TextEdit

This directory contains the source code for the TextEdit demo, which is a simple text editor based on the 4.0 text system.

Changes since 4.1

· Changed #ifdef WIN32 in Document.m to more dynamic calls to NSInterfaceStyleForKey()
· Added support for the new hyphenation feature in the text system
· Got tab stop locations in plain text mode to be recomputed when needed

Major source files and what's interesting about them:

Document.m

Contains most of the implementation for the **Document** class. One instance of this is created for every document (new or saved) in TextEdit. The internal designated initializer for this class is **init**; however, external clients should go through **initWithPath:encoding:uniqueZone:**; or better yet **openUntitled** or **openDocumentWithPath:encoding:**.

The text contents (characters, attachments, attributes) of the document are kept in an instance of NSTextStorage. There is also one NSLayoutManager, and one or more NSTextViews (depending on whether

"wrap to page" mode is selected). **setHasMultiplePages:** determines whether the document is in "wrap to page" mode or not; study this method, **addPage**, and **removePage** to see how to create and manipulate NSTextViews programmatically.

setRichText: demonstrates what needs to happen when you want to convert an attributed string from being "rich" to "plain" and back. Note that in the new text system setting setRichText: on a text view simply prevents the user from manipulating attributes; however, the backing store can still contain multiple fonts, colors, etc. This is why this method needs to do the things it does.

As you will note, there is a good deal of code to deal with encoding of the characters in the document when the document contains plain text. The instance variable **encodingIfPlainText** stores the encoding of the document; this is either deduced from the file or specified by the user when the document is opened. Keeping this encoding around allows the document to be saved with the same encoding as it was read. (When in memory the character encoding of the document is somewhat meaningless, because the characters in the document are stored in an NSString, whose backing stores are always expressed in terms of Unicode characters. The encoding determines how to save the document when saved as plain text.)

The encodings recognized by NSString are listed in NSString.h; the encodings listed in the open panel accessory view are listed in the resource file **Encodings.txt**. TextEdit uses three additional values for encodings to denote unknown encoding, RTF, and RTFD; this is somewhat of a hack, but the chosen values are unlikely to be used as encoding values in the system.

Document class contains an instance of NSPrintInfo to store the default settings for printing. Setting horizontal

pagination to **NSFitPagination** allows the text to be printed with the same wrapping as on the screen. In non-"wrap to page" mode this means the text might need to be scaled smaller when printed.

The method **doForegroundLayoutToCharacterIndex:** shows how to get the text system to lay text out in the foreground up to a certain character location. By default the text system does its layout in the background, which allows bringing up the window fairly quickly. The user can even edit, print, or save the document while the background layout is going on. Although this is rather interesting in practice, having the scrollbar race down the page when the document first comes up can be confusing to some users. Thus TextEdit causes a portion of the document (by default the first 100,000 characters) to be laid out before the document is displayed. For documents that are larger the layout will still continue in the background once the document is displayed. Note that the user can always cause a full layout to happen by hitting the "End" key on the keyboard.

There are a couple of instances of Windows-specific code in Document.m. Both are concerned with tweaking the look of the document scrollview to match the Windows guidelines. Note that rather than using a compile-time default (#ifdef WIN32), we look at the interface style to dynamically match the look & feel.

Features poorly implemented in Document include **saveTo:** (which would save the document elsewhere without changing its current path) and dynamically changing the tab stops in plain text documents on font changes (this should really be handled by the text system at some point). It would also be great (and easy!) to add undo.

One final note on the Document class: The example program **Yap** has a cleaner Document, which Yap

subclasses to add Yap-specific functionality. If you need a very simple document class to handle text documents, you might want to check out Yap's Document.m.

DocumentReadWrite.m

This file contains the file I/O code, including a few convenience methods on NSAttributedString. The method **loadFromPath:encoding:** opens the specified file using the specified encoding, and the method **saveToPath:encoding:updateFilenames:** saves the document using the specified name and encoding.

Controller.m

This file contains the central controller object for TextEdit, which responds to application messages such as openFile:, printFile:, etc, and also implements the info panel.

A piece of Windows-specific code in this file assures that when the users double-click on a TXT.rtf within a .rtfd directory, the .rtfd is opened instead of the TXT.rtf. This gets around the fact that we cannot (yet) allow opening directories as documents from the Explorer.

MultiplePageView.m

In "wrap to page" mode there is one NSTextView per page. **MultiplePageView** is the top level view which groups all of these views. It is inserted as the document view of the scroll view in the document window.

MultiplePageView is fairly simple, providing support for conversions between page numbers and rects, and drawing the background for the pages.

A possible enhancement to this class would be to have it allow the user to manipulate the page margins by dragging guides around. An advanced exercise would be to add custom markers to the ruler to allow changing the page margins via the ruler as well.

ScalingScrollView.m

Contains **ScalingScrollView**, a subclass of NSView to implement a scroll view with a popup to allow setting the zoom factor. This class is fairly generic and can easily be used in a variety of cases.

TextFinder.m

Along with the corresponding nib file **FindPanel.nib** implements basic replace and find functionality on text stored in an NSTextStorage. Among other things demonstrates communicating with the pasteboard to get/set the find string.

Preferences.m

Provides a preferences controller to read/write preferences stored in the defaults database. Provides support for specifying default values for the preferences, and tries to minimize the amount of information written out to

the database by removing entries which have default values.

Note that the preferences panel in TextEdit does not have an "OK" button to confirm the changes; changes take place immediately. However the Preferences class supports a panel with OK/Revert buttons; simply connect the OK button to **ok:**, and the Revert button to **revert:**.