NSBottom = 4
    } NSPosition;
```

A new method has been added to NSView, which is to be overridden by subclassers to find when their view is installed in a new superview.

```
- (void)viewWillMoveToSuperview:(NSView *)newSuperview;
```

Two new display methods have been added to round out the existing set.

```
- (void)displayIfNeededInRect:(NSRect)rect;
- (void)displayIfNeededInRectIgnoringOpacity:(NSRect)rect;
```

The **NSViewBoundsChangedNotification** notification present in PR1 has been added to the OpenStep spec.

Updating in NSApplication, NSWindow and DPSClient

Various updating mechanisms in the Application Kit are now driven by a new Foundation feature, where NSRunLoop will perform selectors registered by clients in a given order during it's next cycle. (See **performSelector:target:argument:order:modes:** in NSRunLoop.) The following constants represent these uses within the Application Kit:

NSUpdateWindowsRunLoopOrdering (500000)
: All Windows are sent the update message. This is also the time when menu items are enabled and disabled.

NSDisplayWindowRunLoopOrdering (600000)
All Windows holding Views needing redisplay are given the chance to redraw.

NSResetCursorRectsRunLoopOrdering (700000)
If needed, cursor rectangles are reestablished.

DPSFlushContextRunLoopOrdering (800000)
The IPC channel to the Window Server is flushed, ensuring any buffered data is executed.

NSResponder

The **insertNewLine:** method was renamed to **insertNewline:**.

NSApplicationMain

The AppKit now provides a function NSApplicationMain() to take care of the initialization and startup of your application. This function is declared in NSApplication.h. Because of this, ProjectBuilder will now generate a trivial main function for your application:

```
#import <AppKit/NSApplication.h>

int main(int argc, const char *argv[]) {
    NSApplicationMain(argc, argv);
}
```

Although developer's should not need access to the body of NSApplicationMain(), it is documented here in the rare case that it is necessary. If at all possible, avoid copying this code, as doing so may prevent your application from automatically inheriting features added in future releases.

```
int NSApplicationMain(int argc, const char *argv[]) {
    NSAutoreleasePool * pool     = [[NSAutoreleasePool alloc] init];
    NSDictionary *info           = [[NSBundle mainBundle] infoDictionary];
    NSString *principalClassName = [info objectForKey: @"NSPrincipalClass"];
    NSString *mainNibFile        = [info objectForKey: @"NSMainNibFile"];

    if (principalClassName) {
      Class principalClass = NSClassFromString(principalClassName);
      if (principalClass) {
          [principalClass sharedApplication];
          if ([NSBundle loadNibNamed: mainNibFile owner: NSApp]) {
             [pool release];
             [(NSApplication *)NSApp run];
             [NSApp release];
             exit(0);
          }
          else NSLog(@"Unable to load nib file: %@, exiting", mainNibFile);
      }
      else NSLog(@"Unable to find class: %@, exiting", principalClassName);
    }
    else NSLog(@"No NSPrincipalClass specified in info dictionary, exiting");
    [NSApp release];
    [pool release];
    exit(1);
    return 0;
}
```

NSHost

NSHost (or NXHost)'ing is supported only between OPENSTEP for Mach machines or from NEXTSTEP 3.x machines to OPENSTEP for Mach. You cannot run an OPENSTEP application NSHost'ed to a 3.x machine. NSHost'ing is also not supported on OPENSTEP for Windows.

ObjectLinks

ObjectLinks have been removed from the OpenStep specification, and in general the feature is not supported in OPENSTEP for Mach or Windows. The exception is that the following subset of the API may be used on either platform to walk through the linked data from documents written in NEXTSTEP 3.X:

```
@interface NSDataLinkManager : NSObject <NSCoding> {
- (id)initWithDelegate:(id)obj fromFile:(NSString *)path;
- (NSEnumerator *)sourceLinkEnumerator;
- (NSEnumerator *)destinationLinkEnumerator;
@end

@interface NSDataLink : NSObject <NSCopying, NSCoding>
- (NSArray *)types;
- (NSSelection *)sourceSelection;
- (NSSelection *)destinationSelection;
- (NSString *)sourceFilename;
- (NSString *)currentSourceFilename;   - (NSString *)destinationFilename;
- (NSString *)sourceApplicationName;
- (NSString *)destinationApplicationName;
- (NSDataLinkManager *)manager;
- (NSDate *)lastUpdateTime;
```

```
- (NSDataLinkNumber)linkNumber;
- (NSDataLinkDisposition)disposition;
@end

@interface NSSelection : NSObject <NSCopying, NSCoding>
+ (NSSelection *)emptySelection;
+ (NSSelection *)allSelection;
+ (NSSelection *)currentSelection;
- (NSData *)descriptionData;
- (BOOL)isWellKnownSelection;
@end
```

Note that this API is a subset of the previously provided ObjectLinks API, with one exception: The **currentSourceFilename** method has been added to NSDataLink to allow the application to query the current location of the source file (whereas the **sourceFilename** method returns the original name).

-NSShowPS

On Windows NT, when apps are run with the arguments "-NSShowPS YES", they will produce a dump of their PostScript output in the file **$TEMP\showps.txt** (where $TEMP is an environment variable usually containing "C:\temp".

NSColor

The following methods have been added to NSColor:

```
@interface NSColor (NSDynamicSystemColors)          // OpenStep/Mach:    Windows Name:
+ (NSColor *)controlShadowColor;                    // darkGrayColor     COLOR_3DSHADOW
```

```
+ (NSColor *)controlDarkShadowColor;              // blackColor        COLOR_3DDKSHADOW
+ (NSColor *)controlColor;                        // lightGrayColor    COLOR_3DFACE
+ (NSColor *)controlHighlightColor;               // lightGrayColor    COLOR_3DLIGHT
+ (NSColor *)controlLightHighlightColor;          // whiteColor        COLOR_3DHILIGHT
+ (NSColor *)textColor;                           // blackColor        COLOR_WINDOWTEXT
+ (NSColor *)textBackgroundColor;                 // whiteColor        COLOR_WINDOW
+ (NSColor *)selectedTextColor;                   // blackColor        COLOR_HIGHLIGHTTEXT
+ (NSColor *)selectedTextBackgroundColor;         // lightGrayColor    COLOR_HIGHLIGHT
+ (NSColor *)controlBackgroundColor;              // lightGrayColor    COLOR_WINDOW
+ (NSColor *)controlTextColor;                    // blackColor        COLOR_BTNTEXT
+ (NSColor *)selectedControlTextColor;            // blackColor        COLOR_HIGHLIGHTTEXT
+ (NSColor *)selectedControlColor;                // whiteColor        COLOR_HIGHLIGHT
+ (NSColor *)scrollBarColor;                      // grayColor         COLOR_SCROLLBAR
+ (NSColor *)gridColor;                           // grayColor         COLOR_3DFACE

@end
```

These colors should be used by developers who want to create custom controls or subclass existing controls which honor the user's color preferences. For example, where

```
PSsetgray(NX_LTGRAY);
```

was used,

```
[[NSColor controlColor] set];
```

is now used.

These colors belong to their own color space, namely:

```
extern NSString *NSDynamicSystemColorSpace;
```

Since these colors may change dynamically, they do not respond to the component accessor methods declared in NSColor.h. To extract the components of a system color, you must use NSColor's **colorUsingColorSpaceName:** method to convert the color to a color space known to respond to the component accessor methods you need.

The following notification has been added which is sent when the system colors have been changed (such as through a system control panel interface):

```
extern NSString *NSSystemColorsDidChangeNotification;
```

The default **NSCMYKAdjust** is now YES by default. This basically enables the WindowServer's ability to use an enhanced algorithm for rendering device-dependent CMYK colors on an RGB device, resulting in a greater accuracy in the on-screen appearance of CMYK colors.

Applications can turn this off by registering this default with a NO value at startup time.

Platform specific resources

Under PR1 and PR2 including platform-specific versions of resource files (nibs, for instance), required Makefile kludgery and/or source changes. This is no longer necessary.

NSBundle has been changed so that when it fetches a resource, it will look in the directory where the resource was found one additional time to check for a platform-specific version. The platform specific versions are indicated by appending "*-platformname*" to the base name of the resource. Valid platform names include **winnt** and **nextstep**.

For example, to include a Windows-specific version of Doc.draw in your project, simply add it as Doc-winnt.draw. The call [bundle pathForResource:@"Doc" ofType:@"draw"] will find the latter under OPENSTEP for Windows. Similarly a Windows-specific version of Edit.nib can be added as Edit-winnt.nib.

Note that ProjectBuilder also provides ways to specify platform-specific app and document icons, paths, and build options.

NSMenu

The OpenStep spec was amended to make NSMenu a subclass of NSObject, not NSPanel, and to replace NSMenuCell with the NSMenuItem protocol. NeXT's OPENSTEP products also provide an NSMenuItem class which conforms to the NSMenuItem protocol.

NSMenu and NSMenuItem provide almost all of the functionality of the old classes, but some window and cell specific features are no longer available. This change was motivated by the very different ways menus are handled on different operating systems (such as Microsoft Windows). See the headers for more detail.

The NSWindow and NSView have additional support for menus to support Microsoft Windows conventions. Each NSWindow can have its own menu bar or no menu bar at all. By default NSWindow objects have the application's main menu as their menu bar while NSPanels have no menu bar. Use -[NSWindow setMenu:] to set a different NSMenu to be used as the menu bar in an NSWindow. **setMenu:** in NSWindow only has an affect in OPENSTEP for Windows since on Mach, windows do not have menu bars, but the API does exist on both platforms.

NSViews can now have menus associated with them as well. An NSView's menu is used as a contextual popup menu accessed by right-clicking the view (this should be a familiar feature to users of Windows 95). By default an NSView has no contextual menu, but some AppKit objects will provide default context menus. Currently only NSTextView (and

therefore also editable controls) have a default menu, but this is likely to change. You can set a menu to be used as the context menu for a view through -[NSView setMenu:]. Context menus currently only work on OPENSTEP for Windows.

Setting a menu for either a window or view on Mach has no affect, so there is no need to do this conditionally inside an #ifdef WIN32 or anything. Context menus for views may even be supported on Mach in a later release.

Interface Style

Some aspects of the new support for multiple interface styles (or UIs) have been made public.

The header NSInterfaceStyle.h is now public and can be used by developers writing classes that need to tailor their appearance to the current interface style (usually NSControl or NSView subclasses, but not always). The function **NSInterfaceStyleForKey()** can be used to determine which style is in effect at a given time. This API should be used instead of relying on #ifdef WIN32 so that your drawing will be correct in all cases. For instance, InterfaceBuilder lets the user specify what interface style to show when editing nibs so that you can see what your interface will look like on any supported platform without actually having to be running on that platform.

The default **NSInterfaceStyle** can be used to control what style to use. By default the native interface style will be used, but you can override that with this default. Currently supported values are **Windows95** and **Mach**. Not everything can obey this setting, but most AppKit objects do.

NSPopUpButton

NSPopUpButtons no longer have NSButtonCells for items, the items in a popup now have the type **(id <NSMenuItem>)**. Also, The -**(NSMatrix *)itemMatrix** method has been replaced with **-(NSArray *)itemArray** which returns an NSArray instead of an NSMatrix.

NSButton and NSButtonCell

The **setType:** method in NSButton and NSButtonCell has been renamed to **setButtonType:**, because the old name overloaded a method inherited from NSCell.

NSMatrix

Two new convenience methods have been added to NSMatrix:

```
- (int)numberOfRows;
- (int)numberOfColumns;
```

NSSavePanel

NSSavePanel now inherits from NSObject in the OpenStep specification to gain greater portability. This means that although on Mach NSSavePanel is implemented as a subclass of NSPanel, methods inherited from NSPanel and NSWindow are not considered part of the spec. In fact, on Windows NSSavePanel is implemented as subclass of NSObject since we are using the native UI.

A new method has been added to the SavePanel to validate the visible columns. For instance, say your filtering criteria have changed. Call this method to get the correct files displayed:

```
- (void)validateVisibleColumns;
```

NSScrollView

Methods have been added to access the content view of the ScrollView. The content view is the scroll view's clip view:

```
- (void)setContentView:(NSClipView *)contentView;
- (NSClipView *)contentView;
```

Windows application lifetime issues

Applications under Windows introduce some interesting life cycle issues. One issue is that because the menus are only present inside of windows, an application with no open windows has no UI. Another issue is that when you double-click a file, Windows generally starts a fresh copy of the associated application to open it even if there's another copy already running. In OPENSTEP we have addressed these issues by defining some default behaviors and providing some new API to ensure that developers can control the behaviors if necessary.

By default, on Windows, an Application will terminate when the last menu-bearing window is closed. In general, this is the right thing to do. Sometimes, however, you may need to do something different. A new NSApplication delegate message has been added to allow you to override this default behavior. When NSApplication notices the last menu-bearing window has been closed, it will send **-(BOOL)applicationShouldTerminateAfterLastWindowClosed: (NSApplication *)sender** to the application's delegate if it responds. This message is sent on both Mach and Windows so it may be necessary to tailor your app's response for the platform. On Mach, application do not, by default, terminate when windows are closed. If you implement this method you should return YES if the application should terminate and NO if it should not. If you return YES you will shortly receive an -applicationShouldTerminate: messageas well. If you return NO you should also have arranged for some UI to remain. If you return NO and do nothing else, you are likely to lose all UI to your app and leave the user with no way to quit or do anything else.

On the other end of the life cycle, a new Windows-only feature has been added to NSApplication. If the arguments to the application on the command line includes the argument pair "-NSUseRunningCopy YES" then the app will check to see if a copy of itself is already running, and if it is, it will hand off to the running copy and exit. Any -NSOpen or

-NSPrint requests will be forwarded on to the existing copy. When file associations are made for OpenStep apps, these arguments are included in the association. Therefore, double-clicking multiple .draw documents should end up opening them in a single copy of Draw. This feature is supported through a single NSApplication method. **+ (void)useRunningCopyOfApplication** is called from the default implementation of the WinMain function. Apps that never want to use this feature can implement their own WinMain and avoid calling this method, but there should be very few good reasons to want to do that.

NSPasteboard and the NSFileContentsPboardType

NSFileContentsPboardType has been modified slightly to be the same as NSFileWrapper's serializedRepresentation. We expect this change to make the type much more useful. Among other things this means that NSFileContentsPboardType and NSRTFDPboardType use identical formats. There is still a semantic difference between the two, however. NSRTFDPboardType is used only for RTFD text and it must satisfy certain constraints on the actual file contents written (eg there must be a file inside called TXT.rtf and so forth).

In support of this change, and to make it easier to use NSFileContentsPboardType in your applications, NSPasteboard has some new API for dealing with this type. In the past you had to write this type onto the pasteboard from the filesystem. And when you read it from the pasteboard you had to write it to the file system. Now it is possible to wirte this type from an NSFileWrapper instance and to read it out and get a new NSFileWrapper instance. The disk need not be involved anymore. The new API is:

```
- (BOOL)writeFileWrapper:(NSFileWrapper *)wrapper;
- (NSFileWrapper *)readFileWrapper;
```

Notification name changes

Some of the names of AppKit notifications have changed to follow the general naming guidelines.

Old Name	New Name
NSColorListChangedNotification	NSColorListDidChangeNotification
NSColorPanelColorChangedNotification	NSColorPanelColorDidChangeNotification
NSImageRepRegistryChangedNotification	NSImageRepRegistryDidChangeNotification
NSViewFocusChangedNotification	NSViewFocusDidChangeNotification
NSViewFrameChangedNotification	NSViewFrameDidChangeNotification
NSViewBoundsChangedNotification	NSViewBoundsDidChangeNotification

NSDragOperation removed

The **NSDragOperation** typedef has been removed from the API, and all uses are replaced by **unsigned int**, since the values used for this type must be often or'ed together.

NSPageLayout

NSPageLayout now inherits from NSObject in the OpenStep specification to gain greater portability. This means that although on Mach NSPageLayout is implemented as a subclass of NSPanel, methods inherited from NSPanel and NSWindow are not considered part of the spec. In fact, on Windows NSPageLayout is implemented as subclass of NSObject since we are using the native UI.

In addition, the following methods and types have been removed from the OpenStep specification, but are still available on the Mach platform:

```
- (void)convertOldFactor:(float *)old newFactor:(float *)new
- (void)pickedButton:(id)sender.
- (void)pickedOrientation:(id)sender
- (void)pickedPaperSize:(id)sender
```

```
- (void)pickedUnits:(id)sender

enum {
       NSPLImageButton,   NSPLTitleField,
       NSPLPaperNameButton,
       NSPLUnitsButton,
       NSPLWidthForm,
       NSPLHeightForm,
       NSPLOrientationMatrix,
       NSPLCancelButton,
       NSPLOKButton
};
```

The **runModal** and **runModalWithPrintInfo:** methods now return NSOKButton or NSCancelButton on all platforms.

NSPrintInfo

There are two new keys available in the NSPrintInfo dictionary on the Windows platform:

```
NSPrintMustCollate
NSPrintFormName
```

NSPrintMustCollate indicates whether collation occurs when multiple copies are printed. NSPrintFormName is the form name associated with the current print request, which might be different from the NSPrintPaperName.

NSPrinter

A new method has been added to return the names of configured printers available to the system. This name can then be passed to **printerWithName:** to create an NSPrinter. If no printers are available, an empty array is returned. If an error occurs, nil is returned.

+ (NSArray *)printerNames

NSPrintPanel

The NSPrintPanel class has been removed from the OpenStep specification. In addition, these related methods in NSPrintOperation have been removed from the specification :

```
- (void)setPrintPanel:(NSPrintPanel *)panel;
- (NSPrintPanel *)printPanel;
```

NSPrintPanel is still available as a platform specific interface; however, it now inherits from NSObject rather than NSPanel.

NSPrintOperation

The following methods were added to NSPrintOperation in the OpenStep specification to allow the accessory view to be set in the print panel:

```
- (void)setAccessoryView:(NSView *)aView;
- (NSView *)accessoryView;
```

The following methods have also been added to the specification. These methods are sent to the accessory view passed to the above methods if it responds to them.

```
- (void)updateFromPrintInfo;
- (void)finalWritePrintInfo;
```

New Graphics functions

Four new functions have been added to support conformance to the Windows UI look:

```
void NSRectFillListWithColors(const NSRect *rects, NSColor *colors, int count);
```

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

```
void NSDrawDarkBezel(NSRect aRect, NSRect clipRect)
```

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

```
NSRect NSDrawColorTiledRects(NSRect boundsRect, NSRect clipRect, const NSRectEdge *sides,
                        NSColor *colors, int count)
```

Draws an unfilled rectangle, clipped by *clipRect*, whose border is defined by the parallel arrays *sides* and *colors*, both of length *count*. Each element of *sides* specifies an edge of the rectangle, which is drawn with a width of 1.0 using the corresponding color from *colors*. If the *edges* array contains recurrences of the same edge, each is inset within the previous edge.

```
void NSDrawLightBezel(NSRect aRect, NSRect clipRect)
```

Draws a rectangle with the appearance of an editable text field. Only the area that intersects *clipRect* is drawn.

NSFont

The method **maximumAdvancement** was added to the API to return the advancement of the widest glyph in the given font. If the font is fixed-pitch, this is the advancement of all the glyphs.

The following API additions (which are not in OPENSTEP) allow one to get/set a list of the user's "preferred" fonts, which are checked (in order) before any other fonts when searching for a font to render some character that cannot be rendered in a pre-given or specified font. The list should typically contain fonts with different encodings and glyph complements, as available.

```
+ (NSArray *)preferredFontNames;
+ (void)setPreferredFontNames:(NSArray *)array;
```

API has also been added to access the Windows system fonts.

```
+ (NSFont *)menuFontOfSize:(float)fontSize;
+ (NSFont *)titleBarFontOfSize:(float)fontSize;
+ (NSFont *)messageFontOfSize:(float)fontSize;
+ (NSFont *)paletteFontOfSize:(float)fontSize;
+ (NSFont *)toolTipsFontOfSize:(float)fontSize;
```

The existing NSFont methods systemFontOfSize: and boldSystemFontOfSize: now return the same as messageFontOfSize: and titleBarFontOfSize: respectively. This API is valid on Mach and Windows. On Mach it returns the standard system or bold system fonts. On WIndows it will return the user's font settings from the Display options control panel if the user's chosen fonts can be supported (currently bitmap fonts are not supported, but TrueType fonts are).

NSFontManager

NSFixedPitchFontMask mask was added so that the font manager can determine whether or not a font is fixed-pitch. It may be used in trait queries.

The method **fontNamed:hasTraits:** was added to help in searching for fonts with specified traits. The available traits are specified by the trait masks.

The new method **availableFontNamesWithTraits:** uses the given mask (which may contain several one bits) in testing the available fonts. The names of all fonts matching the given bits are returned in an array.

NSInputManager/NSInputServer

The method **doCommandByName:** was removed from the API and from the NSTextInput protocol in favor of **doCommandBySelector:**. (The latter was already in the API, but has been added to the NSTextInput protocol.) Along the same lines the NSInputServer method **doCommandByName:sender:conversation:** was also obsoleted.

The method **localizedInputManagerName** replaces the method **inputManagerName**.

Dead Keys

Dead key handling in DPSClient is now off by default. The new text object and keyboard UI rely on the input management layer to simulate dead keys. NSCStringText will turn dead key handling on and off as it gains and loses first responder status. Other custom views which need dead keys to be handled by DPSClient should also use the same approach.

Help System

The AppKit's Help API has changed significantly. NSHelpPanel has been obsoleted in favor of a new class,

NSHelpManager. NSHelpManager provides a more platform-independent approach to presenting help.

Previously, the only way for your application to present help was through the NEXTSTEP Help panel. Context-sensitive help and all other help for your application was displayed in the Help panel. This made sense when NEXTSTEP was the only platform for your application.

OpenStep applications, however, can run on multiple platforms, and each platform provides its own support for online help. It's important to users that applications use the native online help system (on Windows, for instance, users want the Windows help system and don't want to have to learn how to use a different help system), so NSHelpManager does not provide a comprehensive solution for presenting help. Instead, it provides cross-platform support for context-sensitive help, and allows you to present more comprehensive help (conceptual and task-based help) in any way you choose.

Context Help

Context-sensitive help (also referred to as context help) gives the user a small amount of information when they help-click on an interface item. For example, if the user help-clicks on a menu item called "Copy", they should get context help that says something like "Copies the currently selected text to the pasteboard." This text appears in a small window near where the user help-clicked, and the window disappears when the user clicks anywhere else in the application.

To provide context help for your application, follow these steps:

1. For each interface item that needs context help, create an .rtf or .rtfd file containing the text and any images you want to appear when the user help-clicks on that interface item. Try to keep the text as brief as possible and the images as small as possible.

2. If you don't need to localize your context help files, in Project Builder simply add these files to the Context Help suitcase of your project.

   If you do need to localize your context help files, first copy the files into the appropriate **.lproj** directory of your project, then use Project Builder to add them to the Context Help suitcase of your project.

3. In Interface Builder, connect each interface item to its context help file by doing the following:
   a. Bring up the Interface Builder inspector and choose the Help display. The Help display lists all the context help files associated with your application.
   b. Select an interface item.
   c. In the inspector, choose the appropriate help file.

When you compile your application, **/usr/bin/compileHelp** packages your help files into a property list named Help.plist. NSHelpManager knows how to extract context help from an Help.plist file.

Comprehensive Help

Most applications provide some form of online help that is more comprehensive and detailed than context-sensitive help, such as conceptual or task help. NSHelpManager allows you to provide this sort of comprehensive help in any way you choose. Some help authors prefer to provide comprehensive help in HTML using a World-Wide Web browser; others use tools such as Digital Librarian or Concurrence; on Windows a full-featured native help system is available. Given the availability of rich tools like Digital Librarian and HTML browsers, OpenStep on Mach no longer supplies a native help system for comprehensive help.

When the user chooses the Help menu item, NSHelpManager simply asks the Workspace Manager to open the help file

you have specified for your application. That file should be the starting point of your help, and should allow users to access whatever information they might need.

To specify a help file for your application, do one of the following:

- The simplest approach, not requiring you to specify the help file anywhere in your application, is to place the help file in your app wrapper and name it after your application. If you haven't specified a help file, NSHelpManager looks in the app wrapper for an appropriately named file.

  On Mach, it must be an rtf file called <appName>.rtf.

  On Windows, it must be a Windows help file called <appName>.hlp.

- To specify a help file that applies to all platforms, define the NSHelpFile key in your project's CustomInfo.plist file:

  ```
  {
      NSHelpFile = "ApplicationHelp.rtfd";
  }
  ```

  If your project does not have a CustomInfo.plist, simply create one (as in the example), and add it to your project's "Supporting Files" file category. When you build your project, the contents of CustomInfo.plist will be merged into your application's Info.plist

  The specified value for the NSHelpFile key can be a full or relative path, and if it is relative, it is assummed to be a bundle resource (resolved using [[NSBundle mainBundle] pathForResource:<NSHelpFile value> ofType:nil]).