Improving Launch Times of the Application and Panels

Be Lazy

Often launch times can be dramatically improved by taking a lazy approach to the creation and initialization of data structures and objects in your application.   That is, don't spend launch time creating and initializing objects unless they're central to the working of your application, or unless they'll benefit the typical user and you want to optimize response time.   Panels are a classic example of this.

Panels should always be put in separate Interface Builder files so that interface objects are loaded on demand rather than at startup time.   Even if the panel is deferred (that is, the creation of the PostScript window is deferred until the window is about to become visible), the time required to read in the interface data and do initialization is substantial.   For example, Preferences panels or Info panels should never be in your "main" nib file.

In many instances it may make sense to create a separate class that owns a panel and is the target for the panel's controls.   In this case the interface is typically loaded within the class method responsible for creating an instance of the class.   The OpenPanel, SavePanel, and PageLayoutPanel classes all use this technique.   Once again, create the object only when you absolutely need to.

Interface Builder associates a name with every object in an interface file.   When **loadNibSection:owner:** is invoked, the names are read in along with the objects themselves.   Subsequently, **NXGetNamedObject()** can be used to return the **id** of an object associated with a given name.   However, if you don't need this functionality, you can save time and memory by not having the names read into memory.   If you use **loadNibSection:owner:withNames:** rather than **loadNibSection:owner:** and pass NO as the argument for **withNames:**, the names will not be read in.

Another consideration for panels is whether to make them **one-shot**. If a window is one-shot, the underlying PostScript window is freed when the panel is closed. On a color system, this is a substantial amount of memory to be reclaimed. With the drawing speed of today's CPUs, only panels that are very complex and take a long time to draw should not be one-shot. For example, all panels in the Application Kit are one-shot.

Procedure Reordering

Procedure reordering (sometimes called "Scatter Loading") is the process of reordering the functions and methods of your application to reduce the number of virtual memory pages required for a given operation. For example, if you reorder all your application's initialization code so that these routines are contiguous instead of spread throughout your application's text segment, fewer pages of your application's code will need to be loaded at launch time. This also has the additional benefit that these once-used routines can then page out together, instead of being mixed with other routines that are used later. This reduces the working set of your application during subsequent use. For more information, see the performance note on "Link Optimization".