3.0 Enhancement Checklist

(or "How to be 3.0 Hip")

When you update your application to Release 3.0, the following is a list of features you should strongly consider adding to your application to take advantage of the new 3.0 feature set. This information is provided as a checklist and primarily consists of features *every* 3.0 application should probably support. There are many other new and exciting 3.0 features (such as Distributed Objects, 3DKit, DBKit, Indexing Kit) that are not mentioned here because they are relevant to a smaller set of applications.

- **Object links**. Every application running under NEXTSTEP 3.0 should take advantage, if at all possible, of the powerful ability to link external data or files into the documents of an application. Likewise a document- or data-oriented application should support exporting data links. Linking is performed using the familiar Copy/Paste metaphor for data and the Drag-n-Drop metaphor for filesÐprogrammers should consider implementing both forms of object links. Bundled applications that import data links are Edit and Draw. Bundled applications that export data links are Draw, IconBuilder, and Graph. See the 3.0 Examples Graph and Draw, the AppKit and ObjectLinks Release Notes, and the NXDataLink spec sheets for more information on how to implement object linking.

- **Filter services**. A Filter service is a new 3.0 type of Service which has no menu item but supports the ability to convert a piece of data from one type to another. Every application can take advantage of filter services when opening files (via **+typesFilterableTo:** if it can open a standard type, like RTF) and when importing data via copy/paste or dragging (via **+imagePasteboardTypes**).

  Other applications can provide filter services: for example, if you have some code

which converts your document format [the *WonderFormat*] to RTFD, then you should provide this functionality as a filter service. Another application which understands RTFD but not *WonderFormat* can still import *WonderFormat* documents and the filter service will do the work of the conversion. The other advantage of providing a filter to ascii is that your documents then become indexible by Digital Librarian. See the 3.0 AppKit Release Notes and the Pasteboard specification sheet for more information on: **+typesFilterableTo:**, **+newByFilteringFile:**, **+newByFilteringData:**, +**newByFilteringTypesInPasteboard:**.

- **New dragging mechanism**. The drag and drop mechanism is new and improved in 3.0. Under 2.x, drag and drop was implemented using the **registerWindow:**, etc. methods. (These methods will continue to work in 3.0 but are being obsoleted.) For 3.0, a new Workspace Protocol has been added which supports interapplication dragging in View and Window classes and should actually simplify your code. The new dragging mechanism can be used both for *inter*-application dragging and *intra*-application dragging (as in palettes). See the 3.0 AppKit Release Notes and the View, Window and NXDraggingDestination spec sheets for more information on (to name some of the methods involved): -**registerForDraggedTypes:**, -**draggingEntered:**, -**draggingUpdated:**, -**draggingExited:**, -**prepareForDragOperation:**, -**performDragOperation:** and -**concludeDragOperation:**.

  - **Dragging TIFF and EPS [i.e. image] data**. As a side note, any application which supports drag and drop of TIFF or EPS should (instead of registering for ".tiff" and ".eps" explicitly) register for **[NXImage imagePasteboardTypes]**. This method returns a NULL terminated list of strings and allows your application to take advantage of other image types that NXImage knows how to

convert between, such as the RIB format (if libMedia is linked with your application) or other formats supported via the filter services mechanism. See the AppKit Release Notes and the NXImage spec sheet for more information on +**imagePasteboardTypes**. (Note: all applications that import data via NXImage should start linking with libMedia.)

- **Dragging colors**. Another side note: applications should no longer use the **-acceptColor:atPoint:** method for accepting colors. (The **-acceptColor:** method will continue to work for 3.0 but should be avoided.) Dragging colors should be implemented via the normal drag/drop protocols with the NXColorPBoardType. See the 3.0 AppKit Release Notes for more information on **-acceptColor:** and NXColorPBoardType.

- **Undo**. Undo is a very powerful mechanism that we strongly encourage all 3.0 applications to support. To this end, the Draw Example (in /NextDeveloper/Examples/Appkit/Draw) illustrates a whole framework for supporting Undo. Draw has three subprojects which implement Undo and two of these subprojects have been designed so that they can be added to an existing application with minimal effort: a reusable Change class, a reusable UndoText class, and a class specific to Draw. See the UndoDoc.rtf and UndoREADME.rtf files in the Draw Example for more information.

- **Help**. The Help facility is another powerful user mechanism that has been integrated into InterfaceBuilder for 3.0 that every application should support. The easiest way to implement Help in your application is to use the Help panel to display any help files you may have already written. The next level of implementation involves attaching help files to each of the buttons, menus and other controls in your application. For more information on how to implement

Help, see the 3.0 InterfaceBuilder and AppKit Release Notes, the NXHelpPanel spec sheet and the Draw Example.

- **Device independent color: calibrated RGB and Pantone**. Colors should be set via the **NXSetColor()** function and should use NXColor for color storage. (Colors should *not* be set via **PSsetrgbcolor()** if you wish to take advantage of device independent color.) Applications using **NXSetColor()** and NXColor can take advantage (for free) of the calibrated RGB color space in Release 3.0 which emits device independent colors to all Level 2 devices. See the 3.0 AppKit and WindowServer Release Notes for further information on **NXSetColor()** and the NXColor spec sheet.

- **Restoring Windows to previous location and size**. Persistent windows can now take advantage of the new Window methods which allow saving and restoring a window by name. These methods use the defaults database to store the position and size of the window so that when you recreate the window it will adopt the same characteristics. See the 3.0 AppKit Release Notes and the Window class specification sheet for more information on: -**saveFrameUsingName:**, -**setFrameUsingName:**, -**setFrameAutosaveName:**, -**frameAutosaveName** and -**removeFrameUsingName:**.

- **Creating a Workspace contents inspector**. You can now create your own contents inspector which Workspace can use when one of your documents is selected. See the 3.0 IntroWorkspace.rtf document and the WMInspector class spec sheet for more information on building an inspector module.

- **Find text object**. The ability to find text has been added to the Text object making it much easier to implement a text search panel. See the Text class

specification sheet for more information on **-findText:**.

- **RTFD**. The Text object now supports reading, writing and editing the RTFD file format. (Under 2.x the Text object only supported reading and writing of straight ascii and RTF files.) See the 3.0 AppKit Release Notes and the Text class spec sheet for more information on (to name a few of the methods): **-setGraphicsImportEnabled:**, **-saveRTFDTo:**, **-openRTFDFrom:** and **-writeRTFDTo:**.

- **Localization**. While making an application localizable to other languages was encouraged under the 2.x release of NeXTSTEP, it is now easier to implement in 3.0 with additional support in InterfaceBuilder and ProjectBuilder. Under 3.0, any application created via ProjectBuilder will automatically bundle the nib files, binary and other necessary files into an NXBundle (which looks suspiciously like an app wrapper in days of yore). All methods in the AppKit which used to look in the mach-O section to find something, now look in **[NXBundle mainBundle]** if the desired object is not in the mach-O section. See the 3.0 Localization Chapter of the Concepts Manual for more information on how to make your application localizable for 3.0. (Note: Localizing an application is different than making that application localizable. Localizing involves the actual translation while making it localizable puts the framework in place so that it can be translatedÐusually by someone other than the programmer.)

- **Keyboard key codes are not portable to other platforms**. The keyCode information (available in the event data for key events) is device-dependent for a particular keyboard (and is documented as such). NeXT recently started shipping a keyboard with new key codes and any application which relied on the old raw key codes will almost certainly perform differently. In order to keep software as

portable to as many platforms as possible, an application should avoid examining raw key codes at all costs. (It is safe to use charSet and event flags along with charCode to distinguish between key events.) To quickly determine if your application uses raw key codes, search your source files for the string "**keyCode**".

- **Copying/moving/compressing/destroying files via the Workspace**. Any application which supports copying or moving files around (or compressing or destroying files, to name some of the supported functions) should use the new Workspace Protocol method **-performFileOperation:**. This method allows you to ask the Workspace to perform an operation such as copying or moving a file and will update the Processes panel with the Background process information and the animated pie chart. See the 3.0 AppKit Release Notes and NXWorkspaceRequestProtocol specification sheet for more information on **-performFileOperation:** and the list of operations it supports.

QA845

Valid for 3.0, 3.1