NSMatrix

Inherits From:	NSControl : NSView : NSResponder : NSObject
Conforms To:	NSCoding (from NSResponder) NSObject (from NSObject)
Declared In:	AppKit/NSMatrix.h

Class Description

NSMatrix is a class used for creating groups of NSCells that work together in various ways. It includes methods for arranging NSCells in rows and columns, either with or without space between them. NSCells in an NSMatrix are numbered by row and column, each starting with 0; for example, the top left NSCell would be at (0, 0), and the NSCell 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 can be set up to create new NSCells 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 adds to NSControl's target/action paradigm by allowing a separate target and action for each of its NSCells in addition to its own target and action. It also allows for an action message that's sent when the user double-clicks an NSCell, which is sent in addition to the single-click action message. If an NSCell doesn't have an action, the NSMatrix sends its own action to its own target. If an NSCell doesn't have a target, the NSMatrix sends the NSCell's action to its own target. The double-click action of an NSMatrix is always sent to the target of the NSMatrix.

Since the user might press the mouse button while the cursor is within the NSMatrix and then drag the mouse around, NSMatrix offers four "selection modes" that determine how NSCells behave when the NSMatrix is tracking the mouse:

- NSTrackModeMatrix is the most basic mode of operation. In this mode the NSCells are asked to track the mouse with **trackMouse:inRect:ofView:untilMouseUp:** 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 the option of allowing no button to be selected). Any time an NSCell is selected, the previously selected NSCell is unselected. The canonical example of this mode is a set of 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 discontinuous selections, and using the alternate key to extend selections. Browsers (as used, for instance, in NeXT's File Viewer) use this mode.

Method Types

Initializing the NSMatrix class	+ cellClass + setCellClass
Initializing an NSMatrix object	– initWithFrame: – initWithFrame:mode:cellClass:numberOfRows:numberOfColums: – initWithFrame:mode:prototype:numberOfRows:numberOfColumns:
Setting the selection mode	– mode – setMode:
Configuring the NSMatrix	– allowsEmptySelection – isSelectionByRect – setAllowsEmptySelection – setSelectionByRect:
Setting the cell class	– cellClass – prototype – setCellClass: – setPrototype:

Laying out the NSMatrix	– addColumn – addColumnWithCells: – addRow – addRowWithCells: – cellFrameAtRow:column: – cellSize – getNumberOfRows:columns: – insertColumn: – insertColumn:withCells: – insertRow: – insertRow:withCells: – intercellSpacing – makeCellAtRow:column: – numberOfColumns – numberOfRows: – putCell:atRow:column: – removeColumn: – removeRow: – renewRows:columns: – setCellSize: – setIntercellSpacing: – sortUsingFunction:context: – sortUsingSelector:
Finding matrix coordinates	– getRow:column:forPoint: – getRow:column:ofCell:
Modifying individual cells	– setState:atRow:column:
Selecting cells	– deselectAllCells – deselectSelectedCell – keyCell – selectAll: – selectCellAtRow:column: – selectCellWithTag: – selectedCell – selectedCells – selectedColumn – selectedRow – setKeyCell: – setSelectionFrom:to:anchor:highlight:
Finding cells	– cellAtRow:column: – cellWithTag: – cells

Modifying graphic attributes	– backgroundColor – cellBackgroundColor – drawsBackground – drawsCellBackground – setBackgroundColor: – setCellBackgroundColor: – setDrawsBackground: – setDrawsCellBackground:
Editing text in cells	– selectText: – selectTextAtRow:column: – textDidBeginEditing: – textDidChange: – textDidEndEditing: – textShouldBeginEditing: – textShouldEndEditing:
Setting tab key behavior	– nextText – previousText – setNextText: – setPreviousText: – setTabKeyTraversesCells: – tabKeyTraversesCells
Assigning a delegate	– delegate – setDelegate:
Resizing the matrix and its cells	– autosizesCells – setAutosizesCells: – setValidateSize: – sizeToCells
Scrolling	– isAutoscroll – scrollCellToVisibleAtRow:column: – setAutoscroll: – setScrollable:
Displaying	– drawCellAtRow:column: – highlightCell:atRow:column:
Target and action	– doubleAction – errorAction – sendAction – sendAction:to:forAllCells: – sendDoubleAction – setDoubleAction: – setErrorAction:

Handling event and action messages	– acceptsFirstMouse: – mouseDown: – mouseDownFlags – performKeyEquivalent:
Managing the cursor	– resetCursorRects

Class Methods

cellClass

+ (Class)**cellClass**

Returns the default class that the NSMatrix class will use to make cells.

See also: – **cellClass**, – **makeCellAtRow:column:**, – **prototype**

setCellClass:

+ (void)**setCellClass:**(Class)*factoryId*

Sets the default class that the NSMatrix class will use to make cells. *factoryId* should be the **id** of a subclass of NSCell (usually NSActionCell), obtained by sending the **class** message to either the NSCell subclass object or to an instance of that subclass. The default NSCell class is NSActionCell.

Your code should rarely need to invoke this method, since each instance of NSMatrix can be configured to use its own NSCell class (or a prototype that gets copied). The NSCell class set with this method is simply a fall-back for matrices initialized with **initWithFrame:**.

See also: – **setCellClass:**, – **setPrototype:**, – **makeCellAtRow:column:**,
– **initWithFrame:mode:cellClass:numberOfRows:numberOfColumns:**,
– **initWithFrame:mode:prototype:numberOfRows:numberOfColumns:**

Instance Methods

acceptsFirstMouse:

– (BOOL)**acceptsFirstMouse:**(NSEvent *)*theEvent*

Returns NO if the selection mode of the NSMatrix is NSListModeMatrix, YES if the NSMatrix is in any other selection mode. The NSMatrix does not accept first mouse in NSListModeMatrix to prevent the loss of multiple selections. The NSEvent parameter, *theEvent*, is ignored.

See also: – **mode**

addColumn

– (void)**addColumn**

Adds a new column of cells to the right of the last column, creating new cells as needed with **makeCellAtRow:column:**.

If the number of rows or columns in the NSMatrix has been changed with **renewRows:columns:**, new cells are created only if they are needed. This allows you to grow and shrink an NSMatrix without repeatedly creating and freeing the cells.

This method doesn't redraw the NSMatrix. Accordingly, after calling this method you should send **setNeedsDisplay:YES** to the NSMatrix. Your code may also need to use **sizeToCells** after sending this method to resize the NSMatrix to fit the newly added cells.

See also: – **cellClass**, – **insertColumn:**, – **prototype**

addColumnWithCells:

– (void)**addColumnWithCells:**(NSArray *)*newCells*

Adds a new column of cells to the right of the last column. The new column is filled with objects from *newCells*, starting with the object at index 0. Each object in *newCells* should be a an NSCell or one of its subclasses (usually NSActionCell). *newCells* should have a sufficient number of cells to fill the entire column; extra cells are ignored.

This method doesn't redraw the NSMatrix. Accordingly, after calling this method you should send **setNeedsDisplay:YES** to the NSMatrix. Your code may also need to use **sizeToCells** after sending this method to resize the NSMatrix to fit the newly added cells.

See also: – **insertColumn:withCells:**

addRow

– (void)**addRow**

Adds a new row of cells below the last row, creating new cells as needed with **makeCellAtRow:column:**.

If the number of rows or columns in the NSMatrix has been changed with **renewRows:columns:**, then new cells are created only if they are needed. This allows you to grow and shrink an NSMatrix without repeatedly creating and freeing the cells.

This method doesn't redraw the NSMatrix. Accordingly, after calling this method you should send **setNeedsDisplay:YES** to the NSMatrix. Your code may also need to use **sizeToCells** after sending this method to resize the NSMatrix to fit the newly added cells.

See also: – **cellClass**, – **insertRow:**, – **prototype**

addRowWithCells:

– (void)**addRowWithCells:**(NSArray *)*newCells*

Adds a new row of cells below the last row. The new row is filled with objects from *newCells*, starting with the object at index 0. Each object in *newCells* should be a an NSCell or one of its subclasses (usually NSActionCell). *newCells* should have a sufficient number of cells to fill the entire row; extra cells are ignored.

This method doesn't redraw the NSMatrix. Accordingly, after calling this method you should send **setNeedsDisplay:YES** to the NSMatrix. Your code may also need to use **sizeToCells** after sending this method to resize the NSMatrix to fit the newly added cells.

See also: – **insertRow:withCells:**

allowsEmptySelection

– (BOOL)**allowsEmptySelection**

Returns whether its possible to have no cells selected in a radio-mode matrix.

See also: – **mode**

autosizesCells

– (BOOL)**autosizesCells**

Returns YES if cells are resized proportionally to the NSMatrix when its size changes (and inter-cell spacing is kept constant). Returns NO if the cell size remains constant (and inter-cell spacing changes).

backgroundColor

– (NSColor *)**backgroundColor**

Returns the color used to draw the background of the NSMatrix (the space between the cells).

See also: – **cellBackgroundColor**, – **drawsBackground**

cellAtRow:column:

– (id)**cellAtRow:**(int)*row* **column:**(int)*column*

Returns the NSCell object at *row* and *column*, or **nil** if either *row* or *column* are outside the bounds of the NSMatrix.

See also: – **getRow:column:ofCell:**

cellBackgroundColor

– (NSColor *)**cellBackgroundColor**

Returns the color used to fill the background of the NSMatrix's cells.

See also: – **backgroundColor**, – **drawsCellBackground**

cellClass

– (Class)**cellClass**

Returns the subclass of NSCell that the NSMatrix uses when creating new (empty) cells.

See also: – **prototype**, – **makeCellAtRow:column:**

cellFrameAtRow:column:

– (NSRect)**cellFrameAtRow:**(int)*row* **column:**(int)*column*

Returns the frame rectangle of the cell that would be drawn at the specified *row* and *column* (whether or not the specified cell actually exists).

See also: – **cellSize**

cellSize

– (NSSize)**cellSize**

Returns the width and the height of each cell in the NSMatrix (all cells in an NSMatrix are the same size).

See also: – **cellFrame:atRow:column**:, – **intercellSpacing**

cellWithTag:

– (id)**cellWithTag:**(int)*anInt*

Searches the NSMatrix and returns the last (when viewing the matrix as a row-ordered array) NSCell object which has a tag matching *anInt*, or **nil** if no such cell exists.

See also: – **selectCellWithTag:**, – **setTag:** (NSActionCell)

cells

– (NSArray *)**cells**

Returns an NSArray that contains the NSMatrix's cells. The cells in the array are row-ordered; that is, the first row of cells appear first in the array, followed by the second row, and so forth.

See also: – **cellAtRow:column:**

delegate

– (id)**delegate**

Returns the delegate for messages from the field editor.

See also: – **textShouldBeginEditing:**, – **textShouldEndEditing:**

deselectAllCells

– (void)**deselectAllCells**

Deselects all cells in the NSMatrix and, if necessary, redisplays the NSMatrix. If the selection mode is NSRadioModeMatrix and empty selection is not allowed, this method does nothing.

See also: – **allowsEmptySelection**, – **mode**, – **selectAll:**

deselectSelectedCell

– (void)**deselectSelectedCell**

Deselects the selected cell or cells. If the selection mode is NSRadioModeMatrix and empty selection is not allowed, or if nothing is currently selected, this method does nothing. This method doesn't redisplay the NSMatrix.

See also: – **allowsEmptySelection**, – **mode**, – **selectCellAtRow:column:**

doubleAction

– (SEL)**doubleAction**

Returns the action method sent by the NSMatrix to its target when the user double-clicks an entry, or NULL if there's no double-click action. The double-click action of an NSMatrix is sent after the appropriate single-click action (for the NSCell clicked or for the NSMatrix if the NSCell doesn't have its own action).

If there is no double-click action and the NSMatrix doesn't ignore multiple clicks, the single-click action is sent twice.

See also: – **action** (NSControl), – **target** (NSControl), – **sendDoubleAction**, – **ignoresMultiClick** (NSControl)

drawCellAtRow:column:

– (void)**drawCellAtRow:**(int)*row* **column:**(int)*column*

Displays the cell at the specified row and column, providing that *row* and *column* reference a cell that's within the NSMatrix.

See also: – **drawCell:** (NSControl), – **drawCellInside:** (NSControl)

drawsBackground

– (BOOL)**drawsBackground**

Returns whether the receiver draws its background (the space between the cells).

See also: – **backgroundColor**, – **drawsCellBackground**

drawsCellBackground

– (BOOL)**drawsCellBackground**

Returns whether the receiver draws the background within each of its cells.

See also: – **cellBackgroundColor**, – **drawsBackground**

errorAction

– (SEL)**errorAction**

Returns the action that's sent to the target of the NSMatrix when the user enters an illegal value for the selected cell.

See also: – **action** (NSControl), – **target** (NSControl), – **textShouldEndEditing:**

getNumberOfRows:columns:

– (void)**getNumberOfRows:**(int *)*rowCount* **columns:**(int *)*columnCount*

Returns by reference the number of rows and columns in the NSMatrix.

See also: – **numberOfColumns**, – **numberOfRows**

getRow:column:forPoint:

– (BOOL)**getRow:**(int *)*row* **column:**(int *)*column* **forPoint:**(NSPoint)*aPoint*

Returns YES if *aPoint* lies within one of the cells in the NSMatrix, and returns by reference the row and column for the cell within which the specified point lies. If *aPoint* falls outside the bounds of the Matrix or lies within an intercell spacing, **getRow:column:forPoint:** returns NO.

Make sure that *aPoint* is in the coordinate system of the NSMatrix.

See also: – **getRow:column:ofCell:**

getRow:column:ofCell:

– (BOOL)**getRow:**(int *)*row* **column:**(int *)*column* **ofCell:**(NSCell *)*aCell*

Searches the NSMatrix and returns YES if *aCell* is one of the cells in the NSMatrix, and returns by reference the row and column of the cell. If *aCell* is not found within the Matrix, **getRow:column:ofCell:** returns NO.

See also: – **getRow:column:forPoint:**

highlightCell:atRow:column:

– (void)**highlightCell:**(BOOL)*flag* **atRow:**(int)*row* **column:**(int)*column*

Assuming that *row* and *column* indicate a valid cell within the NSMatrix, **highlightCell:atRow:column:** highlights (if *flag* is YES) or unhighlights (if *flag* is NO) the specified cell.

initWithFrame:

– (id)**initWithFrame:**(NSRect)*frameRect*

Initializes and returns the receiver, a newly-allocated instance of NSMatrix, with default parameters in the frame specified by *frameRect*. The new NSMatrix contains no rows or columns. The default mode is NSRadioModeMatrix. The default cell class is NSActionCell.

initWithFrame:mode:cellClass:numberOfRows:numberOfColumns:

– (id)**initWithFrame:**(NSRect)*frameRect* **mode:**(int)*aMode* **cellClass:**(Class)*factoryId*
numberOfRows:(int)*numRows* **numberOfColumns:**(int)*numColumns*

Initializes and returns the receiver, a newly-allocated instance of NSMatrix, in the frame specified by *frameRect*. The new NSMatrix contains *numRows* rows and *numColumns* columns. *aMode* is set as the tracking mode for the NSMatrix, and can be one of the following four constants, all of which are described in the class description:

NSTrackModeMatrix	Cells track the mouse, but do not highlight
NSHighlightModeMatrix	Cells highlight as they track the mouse
NSRadioModeMatrix	Allows no more than one selected cell
NSListModeMatrix	Cells are highlighted, but don’t track the mouse

The new NSMatrix creates and uses cells of class *classId*, which can be obtained by sending a **class** message to the desired subclass of NSCell.

This method is the designated initializer for matrices that add cells by creating instances of an NSCell subclass.

initWithFrame:mode:prototype:numberOfRows:numberOfColumns:

– (id)**initWithFrame:**(NSRect)*frameRect* **mode:**(int)*aMode* **prototype:**(NSCell *)*aCell*
numberOfRows:(int)*numRows* **numberOfColumns:**(int)*numColumns*

Initializes and returns the receiver, a newly-allocated instance of NSMatrix, in the frame specified by *frameRect*. The new NSMatrix contains *numRows* rows and *numColumns* columns. *aMode* is set as the tracking mode for the NSMatrix, and can be one of the following four constants, all of which are described in the class description:

NSTrackModeMatrix	Cells track the mouse, but do not highlight
NSHighlightModeMatrix	Cells highlight as they track the mouse
NSRadioModeMatrix	Allows no more than one selected cell
NSListModeMatrix	Cells are highlighted, but don’t track the mouse

The new Matrix creates cells by copying *aCell*, which should be an instance of a subclass of NSCell.

This method is the designated initializer for matrices that add cells by copying an instance of an NSCell subclass.

insertColumn:

– (void)**insertColumn:**(int)*column*

Inserts a new column of cells before *column*, creating new cells if needed with **makeCellAtRow:column:**. If *column* is greater than the number of columns in the NSMatrix, enough columns are created to expand the NSMatrix to be *column* columns wide. This method doesn't redraw the NSMatrix. Accordingly, after calling this method you should send **setNeedsDisplay:YES** to the NSMatrix. Your code may also need to use **sizeToCells** after sending this method to resize the NSMatrix to fit the newly added cells.

If the number of rows or columns in the NSMatrix has been changed with **renewRows:columns:**, then new cells are created only if they're needed. This allows you to grow and shrink an NSMatrix without repeatedly creating and freeing the cells.

See also: – **addColumn**, – **insertRow:**

insertColumn:withCells:

– (void)**insertColumn:**(int)*column* **withCells:**(NSArray *)*newCells*

Inserts a new column of cells before *column*. The new column is filled with objects from *newCells*, starting with the object at index 0. Each object in *newCells* should be a an NSCell or one of its subclasses (usually NSActionCell). If *column* is greater than the number of columns in the NSMatrix, enough columns are created to expand the NSMatrix to be *column* columns wide. *newCells* should have a sufficient number of cells to fill each new column; extra cells are ignored.

This method doesn't redraw the NSMatrix. Accordingly, after calling this method you should send **setNeedsDisplay:YES** to the NSMatrix. Your code may also need to use **sizeToCells** after sending this method to resize the NSMatrix to fit the newly added cells.

See also: – **addColumnWithCells:**, – **insertRow:withCells:**

insertRow:

– (void)**insertRow:**(int)*row*

Inserts a new row of cells before *row*, creating new cells if needed with **makeCellAtRow:column:**. If *row* is greater than the number of rows in the NSMatrix, enough rows are created to expand the NSMatrix to be *row* rows high. This method doesn't redraw the NSMatrix. Accordingly, after calling this method you should send **setNeedsDisplay:YES** to the NSMatrix. Your code may also need to use **sizeToCells** after sending this method to resize the NSMatrix to fit the newly added cells.

If the number of rows or columns in the NSMatrix has been changed with **renewRows:columns:**, then new cells are created only if they're needed. This allows you to grow and shrink an NSMatrix without repeatedly creating and freeing the cells.

See also: – **addRow**, – **insertColumn:**

insertRow:withCells:

– (void)**insertRow:**(int)*row* **withCells:**(NSArray *)*newCells*

Inserts a new row of cells before *row*. The new row is filled with objects from *newCells*, starting with the object at index 0. Each object in *newCells* should be an NSCell or one of its subclasses (usually NSActionCell). If *row* is greater than the number of rows in the NSMatrix, enough rows are created to expand the NSMatrix to be *row* rows high. *newCells* should have a sufficient number of cells to fill each new row (**[***newCells* **count]** must be greater than or equal to **[self numberOfColumns]**); extra cells are ignored.

This method doesn't redraw the NSMatrix. Accordingly, after calling this method you should send **setNeedsDisplay:YES** to the NSMatrix. Your code may also need to use **sizeToCells** after sending this method to resize the NSMatrix to fit the newly added cells.

See also: – **addRowWithCells:**, – **insertColumn:withCells:**

intercellSpacing

– (NSSize)**intercellSpacing**

Returns the vertical and horizontal spacing between cells in the NSMatrix.

See also: – **cellSize**

isAutoscroll

– (BOOL)**isAutoscroll**

Returns whether the NSMatrix will be automatically scrolled whenever the mouse is dragged outside the NSMatrix after a mouse-down event within its bounds.

See also: – **scrollCellToVisibleAtRow:column:**, – **setScrollable:**

isSelectionByRect

– (BOOL)**isSelectionByRect**

Returns YES if a the user can select a rectangle of cells in the NSMatrix by dragging the cursor, NO otherwise.

See also: – **setSelectionFrom:to:anchor:highlight:**

keyCell

– (id)**keyCell**

Returns the cell that will be clicked when the user presses the Return key.

See also: – **nextText**, – **tabKeyTraversesCells**

makeCellAtRow:column:

– (NSCell *)**makeCellAtRow:**(int)*row* **column:**(int)*column*

Creates a new cell at the specified location in the NSMatrix. If the NSMatrix has a prototype cell, it's copied to create the new cell. If not, and if the NSMatrix has a cell class set, it allocates and initializes (with **init**) an instance of that class. If the NSMatrix hasn't had either a prototype cell or a cell class set, **makeCellAtRow:column:** creates an NSActionCell. Returns the newly created cell.

Your code should never invoke this method directly; it's used by **addRow** and other methods when a cell must be created. It may be overridden to provide more specific initialization of cells.

See also: – **addColumn**, – **addRow**, – **insertColumn:**, – **insertRow:**, – **setCellClass:**, – **setPrototype:**

mode

– (NSMatrixMode)**mode**

Returns the selection mode of the NSMatrix. Possible return values are defined in NSMatrix.h, and are also listed here:

NSTrackModeMatrix	Cells track the mouse, but do not highlight
NSHighlightModeMatrix	Cells highlight as they track the mouse
NSRadioModeMatrix	Allows no more than one selected cell
NSListModeMatrix	Cells are highlighted, but don't track the mouse

These modes are explained in detail in the class description.

See also: – **initWithFrame:mode:cellClass:numberOfRows:numberOfColumns:**,
– **initWithFrame:mode:prototype:numberOfRows:numberOfColumns:**

mouseDown:

– (void)**mouseDown:**(NSEvent *)*theEvent*

Responds to a mouse-down event. A mouse-down event in a text cell initiates editing mode. A double-click in any cell type except a text cell sends the double-click action of the NSMatrix (if there is one) in addition to the single-click action.

Your code should never invoke this method, but you may override it to implement different mouse tracking than NSMatrix does. The response of the NSMatrix depends on its selection mode, as explained in the class description.

See also: – **sendAction**, – **sendDoubleAction**

mouseDownFlags

– (int)**mouseDownFlags**

Returns the flags that were in effect at the mouse-down event that started the current tracking session (NSMatrix's **mouseDown:** method obtains these flags by sending a **modifierFlags** message to the event passed into **mouseDown:**). Use this method if you want to access these flags. This method is valid only during tracking; it isn't useful if the target of the NSMatrix initiates another tracking loop as part of its action method (as a cell that pops up a PopUpList does, for example).

See also: – **sendActionOn:** (NSCell)

nextText

– (id)**nextText**

Returns the object that would be selected if the user presses Tab while editing the last text cell in the NSMatrix.

See also: – **nextKeyView** (NSView), – **previousText**, – **setNextText:**

numberOfColumns

– (int)**numberOfColumns**

Returns the number of columns in the NSMatrix.

See also: – **getNumberOfRows:columns:**

numberOfRows

– (int)**numberOfRows**

Returns the number of rows in the NSMatrix.

See also: – **getNumberOfRows:columns:**

performKeyEquivalent:

– (BOOL)**performKeyEquivalent:**(NSEvent *)*theEvent*

If there's a cell in the NSMatrix that has a key equivalent equal to the character in **[***theEvent* **charactersIgnoringModifiers]** (taking into account any key modifier flags) and that cell is enabled, that cell is made to react as if the user had clicked it: by highlighting, changing its state as appropriate, sending its action if it has one, and then unhighlighting. Returns YES if a cell in the NSMatrix responds to the key equivalent in *theEvent*, NO if no cell responds.

Your code should never send this message; it is sent when the NSMatrix or one of its superviews is the first responder and the user presses a key. You may want to override this method to change the way key equivalents are performed or displayed, or to disable them in your subclass.

previousText

– (id)**previousText**

Returns the object that would be selected if the user presses Shift-Tab while editing the first text cell in the NSMatrix.

See also: – **nextKeyView** (NSView), – **nextText**, – **setPreviousText:**

prototype

– (id)**prototype**

Returns the prototype cell that's copied whenever a new cell needs to be created, or **nil** if there is none.

See also: – **initWithFrame:mode:prototype:numberOfRows:numberOfColumns:**,
– **makeCellAtRow:column:**

putCell:atRow:column:

– (void)**putCell:**(NSCell *)*newCell* **atRow:**(int)*row* **column:**(int)*column*

Replaces the cell at the specified *row* and *column* with *newCell*, and redraws the cell.

removeColumn:

– (void)**removeColumn:**(int)*column*

Removes the column at position *column* from the NSMatrix and autoreleases the column's cells. Doesn't redraw the NSMatrix. Your code should normally send **sizeToCells** after invoking this method to resize the NSMatrix so that it fits the reduced cell count.

See also: – **removeRow:**, – **addColumn**, – **insertColumn:**

removeRow:

– (void)**removeRow:**(int)*row*

Removes the row at position *row* from the NSMatrix and autoreleases the row's cells. Doesn't redraw the NSMatrix. Your code should normally send **sizeToCells** after invoking this method to resize the NSMatrix so that it fits the reduced cell count.

See also: – **removeColumn:**, – **addRow**, – **insertRow:**

renewRows:columns:

– (void)**renewRows:**(int)*newRows* **columns:**(int)*newCols*

Changes the number of rows and columns in the NSMatrix. This method uses the same cells as before, creating new cells only if the new size is larger; it never frees cells. Doesn't redisplay the NSMatrix. Your code should normally send **sizeToCells** after invoking this method to resize the NSMatrix so that it fits the changed cell arrangement. This method deselects all cells in the NSMatrix.

See also: – **addColumn**, – **addRow**, – **removeColumn:**, – **removeRow:**

resetCursorRects

– (void)**resetCursorRects**

Resets cursor rectangles so that the cursor becomes an I-beam over text cells. It does this by sending **resetCursorRect:inView:** to each cell in the NSMatrix. Any cell that has a cursor rectangle to set up should then send **addCursorRect:cursor:** back to the NSMatrix.

See also: – **resetCursorRect:inView:** (NSCell), – **addCursorRect:cursor:** (NSView)

scrollCellToVisibleAtRow:column:

– (void)**scrollCellToVisibleAtRow:**(int)*row* **column:**(int)*column*

If the NSMatrix is in a scrolling view, and *row* and *column* represent a valid cell within the NSMatrix, this method scrolls the NSMatrix so that the specified cell is visible.

See also: – **scrollRectToVisible:** (NSView)

selectAll:

– (void)**selectAll:**(id)*sender*

Selects and highlights all of the cells in the NSMatrix, except for editable text cells and disabled cells. Redisplays the NSMatrix. *sender* is ignored.

See also: – **selectCell:** (NSControl)

selectCellAtRow:column:

– (void)**selectCellAtRow:**(int)*row* **column:**(int)*column*

Selects the cell at the specified *row* and *column* within the NSMatrix. If the specified cell is an editable text cell, its text is selected. If either *row* or *column* is –1, then the current selection is cleared (unless the NSMatrix is in NSRadioModeMatrix and doesn't allow empty selection). Redraws the affected cells.

See also: – **allowsEmptySelection**, – **mode**, – **selectCell:** (NSControl)

selectCellWithTag:

– (BOOL)**selectCellWithTag:**(int)*anInt*

If the NSMatrix has at least one cell whose tag is equal to *anInt*, the last cell (when viewing the matrix as a row-ordered array) is selected. If the specified cell is an editable text Cell, its text is selected. Returns YES if the NSMatrix contains a cell whose tag matches *anInt*, or NO if no such cell exists.

See also: – **cellWithTag:**, – **selectCell:** (NSControl)

selectText:

– (void)**selectText:**(id)*sender*

If the currently selected cell is editable and enabled, its text is selected. Otherwise, the key cell is selected.

See also: – **keyCell**, – **selectText:** (NSTextField)

selectTextAtRow:column:

– (id)**selectTextAtRow:**(int)*row* **column:**(int)*column*

If *row* and *column* indicate a valid cell within the NSMatrix, and that cell is both editable and selectable, **selectTextAtRow:column:** selects and then returns the specified cell. If the cell specified by *row* and *column* is either not editable or not selectable, **selectTextAtRow:column:** does nothing, and returns **nil**. Finally, if *row* and *column* indicate a cell that is outside the NSMatrix, **selectTextAtRow:column:** does nothing and returns the receiver.

See also: – **selectText:**

selectedCell

– (id)**selectedCell**

Returns the most recently selected cell, or **nil** if no cell is selected. If more than one cell is selected, **selectedCell** returns the cell that is lowest and furthest to the right in the NSMatrix.

selectedCells

– (NSArray *)**selectedCells**

Returns an array containing each of the cells in the receiver that is currently highlighted.

See also: – **selectedCell**

selectedColumn

– (int)**selectedColumn**

Returns the column number of the selected cell, or –1 if no cells are selected. If cells in multiple columns are selected, this method returns the number of the last (right-most) column containing a selected cell.

selectedRow

– (int)**selectedRow**

Returns the row number of the selected cell, or –1 if no cells are selected. If cells in multiple rows are selected, this method returns the number of the last row containing a selected cell.

sendAction

– (BOOL)**sendAction**

If the selected cell has both an action and a target, its action is sent to its target. If the cell has an action but no target, its action is sent to the target of the NSMatrix. If the cell doesn't have an action, or if there is no selected cell, the NSMatrix sends its own action to its target. Returns YES if an action was successfully sent to a target.

If the selected cell is disabled, this method does nothing and returns NO.

See also: – **sendDoubleAction**, – **action** (NSCell), – **target** (NSCell)

sendAction:to:forAllCells:

– (void)**sendAction:**(SEL)*aSelector* **to:**(id)*anObject* **forAllCells:**(BOOL)*flag*

Iterates through all of the cells in the NSMatrix (if *flag* is YES), or just the selected cells in the NSMatrix (if *flag* is NO), sending *aSelector* to *anObject* for each. Iteration begins with the cell in the upper-left corner of the NSMatrix, proceeding through the appropriate entries in the first row, then on to the next.

aSelector must represent a method that takes a single argument: the **id** of the current cell in the iteration. *aSelector*'s return value must be a BOOL. If *aSelector* returns NO for any cell, **sendAction:to:forAllCells:** terminates immediately, without sending the message to the remaining cells. If it returns YES, **endAction:to:forAllCells:** proceeds on to the next cell.

This method is *not* invoked to send action messages to target objects in response to mouse-down events in the NSMatrix. Instead, you can invoke it if you want to have multiple cells in an NSMatrix interact with an object. For example you could use it to verify the titles in a list of items, or to enable a series of radio buttons based on their purpose in relation to *anObject*.

sendDoubleAction

– (void)**sendDoubleAction**

If the NSMatrix has a double-click action, **sendDoubleAction** sends that message to the target of the NSMatrix. If not, then if the selected cell (as returned by **selectedCell**) has an action, that message is sent to the selected cell's target. Finally, if the selected cell also has no action, then the single-click action of the NSMatrix is sent to the target of the NSMatrix.

If the selected cell is disabled, this method does nothing.

Your code shouldn't invoke this method; it's sent in response to a double-click event in the NSMatrix. Override it if you need to change the search order for an action to send.

See also: – **sendAction**, – **ignoresMultiClick:** (NSControl)

setAllowsEmptySelection:

– (void)**setAllowsEmptySelection:**(BOOL)*flag*

If *flag* is YES, then the NSMatrix will allow one or zero cells to be selected. If *flag* is NO, then the NSMatrix will allow one and only one cell (not zero cells) to be selected. This setting has effect only in the NSRadioModeMatrix selection mode.

setAutoscroll:

– (void)**setAutoscroll:**(BOOL)*flag*

If *flag* is YES and the NSMatrix is in a scrolling view, it will be automatically scrolled whenever a the mouse is dragged outside the NSMatrix after a mouse-down event within the bounds of the NSMatrix.

setAutosizesCells:

– (void)**setAutosizesCells:**(BOOL)*flag*

If *flag* is YES, then whenever the NSMatrix is resized, the sizes of the cells change in proportion, keeping the inter-cell space constant; further, this method verifies that the cell sizes and inter-cell spacing add up to the exact size of the NSMatrix, adjusting the size of the cells and updating the NSMatrix if they don't. If *flag* is NO, then the inter-cell space changes when the NSMatrix is resized, with the cell size remaining constant.

setBackgroundColor:

– (void)**setBackgroundColor:**(NSColor *)*aColor*

Sets the background color for the NSMatrix to *aColor*, and redraws the NSMatrix. This color is used to fill the space between cells or the space behind any non-opaque cells. The default background color is NSColor's **controlColor**.

See also: – **drawsBackground**, – **setCellBackgroundColor:**

setCellBackgroundColor:

– (void)**setCellBackgroundColor:**(NSColor *)*aColor*

Sets the background color for the cells in the NSMatrix to *aColor*. This color is used to fill the space behind non-opaque cells. The default cell background color is NSColor's **controlColor**.

See also: – **drawsCellBackground**, – **setBackgroundColor:**

setCellClass:

– (void)**setCellClass:**(Class)*factoryId*

Configures the receiver to use instances of *factoryId* when creating new cells. *factoryId* should be the **id** of a subclass of NSCell, which can be obtained by sending the **class** message to either the NSCell subclass object or to an instance of that subclass. The default cell class is that set with the class method **setCellClass:**, or NSActionCell if no other default cell class has been specified.

You only need to use this method with matrices initialized with **initWithFrame:**, since the other initializers allow you to specify an instance-specific cell class or cell prototype.

See also: – **addColumn**, – **addRow**, – **insertColumn:**, – **insertRow:**, – **makeCellAtRow:column:**, – **setPrototype:**

setCellSize:

– (void)**setCellSize:**(NSSize)*aSize*

Sets the width and the height of each of the cells in the NSMatrix to those in *aSize*. This may change the size of the NSMatrix. Does not redraw the NSMatrix.

See also: – **calcSize** (NSControl)

setDelegate:

– (void)**setDelegate:**(id)*anObject*

Sets the delegate for messages from the field editor.

See also: – **textShouldBeginEditing:**, – **textShouldEndEditing:**

setDoubleAction:

– (void)**setDoubleAction:**(SEL)*aSelector*

Makes *aSelector* the action sent to the target of the NSMatrix when the user double-clicks a cell. A double-click action is always sent after the appropriate single-click action; the cell's if it has one, otherwise the single-click action of the NSMatrix.

If an NSMatrix has no double-click action set, then by default a double-click is treated as a single-click.

See also: – **sendDoubleAction**, – **setAction:** (NSControl), – **setTarget:** (NSControl)

setDrawsBackground:

– (void)**setDrawsBackground:**(BOOL)*flag*

Sets whether the receiver draws its background (the space between the cells).

See also: – **backgroundColor**, – **setDrawsCellBackground**

setDrawsCellBackground:

– (void)**setDrawsCellBackground:**(BOOL)*flag*

Sets whether the receiver draws the background within each of its cells.

See also: – **cellBackgroundColor**, – **setDrawsBackground**

setErrorAction:

– (void)**setErrorAction:**(SEL)*aSelector*

Sets the action that's sent to the target of the NSMatrix when the user enters an illegal value for the selected cell.

See also: – **action** (NSControl), – **target** (NSControl)

setIntercellSpacing:

– (void)**setIntercellSpacing:**(NSSize)*aSize*

Sets the vertical and horizontal spacing between cells in the NSMatrix. By default, both values are 1.0 in the NSMatrix's coordinate system.

See also: – **cellSize**

setKeyCell:

– (void)**setKeyCell:**(NSCell *)*aCell*

Sets to *aCell* the cell that will be clicked when the user presses the Return key.

See also: – **setNextText:**, – **setTabKeyTraversesCells:**

setMode:

– (void)**setMode:**(NSMatrixMode)*aMode*

Sets the selection mode of the NSMatrix. Possible values for *aMode* are defined in NSMatrix.h, and include:

NSTrackModeMatrix	Cells track the mouse, but do not highlight
NSHighlightModeMatrix	Cells highlight as they track the mouse
NSRadioModeMatrix	Allows no more than one selected cell
NSListModeMatrix	Cells are highlighted, but don't track the mouse

These modes are explained in detail in the class description.

See also: – **initWithFrame:mode:cellClass:numberOfRows:numberOfColumns:**,
– **initWithFrame:mode:prototype:numberOfRows:numberOfColumns:**

setNextText:

– (void)**setNextText:**(id)*anObject*

If the NSMatrix doesn't already have a next key view, inserts *anObject* after the receiver in the key view loop of the receiver's NSWindow. *anObject* thus becomes the object that would be selected if the user presses Tab while editing the last text cell in the NSMatrix. If the NSMatrix already has a next key view, this method does nothing.

See also: – **setNextKeyView:** (NSView), – **setTabKeyTraversesCells:**

setPreviousText:

– (void)**setPreviousText:**(id)*anObject*

If *anObject* doesn't already have a next key view, inserts the receiver after *anObject* in the key view loop of *anObject*'s NSWindow. If *anObject* already has a next key view, this method does nothing.

See also: – **setNextKeyView:** (NSView)

setPrototype:

– (void)**setPrototype:**(NSCell *)*aCell*

Sets the prototype cell that's copied whenever a new cell needs to be created.

See also: – **initWithFrame:mode:prototype:numberOfRows:numberOfColumns:**,
– **makeCellAtRow:column:**

setScrollable:

– (void)**setScrollable:**(BOOL)*flag*

If *flag* is YES, sets all the cells to be scrollable, so that the text they contain scrolls to remain in view if the user types past the edge of the cell. If *flag* is NO, all cells are made to be non-scrolling. The prototype cell, if there is one, is also set accordingly.

See also: – **prototype**, – **setScrollable:** (NSCell)

setSelectionByRect:

– (void)**setSelectionByRect:**(BOOL)*flag*

Sets whether the user can select a rectangle of cells in the NSMatrix by dragging the cursor. If *flag* is NO, selection is on a row-by-row basis. The default is YES.

See also: – **setSelectionFrom:to:anchor:highlight:**

setSelectionFrom:to:anchor:highlight:

– (void)**setSelectionFrom:**(int)*startPos* **to:**(int)*endPos* **anchor:**(int)*anchorPos* **highlight:**(BOOL)*lit*

Programmatically selects a range of cells. *startPos*, *endPos*, and *anchorPos* are cell positions, counting from 0 at the upper left cell of the NSMatrix, in row order. For example, the third cell in the top row would be number 2.

startPos and *endPos* are used to mark where the user would have pressed the mouse button and released it, respectively. *anchorPos* locates the “last selected cell” with regard to extending the selection by Shift- or Alternate-clicking. Finally, *lit* determines whether cells selected by this method should be highlighted.

See also: – **isSelectionByRect**, – **selectedCells:**

setState:atRow:column:

– (void)**setState:**(int)*value* **atRow:**(int)*row* **column:**(int)*column*

Sets the state of the cell at *row* and *column* to *value*. For radio-mode matrices, if *value* is non-zero the specified cell is selected before its state is set to *value*. If *value* is zero and the receiver is a radio-mode matrix, then the currently-selected cell is deselected (providing that empty selection is allowed).

This method redraws the affected cell.

See also: – **allowsEmptySelection:**, – **setState:** (NSCell), – **selectCellAtRow:column:**

setTabKeyTraversesCells:

– (void)**setTabKeyTraversesCells:**(BOOL)*flag*

Sets whether pressing the Tab key advances the key cell to the next selectable cell in the NSMatrix. If *flag* is NO, or if there aren't any selectable cells after the current one, when the user presses the Tab key the next view in the window becomes key. Pressing Shift-Tab causes the key cell to advance in the opposite direction (if *flag* is NO, or if there aren't any selectable cells before the current one, the previous view in the window becomes key).

See also: – **selectKeyViewFollowingView:** (NSWindow), – **selectNextKeyView:** (NSWindow), – **setKeyCell:**, – **setNextText:**

setValidateSize:

– (void)**setValidateSize:**(BOOL)*flag*

If *flag* is YES, then the size information in the NSMatrix is assumed to be correct. If *flag* is NO, then **calcSize** will be invoked before any further drawing is done.

See also: – **calcSize** (NSControl)

sizeToCells

– (void)**sizeToCells**

Changes the width and the height of the NSMatrix frame so that it exactly contains the cells. Does not redraw the NSMatrix.

See also: – **setFrameSize:** (NSView), – **sizeToFit** (NSControl)

sortUsingFunction:context:

– (void)**sortUsingFunction:**(int (*)(id, id ,void *))*comparator* **context:**(void *)*context*

Sorts the receiver's cells in ascending order as defined by the comparison function *comparator*. The comparison function is used to compare two elements at a time and should return NSOrderedAscending if the first element is smaller than the second, NSOrderedDescending if the first element is larger than the second, and NSOrderedSame if the elements are equal. Each time the comparison function is called, it's passed *context* as its third argument. This allows the comparison to be based on some outside parameter, such as whether character sorting is case-sensitive or case-insensitive.

See also: – **sortUsingFunction:context:** (NSMutableArray)

sortUsingSelector:

– (void)**sortUsingSelector:**(SEL)*comparator*

Sorts the receiver's cells in ascending order as defined by the comparison method *comparator*. The *comparator* message is sent to each object in the matrix, and has as its single argument another object in the array. The comparison method is used to compare two elements at a time and should return NSOrderedAscending if the receiver is smaller than the argument, NSOrderedDescending if the receiver is larger than the argument, and NSOrderedSame if they are equal.

See also: – **sortUsingSelector:** (NSMutableArray)

tabKeyTraversesCells

– (BOOL)**tabKeyTraversesCells**

Returns whether pressing the Tab key advances the key cell to the next selectable cell in the NSMatrix.

See also: – **keyCell:**, – **setTabKeyTraversesCells:**

textDidBeginEditing:

– (void)**textDidBeginEditing:**(NSNotification *)*notification*

Invoked when there's a change in the text after the receiver gains first responder status. This method's default behavior is to post an NSControlTextDidBeginEditingNotification along with the receiving object to the default notification center. The posted notification's user info contains the contents of *notification*'s user info dictionary, plus an additional key/value pair. The additional key is "NSFieldEditor"; the value for this key is the text object that began editing.

See also: – **textDidChange:**, – **textDidEndEditing:**, – **textShouldBeginEditing:**

textDidChange:

– (void)**textDidChange:**(NSNotification *)*notification*

Invoked upon a key-down event or paste operation that changes the receiver's contents. This method's default behavior is to pass this message on to the selected cell (if the selected cell responds to **textDidChange:**), and then to post an NSControlTextDidChangeNotification along with the receiving object to the default notification center. The posted notification's user info contains the contents of *notification*'s user info dictionary, plus an additional key/value pair. The additional key is "NSFieldEditor"; the value for this key is the text object that changed.

See also: – **textDidBeginEditing:**, – **textDidEndEditing:**

textDidEndEditing:

– (void)**textDidEndEditing:**(NSNotification *)*notification*

Invoked when text editing ends. This method's default behavior is to post an NSControlTextDidEndEditingNotification along with the receiving object to the default notification center. The posted notification's user info contains the contents of *notification*'s user info dictionary, plus an additional key/value pair. The additional key is "NSFieldEditor"; the value for this key is the text object that began editing. After posting the notification, **textDidEndEditing:** sends an **endEditing:** message to the selected cell, draws and makes the selected cell key, and then takes the appropriate action based on which key was used to end editing (Return, Tab, or Back-Tab).

See also: – **textDidBeginEditing:**, – **textDidChange:**, – **textShouldEndEditing:**

textShouldBeginEditing:

– (BOOL)**textShouldBeginEditing:**(NSText *)*textObject*

Invoked to let the NSTextField respond to impending changes to its text. This method's default behavior is to send **control:textShouldBeginEditing:** to the receiver's delegate (passing the receiver and *textObject* as parameters). **textShouldBeginEditing:** returns the value obtained from **control:textShouldBeginEditing:**, unless the delegate doesn't respond to that method, in which case it returns YES, thereby allowing text editing to proceed.

See also: – **delegate**

textShouldEndEditing:

– (BOOL)**textShouldEndEditing:**(NSText *)*textObject*

Invoked to let the NSTextField respond to impending loss of first-responder status. This method's default behavior checks the text field for validity; providing that the field contents are deemed valid, and providing that the delegate responds, **control:textShouldEndEditing:** is sent to the receiver's delegate (passing the receiver and *textObject* as parameters). If the contents of the text field aren't valid, **textShouldEndEditing:** sends the error action to the selected cell's target.

textShouldEndEditing: returns NO if the text field contains invalid contents, otherwise it returns the value passed back from **control:textShouldEndEditing:**.

See also: – **delegate**, – **errorAction**