Matrix Programming Guide for Cocoa

Cocoa > User Experience

2006-11-07

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Introduction to Matrices

NSMatrix is a class used for creating groups of NSCells that work together in various ways.

Organization of This Document

This programming topic contains the following articles:

- "About Matrices" (page 7) provides basic information about matrices
- "Matrix Selection Modes" (page 9) describes how the cells in a matrix behave when the matrix is tracking the mouse.
- "Managing the Matrix's Cells" (page 11) discusses how to add and remove cells programatically.
- "Setting a Matrix's Appearance" (page 13) discusses how to change the appearance of a matrix and its cells.

See Also

If you want to group several elements visually, see the *Boxes* programming topic.

About Matrices

`NSMatrix` is a class used for creating groups of `NSCell` objects (or simply, cells or cell objects) that work together in various ways. It includes methods for arranging cells in rows and columns, either with or without space between them. Cell objects in an NSMatrix are numbered by row and column, each starting with 0; for example, the top left cell would be at (0, 0), and the cell that's second down and third across would be at (1, 2).

The cell objects that an `NSMatrix` contains are usually of a single subclass of `NSCell`, but they can be of multiple subclasses of `NSCell`. The only restriction is that all cell objects must be the same size. An `NSMatrix` object can be set up to create new cell objects by copying a prototype object, or by allocating and initializing instances of a specific `NSCell` class. Cells created by or added to an `NSMatrix` are retained by the matrix.

An `NSMatrix` object (or, simply, matrix) adds to the target-action paradigm implemented by cell objects (specifically, cells that inherit from `NSActionCell`) by maintaining its own target and action in addition to the targets and actions of its cell objects. A matrix's target and action are used if one of its cells doesn't have a target or action set. This design allows for common usage patterns, including the following:

- If none of the cells of the `NSMatrix` object has either target or action set, the target and action of the `NSMatrix` object is always used.
- If only the actions of each of the cells is set, they share the target specified by their `NSMatrix` object, but send different messages to it.
- If only the targets of each of the cells is set, they all send the action message specified by the `NSMatrix` object, but to different targets.

When the user double-clicks an `NSMatrix` object, it can dispatch a separate action message (the selector for which is set via `setDoubleAction:`); this double-click action message is in addition to any cell's single-click action message. The double-click action of an `NSMatrix` object is always sent to its target.

Matrix Selection Modes

Since users frequently press the mouse button while the cursor is within the NSMatrix and then drag the mouse around, NSMatrix offers several methods that determine how it tracks the mouse. `setMode:` lets you choose among four "selection modes" that broadly determine how the matrix tracks the mouse. The `setAllowsEmptySelection:` and `setSelectionByRect:` methods let you refine how those modes work.

The `setMode:` method lets you choose one of these four modes:

- `NSTrackModeMatrix` is the most basic mode of operation. In this mode the NSCells are asked to track the mouse with `trackMouse` whenever the mouse is inside their bounds. No highlighting is performed. An example of this mode might be a "graphic equalizer" NSMatrix of sliders, where moving the mouse around causes the sliders to move under the mouse.
- `NSHighlightModeMatrix` is a modification of `NSTrackModeMatrix`. In this mode, an NSCell is highlighted before it's asked to track the mouse, then unhighlighted when it's done tracking. This is useful for multiple unconnected NSCells that use highlighting to inform the user that they are being tracked (like push-buttons and switches).
- `NSRadioModeMatrix` is used when you want no more than one NSCell to be selected at a time. It can be used to create a set of buttons of which one and only one is selected. (There's also the option of allowing no button to be selected.) Any time an NSCell is selected, the previously selected NSCell is unselected. This is most commonly used with groups of radio buttons. You might also use it with a group of push buttons that you want to behave like radio buttons.
- `NSListModeMatrix` is the opposite of `NSTrackModeMatrix`. NSCells are highlighted, but don't track the mouse. This mode can be used to select a range of text values, for example. NSMatrix supports the standard multiple-selection paradigms of dragging to select, using the Shift key to make continuous selections, and using the Command key to make discontinuous selections. Browsers (as used, for instance, by NSOpenPanel objects) use this mode.

`setAllowsEmptySelection:` has an effect only if the selection mode is `NSRadioModeMatrix`. It lets you choose whether, in a group of radio buttons, it's allowed for none of them to be on. For example, say the user clicks on the one radio button in a matrix that's on. If `allowsEmptySelection` is `YES`, that button turns off and none of the radio buttons is on. If `allowsEmptySelection` is `NO`, the button remains on, and the only way to turn it off is to click another button.

`setSelectionByRect:` sets whether the user can select a range of cells by dragging the mouse. If `isSelectionByRect` is `NO`, dragging over a range selects only the last cell only. If `isSelectionByRect` is `YES`, dragging over a range selects all the cells the user drags over.

Managing the Matrix's Cells

These methods dynamically add and remove columns:

- To add a column of empty cells beyond the last column, use `addColumn`.
- To add a column of filled cells beyond the last column, use `addColumnWithCells:`.
- To insert a column of empty cells at a specified location, use `insertColumn:`.
- To insert a column of filled cells at a specified location, use `insertColumn:withCells:`.
- To remove a column of cells, use `removeColumn:`.

These methods dynamically add and remove rows:

- To add a row of empty cells below the last row, use `addRow`.
- To add a row of filled cells below the last row, use `addRowWithCells:`.
- To insert a row of empty cells at a specified location, use `insertRow:`.
- To insert a row of filled cells at a specified location, use `insertRow:withCells:`.
- To remove a row of cells, use `removeRow:`.

This method creates individual cells:

- To replace an specific cell with a new cell, use `putCell:atRow:column:`.

This method retrieves information about individual cells:

- To get the frame of a specified cell, use `cellFrameAtRow:column:`.

These methods retrieve information about the matrix:

- To get the numbers of rows and columns, use `numberOfColumns` and `numberOfRows`.

These methods locate particular cells:

- To find a cell at a particular location, use `cellAtRow:column:`.
- To find a cell with a particular tag, use `cellWithTag:`.
- To get a list of all the cells, use `cells`.

These methods manage the selection:

- To retrieve the current selection, use `selectedCell` or `selectedCells`.
- To select a particular cell, use `selectCellAtRow:column:` or `selectCellWithTag`.

- To select a range of cells, use `selectAll:` or `setSelectionFrom:to:anchor:highlight:`.
- To deselect cells, use `deselectAllCells` or `deselectSelectedCell`.
- To retrieve the column and row number for the selection, use `selectedColumn` and `selectedRow`.
- To select the text in a cell (for a matrix that contains text fields), use `selectTextAtRow:column:`.

Setting a Matrix's Appearance

These methods control the sizes of the matrix's cells:

- To set the height and width of each cell, use `setCellSize:`.
- To set the amount of space that surrounds each cell, use `setIntercellSpacing:`.

These methods control whether the matrix displays its background. If the matrix's background isn't displayed, what's behind it shows through. If the matrix shows its background but the cells don't, the matrix's background shows through the cells.

- To set whether the matrix draws its background, use `setDrawsBackground:`.
- To set whether the matrix's cells draw their backgrounds, use `setDrawsCellBackground:`.

These methods control the color of the matrix and its cells:

- To set the background color of each cell, use `setCellBackgroundColor:`.
- To set the background color of the space between each cell, use `setBackgroundColor:`.

Document Revision History

This table describes the changes to *Matrix Programming Guide for Cocoa*.

Date	Notes
2006-11-07	Clarified semantics of double-click actions.
2006-04-04	Clarified interaction of NSMatrix action and target and those of its cells. Also corrected an error about modifier keys used for selections.
2005-07-07	Fixed bug and changed title from "Matrices."
2004-06-28	Reorganized Introduction. Corrected typos.
2002-11-12	Revision history was added to existing topic.