

# ScrollDoodScroll

by Jayson Adams

## Overview

This example demonstrates how to create a “scrollable matrix” — a scrollView containing a matrix of cells — as used by applications such as Mail and PrintManager. The application's controller object (of the Controller class) creates a matrix (of the NiftyMatrix class), fills it with cells (of the CustomCell class) and places the matrix within a scrollView. The example also shows how to place “rulers” and controls within a scrollView. WriteNow, for example, places rulers above its documentViews, constraining them to scroll horizontally with the document; when the user scrolls the document vertically, the rulers do not scroll out of sight. WriteNow also places a “scale” popup list in each document's lower right-hand corner, a good example of why you might want to place controls within a scrollView.

## Important classes within ScrollDoodScroll

### NiftyMatrix class

The NiftyMatrix class differs from the Matrix class in that a niftyMatrix allows the user to rearrange cells, as InterfaceBuilder lets you do with menu items. If a user control-clicks on a cell, that cell will follow the mouse as the user drags it, leaving a “well” in its place. When the user releases the cell over another cell in the matrix, the niftyMatrix places the control-dragged cell in that location and slides the other cells up or down in order to fill the vacant spot.

### CustomCell class

The CustomCell class demonstrates how, by overriding the `drawInside:inView:` method, you can display graphics and/or text at different locations within the cell.

### TileScrollView class

The TileScrollView class shows how overriding the `tile` method lets you add elements to a scrollView. The `tile` method places a popup list to the right of the horizontalScroller, as WriteNow does. It also places a ClipView instance above the contentView; this clipView has a RulerView instance as its documentView (the rulerView displays a TIFF image of a ruler). The tileScrollView implements the `scrollClip:to:` method, which lets it regulate the scrolling for its clipViews. The tileScrollView instructs its clipViews how to scroll their documentViews by sending

them the `rawScroll:` message.

**PostScriptView class**

The `tileScrollView` has a `PostScriptView` , which displays a sample PostScript image, as its `documentView`. The `postScriptView` images the PostScript code into an `NXImage` to speed scrolling: whenever the user scrolls the `postScriptView`, the `tileScrollView` asks the `postScriptView` to draw the portion that just became visible. With an `NXImage` of the PostScript image, the `postScriptView` can composite the `updateRects` from the `NXImage` instead of reexecuting the PostScript code that created the image. When the user asks that the image be scaled, the `postScriptView` redraws the PostScript into the `NXImage` (with the new scale, of course), and then composites the image into itself.

**Controller class**

The `Controller` class creates and initializes a `niftyMatrix`, fills it with cells and places the matrix within a `scrollView`. It implements the methods the matrix sends on single- and double-clicks. The controller also shows how to detect when a matrix has multiple cells selected. Lastly, it initializes the `tileScrollView`.

**Interesting Stuff**

**Controller**

How to detect that a matrix has multiple cells selected

**CustomCell**

How to draw whatever you want within a cell

**NiftyMatrix**

- How to implement autoscrolling
- Using off-screen image buffers for fast drawing

**PostScriptView**

Using `NXImage`

**TileScrollView**

How to place WriteNow-style popup lists within a `scrollView`

How to implement rulers (views that scroll along with the documentView)