CellScrollView

by R. Dunbar Poor and Mai Nguyen, NeXT Developer Support

CellScrollView is a simple example that installs a matrix of custom cells (FooCells) in a subclass of ScrollView (CellScrollView). This forms a basis for a user interface that lets the user add and delete objects (FooObjects) from a list. The list is maintained separately from the scrollView.

Files in CellScrollView:

CellScrollView.m	A subclass of ScrollView, which has a matrix installed as its document at initialization time.
Controller.m	The basic user interface. Lets the user add new fooObjects to the list and delete fooObjects whose corresponding fooCells are selected in the CellScrollView.
FooCell.m	A simple custom cell. Each FooCell "owns" a fooObject and displays the fooObject's contents.
FooObject.m	A really simple data structure for demonstration purposes only.

Key Points:

CellScrollView demonstrates several points:

How to set up a ScrollView to contain a matrix of custom cells. The ScrollView class is complicated and subtle, and it's not always obvious how to get everything set up to do resizing, autoscrolling, etc. The file CellScrollView.m demonsrates how to install a matrix of custom cells in the scrollView and get everything set up properly.

How to make a custom cell. The FooCell class is very simple, but it shows you the basics. A more advanced example might associate an action with each cell.

The benefits of separating data structures from the user interface. In this example,

the list of FooObjects is maintained entirely separately from any aspect of the user interface. This approach lets you separate the design of the user interface from the layout of the underlying data structures, which can become very important in a larger programming project. For example, you could create additional ways of viewing the list of FooObjects (not just the FooCells given here). You can read and write the list of FooObjects to a file without the overhead of reading and writing views, cells, and windows.

An effective way to use the renewRows:cols: (Matrix) method. Using the renewRows:cols method can reduce the number of calls to alloc and free. When you decrease the number of cells displayed in a matrix via a call to renewRows:cols:, no cells are actually deallocated. Instead, you'll get those "recycled" cells back at subsequent calls to addRow (or addCol). But there are subtleties: you must make sure that these recycled cells are unhighlited and unselected before you use them. The loadCellsFrom: method in CellScrollView.m shows you one way to handle this.

The advantage of a "nib-less" custom view. Since CellScrollView is generated entirely from code (not from Interface Builder), you can use it as a building block for other applications. For example, you could install a CellScrollView as one of the subviews of a SplitView. While you *could* use Interface Builder to instantiate a scrollview, it gets ugly. (You'd create a custom object with an outlet to a scrollView, instantiate the object and install its scrollView in the SplitView.)

Valid for 3.0