Paul S. McCarthy and Eric Sunshine   --   March 25, 1998

MiscTableScroll

Inherits From: NSScrollView : NSView : NSResponder : NSObject

Conforms To: NSCoding (NSObject)

Declared In: MiscTableScroll.h

Class Description

This class provides a convenient and powerful user-interface object for displaying and manipulating tabular data. The appearance and behavior is similar to NSTableView but is functionally superior.   Although it inherits from NSScrollView, the programmatic interface is similar to the NSMatrix class.

User Interface Highlights

Scrollable matrix.
Column and row titles.
Columns and rows can be resized.
Columns and rows can be dragged (rearranged).
Automatically sorts rows when columns are rearranged.

Direct user control of sort direction (ascending vs. descending).
Incremental search.
Exports contents as ASCII, ASCII-delimited, or dBASE.
Prints column and row titles.
Keyboard & mouse control.
Interacts with the Pasteboard, Font Panel, and Services.
Alt-click works in highlight mode.

Programmatic Interface Highlights

Each column and row can have its own size.
Each column can have its own cell-type.
Lazy-mode for large amounts of data.
Programmatic interface for multiple selection.
Built-in sort support.
Built-in image-dragging support.
Smart memory management.
Delegate methods for most features.
Simple indexed access to rows and columns.
Easy-to-use text-cell editing.
Full control over selection.

Similarities between Rows and Columns -- Slots and Borders

Rows and columns are treated equally wherever it is practical and desirable to do so.   Almost every action and option that is available for columns is also available for rows and vice versa.   *Slot* is the generic term for a single column or

row.   *Border* is the generic term for row or column orientation.   *Size* is the generic term for width or height.   Most methods come in two flavors: a row/column specific flavor that uses *row* or *column* as part of the name; and a generic flavor that has a *border* argument and (when needed) a *slot* argument.   Here are some examples:

Generic	Specific
- border:setSlotTitlesOn:	- setColumnTitlesOn:
	- setRowTitlesOn:
- border:setSlot:size:	- setColumn:size:
	- setRow:size:

Differences between Rows and Columns

There are some differences between rows and columns.   This object is designed to maximize the efficiency of displaying many rows of data.   Hence, it is faster to add and remove rows than columns.   You should set up all your columns in InterfaceBuilder, or while the table is empty, then add and remove rows afterwards.   Rows are cached for re-use with the `-renewRows:` method.   This makes it very fast to change the contents of the table on a row-oriented basis.   Despite the row-oriented bias, column-oriented operations can be performed at any time; they will just be slower than the corresponding row-oriented operations.   This behavior is intrinsic to the implementation, it cannot be changed.

Cell-prototypes are only used for columns.   This behavior can only be changed by subclassing.

Selection in the body of a table performs selection on a row-wise basis.   This behavior can be changed programmatically via the `-trackBy:` method.

There are numerous default settings which differ between columns and rows.   Most of these options can be changed in InterfaceBuilder; all of them can be changed programmatically.   Here is a summary of the defaults which differ

between rows and columns:

Option	Column Default	Row Default
modifier drag	NO	YES
uniform size	NO	YES
user sizeable	YES	NO
user draggable	YES	NO
titles displayed	YES	NO
title mode	Custom	Auto-Numbered

Slot Sizing

Uniform size is the simplest sizing mode.   When you set the uniform size of a border to any non-zero value, all slots in that border will have the same (uniform) size.   Setting the uniform size of a border to zero enables slots to have individual sizes.   By default, rows are uniform size, columns are not.

If uniform sizing is not set for a border, the following sizing information is maintained for each slot in the border:

target size
minimum size
maximum size
adjusted size
user-sizeable-flag
autosize-flag

Target size is the desired size for a particular slot.   Minimum and maximum sizes are the lower and upper bounds for the size of a slot.   Adjusted size is the final display size of the slot after all other factors have been taken into account.

When a slot is marked as user-sizeable, the user will be able to resize the slot (subject to further conditions described below).   The adjusted size of autosize slots will be increased if needed to prevent a "gap" from appearing when the table is smaller than the display region.   The flag values are mutually independent, but user-sizeable and autosize do not mix well.   They cause bizarre, counter-intuitive behavior on narrow tables in wide views.

Users resize columns by dragging the right-hand edge of the column's title cell to the desired width.   Likewise, users resize rows by dragging the bottom edge of the row's title cell to the desired height.   The cursor changes to a horizontal or vertical resize cursor whenever the cursor is over one of the resizing areas.   When users resize a slot, they are setting the target size for the slot on non-uniform-size borders.   On uniform-size borders, they are setting the new uniform-size for the border.

All of the following conditions must be met to enable the user to resize a particular slot:

(a) The title cells must be displayed (`-border:setSlotTitlesOn:YES`)
(b) The border in question must allow user-sizing of slots (`-border:setSizeableSlots:YES`)
(c) The border must be uniform-size (`-setBorder:uniformSizeSlots:`) or the slot in question must be user-sizeable (`-border:setSlot:sizeable:YES`).
(d) There must be some room to grow or shrink between the slot's current adjusted size and the slot's minimum and maximum sizes.

All of these conditions are met by default for new columns, unless you explicitly disable one of the global options for column sizing.

Slot Dragging and Indexing -- Visual vs. Physical

Dragging and sizing are independent of each other.   You can have borders that are not sizeable in any way, but are still draggable, and vice versa.   You can also have borders that are both draggable and sizeable, or neither draggable

nor sizeable.

Users drag slots by dragging the title cells until the leading edge is over the desired new location and "dropping" the slot there.   For example, the left edge of the cell shows you where the column will end up when dragging a column to the left.   Likewise, the right edge is used when the column is dragged to the right.   This makes it possible to clearly see the new location without guessing.

Dragging must be enabled for that border.   By default, columns are dragged with an unmodified drag, and they are selected with a command-drag.   By default, rows are selected with an unmodified drag, and they are dragged with a command-drag.   By default, dragging is enabled for columns, but not for rows.

If slot-dragging is enabled for a border then an internal mapping vector is maintained which translates the original physical position of the slot to its current visual position.   All programmer-interface methods and all delegate call-back methods use the original physical position of the slot so you can ignore the current visual ordering in your programs. If you need or want to examine the current visual ordering, you can do so with the `-border:slotPosition:` and `-border:slotAtPosition:` methods.

Keyboard Operations

MiscTableScroll provides keyboard control over almost all functions.   Keyboard equivalents are available for scrolling, selection, and performing actions (simulating double-click).   This class displays a dashed rectangle around the slot which is currently the focus of keyboard operations.   Display of the cursor can be disabled and re-enabled with calls to the methods `-disableCursor` and `-enableCursor`.

MiscTableScroll instances can be linked into the next-key-view chain just like all other Views.   This can be done directly in InterfaceBuilder™, or programmatically via `-nextKeyView`, `-setNextKeyView:`, and related functions.

The keyboard *cursor* can be moved with the standard arrow keys as well as the editing keys on the numeric keypad,

including the page-up, page-down, home, and end keys -- as well as the *real* page-up, page-down, home, and end on keyboards which actually supply these keys. For functions which do not normally appear on any keyboard -- such as page-left and page-right -- modified arrow-keys can be used. Please refer to the following table:

Key[2]	Action	Action Modified[3]
space	select slot	select slot
return	perform action	perform action
up-arrow	up	page-up
down-arrow	down	page-down
left-arrow	left	page-left
right-arrow	right	page-right
page-up	page-up	top-edge
page-down	page-down	bottom-edge
home	top-edge	left-edge
end	bottom-edge	right-edge
enter[*]	perform action	perform action
insert[*]	select slot	select slot

[2] These functions are recognized when generated from both the standard editing keys as well as those on the numeric keypad.

[3] Modified actions are produced by holding down a modifier while typing the primary key. Any of Shift, Control, or Alt can be used to produce a modified action -- and all have the same affect. So, for instance, one can generate a page-left from any of ctrl-left-arrow, shift-left-arrow, or alt-left-arrow.

[*] Applicable to the numeric keypad only.

Keys which perform the *select-slot* function simulate a single mouse-click. Keys which perform *perform-action* simulate a double mouse-click. When performing a *select-slot* via the keyboard, one can use the same modifier keys one uses when selecting via the mouse. In other words, one can use shift-single-click to select multiple slots (one at a time) with the mouse. Likewise one can use shift-space to select multiple slots (one at a time) with the keyboard.

Selection Modes

Three selection modes are supported: List, Highlight, and Radio.   Unlike NSMatrix, this class treats the different selection modes as uniformly as possible (see *Mouse Tracking* below).   Methods such as `-selectionMode` and `-setSelectionMode:` allow direct control over the mode.

Methods such as `-hasRowSelection, -numberOfSelectedRows, -rowIsSelected:,` as well as the border and slot variations allow selection querying, while methods such as `-selectedRows, -selectedRowTags, -selectRows:,` and `-selectRowTags:` along with their variations allow batch-oriented selection modification.

Mouse Tracking

The manner in which MiscTableScroll performs mouse-tracking is different from the manner in which NSMatrix does so.   This class gives the cells the opportunity to track the mouse in *all* selection modes, whereas NSMatrix allows the cells to participate in mouse-tracking in all *but* List mode.   NSMatrix make a special case of List mode, and in addition to the mouse-tracking difference it also modifies the cells' *state* variables in this mode.   In all other modes NSMatrix allows the cell's mouse-tracking methods to manipulate the *state* rather than doing so itself.   MiscTableScroll, on the other hand, treats all selection modes uniformly.   It always highlights a cell via its *highlight* flag and *never* alters the cell's *state* -- instead it leaves alteration of *state* to the cell's mouse-tracking methods.

Upon receipt of a `-mouseDown:`, this class gives the cell at the mouse-down location an opportunity to track the mouse by invoking its `-trackMouse:inRect:ofView:untilMouseUp:` method.   That method normally tracks the mouse until either a mouse-up event in which case it returns YES, or until the mouse leaves the cell-frame in which case it returns NO.   If `-trackMouse:inRect:ofView:untilMouseUp:` returns NO, then MiscTableScroll goes into its own modal-responder loops, continues tracking the mouse itself, and updates the selection appropriately until a mouse-up event. Only the cell under the mouse-down event is given a chance to participate in tracking -- after that no other cells are

offered the opportunity.

Image Dragging

MiscTableScroll has built in support for dragging images right out of cells using the standard NeXT dragging services. Dragging is enabled by implementing a few of simple methods in the delegate or dataDelegate.   Two required methods are `-tableScroll:allowDragOperationAtRow:column:` and `-tableScroll:preparePasteboard:forDragOperationAtRow:column:`.   Each method is passed a pointer to the MiscTableScroll and the cell's physical coordinates.   The first method gives the delegate or dataDelegate a chance to allow or veto the drag operation.   The second method is responsible for declaring types and, possibly, providing data for the pasteboard.

A third delegate method `-tableScroll:imageForDragOperationAtRow:column:` is required for non-image cells and optional for cells which contain an image.   This method allows the delegate or dataDelegate to supply an image for dragging.   If the cell from which dragging is taking place contains its own image, then this method need not be implemented or can return 0, in which case the cell's own image is used by default.

Other methods allow the delegate or dataDelegate to respond to the standard dragging source protocol methods.

As usual, when dealing with pasteboards, keep in mind that if a non-nil *owner* is specified, the NSPasteboard will retain it.   Only upon the initiation of another dragging operation will the owner receive the `-pasteboardChangedOwner:` message and be released.   Therefore the *owner* needs to remain in a valid state, along with any necessary data, even after completion of the drag operation.

Prototype Cells

Each column maintains a prototype cell which is used when new rows are created.   When new rows are created, a

`-copyWithZone:` message is sent to the prototype cell for each column, and the new copy of the prototype cell is put into the new row.   This means that all prototype cells must implement the `-copyWithZone:` method appropriately. Generally this implies performing a *deep* copy.

The prototype cell can be one of the built-in types (text, icon, or button), it can be supplied by the delegate, or you can set it programmatically.   If you set a prototype cell programmatically, the MiscTableScroll object will retain the prototype cell, and will release it when it is finished with it.   If the delegate provides the prototype cell, the delegate retains ownership -- the MiscTableScroll object will neither retain nor release prototype cells provided by the delegate.

Cell Owner and Inherited Font and Color Attributes

The MiscTableScroll class implements an informal *owner* protocol with the cells that it manages.   The MiscTableCell class implements the other side this informal protocol.   This protocol enables the cells to specify that they want to inherit their font and color attributes from their owner.   In this case, the owner is the MiscTableScroll object.   This makes it possible to propagate global *default* font and color attributes to all such cells easily and efficiently.   At the same time, individual cells can use custom font and color attributes which will override the global default inherited values.   Cells that use the inherited values do not need to store copies of those inherited values.   Only cells that use custom values need to remember and store their own custom values.   See the documentation for the MiscTableCell class for more details.

Cell Owner and Owner-Draw

Another feature of the informal *owner* protocol is a specialized notion of delegated drawing.   All cells that respond YES to the `-ownerDraw` message are drawn by the MiscTableScroll rather than drawing themselves.   Since the MiscTableScroll object typically manages many cells with similar font and color attributes, the MiscTableScroll object can eliminate large amounts of redundant font and color setting PostScript code.   Likewise, contiguous cells with the same background colors have all of their backgrounds drawn with a single PostScript operator rather than several.

The built-in drawing mechanism also eliminates a lot of very expensive clipping path operations by simply not drawing partial characters that would be clipped.   (You can force the partially visible text to be displayed with the `-setDrawClippedText:YES` message.)   Any object used as a cell in an instance of MiscTableScroll object can take advantage of these facilities as long as the cell consists of plain (single font) text with a single (optional) icon, and does not have borders or other drawing requirements.   This results in a very considerable improvement in drawing performance -- especially noticable on older, slower CPUs.   <FIXME: Owner-draw is currently restricted to cells containing 7-bit ASCII text.   See OPENSTEP-BUGS.txt>

Lazy vs. Eager

In general, eager mode is much easier to use than lazy mode.   In eager mode, you can usually take advantage of the cells themselves to store the data that you are retrieving and displaying.   Even complex data types can be stored by allocating the record and a storing pointer in the cell's tag or represented-object, or the tag for a row.   By contrast, lazy mode forces you to manage all the storage yourself.   In eager mode, the MiscTableScroll object allocates and manages a dense matrix of cells, which you fill in with data as needed.   In lazy mode, the MiscTableScroll object does not manage any cells at all.   You are responsible for implementing `-tableScroll:cellAtRow:column:` to provide the MiscTableScroll object with a cell whenever it needs one.   In lazy mode, you generally want to implement many of the `-tableScroll:...ValueAtRow:column:` methods to improve performance.

Usage Tips

For simple, flexible and maintainable access to the columns of the table scroll, you should declare an **enum** which identifies the columns in the MiscTableScroll, for instance:

```
enum
  {
  PHOTO_SLOT,
```

```
        LAST_NAME_SLOT,
        FIRST_NAME_SLOT,
        AGE_SLOT
        };
```

Then you use the enumeration identifiers whenever you need to specify a column.   Using an enumeration this way lets you add, remove and shuffle the slots just by updating the enum declaration, rather than searching through the code to find all the places that need to be fixed.   It also makes your code more readable.

There are two standard patterns for putting the data into eager (non-lazy) MiscTableScroll objects: `-renewRows:` and `-addRow`.

-renewRows:
When you know the number of rows in advance, it is most efficient to use the `-renewRows:` method to tell the MiscTableScroll object the number of rows that you will need.   Your code will usually be structured as follows:

```
    int row;
    int const nrows = [dataSource count];

    [tableScroll renewRows:nrows];
    for (row = 0;   row < nrows;   row++)
        {
        id item = [dataSource itemAtRow:row];
        [tableScroll setRow:row tag:(int)[item retain]];
        [[tableScroll cellAtRow:row column:LAST_NAME_SLOT] setStringValue:[item lastName]];
        [[tableScroll cellAtRow:row column:AGE_SLOT] setIntValue:[item age]];
        //... and so on ...
        }
```

```
    if ([tableScroll autoSortRows])
        [tableScroll sortRows];
```

-addRow

When you do not know the final number of rows in advance, your code will usually be structured as follows:

```
id item;
int row = 0;
[tableScroll empty];

while ((item = [self getNextItem]) != 0)
    {
    [tableScroll addRow];
    [tableScroll setRow:row tag:(int)[item retain]];
    [[tableScroll cellAtRow:row column:LAST_NAME_SLOT] setStringValue:[item lastName]];
    [[tableScroll cellAtRow:row column:AGE_SLOT] setIntValue:[item age]];
    //... and so on ...
    row++;
    }

[tableScroll sizeToCells];

if ([tableScroll autoSortRows])
    [tableScroll sortRows];
```

A common programming mistake is forgetting to call `-sizeToCells`. You must call `-sizeToCells` after you have finished adding rows so that the MiscTableScroll can update the frames of its various subviews.   If you forget to call `-sizeToCells`, the MiscTableScroll will appear to be empty when it is displayed.

Smart Memory Management

The MiscTableScroll class implements *smart* memory management.   It does not allocate support structures until and unless they are needed.   For example, since rows are uniform-size by default, the MiscTableScroll will not allocate the array of sizing-info structures until and unless you make the rows non-uniform size.   In a complimentary fashion, if you make the columns uniform size, the MiscTableScroll object will release the sizing-info array for the columns. Similarly, custom titles must be stored in an array.   However, no other title-mode requires this array, and the array will only exist   for borders that have custom titles.   Likewise, the visual-to-physical mapping vector that supports user-draggable slots is only created when the first slot is actually moved.   Even if the draggable option is turned on, you will not incur the memory overhead until the option is used.   The net result of all this is that you only pay for the features that you use.

On the other hand, you do pay for the features that you do use.   These extra features exact a price in storage and cpu.   You should be careful about using them for rows when you expect thousands of rows.

The MiscTableScroll class is designed to provide high-quality, consistent, flexible behavior to the user while supporting a wide range of load requirements -- from dozens of   rows to hundreds of thousands of rows.   Smart memory management is an important element in achieving that goal.

Errors

The only exception currently raised by MiscTableScroll is NSInternalInconsistencyException.   This exception is raised when MiscTableScroll detects an internal inconsistency.   This is typically indicative of a bug within this class.

This exception is also currently raised when an out-of-range slot index is passed as an argument to a method which can not sensibly handle the bad index.   For instance, there is no sensible value which `-rowTag:` can return when given a bad index, hence it raises an exception.   On the other hand no exception is raised by `-selectRow:` since it can simply ignore the bad index.   <FIXME: In the future such methods will raise a more suitable exception, such as

NSRangeException.>

Method Types

Creating and destroying instances	±€initWithFrame: ±€dealloc
Delegates	±€setDelgate: ±€delegate ± setDataDelegate: ± dataDelegate
Transmitting action	±€setTarget: ±€target ± setDoubleTarget: ± doubleTarget ± setAction: ± action ± setDoubleAction: ± doubleAction ± sendAction:to:forAllCells: ± sendAction:to: ± sendAction ± sendDoubleAction ± sendActionIfEnabled

± sendDoubleActionIfEnabled

± tracking

± clickedSlot:
± clickedRow
± clickedColumn
± clickedCell

Enabling and disabling   ±€setEnabled:
±€isEnabled

Selection   ± setSelectionMode:
±€selectionMode

± border:slotIsSelected:
± rowIsSelected:
± columnIsSelected:
± cellIsSelectedAtRow:column:

± selectedSlot:
± selectedRow
± selectedColumn
± selectedCell
± selectedSlotTags:
± selectedRowTags
± selectedColumnTags
± selectedSlots:

± selectedRows
± selectedColumns

± border:selectSlot:byExtension:
± border:selectSlot:
± selectRow:byExtension:
± selectRow:
± selectColumn:byExtension:
± selectColumn:
± border:selectSlotTags:byExtension:
± border:selectSlotTags:
± selectRowTags:byExtension:
± selectRowTags:
± selectColumnTags:byExtension:
± selectColumnTags:
± border:selectSlots:byExtension:
± border:selectSlots:
± selectRows:byExtension:
± selectRows:
± selectColumns:byExtension:
± selectColumns:
± selectAllSlots:
± selectAllRows
± selectAllColumns
± selectAll:

±€border:deselectSlot:
±€deselectRow:
±€deselectColumn:
± border:deselectSlotTags:
±€deselectRowTags:
±€deselectColumnTags:
± border:deselectSlots:
±€deselectRows:
±€deselectColumns:
± clearSlotSelection:
± clearRowSelection
± clearColumnSelection
±€clearSelection
±€deselectAll:

± hasSlotSelection:
± hasRowSelection
± hasColumnSelection
± hasMultipleSlotSelection:
± hasMultipleRowSelection
± hasMultipleColumnSelection
± numberOfSelectedSlots:
± numberOfSelectedRows
± numberOfSelectedColumns

± trackBy:

± trackingBy

± selectionChanged

Keyboard cursor
± border:setCursorSlot:
± clearCursorSlot:
± cursorSlot:
± hasValidCursorSlot:
± clearCursor
± clearCursorColumn
± clearCursorRow
± cursorColumn
± cursorRow
± disableCursor
± enableCursor
± hasValidCursorColumn
± hasValidCursorRow
± isCursorEnabled
± setCursorColumn:
± setCursorRow:

Incremental search
± incrementalSearch:
± doIncrementalSearch:column:
± getISearchColumn:
± doGetISearchColumn:

Scrolling
± scrollCellToVisibleAtRow:column:

± scrollColumnToVisible:
± scrollRowToVisible:
± scrollSelectionToVisible

± border:setFirstVisibleSlot:
± border:setLastVisibleSlot:
± border:slotIsVisible:
± firstVisibleSlot:
± lastVisibleSlot:
± numberOfVisibleSlots:

± columnIsVisible:
± firstVisibleColumn
± lastVisibleColumn
± numberOfVisibleColumns
± setFirstVisibleColumn:
± setLastVisibleColumn:

± rowIsVisible:
± firstVisibleRow
± lastVisibleRow
± numberOfVisibleRows
± setFirstVisibleRow:
± setLastVisibleRow:

Titles   ± border:setSlotTitlesOn:
                          ± setColumnTitlesOn:

± setRowTitlesOn:
± slotTitlesOn:
± columnTitlesOn
± rowTitlesOn

± border:setSlotTitleMode:
± setColumnTitleMode:
± setRowTitleMode:
± slotTitleMode:
± columnTitleMode
± rowTitleMode

± border:setSlot:title:
± setColumn:title:
± setRow:title:
± border:slotTitle:
± columnTitle:
± rowTitle:

± cornerTitle
± setCornerTitle:

Sizing   ± border:setUniformSizeSlots:
± border:setMinUniformSizeSlots:
± border:setMaxUniformSizeSlots:
± setUniformSizeColumns:
± setMinUniformSizeColumns:

± setMaxUniformSizeColumns:
± setUniformSizeRows:
± setMinUniformSizeRows:
± setMaxUniformSizeRows:
± uniformSizeSlots:
± minUniformSizeSlots:
± maxUniformSizeSlots:
± uniformSizeColumns
± minUniformSizeColumns
± maxUniformSizeColumns
± uniformSizeRows
± minUniformSizeRows
± maxUniformSizeRows

± border:setSizeableSlots:
± setSizeableColumns:
± setSizeableRows:
± sizeableSlots:
± sizeableColumns:
± sizeableRows:

± border:setSlot:size:
± setColumn:size:
± setRow:size:
± border:slotSize:
± columnSize:

± rowSize:

± border:setSlot:minSize:
± setColumn:minSize:
± setRow:minSize:
± border:slotMinSize:
± columnMinSize:
± rowMinSize:

± border:setSlot:maxSize:
± setColumn:maxSize:
± setRow:maxSize:
± border:slotMaxSize:
± columnMaxSize:
± rowMaxSize:

± border:slotAdjustedSize:
± columnAdjustedSize:
± rowAdjustedSize:

± border:setSlot:autosize:
± setColumn:autosize:
± setRow:autosize:
± border:slotIsAutosize:
± columnIsAutosize:
± rowIsAutosize:

± border:setSlot:sizeable:

± setColumn:sizeable:
± setRow:sizeable:
± border:slotIsSizeable:
± columnIsSizeable:
± rowIsSizeable:

± constrainSize
± totalSize:
± totalHeight
± totalWidth

± border:slotResized:

± border:setSlotTitlesSize:
± slotTitlesSize:
± setColumnTitlesHeight:
± columnTitlesHeight
± setRowTitlesWidth:
± rowTitlesWidth

± sizeToCells
± sizeToFit

Dragging
± border:setDraggableSlots:
± setDraggableColumns:
± setDraggableRows:
± draggableSlots:
± draggableColumns

± draggableRows

± border:setModifierDragSlots:
± setModifierDragColumns:
± setModifierDragRows:
± modifierDragSlots:
± modifierDragColumns
± modifierDragRows

± border:moveSlotFrom:to:
± moveColumnFrom:to:
± moveRowFrom:to:

± border:slotAtPosition:
± columnAtPosition:
± rowAtPosition:

± border:slotPosition:
± columnPosition:
± rowPosition:

± border:physicalToVisual:
± border:visualToPhysical:

± border:slotDraggedFrom:to:

Inserting and deleting   ± numberOfSlots:
± numberOfColumns

± numberOfRows

± addSlot:
± addColumn
± addRow

± border:insertSlot:
± insertColumn:
± insertRow:

± border:removeSlot:
± removeColumn:
± removeRow:

± empty
± emptyAndReleaseCells
± renewRows:

Cell prototypes   ± border:setSlot:cellType:
± setColumn:cellType:
± setRow:cellType:
± border:slotCellType:
± columnCellType:
± rowCellType:

± border:setSlot:cellPrototype:
± setColumn:cellPrototype:
± setRow:cellPrototype:

± border:slotCellPrototype:
± columnCellPrototype:
± rowCellPrototype:

Tags

± tag
± setTag:

± border:setSlot:tag:
± setColumn:tag:
± setRow:tag:
± border:slotTag:
± columnTag:
± rowTag:
± tagAtRow:column:

Drawing

± drawCellAtRow:column:
± border:drawSlot:
± drawRow:
± drawColumn:
± border:drawSlotTitle:
± drawRowTitle:
± drawColumnTitle:
± selectionChanged
±€cellFrameAtRow:column:
± documentClipRect
± drawClippedText
± setDrawClippedText:

Editing	± abortEditing
	± suspendEditing
	± resumeEditing
	± isEditing
	± getPreviousEditRow:column:
	± getNextEditRow:column:
	± getNext:editRow:column:
	± edit:atRow:column:
	± canEdit:atRow:column:
	± editIfAble:atRow:column:
	± editCellAtRow:column:
	± textDidBeginEditing:
	± textDidChange:
	± textDidEndEditing:
	± textShouldBeginEditing:
	± textShouldEndEditing:
Data control	± setLazy:
	± isLazy
	± cellAtRow:column:
	± bufferCount
	± empty
	± emptyAndReleaseCells
	± sizeToCells

± addSlot:
± border:insertSlot:
± border:removeSlot:
± numberOfSlots:

± addColumn
± insertColumn:
± removeColumn:
± numberOfColumns

± addRow
± insertRow:
± removeRow:
± numberOfRows
± renewRows:

± doRetireCell:atRow:column:
± doReviveCell:atRow:column:
± retireCell:atRow:column:
± reviveCell:atRow:column:

± tagAtRow:column:
± intValueAtRow:column:
± floatValueAtRow:column:
± doubleValueAtRow:column:
± stringValueAtRow:column:
± titleAtRow:column:

± stateAtRow:column:

Sorting

± autoSortColumns
± autoSortRows
± autoSortSlots:
± border:setAutoSortSlots:
± border:setSlot:sortDirection:
± border:setSlot:sortFunction:
± border:setSlot:sortType:
± border:setSlotSortVector:
± border:slotSortDirection:
± border:slotSortFunction:
± border:slotSortType:
± border:sortSlot:
± slotSortVector:
± columnSortDirection:
± columnSortFunction:
± columnSortType:
± columnSortVector
± compareSlotFunction
± rowSortDirection:
± rowSortFunction:
± rowSortType:
± rowSortVector
± setAutoSortColumns:
± setAutoSortRows:

± setColumn:sortDirection:
± setColumn:sortFunction:
± setColumn:sortType:
± setColumnSortVector:
± setCompareSlotFunction:
± setRow:sortDirection:
± setRow:sortFunction:
± setRow:sortType:
± setRowSortVector:
± sortColumn:
± sortColumns
± sortRow:
± sortRows
± sortSlots:

± slotsAreSorted:
± rowsAreSorted
± columnsAreSorted
± border:slotIsSorted:
± columnIsSorted:
± rowIsSorted:

± border:compareSlots::
± border:compareSlots::info:
± compareColumns::
± compareColumns::info:

	± compareRows::
	± compareRows::info:
	± sortInfoInit:border:
	± sortInfoDone:
Font	+ defaultFont
	± font
	± setFont:
Color	± backgroundColor
	± color
	+ defaultBackgroundColor
	+ defaultSelectedBackgroundColor
	+ defaultSelectedTextColor
	+ defaultTextColor
	± selectedBackgroundColor
	± selectedTextColor
	± setBackgroundColor:
	± setColor:
	± setSelectedBackgroundColor:
	± setSelectedTextColor:
	± setTextColor:
	± textColor
Multicast	± makeCellsPerformSelector:
	± makeCellsPerformSelector:selectedOnly:

± makeCellsPerformSelector:with:
± makeCellsPerformSelector:with:selectedOnly:
± makeCellsPerformSelector:with:with:
± makeCellsPerformSelector:with:with:selectedOnly:

Finding cells / tags

± border:slotWithTag:
± cellWithTag:
± columnWithTag:
± rowWithTag:
± getRow:column:ofCell:
± getRow:column:ofCellWithTag:
±€getRow:column:forPoint:

Save / restore

± border:setSlotOrder:
± border:setSlotOrderFromString:
± slotOrder:
± slotOrderAsString:
± border:setSlotSizes:
± border:setSlotSizesFromString:
± slotSizes:
± slotSizesAsString:
± columnOrder
± columnOrderAsString
± columnSizes
± columnSizesAsString
± rowOrder

± rowOrderAsString
± rowSizes
± rowSizesAsString
± setColumnOrder:
± setColumnOrderFromString:
± setColumnSizes:
± setColumnSizesFromString:
± setRowOrder:
± setRowOrderFromString:
± setRowSizes:
± setRowSizesFromString:

Pasteboard and services

± copy:
± cut:
± builtinCanWritePboardType:
± builtinReadSelectionFromPasteboard:
± builtinRegisterServicesTypes
± builtinValidRequestorForSendType:returnType:
± builtinStringForPboardType:
± builtinWriteSelectionToPasteboard:types:
± canWritePboardType:
± readSelectionFromPasteboard:
± registerServicesTypes
± validRequestorForSendType:returnType:
± stringForNSStringPboardType
± stringForNSTabularTextPBoardType

	± stringForPboardType:
	± writeSelectionToPasteboard:types:
Printing	± print:
Methods implemented by delegate	± tableScroll:abortEditAtRow:column:
	± tableScroll:allowDragOperationAtRow:column:
	± tableScroll:backgroundColorChangedTo:
	± tableScroll:border:slotDraggedFrom:to:
	± tableScroll:border:slotPrototype:
	± tableScroll:border:slotResized:
	± tableScroll:border:slotSortReversed:
	± tableScroll:border:slotTitle:
	± tableScrollBufferCount:
	± tableScroll:canEdit:atRow:column:
	± tableScroll:canWritePboardType:
	± tableScroll:cellAtRow:column:
	± tableScroll:changeFont:to:
	± tableScroll:didEdit:atRow:column:
	± tableScrollDidPrint:
	± tableScroll:draggingSourceOperationMaskForLocal:
	± tableScroll:doubleValueAtRow:column:
	± tableScroll:floatValueAtRow:column:
	± tableScroll:fontChangedFrom:to:
	± tableScroll:getISearchColumn:
	± tableScrollIgnoreModifierKeysWhileDragging:

± tableScroll:imageForDragOperationAtRow:column:
± tableScroll:intValueAtRow:column:
± tableScroll:preparePasteboard:forDragOperationAtRow:column:
± tableScroll:readSelectionFromPasteboard:
± tableScrollRegisterServicesTypes:
± tableScroll:retireCell:atRow:column:
± tableScroll:reviveCell:atRow:column:
± tableScroll:selectedBackgroundColorChangedTo:
± tableScroll:selectedTextColorChangedTo:
± tableScroll:setStringValue:atRow:column:
± tableScroll:shouldDelayWindowOrderingForEvent:
± tableScroll:stateAtRow:column:
± tableScroll:stringForPboardType:
± tableScroll:stringValueAtRow:column:
± tableScroll:tagAtRow:column:
± tableScroll:textColorChangedTo:
± tableScroll:titleAtRow:column:
± tableScroll:validRequestorForSendType:returnType:
± tableScroll:willEditAtRow:column:
±€tableScrollWillPrint:
± tableScroll:writeSelectionToPasteboard:types:

Methods implemented by cells   ± tableScroll:retireAtRow:column:
                                      ± tableScroll:reviveAtRow:column:

Class Methods

defaultBackgroundColor
+ (NSColor*)**defaultBackgroundColor**

Returns `[NSColor lightGrayColor]`. This is the default background color for new MiscTableScroll objects.

defaultFont
+ (NSFont*)**defaultFont**

Returns the user's preferred font at 12pt size.   This is the default font for new MiscTableScroll objects.

defaultSelectedBackgroundColor
+ (NSColor*)**defaultSelectedBackgroundColor**

Returns `[NSColor whiteColor]`.   This is the default selected background color for new MiscTableScroll objects.

defaultSelectedTextColor
+ (NSColor*)**defaultSelectedTextColor**

Returns `[NSColor blackColor]`.   This is the default selected text color for new MiscTableScroll objects.

defaultTextColor
+ (NSColor*)**defaultTextColor**

Returns `[NSColor blackColor]`.   This is the default text color for new MiscTableScroll objects.

Instance Methods

abortEditing
-   (void)**abortEditing**

Abort cell editing.   Does not go though the normal `-control:textShouldEndEditing:` validation method, nor does it go through the `-controlTextDidEndEditing:` method.

action
-   (SEL)**action**

Returns the action associated with a single click

addColumn
-   (void)**addColumn**

Appends a new column.   See **Usage Tips** in the introduction for a more complete discussion.   Equivalent to:
`-addSlot:MISC_COL_BORDER`.

addRow
-   (void)**addRow**

Appends a new row to the table.   If you know how many rows you will need in advance, you should use **-renewRows:** instead; it will be faster.   If you do not know the number of rows in advance, use this method.   This method is faster than `-insertColumn:`.   Internally, the table pre-allocates rows with a geometric growth pattern so there are only a logarithmic number of allocations.   See **Usage Tips** in the introduction for a more complete discussion.   Equivalent to: `-addSlot:MISC_ROW_BORDER`.

This method does no drawing, nor does it update the frames of the various subviews.   (This enhances performance when adding hundreds or thousands of rows.)   After you have finished adding rows, you must call `-sizeToCells` so that the MiscTableScroll can update the frames of the various subviews.

addSlot:
-   (void)**addSlot:**(MiscBorderType)*b*

Appends a new row or column to the table.   Appending rows is fast (geometric growth, logarithmic allocations, no shifting).   Appending columns is slower (linear growth, linear allocations, lots of shifting).   See **Usage Tips** in the introduction for a more complete discussion.

See also: -renewRows:

autoSortColumns
- (BOOL)**autoSortColumns**

Indicates whether columns will be automatically sorted when the user drags rows.   Equivalent to `-autoSortSlots:MISC_COL_BORDER`.

autoSortRows
- (BOOL)**autoSortRows**

Indicates whether rows will be automatically sorted when the user drags columns.   Equivalent to `-autoSortSlots:MISC_ROW_BORDER`.

autoSortSlots:
- (BOOL)**autoSortSlots:**(MiscBorderType)*b*

Indicates whether or not slots on the given border will be automatically sorted when the user drags (rearranges) slots on the other border.

backgroundColor
- (NSColor*)**backgroundColor**

Returns the current background color for the MiscTableScroll object.   The background color is used as the background color of unselected cells in the table body as well as the *exposure color* for areas not covered by cells.

border:compareSlots::
- (int)**border:**(MiscBorderType)*b* **compareSlots:**(int)*slot1* **:**(int)*slot2*

This method compares two slots.   Returns a value less than zero if *slot1* should sort before *slot2*, zero if *slot1* should sort equally with *slot2*, or greater than zero if *slot1* should sort after *slot2*.   It calls `-sortInfoInit:border:` to compute the sorting information, then calls `-border:compareSlots::info:`, and finally cleans up with `-slotInfoDone:`.

See also: **-border:compareSlots::info:, -border:slotIsSorted:, -border:sortSlot:**

border:compareSlots::info:
- (int)**border:**(MiscBorderType)*b* **compareSlots:**(int)*slot1* **:**(int)*slot2* **info:**(MiscSlotSortInfo*)*sortInfo*

This method compares two slots, given a pointer to the precomputed sorting information.   If you call this method, you are responsible for initializing *sortInfo* by calling `-sortInfoInit:border:`, and then releasing the resources by calling `-sortInfoDone:`.

See also: **-border:compareSlots::, -sortInfoDone:, -sortInfoInit:border:**

border:deselectSlot:
- (void)**border:**(MiscBorderType)*b* **deselectSlot:**(MiscCoord_P)*slot*

Deselects the indicated slot.

border:deselectSlots:
-   (void)**border:**(MiscBorderType)*b* **deselectSlots:**(NSArray*)*slots*

Deselects each slot in *slots*, which should be an array of NSNumber objects each containing a slot index.

border:deselectSlotTags:
-   (void)**border:**(MiscBorderType)*b* **deselectSlotTags:**(NSArray*)*tags*

Deselects all slots whose tag value can be found in *tags*, which should be an array of NSNumber objects.

border:drawSlot:
-   (void)**border:**(MiscBorderType)*b* **drawSlot:**(int)*n*

Draws a single row or column.   This method locks focus on the view if needed.

See also: **-drawCellAtRow:column:, -drawColumn:, -drawRow:**

border:drawSlotTitle:
-   (void)**border:**(MiscBorderType)*b* **drawSlotTitle:**(int)*n*

Draws the title cell for a single row or column.   This method locks focus on the view if needed.   You should never need to call this method in normal use, though it might be useful for subclasses.

See also: **-drawColumnTitle:, -drawRowTitle:**

border:insertSlot:
   - (void)**border:**(MiscBorderType)*b* **insertSlot:**(int)*pos*

Inserts a single row or column at the indicated position.   Position is a zero-based index.   The slot is inserted at physical index *pos*, and also at visual index *pos*.   This method performs linear allocation, and is slower than the corresponding `-addRow` method when adding new rows to a table.

border:moveSlotFrom:to:
   - (void)**border:**(MiscBorderType)*b* **moveSlotFrom:**(int)*from_pos* **to:**(int)*to_pos*

This is equivalent to the user dragging a slot from *from_pos* to *to_pos*.   Both *from_pos* and *to_pos* are zero-based indexes into the current visual ordering of the slots.

border:physicalToVisual:
   - (NSArray*)**border:**(MiscBorderType)*b* **physicalToVisual:**(NSArray*)*list*

This method accepts an array of NSNumber objects each containing a *physical* (original) slot index, and returns an array containing the corresponding *visual* (current) slot indexes.   This is accomplished via a succession of calls to

`-border:slotPosition:.`

See also: **-border:slotPosition:, -border:visualToPhysical:**

border:removeSlot:
-   (void)**border:**(MiscBorderType)*b* **removeSlot:**(int)*pos*

Removes a single row or column.   All cells are released and other internal resources for the slot are deallocated immediately.   If you are just emptying the table so that you can refill it with new data, use `-renewRows:` or `-empty` instead since these methods cache the cells for later re-use.   See **Usage Tips** in the introduction for a more complete discussion.

See also: -empty, -emptyAndReleaseCells, -renewRows:

border:selectSlot:byExtension:
-   (void)**border:**(MiscBorderType)*b* **selectSlot:**(MiscCoord_P)*slot* **byExtension:**(BOOL)*flag*

Selects the indicated slot.   If *flag* is YES then does not clear the previous selection, hence *slot* is added to the existing selection.   If *flag* is NO then the previous selection is cleared.

border:selectSlot:
-   (void)**border:**(MiscBorderType)*b* **selectSlot:**(MiscCoord_P)*slot*

Equivalent to: `-border:`*b* `selectSlot:`*slot* `byExtension:NO.`

border:selectSlots:byExtension:

- (void)**border:**(MiscBorderType)*b* **selectSlots:**(NSArray*)*slots* **byExtension:**(BOOL)*flag*

Selects each slot in *slots*, which should be an array of NSNumber objects each containing a slot index.   If *flag* is YES then does not clear the previous selection, hence *slots* are added to the existing selection.   If *flag* is NO then the previous selection is cleared.

border:selectSlots:

- (void)**border:**(MiscBorderType)*b* **selectSlots:**(NSArray*)*slots*

Equivalent to: `-border:`*b* `selectSlots:`*slots* `byExtension:NO`.

border:selectSlotTags:byExtension:

- (void)**border:**(MiscBorderType)*b* **selectSlotTags:**(NSArray*)*tags* **byExtension:**(BOOL)*flag*

Selects all slots whose tag value can be found in *tags*, which should be an array of NSNumber objects.   If *flag* is YES then does not clear the previous selection, hence the slots found in *tags* are added to the existing selection.   If *flag* is NO then the previous selection is cleared.   This method is useful in conjunction with `-selectedSlotTags:` to save and restore the user's selection when you have tag values that uniquely identify the slots.

border:selectSlotTags:
- (void)**border:**(MiscBorderType)*b* **selectSlotTags:**(NSArray*)*tags*

Equivalent to: `-border:`*b* `selectSlotTags:`*tags* `byExtension:NO.`

border:setAutoSortSlots:
- (void)**border:**(MiscBorderType)*b* **setAutoSortSlots:**(BOOL)*flag*

Instructs the MiscTableScroll object whether or not to automatically sort the slots in border *b*, when the user drags a slot from the other border.   For example, when you tell the MiscTableScroll object to auto-sort rows, the rows will be automatically sorted every time the user drags a column to a new position.   AutoSort is off by default.

border:setCursorSlot:
- (void)**border:**(MiscBorderType)*b* **setCursorSlot:**(MiscCoord_P)*slot*

Sets the keyboard cursor to *slot*.

border:setDraggableSlots:
- (void)**border:**(MiscBorderType)*b* **setDraggableSlots:**(BOOL)*flag*

Enables or disables reordering of the slots.   To let the user drag slots, the titles must be displayed, and the slots must be draggable.

See also: -border:setModifierDragSlots:, -border:setSizeableSlots:, -border:setSlotTitlesOn:

border:setFirstVisibleSlot:
- (void)**border:**(MiscBorderType)*b* **setFirstVisibleSlot:**(int)*n*

Scrolls the table so that slot *n* is the first slot displayed if possible.

border:setLastVisibleSlot:
- (void)**border:**(MiscBorderType)*b* **setLastVisibleSlot:**(int)*n*

Scrolls the table so that slot *n* is the last slot displayed if possible.

border:setMaxUniformSizeSlots:
- (void)**border:**(MiscBorderType)*b* **setMaxUniformSizeSlots:**(float)*size*

Sets the upper bound for user-sizing of a uniform-sized border, *b*.   See **Slot Sizing** in the introduction for more details.

border:setMinUniformSizeSlots:
- (void)**border:**(MiscBorderType)*b* **setMinUniformSizeSlots:**(float)*size*

Sets the lower bound for user-sizing of a uniform-sized border, *b*.   See **Slot Sizing** in the introduction for more details.

border:setModifierDragSlots:
- (void)**border:**(MiscBorderType)*b* **setModifierDragSlots:**(BOOL)*flag*

This option controls whether an unmodified mouse-down initiates selection, or slot-dragging.   When *flag* is YES, an unmodified mouse-down initiates selection, and the user must hold down the command-key to drag a slot.   When *flag* is NO, an unmodified mouse-down initiates dragging, and the user must hold down the command-key to select a slot.   By default, columns are dragged with an unmodified mouse-down and selected when the command-key modifier is used.   By default, rows behave the other way; an unmodified mouse-down initiates selection, and a command-key modifier must be used to initiate dragging.

border:setSizeableSlots:
- (void)**border:**(MiscBorderType)*b* **setSizeableSlots:**(BOOL)*flag*

Enables or disables user-sizing of the slots.   Many conditions must be met to enable the user to resize a particular slot.   See **Slot Sizing** in the introduction for details.

border:setSlot:autosize:
- (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **autosize:**(BOOL)*flag*

Enables or disables *autosizing* for a particular slot.   When YES, the slot will be adjusted proportionately with all other *autosize* slots in the border to meet global minimum or maximum size restrictions for the border as a whole.
Currently, this only has effect for columns in narrow tables displayed in wide NSScrollViews.   See **Slot Sizing** in the

introduction for more details.

border:setSlot:cellPrototype:
- (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **cellPrototype:**(id)*cell*

Set the cell prototype for a slot.   Currently, only column cell prototypes are used.   When new rows are allocated for the table, the cell prototype from each column is sent a `-copyWithZone:` message.   The newly created cell is placed into the newly created row.   Thus all prototype cells must implement the `-copyWithZone:` message appropriately.   (In general, this implies making a *deep* copy.)   The MiscTableScroll retains *cell* when this method is invoked and releases it when it is no longer needed.

See also: **-border:setSlot:cellType:**

border:setSlot:cellType:
- (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **cellType:**(MiscTableCellStyle)*t*

Sets the type of cell that will be used for a particular slot.   Currently, only column cell types have any effect; row cell types are ignored.   The cell type, *t*, can be any of the following (declared in `MiscTableTypes.h`):

```
MISC_TABLE_CELL_TEXT
MISC_TABLE_CELL_IMAGE
MISC_TABLE_CELL_BUTTON
MISC_TABLE_CELL_CALLBACK
```

When this method is called, the MiscTableScroll object will create a prototype cell for the indicated slot of the indicated

type. `MISC_TABLE_CELL_TEXT` creates a text-cell; `MISC_TABLE_CELL_IMAGE` creates an image-cell; `MISC_TABLE_CELL_BUTTON` creates a button-cell; and `MISC_TABLE_CELL_CALLBACK` instructs the MiscTableScroll object to ask the delegate for the for the prototype cell.   <FIXME: What is the interaction with setting an explicit prototype cell?   What message is sent to the delegate to get the prototype?   When is the message sent?   Does it ask for the cell itself, or just a prototype?   Is it sent to the delegate, or the data-delegate>

See also: -border:setSlot:cellPrototype:

border:setSlot:maxSize:
- (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **maxSize:**(float)*size*

Sets the maximum size for slot *n*.   The *size* argument is in units of screen pixels.   See **Slot Sizing** in the introduction for more details.

border:setSlot:minSize:
- (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **minSize:**(float)*size*

Sets the minimum size for slot *n*.   The *size* argument is in units of screen pixels.   See **Slot Sizing** in the introduction for more details.

border:setSlot:size:
- (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **size:**(float)*size*

Sets the *target* size for slot *n*.   The *size* argument is in units of screen pixels.   See **Slot Sizing** in the introduction for more details.

border:setSlot:sizeable:
-   (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **sizeable:**(BOOL)*flag*

Sets the *user-sizeable* flag for slot *n*.   When *flag* is YES, the user will be able to resize the slot.   When *flag* is NO, the user will not be able to resize the slot.   There are many conditions which must be met for a user to be able to resize a slot.   See **Slot Sizing** in the introduction for more details.

border:setSlot:sortDirection:
-   (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **sortDirection:**(MiscSortDirection)*x*

Indicates whether slot *n* should be sorted in ascending or descending order.   *X* must be one of the following two values from `MiscTableTypes.h`:

```
MISC_SORT_ASCENDING
MISC_SORT_DESCENDING
```

All other values are ignored.

See also: **-border:setSlot:sortType:, -border:setSlot:sortFunction:, -border:slotSortDirection:**

border:setSlot:sortFunction:
- (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **sortFunction**:(MiscCompareEntryFunc)*func*

Makes *func* the cell-to-cell comparison routine for the cells in slot *n*.   The function, *func*, must match the following prototype from `MiscTableTypes.h`:

```
typedef int (*MiscCompareEntryFunc)
              ( int r1, int c1, int r2, int c2,
             MiscEntrySortInfo const* entry_info,
             MiscSlotSortInfo* sort_info );
```

The function is given the coordinates of the two cells, and two pointers to structures containing additional sorting information.   The function should return an integer that is: (a) less than zero if the cell at (r1,c1) should sort before the cell at (r2,c2), (b) equal to zero if the two cells should sort equally, or (c) greater than zero if the cell at (r1,c1) should sort after the cell at (r2,c2).

The sort direction (ascending or descending) is applied to the value returned by the cell-to-cell comparison function by the slot-to-slot comparison function.   So if you supply a custom cell-to-cell comparison function you should ignore the sort direction for that slot.   You should always return the *ascending* sort-order value.

Use this method when you need to perform custom sorting that the built-in sort-types cannot accommodate.

See also: -border:setSlot:sortDirection:, -border:setSlot:sortType:, -border:slotSortFunction:

border:setSlot:sortType:
- (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **sortType:**(MiscSortType)*x*

Sets the type of sorting to be used by the built-in cell-to-cell comparison function for cells in slot *n*.   The sort type, *x*, must be one of the following values from `MiscTableTypes.h`:

```
MISC_SORT_STRING_CASE_INSENSITIVE
MISC_SORT_STRING_CASE_SENSITIVE
MISC_SORT_INT
MISC_SORT_UNSIGNED_INT
MISC_SORT_TAG
MISC_SORT_UNSIGNED_TAG
MISC_SORT_FLOAT
MISC_SORT_DOUBLE
MISC_SORT_SKIP
MISC_SORT_TITLE_CASE_INSENSITIVE
MISC_SORT_TITLE_CASE_SENSITIVE
MISC_SORT_STATE
MISC_SORT_UNSIGNED_STATE
```

All other values are ignored.   Each of the types is described below.

`MISC_SORT_STRING_CASE_INSENSITIVE`
`MISC_SORT_STRING_CASE_SENSITIVE`

> The cells are compared as strings.   The string values are retrieved using the `-stringValueAtRow:column:` message. `MISC_SORT_STRING_CASE_INSENSITIVE` is the default sort-type.

`MISC_SORT_INT`
`MISC_SORT_UNSIGNED_INT`

> The cells are compared as integers.   The integer values are retrieved using the `-intValueAtRow:column:`

message.

`MISC_SORT_TAG`
`MISC_SORT_UNSIGNED_TAG`

The cells are compared as integers.   The integer values are retrieved using the `-tagAtRow:column:` message. This feature is useful for sorting that is handled "behind-the-scenes".   For example, if the slot holds date information, you can put a numeric representation of the date (such as that obtained with `-[NSDate timeIntervalSinceReferenceDate]`) into the cell's tag, and format the string value any way you wish.   The slot will sort correctly regardless of the display format.   It is also useful for slots that hold keywords from an ordered set of values, like the following enumeration:

`enum Severity { Notice, Warning, Error, Fatal };`

Sorting these alphabetically does not make sense, but if you put the enum value into the tag of the cell, you can sort them correctly.   This sort type also makes it possible to sort slots that hold icons.

`MISC_SORT_FLOAT`

The cells are compared as single-precision floating point numbers.   The values are retrieved using the `-floatValueAtRow:column:` message.

`MISC_SORT_DOUBLE`

The cells are compared as double-precision floating point numbers.   The values are retrieved using the `-doubleValueAtRow:column:` message.

`MISC_SORT_SKIP`

The cells are not compared.   All cells in slots with the `MISC_SORT_SKIP` sort-type are considered equal.   This can

be used for slots that should not affect the sorting.

```
MISC_SORT_TITLE_CASE_INSENSITIVE
MISC_SORT_TITLE_CASE_SENSITIVE
```

The cells are compared as strings.   The string values are retrieved using the `-titleAtRow:column: message`. This is provided to support NSButtonCells.

```
MISC_SORT_STATE
MISC_SORT_UNSIGNED_STATE
```

The cells are compared as integers.   The integer values are retrieved using the `-stateAtRow:column:` message. This is provided to support NSButtonCells.

See also: -border:setSlot:sortDirection:, -border:setSlot:sortFunction:, -border:slotSortType:

border:setSlot:tag:
-   (void)**border:**(MiscBorderType)*b* **tag:**(int)*tag*

Sets the tag for slot *n* to *tag*.

border:setSlot:title:
-   (void)**border:**(MiscBorderType)*b* **setSlot:**(int)*n* **title:**(NSString*)*title*

Sets the title for slot *n* to *title*.   This method only works for borders with custom titles (that is, `-border:`*b* `setSlotTitleMode:MISC_CUSTOM_TITLE`).   If the border does not have custom titles, the request is silently ignored.

border:setSlotOrder:

-   (BOOL)**border:**(MiscBorderType)*b* **setSlotOrder:**(NSArray*)*list*

Rearranges the slots to match the order specified by *list*, which is an array of NSNumber objects.   The list is organized in the *physical* (original) order of the slots.   Each value in the list is the new *visual* position for the corresponding slot.   In other words, *list* is a physical to visual mapping.   This is useful for restoring the user's slot-order preference.

List also encodes the sort direction.   Negative values set the slot sort direction to descending.   Positive values set the slot sort direction to ascending.   Negative values are formed by using the 'C' bitwise complement operator (~).

When *list* is **0** or an empty array, the slots are "unsorted" -- they are returned to their original physical order, and are reset to ascending sort-direction.

Returns YES on success, NO on failure.

border:setSlotOrderFromString:

-   (BOOL)**border:**(MiscBorderType)*b* **setSlotOrderFromString:**(NSString*)*s*

This is a convenience method which invokes `-border:setSlotOrder:` using an `NSArray` constructed from *s*.

border:setSlotSizes:
  - (BOOL)**border:**(MiscBorderType)*b* **setSlotSizes:**(NSArray*)*list*

Sets the sizes of all slots to the values in *list*, which is an array of NSNumber objects.   List is organized in *physical* (original) slot order.   Each value is the size of the corresponding slot.   This method is useful for restoring the user's slot size preferences.   Returns **YES** on success, **NO** on failure.

border:setSlotSizesFromString:
  - (BOOL)**border:**(MiscBorderType)*b* **setSlotSizesFromString:**(NSString*)*s*

This is a convenience method which invokes `-border:setSlotSizes:` using an `NSArray` constructed from *s*.

border:setSlotSortVector:
  - (void)**border:**(MiscBorderType)*b* **setSlotSortVector:**(NSArray*)*v*

Sets the order in which slots are considered when sorting.   Each value in *v*, which is an array of NSNumber objects, is the *physical* (original) position of a slot.   The slots will be compared in the order that they appear in *v*.   Use the one's complement (bitwise negation with the tilde, `~', operator) to reverse the sort direction (ascending / descending)

of a slot.   The current *visual* slot order is used by default; use this method to specify a different ordering.   This method turns off auto-sort for the other border.   That is, setting a *slotSortVector* for the columns turns off auto-sort for the rows.   The auto-sort facility automatically sorts the table whenever a slot is dragged, so setting a slot sort vector invalidates the premise on which auto-sort works.

See also: **-slotSortVector**

border:setSlotTitleMode:
   - (void)**border:**(MiscBorderType)*b* **setSlotTitleMode:**(MiscTableTitleMode)*x*

Sets the *title-mode* for a border.   The title-mode, *x*, can be any of the following (declared in `MiscTableTypes.h`):

```
MISC_NO_TITLE,          // No titles on row/column cells.
MISC_NUMBER_TITLE,      // Titles are sequential numbers.
MISC_ALPHA_TITLE,       // Titles are sequential alphabetics...
MISC_CUSTOM_TITLE,      // Titles are user-supplied strings...
MISC_DELEGATE_TITLE     // Ask the delegate for titles.
```

See also: **-tableScroll:border:slotTitle:** (delegate method)

border:setSlotTitlesOn:
   - (BOOL)**border:**(MiscBorderType)*b* **setSlotTitlesOn:**(BOOL)*on_off*

Determines whether titles will be displayed.   When *on_off* is YES, the titles will be displayed.   When *on_off* is NO, the titles will not be displayed.   The titles must be displayed to let the user resize and drag slots.   See **Slot Sizing** in the introduction for more details.   Returns YES if the titles were changed, NO otherwise.   For example, if this function

is called to turn on the column titles, but the column titles are already on, the function will do nothing and return NO. The function will only return YES if the titles were actually turned on or off by the call.

border:setSlotTitlesSize:
- (void)**border:**(MiscBorderType)*b* **setSlotTitlesSize:**(float)*size*

For column titles, sets their height.   For row titles, sets their width.

border:setUniformSizeSlots:
- (void)**border:**(MiscBorderType)*b* **setUniformSizeSlots:**(float)*uniform_size*

Sets or clears the *uniform-size* for a border.   If *uniform_size* is zero, then each slot on that border will be able to have individually varying sizes.   If *uniform_size* is non-zero, then every slot on that border will have the size, *uniform_size*. When the slots on a border have a uniform size, the user will not be able to resize the slots.   See **Slot Sizing** in the introduction for more details.

border:slotAdjustedSize:
- (float)**border:**(MiscBorderType)*b* **slotAdjustedSize:**(int)*slot*

Returns the current display size of *slot*.

border:slotAtPosition:

- (int)**border:**(MiscBorderType)*b* **slotAtPosition:**(int)*pos*

Returns the original physical position of the slot in visual position *pos*.   This is the visual-to-physical conversion routine.

See also: -border:moveSlotFrom:to:, -border:slotPosition:

border:slotCellPrototype:
- (id)**border:**(MiscBorderType)*b* **slotCellPrototype:**(int)*slot*

Returns the cell prototype for *slot*.

See also: -border:setSlot:cellPrototype:, -border:setSlot:cellType:

border:slotCellType:
- (MiscTableCellStyle)**border:**(MiscBorderType)*b* **slotCellType:**(int)*slot*

Returns the cell type for *slot*.

See also: -border:setSlot:cellPrototype:, -border:setSlot:cellType:

border:slotDraggedFrom:to:
- (void)**border:**(MiscBorderType)*b* **slotDraggedFrom:**(int)*from_pos* **to:**(int)*to_pos*

Internal method, invoked whenever the user drags a slot to a new position.   Can be useful in subclasses to recognize

a user-initiated slot drag event.

border:slotIsAutosize:
- (BOOL)**border:**(MiscBorderType)*b* **slotIsAutosize:**(int)*slot*

Returns the state of the autosize flag for *slot*. See **Slot Sizing** in the introduction for more details.

border:slotIsSelected:
- (BOOL)**border:**(MiscBorderType)*b* **slotIsSelected:**(MiscCoord_P)*slot*

Returns YES if *slot* is selected, else NO.

border:slotIsSizeable:
- (BOOL)**border:**(MiscBorderType)*b* **slotIsSizeable:**(int)*slot*

Returns the state of the user-sizeable flag for *slot*. See **Slot Sizing** in the introduction for more details.

border:slotIsSorted:
- (BOOL)**border:**(MiscBorderType)*b* **slotIsSorted:**(int)*slot*

This method compares *slot* with its neighbors.   It returns YES if these slots are sorted relative to each other.   It returns NO if any of these slots are out of order with respect to the others.   This method can be useful for determining whether or not the table must be resorted when you are changing values in the table.

border:slotIsVisible:
- (BOOL)**border:**(MiscBorderType)*b* **slotIsVisible:**(int)*slot*

Returns YES if any part of *slot* is visible in the scrolling display.   Returns NO otherwise.

border:slotMaxSize:
- (float)**border:**(MiscBorderType)*b* **slotMaxSize:**(int)*slot*

Returns the maximum size for *slot*. See **Slot Sizing** in the introduction for more details.

border:slotMinSize:
- (float)**border:**(MiscBorderType)*b* **slotMinSize:**(int)*slot*

Returns the minimum size for *slot*. See **Slot Sizing** in the introduction for more details.

border:slotPosition:
- (int)**border:**(MiscBorderType)*b* **slotPosition:**(int)*slot*

Returns the current *visual* position of the slot whose original *physical* position was *slot*.   This is the physical-to-visual

conversion routine.

border:slotResized:
- (void)**border:**(MiscBorderType)*b* **slotResized:**(int)*n*

Internal method called whenever the user resizes a slot.   Can be useful in subclasses to recognize user-initiated slot resizing.

border:slotSize:
- (float)**border:**(MiscBorderType)*b* **slotSize:**(int)*slot*

Returns the target size for *slot*. See **Slot Sizing** in the introduction for more details.

border:slotSortDirection:
- (MiscSortDirection)**border:**(MiscBorderType)*b* **slotSortDirection:**(int)*n*

Returns the sort direction (ascending or descending) for slot *n*.

border:slotSortFunction:
- (MiscCompareEntryFunc)**border:**(MiscBorderType)*b* **slotSortFunction**:(int)*n*

Returns the custom sort function for slot *n*, if any, otherwise it returns 0.

border:slotSortReversed:
- (void)**border:**(MiscBorderType)*b* **slotSortReversed:**(int)*n*

Internal method, invoked whenever the user reverses the sort direction of a slot.   Can be useful in subclasses to recognize a user-initiated sort reverse event.

border:slotSortType:
- (MiscSortType)**border:**(MiscBorderType)*b* **slotSortType:**(int)*n*

Returns the sort type of slot *n*.

border:slotTag:
- (int)**border:**(MiscBorderType)*b* **slotTag:**(int)*slot*

Returns the tag value associated with *slot*.

border:slotTitle:
- (NSString*)**border:**(MiscBorderType)*b* **slotTitle:**(int)*slot*

Returns the title for *slot*.

border:slotWithTag:

   - (int)**border:**(MiscBorderType)*b* **slotWithTag:**(int)*x*

Returns the index of the first slot whose tag is *x*, or -1 if no match was found.

border:sortSlot:

   - (BOOL)**border:**(MiscBorderType)*b* **sortSlot:**(int)*slot*

Re-sorts a single slot.   This method can be used to restore the sort order after a single slot has been added or changed in such a way that it might not be in the correct sort position.   The results are unpredictable if the other slots are not already sorted.   Returns YES if the slot sorted to a new visual location, otherwise returns NO.

See also: -sortSlots:, -sortColumn:, -sortRow:

border:visualToPhysical:

   - (NSArray*)**border:**(MiscBorderType)*b* **visualToPhysical:**(NSArray*)*list*

This method accepts an array of NSNumber objects each containing a *visual* (current) slot index, and returns an array containing the corresponding *physical* (original) slot indexes.   This is accomplished via a succession of calls to `-border:slotAtPosition:`.

See also: -border:slotAtPosition:, -border:slotPosition:

bufferCount
- (int)**bufferCount**

This method is only meaningful for lazy tables.   If the *delegate* or *dataDelegate* provide multiple buffers for responding to `-tableScroll:cellAtRow:column:`, they are encouraged to respond to `-tableScrollBufferCount:` with the number of buffers that they provide.   If the *delegate* and *dataDelegate* do not respond, a default value of one (1) is returned, which indicates that all values from a call to `-cellAtRow:column:` must be copied before making a second call to `-cellAtRow:column:`.   This method is called internally during `-sortInfoInit:border:` to determine whether copying must be performed during sorting.

See also: **-cellAtRow:column:, -isLazy, -setLazy:, -tableScrollBufferCount:** (delegate method)**,**
        -tableScroll:cellAtRow:column: (delegate method)

builtinCanWritePboardType:
- (BOOL)**builtinCanWritePboardType:**(NSString*)*type*

The built-in method for determining which data types can be placed on the pasteboard.   This method returns YES for `NSStringPboardType` and `NSTabularTextPboardType`.   Override this method in your subclass if you will provide additional pasteboard datatypes.   This method is called from `-canWritePboardType:`.

See also: **-canWritePboardType:**

builtinReadSelectionFromPasteboard:

- (BOOL)**builtinReadSelectionFromPasteboard:**(NSPasteboard*)*pboard*

This method merely returns NO.   The current implementation of MiscTableScroll never reads anything from the pasteboard.   Override this method in your subclass if you want to read data from the pasteboard.   This method is called from `-readSelectionFromPasteboard:`.

See also: -readSelectionFromPasteboard:

builtinRegisterServicesTypes
- (void)**builtinRegisterServicesTypes**

This method sends `-registerServicesMenuSendTypes:returnTypes:` to NSApp.   It registers `NSTabularTextPboardType` and `NSStringPboardType` send types, and no return types.   Override this method in your subclass if you want to send or return different data types.   Called from `-registerServicesTypes`.

See also: -registerServicesTypes, -registerServicesMenuSendTypes:returnTypes: (NSApplication)

builtinValidRequestorForSendType:returnType:
- (id)**builtinValidRequestorForSendType:**(NSString*)*t_send* **returnType:**(NSString*)*t_return*

This method returns **self** if *t_send* is either `NSTabularTextPboardType` or `NSStringPboardType`, and *t_return* is 0, and there is a selection, otherwise it returns the value from a call to `[super validRequestorForSendType:t_send returnType:t_return]`.   Override this method if your subclass can handle different combinations.   Called from `-validRequestorForSendType:returnType:`.

builtinStringForPboardType:
- (NSString*)**builtinStringForPboardType:**(NSString*)*type*

If *type* is `NSStringPboardType`, then `-stringForNSStringPboardType` is called, else if *type* is `NSTabularTextPboardType`, then `-stringForNSTabularTextPBoardType` is called.   Otherwise it does nothing. Override this method in your subclass if you can write additional datatypes to the pasteboard.   Called from `-writeSelectionToPasteboard:types:`.

builtinWriteSelectionToPasteboard:types:
- (BOOL)**builtinWriteSelectionToPasteboard:**(NSPasteboard*)*pboard* **types:**(NSArray*)*types*

Writes all of the types that can be written to the pasteboard.   Each entry in *types* is tested with `-canWritePboardType:`.   If the result is YES, and there is a selection, it is passed on to `-writeSelectionToPasteboard:types:`.   The data is written immediately, the MiscTableScroll object does not register a pasteboard owner.   Override this method in your subclass if you need different behavior.   Called from `-writeSelectionToPasteboard:types:`.

canEdit:atRow:column:
- (BOOL)**canEdit:**(NSEvent*)*event* **atRow:**(MiscCoord_P)*row* **column:**(MiscCoord_P)*col*

Determines whether or not the cell at *row*, *col* can be edited.   Attempts to send
`-tableScroll:canEdit:atRow:column:` to the *delegate*, the *dataDelegate*, or the cell at *row*, *col*, in that order.   The result is taken from the first of these three to respond to `-tableScroll:canEdit:atRow:column:` and is returned to the caller of `-canEdit:atRow:column:`.   If none of the three respond, then MiscTableScroll applies its own criteria to determine if editing is allowed.   If the cell is not both enabled and editable then NO is returned.   If *event* is NULL or represents a double-click by the mouse then YES is returned, otherwise NO.

It is valid to specify NULL for *event* when editing needs to be invoked for a non-mouse-down event.   (For instance, `-getNext:editRow:column:` operates in this fashion.)   If *event* is non-NULL then it should point at a mouse-down event.

See also: -isEditable (NSCell), **-isEnabled** (NSCell)

canWritePboardType:
- (BOOL)**canWritePboardType:**(NSString*)*type*

Responds to queries from `-builtinWriteSelectionToPasteboard:types:`.   First it gives the *delegate* an opportunity to answer via `-tableScroll:canWritePboardType:`.   If the *delegate* does not respond to that message, it gives the *dataDelegate* an opportunity to answer the same message.   If neither object responds, the built-in implementation, `-builtinCanWritePboardType:` is called.   Called from `-builtinWriteSelectionToPasteboard:types:`.

**See also: -builtinCanWritePboardType:, -builtinWriteSelectionToPasteboard:types:,
          -tableScroll:canWritePboardType:** (delegate method)

cellAtRow:column:
- (id)**cellAtRow:**(int)*row* **column:**(int)*col*

Returns a pointer to the cell located at *row*,*col*.

cellFrameAtRow:column:
- (NSRect)**cellFrameRow:**(int)*row* **column:**(int)*col*

Returns the frame of the cell at the specified coordinates.   If *row* or *col* are out of bounds, then returns the empty rectangle.   < FIXME: Currently the returned rectangle also includes the intercell grid lines which are below and to the right of the cell. >

cellIsSelectedAtRow:column:
- (BOOL)**cellIsSelectedAtRow:**(MiscCoord_P)*row* **column:**(MiscCoord_P)*col*

Returns YES if the cell at *row*, *col* is selected.

cellWithTag:
- (id)**cellWithTag:**(int)*x*

Returns the first cell in the body of the table with tag *x*, otherwise 0.

changeFont:

   - (void)**changeFont:**(id)*sender*

Changes the font of the MiscTableScroll object as well all cells which inherit it.   The NSFontManager sends the `-changeFont:` message whenever the user changes the font using either the NSFontPanel or the Font menu.   *sender* must respond to the `-convertFont:` message and return an NSFont which is then passed to `-setFont:`.   This method sends `-tableScroll:changeFont:to:` and `-tableScroll:fontChangedFrom:to:` messages to the delegate.

See also: - setFont:, ± tableScroll:changeFont:to: (delegate method)**, ± tableScroll:fontChangedFrom:to:** (delegate method)

clearColumnSelection

   - (void)**clearColumnSelection**

Equivalent to: `-clearSlotSelection:MISC_COL_BORDER`.

clearCursor

   - (void)**clearCursor**

Calls `[self clearCursorColumn]` and `[self clearCursorRow]`.

clearCursorColumn
- (void)**clearCursorColumn**

Sets the column border's keyboard cursor slot to -1, effectively hiding it until it is next set to a valid position.

clearCursorRow
- (void)**clearCursorRow**

Sets the row border's keyboard cursor slot to -1, effectively hiding it until it is next set to a valid position.

clearCursorSlot:
- (void)**clearCursorSlot:**(MiscBorderType)*b*

Sets the border's keyboard cursor slot to -1, effectively hiding it until it is next set to a valid position.

clearRowSelection
- (void)**clearRowSelection**

Equivalent to: `-clearSlotSelection:MISC_ROW_BORDER`.

clearSelection
- (void)**clearSelection**

Calls `[self clearRowSelection]` and `[self clearColumnSelection]`.

clearSlotSelection:
- (void)**clearSlotSelection:**(MiscBorderType)*b*

Deselects all slots that were selected in border *b*.

clickedCell
- (id)**clickedCell**

During mouse-tracking, returns the cell underneath the mouse, otherwise returns the cell which was under the mouse when tracking ended.   This method only really makes sense for eager-mode MiscTableScroll objects since a unique cell inhabits each row & column position, whereas in lazy-mode only one cell typically exists per column.

See also: ± clickedSlot:, ± clickedColumn, ± clickedRow, ± tracking

clickedColumn
- (MiscCoord_P)**clickedColumn**

During mouse-tracking, returns the column underneath the mouse, otherwise returns the column which was under the mouse when tracking ended.

See also: ± clickedSlot:, ± clickedCell, ± clickedRow, ± tracking

clickedRow

    - (MiscCoord_P)**clickedRow**

During mouse-tracking, returns the row underneath the mouse, otherwise returns the row which was under the mouse when tracking ended.

See also: **± clickedSlot:, ± clickedCell, ± clickedColumn, ± tracking**

clickedSlot:

    - (MiscCoord_P)**clickedSlot:**(MiscBorderType)*b*

During mouse-tracking, returns the slot underneath the mouse, otherwise returns the slot which was under the mouse when tracking ended.

See also: **± clickedCell, ± clickedColumn, ± clickedRow, ± tracking**

color

    - (NSColor*)**color**

Equivalent to: `-backgroundColor`.

columnAdjustedSize:
  - (float)**columnAdjustedSize:**(int)*col*

Returns the current display width of *col*.   Equivalent to: `-border:MISC_COL_BORDER slotAdjustedSize:`*col*.

See also: -border:slotAdjustedSize:

columnAtPosition:
  - (int)**columnAtPosition:**(int)*pos*

Returns the original *physical* position of the column at the current *visual* position *pos*.   This is the visual-to-physical conversion routine.   Equivalent to: `-border:MISC_COL_BORDER slotAtPosition:`*pos*.

See also: -border:slotAtPosition:, -border:slotPosition:, -columnPosition:

columnCellPrototype:
  - (id)**columnCellPrototype:**(int)*col*

Returns the cell prototype for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotCellPrototype:`*col*.

See also: -border:setSlot:cellPrototype:, -border:slotCellPrototype:

columnCellType:
  - (MiscTableCellStyle)**columnCellType:**(int)*col*

Returns the cell type for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotCellType:`*col*.

See also: **-border:setSlot:cellType:, -border:slotCellType:**

columnIsAutosize:
-   (BOOL)**columnIsAutosize:**(int)*col*

Returns the state of the *autosize* flag for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotIsAutosize:`*col*.

See also: **-border:setSlot:autosize:, -border:slotIsAutosize:**

columnIsSelected:
-   (BOOL)**columnIsSelected:**(MiscCoord_P)*col*

Returns YES if column *col* is selected, else NO.   Equivalent to `-border:MISC_COL_BORDER slotIsSelected:`*col*.

columnIsSizeable:
-   (BOOL)**columnIsSizeable:**(int)*col*

Returns the state of the *user-sizeable* flag for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotIsSizeable:`*col*.

See also: **-border:setSlot:sizeable:, -border:slotIsSizeable:**

columnIsSorted:
- (BOOL)**columnIsSorted:**(int)*col*

Returns YES if *col* is sorted relative to its neighboring columns.   Returns NO otherwise.   Equivalent to
`-border:MISC_COL_BORDER slotIsSorted:`*col*.

columnIsVisible:
- (BOOL)**columnIsVisible:**(int)*col*

Returns YES if any part of *col* is visible in the scrolling display.   Returns NO otherwise.   Equivalent to
`-border:MISC_COL_BORDER slotIsVisible:`*col*.

columnMaxSize:
- (float)**columnMaxSize:**(int)*col*

Returns the maximum size for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotMaxSize:`*col*.

See also: **-border:setSlot:maxSize:, -border:slotMaxSize:**

columnMinSize:
- (float)**columnMinSize:**(int)*col*

Returns the minimum size for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotMinSize:`*col*.

See also: -border:setSlot:minSize:, -border:slotMinSize:

columnOrder
- (NSArray*)**columnOrder**

Equivalent to `-slotOrder:MISC_COL_BORDER`.

columnOrderAsString
- (NSString*)**columnOrderAsString**

Equivalent to `slotOrderAsString:MISC_COL_BORDER`.

columnPosition:
- (int)**columnPosition:**(int)*col*

Returns the current *visual* position of the column whose original *physical* position is *pos*.   This is the physical-to-visual conversion routine.   Equivalent to: `-border:MISC_COL_BORDER slotPosition:`*pos*.

See also: -border:moveSlotFrom:to:, -border:slotAtPosition:, -border:slotPosition:, -columnAtPosition:, -moveColumnFrom:to:

columnsAreSorted
- (BOOL)**columnsAreSorted**

Returns YES if all columns are sorted.   Equivalent to `-slotsAreSorted:MISC_COL_BORDER`.

See also: **-columnIsSorted:, -slotsAreSorted:**

columnSize:
- (float)**columnSize:**(int)*col*

Returns the target size for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotSize:`*col*.

See also: **-border:setSlot:size:, -border:slotSize:**

columnSizes
- (NSArray*)**columnSizes**

Equivalent to: `slotSizes:MISC_COL_BORDER`.

columnSizesAsString
- (NSString*)**columnSizesAsString**

Equivalent to `-slotSizesAsString:MISC_COL_BORDER`.

columnSortDirection:
-   (MiscSortDirection)**columnSortDirection:**(int)*n*

Returns the sort direction (ascending or descending) of column *n*.   Equivalent to: `-border:MISC_COL_BORDER slotSortDirection:`*n*.

columnSortFunction:
-   (MiscCompareEntryFunc)**columnSortFunction**:(int)*n*

Equivalent to: `-border:MISC_COL_BORDER slotSortFunction:`*n*.

columnSortType:
-   (MiscSortType)**columnSortType:**(int)*n*

Equivalent to `-border:MISC_COL_BORDER slotSortType:`*n*.

columnSortVector
-   (NSArray*)**columnSortVector**

Equivalent to: `-slotSortVector:MISC_COL_BORDER`.

columnTag:
- (int)**columnTag:**(int)*col*

Returns the tag for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotTag:`*col*.

columnTitle:
- (NSString*)**columnTitle:**(int)*col*

Returns the title for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotTitle:`*col*.

columnTitleMode
- (MiscTableTitleMode)**columnTitleMode**

Returns the *title-mode* for column *col*.   Equivalent to: `-border:MISC_COL_BORDER slotTitleMode:`*col*.

columnTitlesHeight
- (float)**columnTitlesHeight**

Equivalent to: `-slotTitlesSize:MISC_COL_BORDER`.

columnTitlesOn

- (BOOL)**columnTitlesOn**

Indicates whether or not column titles are displayed.   Equivalent to: `-slotTitlesOn:MISC_COL_BORDER`.

columnWithTag:
- (int)**columnWithTag:**(int)*x*

Returns the index of the first column with tag *x*, or -1 if no columns have tag *x*.

compareColumns::
- (int)**compareColumns:**(int)*col1* **:**(int)*col2*

Compares two columns.   Equivalent to `-border:MISC_COL_BORDER compareSlots:`*col1:col2*.

compareColumns::info:
- (int)**compareColumns:**(int)*col1* **:**(int)*col2* **info:**(MiscSlotSortInfo*)*sortInfo*

Compares two columns.   Equivalent to `-border:MISC_COL_BORDER compareSlots:`*col1:col2* `info:`*sortInfo*.

compareRows::
- (int)**compareRows:**(int)*row1* **:**(int)*row2*

Compares two columns. Equivalent to `-border:MISC_ROW_BORDER compareSlots:`*row1*`:`*row2*.

compareRows::info:
  - (int)**compareRows:**(int)*row1* **:**(int)*row2* **info:**(MiscSlotSortInfo*)*sortInfo*

Compares two columns. Equivalent to `-border:MISC_ROW_BORDER compareSlots:`*row1*`:`*row2* `info:`*sortInfo*.

compareSlotFunction
  - (MiscCompareSlotFunc)**compareSlotFunction**

Returns the slot comparison function.

constrainSize
  - (void)**constrainSize**

Internal method that checks and applies new slot counts and min total size constraints to update the frames of the components of the MiscTableScroll object.

copy:
  - (void)**copy:**(id)*sender*

Copies the selection to the pasteboard. Calls `-writeSelectionToPasteboard:types:`, with

`NSTabularTextPboardType` and `NSStringPboardType` for types that should be written.   Override this method in your subclass if you want to write different datatypes to the pasteboard.

See also: **-writeSelectionToPasteboard:types:**

cornerTitle
-   (NSString*)**cornerTitle**

Returns the title for the corner cell.

cursorColumn
-   (MiscCoord_P)**cursorColumn**

Returns the column that the column keyboard cursor is on.   Meaningless if tracking is by rows.

cursorRow
-   (MiscCoord_P)**cursorRow**

Returns the row that the row keyboard cursor is on.   Meaningless if tracking is by columns.

cursorSlot:
-   (MiscCoord_P)**cursorSlot:**(MiscBorderType)*b*

Returns the index of the slot that the keyboard cursor is currently on, or -1 if the keyboard cursor is not on any slot.

cut:
   - (void)**cut:**(id)*sender*

Calls `[self copy:sender]`.   Nothing is deleted.

dataDelegate
   - (id)**dataDelegate**

Returns the data delegate of the MiscTableScroll object.

See also: **-setDataDelegate**

dealloc
   - (void)**dealloc**

Destroys the MiscTableScroll object, reclaiming all resources allocated by it.

delegate
   - (id)**delegate**

Returns the delegate of the MiscTableScroll object.

See also: **-setDelegate**

deselectAll:
-   (void)**deselectAll:**(id)*sender*

Calls `[self clearSelection]` followed by `[self sendActionIfEnabled]`.

deselectColumn:
-   (void)**deselectColumn:**(MiscCoord_P)*col*

Equivalent to: `-border:MISC_COL_BORDER deselectSlot:`*col*.

deselectColumns:
-   (void)**deselectColumns:**(NSArray*)*cols*

Equivalent to: `-border:MISC_COL_BORDER deselectSlots:`*cols*.

deselectColumnTags:
-   (void)**deselectColumnTags:**(NSArray*)*tags*

Equivalent to: `-border:MISC_COL_BORDER deselectSlotTags:`*tags*.

deselectRow:
- (void)**deselectRow:**(MiscCoord_P)*row*

Equivalent to: `-border:MISC_ROW_BORDER deselectSlot:`*row*.

deselectRows:
- (void)**deselectRows:**(NSArray*)*rows*

Equivalent to: `-border:MISC_ROW_BORDER deselectSlots:`*rows*.

deselectRowTags:
- (void)**deselectRowTags:**(NSArray*)*tags*

Equivalent to: `-border:MISC_ROW_BORDER deselectSlotTags:`*tags*.

disableCursor
- (void)**disableCursor**

Inhibits display of the keyboard cursor.   Calls to this method nest and should be balanced by calls to `-enableCursor`.
The keyboard cursor is a dashed rectangle drawn around a row or column indicating which slot keyboard actions will

affect.   See the discussion of **Keyboard Operations** at the beginning of this document for further information.

See also: ± isCursorEnabled, ± enableCursor

documentClipRect
   - (NSRect)**documentClipRect**

Returns the frame of the NSClipView which contains the document view.

doGetISearchColumn:
   - (BOOL)**doGetISearchColumn:**(int*)*col*

Built-in method to choose the incremental search column.   Returns YES if incremental search should be enabled and sets *col* to the physical index of the column that should be searched,   otherwise returns NO.   To enable incremental search via this function, autoSortRows must be YES; the first sorting column must be string-based (stringValue or title); and there cannot be a custom sort function for the column.   This method works appropriately for normal tables, as long as autoSortRows is turned on.   Whenever the user drags a string-based column to the first position, incremental search will be enabled.   Non-sorting (skip) columns are ignored.   You can override this behavior by implementing the `-tableScroll:getISearchColumn:` method in your delegate.

See also: -incrementalSearch:, -tableScroll:getISearchColumn: (delegate method)

doIncrementalSearch:column:

- (BOOL)**doIncrementalSearch:**(NSEvent*)*event* **column:**(int)*col*

Built-in method that performs incremental search.   *Event* must be the key-down event that invoked incremental searching.   *Col* must be the column that will be searched.   The table must be sorted in *col* order (ascending or descending).   The sort-type for *col* must be string-based (stringValue or title).   *Col* cannot have a custom sort function.   You are responsible for ensuring that the table is sorted in *col* order.   This method runs a modal event loop, processing keystrokes and scrolling the table appropriately.   Returns YES if *col* was acceptable and *event* was processed.   Returns NO if *col* failed any of the tests mentioned.

See also: -incrementalSearch:, -tableScroll:getISearchColumn: (delegate method)

doRetireCell:atRow:column:
- (id)**doRetireCell:**(id)*cell* **atRow:**(int)*row* **column:***(int)col*

This built-in implementation tries to recover storage before the cell is idled.   An attempt is made to send the following messages to the cell in this order: `-setTitle:@""`, `-setStringValue:@""`.   If the cell responds to a message, that message is sent, otherwise the next message is tried.   Override this method in your subclass if you need to do different processing when cells are retired to the cache.   Called from: `-retireCell:atRow:column:`.   Returns *cell*.

doReviveCell:atRow:column:
- (id)**doReviveCell:**(id)*cell* **atRow:**(int)*row* **column:***(int)col*

This method tries to reset the cell so that it will "`useOwner`..." values for font, textColor, backgroundColor, selectedTextColor, and   selectedBackgroundColor.   It tries to set the MiscTableScroll object as the *owner* of the cell.

Then it tries to initialize the font, textColor, backgroundColor, selectedTextColor, and selectedBackgroundColor by first trying the "`setOwner`..." value method, and then trying the straight "`set`..." method if the cell does not respond to the "`setOwner`..." version.   Override this method in your subclass if you need different behavior when a cell is brought into active service.   Called from `-reviveCell:atRow:column:`.   Returns *cell*.

doubleAction
- (SEL)**doubleAction**

Returns the selector message that is sent to the *doubleTarget* on a double-click event.

doubleTarget
- (id)**doubleTarget**

Returns a pointer to the object which will receive the *doubleAction* message on a double-click event.

doubleValueAtRow:column:
- (double)**doubleValueAtRow:**(int)*row* **column:**(int)*col*

Returns the value of sending a `-doubleValue` message to the cell at *row*, *col*.   If the table is lazy, the *delegate*, and then the *dataDelegate* are given the opportunity to reply to the `-tableScroll:doubleValueAtRow:column:` message. This gives lazy tables an opportunity to return this information directly, without the overhead of preparing and formatting a cell.   If the table is not lazy, or the *delegate* and *dataDelegate* do not respond to the `-tableScroll:doubleValueAtRow:column:` message, then the cell is retrieved via `-cellAtRow:column:`.   If the cell

responds to the `-doubleValue` message, that value is returned; otherwise, zero is returned.

See also: **-cellAtRow:column:, -isLazy, -setLazy:, -tableScroll:doubleValueAtRow:column:** (delegate method)

draggableColumns
- (BOOL)**draggableColumns**

Indicates whether or not the user will be allowed to drag (rearrange) the columns.   Equivalent to:
`-draggableSlots:MISC_COL_BORDER`.

draggableRows
- (BOOL)**draggableRows**

Indicates whether or not the user will be allowed to drag (rearrange) the rows.   Equivalent to:
`-draggableSlots:MISC_ROW_BORDER`.

draggableSlots:
- (BOOL)**draggableSlots:**(MiscBorderType)*b*

Indicates whether or not the user will be allowed to drag (rearrange) the slots on this border.   To enable the user to drag slots, the slots must be draggable, and the titles must be displayed.

drawCellAtRow:column:
   - (void)**drawCellAtRow:**(int)*row* **column:**(int)*col*

Instructs the MiscTableScroll object to redraw the cell at position *row*, *col*.   This should be called whenever the contents of a single cell are changed and the screen should be updated to reflect the new state.   This method will lock focus on the view if needed.

See also: **-border:drawSlot:, -display** (View)**, -drawColumn:, -drawRow:**

drawClippedText
   -   (BOOL)**drawClippedText**

Returns YES if the MiscTableScroll object will use clipping rectangles and draw partially visible text.   Returns NO if the MiscTableScroll object will simply not draw partially visible text that would require clipping rectangles.

drawColumn:
   -   (void)**drawColumn:**(int)*col*

Instructs the MiscTableScroll object to redraw all the cells in column *col*.   This method will lock focus on the view if needed.

See also: **-drawCellAtRow:column:**

drawColumnTitle:

- (void)**drawColumnTitle:**(int)*n*

Draws the title for column *col*.   This method will lock focus on the view if needed.   You should never need to call this method in normal use, though it might be useful for subclasses.

drawRow:
   - (void)**drawRow:**(int)*row*

Instructs the MiscTableScroll object to redraw all the cells in row *row*.   This method will lock focus on the view if needed.

See also: -drawCellAtRow:column:

drawRowTitle:
   - (void)**drawRowTitle:**(int)*n*

Draws the title for row *row*.   This method will lock focus on the view if needed.   You should never need to call this method in normal use, though it might be useful for subclasses.

edit:atRow:column:
   - (void)**edit:**(NSEvent*)*event* **atRow:**(int)*row* **column:**(int)*col*

Informs the *delegate* or *dataDelegate* that editing is commencing by sending `-tableScroll:willEditAtRow:column:`, then initiates editing at *row, col*.   It is valid to specify NULL for *event* when editing needs to be invoked for a non-

mouse-down event.   If *event* is non-NULL then it should be the mouse-down event which initiates editing.

editCellAtRow:column:
- (void)**editCellAtRow:**(int)*row* **column:**(int)*col*

Clears the selection, then selects the row (or column, if tracking by columns) of the cell, and invokes: `-edit:0`
`atRow:`*row* `column:`*col*.

See also: -trackingBy

editIfAble:atRow:column:
- (void)**editIfAble:**(NSEvent*)*event* **atRow:**(int)*row* **column:**(int)*col*

Calls `-canEdit:atRow:column:`, and then calls `-edit:atRow:column:` if YES was returned.   Returns YES if editing was initiated, and NO if not.

empty
- (void)**empty**

Resets the number of rows in the MiscTableScroll to zero.   Does not deallocate the rows, nor does it affect the number of columns.   The rows are retained in the cache for future use.   See **Usage Tips** in the introduction for more details.

See also: -addRow, -border:removeSlot:, -removeRow:, -emptyAndReleaseCells, -renewRows:

emptyAndReleaseCells

- (void)**emptyAndReleaseCells**

Resets the number of rows in the MiscTableScroll to zero; releases all cells stored in the cache, and deallocates all cache resources.   Does not affect the number of columns.

See also: -addRow, -border:removeSlot:, -removeRow:, -empty, -renewRows:

enableCursor

- (void)**enableCursor**

Re-enables display of the keyboard cursor after a call to `-disableCursor`.   Calls to this method should be made to balance previous calls to `-disableCursor`.   The keyboard cursor is a dashed rectangle drawn around a row or column indicating which slot keyboard actions will affect.   See the discussion of **Keyboard Operations** at the beginning of this document for further information.

See also: ± disableCursor, ± isCursorEnabled

finishEditing

- (BOOL)**finishEditing**

If cell editing is in progress, then this method attempts to finish it.   This method invokes the normal edit termination routines, and the `-control:textShouldEndEditing:` validation method gets an opportunity to veto the new value.

Returns YES if no cell editing was in progress to start with, or if the editing session terminated successfully.   Returns NO if the new cell value was rejected.

firstVisibleColumn
- (int)**firstVisibleColumn**

Equivalent to: `-firstVisibleSlot:MISC_COL_BORDER`.

firstVisibleRow
- (int)**firstVisibleRow**

Equivalent to: `-firstVisibleSlot:MISC_ROW_BORDER`.

firstVisibleSlot:
- (int)**firstVisibleSlot:**(MiscBorderType)*b*

Returns the physical coordinate of the first fully visible slot, if any.   If there are two partially visible slots, it returns the physical coordinate of the *last* slot.   If there is one partially visible slot, it returns the physical coordinate of that slot.   If there are no slots (the MiscTableScroll is empty), it returns -1.

floatValueAtRow:column:

- (float)**floatValueAtRow:**(int)*row* **column:**(int)*col*

Returns the value of sending a `-floatValue` message to the cell at *row*, *col*.   If the table is lazy, the *delegate*, and then the *dataDelegate* are given the opportunity to reply to the `-tableScroll:floatValueAtRow:column:` message. This gives lazy tables an opportunity to return this information directly, without the overhead of preparing and formatting a cell.   If the table is not lazy, or the *delegate* and *dataDelegate* do not respond to the `-tableScroll:doubleValueAtRow:column:` message, then the cell is retrieved via `-cellAtRow:column:`.   If the cell responds to the `-floatValue` message, that value is returned; otherwise, zero is returned.

See also: -cellAtRow:column:, -isLazy, -setLazy:, -tableScroll:floatValueAtRow:column: (delegate method)

font
- (NSFont*)**font**

Returns the current font for the MiscTableScroll object.   The current font is used to initialize new cells in the table.

getISearchColumn:
- (BOOL)**getISearchColumn:**(int*)*col*

This method is responsible for determining whether incremental searching should be enabled, and identifying the column on which the table is sorted.   If the *delegate* responds to `-tableScroll:getISearchColumn:`, then it is called, otherwise, the *dataDelegate* is tried.   If neither object responds, the built-in `-doGetISearchColumn:` method is called. Returns YES if incremental searching should be enabled, otherwise NO.

See also: -doGetISearchColumn:, -incrementalSearch:, -tableScroll:getISearchColumn: (delegate method)

getNext:editRow:column:
   - (BOOL)**getNext:**(BOOL)*forward* **editRow:**(int*)*row* **column:**(int*)*col*

When *forward* is YES, this method returns the coordinates of the next cell that is editable as determined by `-canEdit:atRow:column:`. When *forward* is NO, this method returns the coordinates of the nearest previous cell that is editable. The search order is based on the visual order of slots in the MiscTableScroll. Normally this method is used by assigning the physical coordinates of the cell that is currently being edited to *row* and *col*. The method then searches for the next/previous cell that is editable and updates the values of *row* and *col* to the coordinates of the next/previous editable cell. This method is used in the `-textDidEndEditing:` method to find the next / previous cell when the user presses TAB / SHIFT-TAB to terminate cell editing. Returns YES if a new editable cell was found. Returns NO if there are no other editable cells.

getNextEditRow:column:
   - (BOOL)**getNextEditRow:**(int*)*row* **column:**(int*)*col*

Equivalent to: `-getNext:YES editRow:row column:col`.

getPreviousEditRow:column:
   - (BOOL)**getPreviousEditRow:**(int*)*row* **column:**(int*)*col*

Equivalent to: `-getNext:NO editRow:row column:col`.

getRow:column:forPoint:
    - (BOOL)**getRow:**(int*)*row* **column:**(int*)*col* **forPoint:**(NSPoint)*point*

Calculates the physical slot row and column coordinates for a *point*, which must be in the coordinate system of the receiving MiscTableScroll object.   If *point* is outside and left of the MiscTableScroll, or there are no columns, then *col* is set to -1.   If *point* is outside and right of the MiscTableScroll, *col* is set to the last valid column index.   Out of range values are handled similarly for *row*.   Returns YES unless *point* was out of bounds, in which case NO is returned.

getRow:column:ofCell:
    - (BOOL)**getRow:**(int*)*row* **column:**(int*)*col* **ofCell:**(NSCell*)*cell*

Finds the location of *cell* in the MiscTableScroll object.   If *cell* is found, *row* and *col* are set to the coordinates of the cell in the table and the method returns YES.   If *cell* is not found, *row* and *col* are set to -1, and the method returns NO.

getRow:column:ofCellWithTag:
    - (BOOL)**getRow:**(int*)*row* **column:**(int*)*col* **ofCellWithTag:**(int)*x*

Assigns the coordinates to *row* and *col* of the first cell in the table with tag *x* and returns YES.   If no cell in the table has tag *x*, *row* and *col* are set to -1, and NO is returned.

hasColumnSelection

- (BOOL)**hasColumnSelection**

Returns YES if any columns are selected, otherwise NO.

hasMultipleColumnSelection
- (BOOL)**hasMultipleColumnSelection**

Returns YES if more than one column is selected, otherwise NO.

hasMultipleRowSelection
- (BOOL)**hasMultipleRowSelection**

Returns YES if more than one row is selected, otherwise NO.

hasMultipleSlotSelection:
- (BOOL)**hasMultipleSlotSelection:**(MiscBorderType)*b*

Returns YES if more than one slot is selected, otherwise NO.

hasRowSelection
- (BOOL)**hasRowSelection**

Returns YES if any rows are selected, otherwise NO.

hasSlotSelection:
- (BOOL)**hasSlotSelection:**(MiscBorderType)*b*

Returns YES if at least one slot is selected, otherwise NO.

hasValidCursorColumn
- (BOOL)**hasValidCursorColumn**

Returns YES if the column keyboard cursor has a valid position in the body of the table, otherwise NO.

hasValidCursorRow
- (BOOL)**hasValidCursorRow**

Returns YES if the row keyboard cursor has a valid position in the body of the table, otherwise NO.

hasValidCursorSlot:
- (BOOL)**hasValidCursorSlot:**(MiscBorderType)*b*

Returns YES if the keyboard cursor is positioned on a valid slot, otherwise NO.

incrementalSearch:
- (BOOL)**incrementalSearch:**(NSEvent*)*event*

Invokes incremental searching if *event* is an appropriate keyboard event to start incremental search, and if `-getISearchColumn:` determines that incremental searching should be enabled.   Returns YES if incremental searching was invoked (and *event* was processed), otherwise returns NO.   This method should be called from within a `-keyDown:` method.

See also: **-doIncrementalSearch:col, -getISearchColumn:**

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

Initializes a newly allocated MiscTableScroll object.   This is the designated initializer for this class.   The newly allocated object will have the following properties set by default: <FIXME: write this.>

insertColumn:
- (void)**insertColumn:**(int)*pos*

Inserts a new column at position *pos*.   Equivalent to: `-border:MISC_COL_BORDER insertSlot:`*pos*.

insertRow:

- (void)**insertRow:**(int)*pos*

Inserts a new row at position *pos*.   Equivalent to: `-border:MISC_ROW_BORDER insertSlot:`*pos*.

intValueAtRow:column:
- (int)**intValueAtRow:**(int)*row* **column:**(int)*col*

Returns the value of sending a `-intValue` message to the cell at *row*, *col*.   If the table is lazy, the *delegate*, and then the *dataDelegate* are given the opportunity to reply to the `-tableScroll:intValueAtRow:column:` message.   This gives lazy tables an opportunity to return this information directly, without the overhead of preparing and formatting a cell.   If the table is not lazy, or the *delegate* and *dataDelegate* do not respond to the `-tableScroll:doubleValueAtRow:column:` message, then the cell is retrieved via `-cellAtRow:column:`.   If the cell responds to the `-intValue` message, that value is returned; otherwise, zero is returned.

See also: **-cellAtRow:column:, -isLazy, -setLazy:, -tableScroll:intValueAtRow:column:** (delegate method)

isCursorEnabled
- (BOOL)**isCursorEnabled**

Indicates whether cursor display is enabled or disabled.   See the discussion of **Keyboard Operations** at the beginning of this document for further information.

See also: **± disableCursor, ± enableCursor**

isEditing

- (BOOL)**isEditing**

Returns YES if a cell editing session is in progress.

isEnabled

- (BOOL)**isEnabled**

Indicates whether or not the MiscTableScroll object is enabled for user interaction.

isLazy

- (BOOL)**isLazy**

Indicates whether or not the MiscTableScroll object is using *lazy-mode* memory management.

lastVisibleColumn

- (int)**lastVisibleColumn**

Equivalent to: `-lastVisibleSlot:MISC_COL_BORDER`.

lastVisibleRow

- (int)**lastVisibleRow**

Equivalent to: `-lastVisibleSlot:MISC_ROW_BORDER`.

lastVisibleSlot:
- (int)**lastVisibleSlot:**(MiscBorderType)*b*

Returns the physical coordinate of the last fully visible slot, if any.   If there are two partially visible slots, it returns the physical coordinate of the *first* slot.   If there is one partially visible slot, it returns the physical coordinate of that slot. If there are no visible slots (the MiscTableScroll is empty), it returns -1.

makeCellsPerformSelector:
- (int)**makeCellsPerformSelector:**(SEL)*aSel*

Calls `[self makeCellsPerformSelector:`*aSel*` selectedOnly:NO]`.

makeCellsPerformSelector:selectedOnly:
- (int)**makeCellsPerformSelector:**(SEL)*aSel* **selectedOnly:**(BOOL)*flag*

Calls `[self makeCellsPerformSelector:`*aSel*` with:0 with:0 selectedOnly:`*flag*`]`.

makeCellsPerformSelector:with:

- (int)**makeCellsPerformSelector:**(SEL)*aSel* **with:**(id)*arg1*

Calls [self makeCellsPerformSelector:*aSel* with:*arg1* selectedOnly:NO].

makeCellsPerformSelector:with:selectedOnly:
- (int)**makeCellsPerformSelector:**(SEL)*aSel* **with:**(id)*arg1* **selectedOnly:**(BOOL)*flag*

Calls [self makeCellsPerformSelector:*aSel* with:*arg1* with:0 selectedOnly:*flag*].

makeCellsPerformSelector:with:with:
- (int)**makeCellsPerformSelector:**(SEL)*aSel* **with:**(id)*arg1* **with:**(id)*arg2*

Calls [self makeCellsPerformSelector:*aSel* with:*arg1* with:*arg2* selectedOnly:NO].

makeCellsPerformSelector:with:with:selectedOnly:
- (int)**makeCellsPerformSelector:**(SEL)*aSel* **with:**(id)*arg1* **with:**(id)*arg2* **selectedOnly:**(BOOL)*flag*

Sends the message *aSel* to the cells in the table.   When *flag* is YES, the message is sent only to selected cells. When *flag* is NO, the message is sent to all cells.   First the cell is tested with -respondsToSelector:*aSel*.   If the cell responds to the message, then the message is sent.   Then the return value from the call is inspected.   If the cell returns any non-zero value, the process continues.   The first cell that returns 0 stops the process.   The process also terminates when all cells have been processed.   This method returns the number of cells that returned non-zero values.

maxUniformSizeColumns
- (float)**maxUniformSizeColumns**

Equivalent to: `-maxUniformSizeSlots:MISC_COL_BORDER`.

maxUniformSizeRows
- (float)**maxUniformSizeRows**

Equivalent to: `-maxUniformSizeSlots:MISC_ROW_BORDER`.

maxUniformSizeSlots:
- (float)**maxUniformSizeSlots:**(MiscBorderType)*b*

Returns the current upper bound for user-sizing of uniform-sized border, *b*.

minUniformSizeColumns
- (float)**minUniformSizeColumns**

Equivalent to: `-minUniformSizeSlots:MISC_COL_BORDER`.

minUniformSizeRows
- (float)**minUniformSizeRows**

Equivalent to: `-minUniformSizeSlots:MISC_ROW_BORDER`.

minUniformSizeSlots:
- (float)**minUniformSizeSlots:**(MiscBorderType)*b*

Returns the current lower bound for user-sizing of uniform-sized border, *b*.

modifierDragColumns
- (BOOL)**modifierDragColumns**

Indicates whether or not the command-key must be held down to drag columns.   It is NO by default.   Equivalent to: `-modifierDragSlots:MISC_COL_BORDER`.

See also: -border:setModifierDragSlots:,-modifierDragSlots:

modifierDragRows
- (BOOL)**modifierDragRows**

Indicates whether or not the command-key must be held down to drag rows.   It is YES by default.   Equivalent to: `-modifierDragSlots:MISC_ROW_BORDER`.

modifierDragSlots:
- (BOOL)**modifierDragSlots:**(MiscBorderType)*b*

Indicates whether or not the command-key must be held down to drag the slots on this border.

moveColumnFrom:to:
- (void)**moveColumnFrom:**(int)*from_pos* **to:**(int)*to_pos*

Moves the column at visual position *from_pos* to visual position *to_pos*. Equivalent to `-border:MISC_COL_BORDER moveSlotFrom:`*from_pos* `to:`*to_pos*.

moveRowFrom:to:
- (void)**moveRowFrom:**(int)*from_pos* **to:**(int)*to_pos*

Moves the row at visual position *from_pos* to visual position *to_pos*. Equivalent to `-border:MISC_ROW_BORDER moveSlotFrom:`*from_pos* `to:`*to_pos*.

numberOfColumns
- (int)**numberOfColumns**

Returns the number of columns in the MiscTableScroll object.   Equivalent to: `-numberOfSlots:MISC_COL_BORDER`.

numberOfRows
  - (int)**numberOfRows**

Returns the number of rows in the MiscTableScroll object.   This is the number of *active* rows currently being displayed.   The MiscTableScroll object performs caching on a row-oriented basis.   There may be additional rows allocated, and stored in the cache.   Equivalent to: `-numberOfSlots:MISC_ROW_BORDER`.

numberOfSelectedColumns
  - (unsigned int)**numberOfSelectedColumns**

Returns the number of selected columns.

numberOfSelectedRows
  - (unsigned int)**numberOfSelectedRows**

Returns the number of selected rows.

numberOfSelectedSlots:
  - (unsigned int)**numberOfSelectedSlots:**(MiscBorderType)*b*

Returns the number of slots that are selected.

numberOfSlots:
- (int)**numberOfSlots:**(MiscBorderType)*b*

Returns the number of slots for the border *b*.

See also: -addSlot:, -border:removeSlot:, -border:insertSlot:, -numberOfColumns, -numberOfRows

numberOfVisibleColumns
- (int)**numberOfVisibleColumns**

Returns the number of columns visible in the scrolling display.   A column is visible if any part of the column (even a single pixel) appears in the scrolling display.   Equivalent to: `-numberOfVisibleSlots:MISC_COL_BORDER`.

numberOfVisibleRows
- (int)**numberOfVisibleRows**

Returns the number of rows visible in the scrolling display.   A row is visible if any part of the row (even a single pixel) appears in the scrolling display.   Equivalent to: `-numberOfVisibleSlots:MISC_ROW_BORDER`.

numberOfVisibleSlots:

- (int)**numberOfVisibleSlots:**(MiscBorderType)*b*

Returns the number of slots visible in the scrolling display.   A slot is visible if any part of the slot (even a single pixel) appears in the scrolling display.

print:
- (void)**print:**(id)*sender*

Prints the MiscTableScroll object, including row and column titles if they are turned on.   This method dispatches the delegate messages `-tableScrollWillPrint:` and `-tableScrollDidPrint:` to bracket the actual printing.   You can implement those methods to perform special operations before and after printing.

readSelectionFromPasteboard:
- (BOOL)**readSelectionFromPasteboard:**(NSPasteboard*)*pboard*

This method is invoked when a service returns some data.   If the *delegate* responds to the `-tableScroll:readSelectionFromPasteboard:` message, it is sent to the *delegate*.   If not, then the *dataDelegate* is given the opportunity.   If neither responds to the message, `-builtinReadSelectionFromPasteboard:` is called. Returns the results of the subroutine that was called.   Override this method in your subclass if you need different behavior.

See also: **-builtinReadSelectionFromPasteboard:, -tableScroll:readSelectionFromPasteboard:** (delegate method)**, -readSelectionFromPasteboard:** (NSServicesRequests)**, -writeSelectionToPasteboard:types:** (NSServicesRequests)

registerServicesTypes
- (void)**registerServicesTypes**

If the *delegate* responds to the `-tableScrollRegisterServicesTypes:` message, the message is sent to the *delegate*.
If not, the *dataDelegate* is tried. If neither responds to the message, `-builtinRegisterServicesTypes` is called.
This method is invoked when an instance of MiscTableScroll object is initialized. Override this method in your subclass if you need different behavior.

See also: -builtinRegisterServicesTypes, -tableScrollRegisterServicesTypes: (delegate method)

removeColumn:
- (void)**removeColumn:**(int)*pos*

Deletes column *n*. Equivalent to: `-border:MISC_COL_BORDER removeSlot:`*n*.

removeRow:
- (void)**removeRow:**(int)*pos*

Deletes row *n*. Equivalent to: `-border:MISC_ROW_BORDER removeSlot:`*n*.

renewRows:

- (void)**renewRows:**(int)*count*

Sets the number of active rows in the MiscTableScroll object to *count*; does not affect the number of columns.   This is the fastest way to change the size of a MiscTableScroll object when you know the number of rows in advance.   See **Usage Tips** in the introduction for more details.

See also: **-addRow, -addSlot:, -border:removeSlot:, -border:insertSlot:, -removeRow:, -empty, -emptyAndReleaseCells, -insertRow:**

resumeEditing:
- (void)**resumeEditing**

Resumes a cell editing session that was suspended by -suspendEditing.

retireCell:atRow:column:
- (id)**retireCell:**(id)*cell* **atRow:**(int)*row* **column:**(int)*col*

Internal method called whenever a cell is being removed from active use.   The method must return *cell*, or a suitable replacement object to place in the cache.   This method provides an opportunity to substitute a different object for *cell* before it is placed in the cache.   This method also provides an opportunity to reclaim storage when a cell is no longer active.   If the *delegate* responds to the `-tableScroll:retireCell:atRow:column:` message, it is sent to the *delegate*.   If not, the *dataDelegate* is tried.   If the *dataDelegate* also does not respond to the message, the cell itself is checked.   If none of these objects responds to the message, a built-in default method, `-doRetireCell:atRow:column:` is called.   Override this method in your subclass if you need different behavior.

See also: **-doRetireCell:atRow:column:, -tableScroll:retireCell:atRow:column:** (delegate method)

reviveCell:atRow:column:
- (id)**reviveCell:**(id)*cell* **atRow:**(int)*row* **column:**(int)

Internal method called whenever a cell is is being moved into active use.   This method is applied to both newly created cells returned by the `-copyWithZone:` method of the column's cell prototype and cells retrieved from the cache.   If the *delegate* responds to the `-tableScroll:reviveCell:atRow:column:` message, it is sent to the *delegate*.   If not, the *dataDelegate* is checked.   If neither the *delegate* nor the *dataDelegate* respond to the message, the cell itself is checked.   If none of these objects respond to the message, a built-in default method `-doReviveCell:atRow:column:` is called.   Override this method in your subclass if you need different behavior.

See also: **-doReviveCell:atRow:column:, -tableScroll:reviveCell:atRow:column:** (delegate method)

rowAdjustedSize:
- (float)**rowAdjustedSize:**(int)*row*

Returns the current display height of *row*.   Equivalent to: `-border:MISC_ROW_BORDER slotAdjustedSize:`*row*.

rowAtPosition:
- (int)**rowAtPosition:**(int)*pos*

Returns the original *physical* position of the row at the current *visual* position *pos*.   This is the visual-to-physical

conversion routine.　Equivalent to: `-border:MISC_ROW_BORDER slotAtPosition:`*pos*.

rowCellPrototype:
- (id)**rowCellPrototype:**(int)*row*

Returns the cell prototype for row *row*.　Equivalent to: `-border:MISC_ROW_BORDER slotCellPrototype:`*row*.

rowCellType:
- (MiscTableCellStyle)**rowCellType:**(int)*row*

Returns the cell type for row *row*.　Equivalent to: `-border:MISC_ROW_BORDER slotCellType:`*row*.

rowIsAutosize:
- (BOOL)**rowIsAutosize:**(int)*row*

Returns the state of the *autosize* flag for row *row*.　Equivalent to: `-border:MISC_ROW_BORDER slotIsAutosize:`*row*.

rowIsSelected:
- (BOOL)**rowIsSelected:**(MiscCoord_P)*row*

Returns YES if *row* is selected, otherwise NO.

rowIsSizeable:
- (BOOL)**rowIsSizeable:**(int)*row*

Returns the state of the *user-sizeable* flag for row *row*.   Equivalent to: `-border:MISC_ROW_BORDER slotIsSizeable:`*row*.

rowIsSorted:
- (BOOL)**rowIsSorted:**(int)*col*

Returns YES if *row* is sorted relative to its neighboring rows.   Returns NO otherwise.   Equivalent to: `-border:MISC_ROW_BORDER slotIsSorted:`*row*.

rowIsVisible:
- (BOOL)**rowIsVisible:**(int)*row*

Returns YES if any part of *row* is visible in the scrolling display.   Returns NO otherwise.   Equivalent to: `-border:MISC_ROW_BORDER slotIsVisible:`*row*.

rowMaxSize:
- (float)**rowMaxSize:**(int)*row*

Returns the maximum size for row *row*.   Equivalent to: `-border:MISC_ROW_BORDER slotMaxSize:`*row*.

rowMinSize:
- (float)**rowMinSize:**(int)*row*

Returns the minimum size for row *row*.   Equivalent to: `-border:MISC_ROW_BORDER slotMinSize:`*row*.

rowOrder
- (NSArray*)**rowOrder**

Equivalent to: `-slotOrder:MISC_ROW_BORDER.`

rowOrderAsString
- (NSString*)**rowOrderAsString**

Equivalent to: `-slotOrderAsString:MISC_ROW_BORDER.`

rowPosition:
- (int)**rowPosition:**(int)*row*

Returns the current *visual* position of the row whose original *physical* position is *pos*.   This is the physical-to-visual conversion routine.   Equivalent to: `-border:MISC_ROW_BORDER slotPosition:`*pos*.

rowsAreSorted
- (BOOL)**rowsAreSorted**

Returns YES if all rows are sorted.   Equivalent to `-slotsAreSorted:MISC_ROW_BORDER`.

rowSize:
- (float)**rowSize:**(int)*row*

Returns the target size for row *row*.   For the actual current display size, use **-rowAdjustedSize:**.   Equivalent to:
`-border:MISC_ROW_BORDER slotSize:`*row*.

rowSizes
- (NSArray*)**rowSizes**

Equivalent to: `-slotSizes:MISC_ROW_BORDER`.

rowSizesAsString
- (NSString*)**rowSizesAsString**

Equivalent to: `-slotSizesAsString:MISC_ROW_BORDER`.

rowSortDirection:
  - (MiscSortDirection)**rowSortDirection:**(int)*n*

Equivalent to: `-border:MISC_ROW_BORDER slotSortDirection:`*n*.

rowSortFunction:
  - (MiscCompareEntryFunc)**rowSortFunction**:(int)*n*

Equivalent to: `-border:MISC_ROW_BORDER slotSortFunction:`*n*.

rowSortType:
  - (MiscSortType)**rowSortType:**(int)*n*

Equivalent to: `-border:MISC_ROW_BORDER slotSortType:`*n*.

rowSortVector
  - (NSArray*)**rowSortVector**

Equivalent to: `-slotSortVector:MISC_ROW_BORDER`.

rowTag:
  - (int)**rowTag:**(int)*row*

Returns the tag for row *row*.   Equivalent to: `-border:MISC_ROW_BORDER slotTag:`*row*.

rowTitle:
-   (NSString*)**rowTitle:**(int)*row*

Returns the title for row *row*.   Equivalent to: `-border:MISC_ROW_BORDER slotTitle:`*row*.

rowTitleMode
-   (MiscTableTitleMode)**rowTitleMode**

Returns the title-mode for row *row*.   Equivalent to: `-border:MISC_ROW_BORDER slotTitleMode:`*row*.

rowTitlesOn
-   (BOOL)**rowTitlesOn**

Indicates whether or not row titles are displayed.   Equivalent to: `-slotTitlesOn:MISC_ROW_BORDER`.

rowTitlesWidth
-   (float)**rowTitlesWidth**

Equivalent to: `-slotTitlesSize:MISC_ROW_BORDER`.

rowWithTag:

- (int)**rowWithTag:**(int)*x*

Returns the index of the first row with tag *x*, or -1 if no row has tag *x*.

scrollCellToVisibleAtRow:column:

- (void)**scrollCellToVisibleAtRow:**(int)*row* **column:**(int)*col*

Scrolls the display as necessary until the cell at position *row*, *col* is visible.

scrollColumnToVisible:

- (void)**scrollColumnToVisible:**(int)*col*

Scrolls the display as necessary until *col* is visible.

scrollRowToVisible:

- (void)**scrollRowToVisible:**(int)*row*

Scrolls the display as necessary until *row* is visible.

scrollSelectionToVisible
- (void)**scrollSelectionToVisible**

Scrolls the display as necessary until the selection is visible.

selectAll:
- (void)**selectAll:**(id)*sender*

Calls `[self selectAllRows]` followed by `[self sendActionIfEnabled]`.

selectAllColumns
- (void)**selectAllColumns**

Equivalent to: `-selectAllSlots:MISC_COL_BORDER`.

selectAllRows
- (void)**selectAllRows**

Equivalent to: `-selectAllSlots:MISC_ROW_BORDER`.

selectAllSlots:
- (void)**selectAllSlots:**(MiscBorderType)*b*

Selects all the slots in border *b*.   Does not send the action to the target.

See also: -selectAll:

selectColumn:byExtension:
- (void)**selectColumn:**(MiscCoord_P)*col* **byExtension:**(BOOL)*flag*

Equivalent to: `-border:MISC_COL_BORDER selectSlot:`*col* `byExtension:`*flag*.

selectColumn:
- (void)**selectColumn:**(MiscCoord_P)*col*

Equivalent to: `-selectColumn:`*col* `byExtension:NO`.

selectColumns:byExtension:
- (void)**selectColumns:**(NSArray*)*cols* **byExtension:**(BOOL)*flag*

Equivalent to: `-border:MISC_COL_BORDER selectSlots:`*cols* `byExtension:`*flag*.

selectColumns:
- (void)**selectColumns:**(NSArray*)*cols*

Equivalent to: `-selectColumns:`*cols* `byExtension:NO`.

selectColumnTags:byExtension:
 - (void)**selectColumnTags:**(NSArray*)*tags* **byExtension:**(BOOL)*flag*

Equivalent to: `-border:MISC_COL_BORDER selectSlotTags:`*tags* `byExtension:`*flag*.

selectColumnTags:
 - (void)**selectColumnTags:**(NSArray*)*tags*

Equivalent to: `-selectColumnTags:`*tags* `byExtension:NO`.

selectedBackgroundColor
 - (NSColor*)**selectedBackgroundColor**

Returns the current selectedBackgroundColor.

selectedCell
 - (id)**selectedCell**

Returns the cell at the intersection of `-selectedColumn` and `-selectedRow` or **nil** if there is no selected cell.   This method really only has meaning in eager-mode, though it can be used in lazy-mode as well.

selectedColumn
- (MiscCoord_P)**selectedColumn**

Equivalent to: `-selectedSlot:MISC_COL_BORDER`.

selectedColumns
- (NSArray*)**selectedColumns**

Equivalent to: `-selectedSlots:MISC_COL_BORDER`.

selectedColumnTags
- (NSArray*)**selectedColumnTags**

Equivalent to: `-selectedSlotTags:MISC_COL_BORDER`.

selectedRow
- (MiscCoord_P)**selectedRow**

Equivalent to: `-selectedSlot:MISC_ROW_BORDER`.

selectedRows
- (NSArray*)**selectedRows**

Equivalent to: `-selectedSlots:MISC_ROW_BORDER`.

selectedRowTags
- (NSArray*)**selectedRowTags**

Equivalent to: `-selectedSlotTags:MISC_ROW_BORDER`.

selectedSlot:
- (MiscCoord_P)**selectedSlot:**(MiscBorderType)*b*

Returns the index of the currently selected slot, or -1 if no slots are selected.

selectedSlots:
- (NSArray*)**selectedSlots:**(MiscBorderType)*b*

Returns an array of NSNumber objects containing indexes of all currently selected slots.

selectedSlotTags:
- (NSArray*)**selectedSlotTags:**(MiscBorderType)*b*

Returns an array of NSNumber objects containing the tags of all currently selected slots.   This method is useful in conjunction with `-border:selectSlotTags:` to save and restore the user's selection when you have tags that uniquely identify the slots.

selectedTextColor
- (NSColor*)**selectedTextColor**

Returns the current selectedTextColor.

selectionChanged
- (void)**selectionChanged**

Invalidates those portions of the display which need to be redrawn in order to reflect the current selection.   When the selection is modified programmatically or via user-interaction this method is called automatically to reflect the new selection.   You need never call this method directly, but subclasses may want to override it.

selectionMode
- (MiscSelectionMode)**selectionMode**

Returns the current setting of the selection mode.

selectRow:byExtension:

- (void)**selectRow:**(MiscCoord_P)*row* **byExtension:**(BOOL)*flag*

Equivalent to: `-border:MISC_ROW_BORDER selectSlot:`*row* `byExtension:`*flag*.

selectRow:

- (void)**selectRow:**(MiscCoord_P)*row*

Equivalent to: `-selectRow:`*row* `byExtension:NO`.

selectRows:byExtension:

- (void)**selectRows:**(NSArray*)*rows* **byExtension:**(BOOL)*flag*

Equivalent to: `-border:MISC_ROW_BORDER selectSlots:`*rows* `byExtension:`*flag*.

selectRows:

- (void)**selectRows:**(NSArray*)*rows*

Equivalent to: `-selectRows:`*rows* `byExtension:NO`.

See also: **±border:selectSlots:, ±selectColumns:, ±selectRows:byExtension:**

selectRowTags:byExtension:
- (void)**selectRowTags:**(NSArray*)*tags* **byExtension:**(BOOL)*flag*

Equivalent to: `-border:MISC_ROW_BORDER selectSlotTags:`*tags* `byExtension:`*flag*.

See also: **±border:selectSlotTags:, ±selectColumnTags:, ±selectRowTags:**

selectRowTags:
- (void)**selectRowTags:**(NSArray*)*tags*

Equivalent to: `-selectRowTags:`*tags* `byExtension:NO`.

See also: **±border:selectSlotTags:, ±selectColumnTags:, ±selectRowTags:byExtension:**

sendAction
- (BOOL)**sendAction**

Sends the *action* message to the *target* object.   Implemented via `-sendAction:to:`. Returns YES if the action is successfully sent, otherwise returns NO.

sendAction:to:
- (BOOL)**sendAction:**(SEL)*theAction* **to:**(id)*theTarget*

Uses the NSApplication class's `-sendAction:to:from:` method to send the message *theAction* to the object *theTarget*

from the MiscTableScroll object itself.   Returns YES if the action is successfully sent, otherwise returns NO.

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

Sends the message *aSelector* to *anObject* for each cell in the table.

sendActionIfEnabled
-   (BOOL)**sendActionIfEnabled**

If `[self isEnabled]` returns YES then `[self sendAction]` is called.   Returns YES if the action is successfully sent, otherwise returns NO.

sendDoubleAction
-   (BOOL)**sendDoubleAction**

Sends the *doubleAction* message to the *doubleTarget* object.   Returns YES if the action is successfully sent, otherwise returns NO.

sendDoubleActionIfEnabled
-   (id)**sendDoubleActionIfEnabled**

If `[self isEnabled]` returns YES then `[self sendDoubleAction]` is called.   Returns YES if the action is successfully sent, otherwise returns NO.

setAction:
- (void)**setAction:**(SEL)*new_sel*

Sets the action method to *new_sel*.   The action message is sent to the *target* upon a single mouse click.   The argument of an action method is the table scroll.

setAutoSortColumns:
- (void)**setAutoSortColumns:**(BOOL)*flag*

Equivalent to: `-border:MISC_COL_BORDER setAutoSortSlots:`*flag*.

setAutoSortRows:
- (void)**setAutoSortRows:**(BOOL)*flag*

Equivalent to: `-border:MISC_ROW_BORDER setAutoSortSlots:`*flag*.

setBackgroundColor:
- (void)**setBackgroundColor:**(NSColor*)*value*

Sets the *backgroundColor*.   The *backgroundColor* is used to initialize new cells, and also to paint the background of areas that are not covered by cells of the table.   By default, this is the value returned by `+defaultBackgroundColor`.

See also: **+defaultBackgroundColor**

setColor:
   -   (void)**setColor:**(NSColor*)*value*

Equivalent to: `-setBackgroundColor:value`.

setColumn:autosize:
   -   (void)**setColumn:**(int)*col* **autosize:**(BOOL)*flag*

Sets the autosize flag for column *col*.   Equivalent to: `-border:MISC_COL_BORDER setSlot:col autosize:flag`.

setColumn:cellPrototype:
   -   (void)**setColumn:**(int)*col* **cellPrototype:**(id)*cell*

Sets the cell prototype for column *col* to *cell*.   Equivalent to: `-border:MISC_COL_BORDER setSlot:col cellPrototype:cell`.

setColumn:cellType:

- (void)**setColumn:**(int)*col* **cellType:**(MiscTableCellStyle)*type*

Sets the cell type for column *col* to *type*. Equivalent to: `-border:MISC_COL_BORDER setSlot:`*col* `cellType:`*type*.

setColumn:maxSize:
- (void)**setColumn:**(int)*col* **maxSize:**(float)*size*

Sets the maximum size of column *col* to *size*. Equivalent to: `-border:MISC_COL_BORDER setSlot:`*col* `maxSize:`*size*.

setColumn:minSize:
- (void)**setColumn:**(int)*col* **minSize:**(float)*size*

Sets the minimum size of column *col* to *size*. Equivalent to: `-border:MISC_COL_BORDER setSlot:`*col* `minSize:`*size*.

setColumn:size:
- (void)**setColumn:**(int)*col* **size:**(float)*size*

Sets the target size of column *col* to *size*. Equivalent to: `-border:MISC_COL_BORDER setSlot:`*col* `size:`*size*.

setColumn:sizeable:
- (void)**setColumn:**(int)*col* **sizeable:**(BOOL)*flag*

Sets the *user-sizeable* flag for column *col* to *flag*.   Equivalent to: `-border:MISC_COL_BORDER setSlot:`*col* `sizeable:`*flag*.

setColumn:sortDirection:
- (void)**setColumn:**(int)*n* **sortDirection:**(MiscSortDirection)*x*

Equivalent to: `-border:MISC_COL_BORDER setSlot:`*n* `sortDirection:`*x*.

setColumn:sortFunction:
- (void)**setColumn:**(int)*n* **sortFunction**:(MiscCompareEntryFunc)*x*

Equivalent to: `-border:MISC_COL_BORDER setSlot:`*n* `sortFunction:`*x*.

setColumn:sortType:
- (void)**setColumn:**(int)*n* **sortType:**(MiscSortType)*x*

Equivalent to: `-border:MISC_COL_BORDER setSlot:`*n* `sortType:`*x*.

setColumn:tag:
- (void)**setColumn:**(int)*col* **tag:**(int)*tag*

Sets the tag for column *col* to *tag*.   Equivalent to: `-border:MISC_COL_BORDER setSlot:`*col* `tag:`*tag*.

setColumn:title:
- (void)**setColumn:**(int)*col* **title:**(NSString*)*title*

Sets the title for column *col* to *title*.   Equivalent to: `-border:MISC_COL_BORDER setSlot:`*col* `title:`*title*.

setColumnOrder:
- (BOOL)**setColumnOrder:**(NSArray*)*list*

Equivalent to: `-border:MISC_COL_BORDER setSlotOrder:`*list*

setColumnOrderFromString:
- (BOOL)**setColumnOrderFromString:**(NSString*)*s*

Equivalent to: `-border:MISC_COL_BORDER setSlotOrderFromString:`*s*.

setColumnSizes:
- (BOOL)**setColumnSizes:**(NSArray*)*list*

Equivalent to: `-border:MISC_COL_BORDER setSlotSizes:`*list*.

setColumnSizesFromString:
- (BOOL)**setColumnSizesFromString:**(NSString*)*s*

Equivalent to: `-border:MISC_COL_BORDER setSlotSizesFromString:s`.

setColumnSortVector:
- (void)**setColumnSortVector:**(NSArray*)*v*

Equivalent to: `-border:MISC_COL_BORDER setSlotSortVector:v`

setColumnTitleMode:
- (void)**setColumnTitleMode:**(MiscTableTitleMode)*x*

Sets the *title-mode* for column *col* to *x*.   Equivalent to: `-border:MISC_COL_BORDER setSlot:col titleMode:x`.

setColumnTitlesHeight:
- (void)**setColumnTitlesHeight:**(float)*height*

Equivalent to: `-border:MISC_COL_BORDER setSlotTitlesSize:size`.

setColumnTitlesOn:
- (BOOL)**setColumnTitlesOn:**(BOOL)*on_off*

Turns the column titles on or off.   When *on_off* is YES, column titles will be displayed.   When *on_off* is NO, column titles will not be displayed.   Column titles are displayed by default.   Equivalent to: `-border:MISC_COL_BORDER setSlotTitlesOn:`*on_off*.

setCompareSlotFunction:
  - (void)**setCompareSlotFunction**:(MiscCompareSlotFunc)*f*

Makes *f* the slot comparison function to be used for sorting.   It must conform to the following prototype from `MiscTableTypes.h`:

```
typedef int (*MiscCompareSlotFunc)( int slot1, int slot2, MiscSlotSortInfo* );
```

The function must return an integer value which is: (a) less than zero if *slot1* should come before *slot2*, or (b) equal to zero if *slot1* should sort equally with *slot2*, or (c) greater than zero if *slot1* should come after *slot2*.   This function is responsible for comparing the cells of the two slots in the order defined by the *slotSortVector*, or visual order if no explicit *slotSortVector* has been set.   This function is also responsible for applying the sort direction to the individual cell-wise comparisons.   This function is also responsible for calling user-installed custom slot sorting functions, or interpreting and applying the sort-type for slots that do not have a custom function.   The default, built-in implementation of this function is `MiscDefaultCompareSlotFunc`.

See also: **-border:setSlot:sortDirection:, -border:setSlot:sortFunction:, -border:setSlot:sortType:, -border:setSlotSortVector:, -compareSlotFunction, -sortInfoDone:, -sortInfoInit:border:**

setCornerTitle:

- (void)**setCornerTitle:**(NSString*)*s*

Sets the title for the corner cell.

setCursorColumn:

- (void)**setCursorColumn:**(MiscCoord_P)*col*

Equivalent to: `-border:MISC_COL_BORDER setCursorSlot:`*col*.

setCursorRow:

- (void)**setCursorRow:**(MiscCoord_P)*row*

Equivalent to: `-border:MISC_ROW_BORDER setCursorSlot:`*row*.

setDataDelegate:

- (void)**setDataDelegate:**(id)*obj*

Makes *obj* the data delegate for the MiscTableScroll object.   Does not retain *obj*.

See also: -setDelegate:, -setLazy:

setDelegate:
  - (void)**setDelegate:**(id)*obj*

Makes *obj* the delegate for the MiscTableScroll object.   Does not retain *obj*.

See also: -setDataDelegate:

setDoubleAction:
  - (void)**setDoubleAction:**(SEL)*new_sel*

Sets the double-action method to *new_sel*.   The double-action message is sent to the *doubleTarget* upon a double mouse click.   The argument of an action method is the table scroll.

setDoubleTarget:
  - (void)**setDoubleTarget:**(id)*obj*

Makes *obj* the *doubleTarget* of the MiscTableScroll object.   Does not retain *obj*.

setDraggableColumns:
  - (void)**setDraggableColumns:**(BOOL)*flag*

Enables or disables *user-dragging* of columns.   When *flag* is YES, columns will be user-draggable.   When *flag* is NO, columns will not be user-draggable.   The column titles must be displayed to enable the user to drag columns.
Equivalent to: `-border:MISC_COL_BORDER setDraggableSlots:`*flag*.

setDraggableRows:
- (void)**setDraggableRows:**(BOOL)*flag*

Enables or disables *user-dragging* of rows.   When *flag* is YES, rows will be user-draggable.   When *flag* is NO, rows will not be user-draggable.   The row titles must be displayed to enable the user to drag rows.   Equivalent to:
`-border:MISC_ROW_BORDER setDraggableSlots:`*flag*.

setDrawClippedText:
- (void)**setDrawClippedText:**(BOOL)*flag*

When *flag* is YES, the MiscTableScroll object will use clipping rectangles to draw partially visible text in cells that respond YES to the `-ownerDraw` message.   When *flag* is NO, the MiscTableScroll object will simply not draw partially visible text that would require clipping rectangles for cells that respond YES to the `-ownerDraw` message. This is a drawing performance optimization.   The clipping rectangles are quite slow (especially noticable on older, slower CPUs).   Drawing clipped text is disabled by default.   You must send this message with *flag* equal to YES to enable partially visible text to be drawn.   If the new setting is different than the existing setting, `[self setNeedsDisplay:YES]` is called.

setEnabled:
- (void)**setEnabled:**(BOOL)*flag*

Enables or disables user-interaction with the MiscTableScroll object.   The only feature affected by this flag is the

dispatch of the *action* and *doubleAction*.   When *flag* is YES, *action* and *doubleAction* are sent as appropriate.   When *flag* is NO, neither messsage is sent.   All other operations are unaffected and remain available.   These include, pasteboard and service operations, selection, column reordering & sizing, etc.

setFirstVisibleColumn:
- (void)**setFirstVisibleColumn:**(int)*col*

Equivalent to: `-border:MISC_COL_BORDER setFirstVisibleSlot:`*col*.

setFirstVisibleRow:
- (void)**setFirstVisibleRow:**(int)*row*

Equivalent to: `-border:MISC_ROW_BORDER setFirstVisibleSlot:`*row*.

setFont:
- (void)**setFont:**(NSFont*)*newFont*

Sets the font for the MiscTableScroll object.   The font is used to initialize new cells in the table.   If rows are uniformly sized, the uniform row size is adjusted proportionately based on the sizes of the old font and the new font.   Then all the cells are updated.   If the cells respond to the `-setOwnerFont:` message, that message is sent.   Otherwise the `-setFont:` message is tried.   Then the `-tableScroll:fontChangedFrom:to:` message is sent to the delegate if the delegate responds to it.   Finally, the display is invalidated.

See also: -tableScroll:fontChangedFrom:to: (delegate method)**, -setOwnerFont:** (MiscTableCell)**, -setFont:**

(NSCell, MiscTableCell)

setLastVisibleColumn:
- (void)**setLastVisibleColumn:**(int)*col*

Equivalent to: `-border:MISC_COL_BORDER setLastVisibleSlot:`*col*.

setLastVisibleRow:
- (void)**setLastVisibleRow:**(int)*row*

Equivalent to: `-border:MISC_ROW_BORDER setLastVisibleSlot:`*row*.

setLazy:
- (void)**setLazy:**(BOOL)*flag*

Enables or disables lazy-mode memory management. When *flag* is YES, the MiscTableScroll object will use lazy-mode memory management, asking the *delegate*, and then if necessary the *dataDelegate* to provide the cells in the body of the table. When *flag* is NO, the MiscTableScroll object will use eager-mode memory management, maintaining a dense, 2-D array of cell pointers, one pointer for each cell in the table, and caching cells on a row-wise basis. MiscTableScroll uses eager-mode memory management by default. See **Usage Tips**, and **Lazy vs. Eager**, in the introduction for more details.

See also: **-dataDelegate, -isLazy, -setDataDelegate:**

setMaxUniformSizeColumns:
 - (void)**setMaxUniformSizeColumns:**(float)*size*

Equivalent to: `-border:MISC_COL_BORDER setMaxUniformSizeSlots:`*size*.

setMaxUniformSizeRows:
 - (void)**setMaxUniformSizeRows:**(float)*size*

Equivalent to: `-border:MISC_ROW_BORDER setMaxUniformSizeSlots:`*size*.

setMinUniformSizeColumns:
 - (void)**setMinUniformSizeColumns:**(float)*size*

Equivalent to: `-border:MISC_COL_BORDER setMinUniformSizeSlots:`*size*.

setMinUniformSizeRows:
 - (void)**setMinUniformSizeRows:**(float)*size*

Equivalent to: `-border:MISC_ROW_BORDER setMinUniformSizeSlots:`*size*.

setModifierDragColumns:

- (void)**setModifierDragColumns:**(BOOL)*flag*

Sets whether or not the command-key must be held down to drag columns.   By default, columns require the command-key to perform selection.   Equivalent to `-border:MISC_COL_BORDER setModifierDragSlots:`*flag*.

setModifierDragRows:
- (void)**setModifierDragRows:**(BOOL)*flag*

Sets whether or not the command-key must be held down to drag rows.   By default, rows do not require the command-key to perform selection.   Equivalent to `-border:MISC_ROW_BORDER setModifierDragSlots:`*flag*.

setRow:autosize:
- (void)**setRow:**(int)*row* **autosize:**(BOOL)*flag*

Sets the *autosize* flag for row *row*.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row* `autosize:`*flag*.

setRow:cellPrototype:
- (void)**setRow:**(int)*row* **cellPrototype:**(id)*cell*

Sets the cell prototype for row *row* to *cell*.   Currently, only column cell prototypes are used.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row* `cellPrototype:`*cell*.

setRow:cellType:
- (void)**setRow:**(int)*row* **cellType:**(MiscTableCellStyle)*type*

Sets the cell type for row *row* to *type*.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row* `cellType:`*type*.

setRow:maxSize:
- (void)**setRow:**(int)*row* **maxSize:**(float)*size*

Sets the maximum size of row *row* to *size*.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row* `maxSize:`*size*.

setRow:minSize:
- (void)**setRow:**(int)*row* **minSize:**(float)*size*

Sets the minimum size of row *row* to *size*.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row* `minSize:`*size*.

setRow:size:
- (void)**setRow:**(int)*row* **size:**(float)*size*

Sets the target size of row *row* to *size*.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row* `size:`*size*.

setRow:sizeable:
- (void)**setRow:**(int)*row* **sizeable:**(BOOL)*flag*

Sets the user-sizeable flag for row *row* to *flag*.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row* `sizeable:`*flag*.

setRow:sortDirection:
- (void)**setRow:**(int)*n* **sortDirection:**(MiscSortDirection)*x*

Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*n* `sortDirection:`*x*.

setRow:sortFunction:
- (void)**setRow:**(int)*n* **sortFunction**:(MiscCompareEntryFunc)*x*

Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*n* `sortFunction:`*x*.

setRow:sortType:
- (void)**setRow:**(int)*n* **sortType:**(MiscSortType)*x*

Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*n* `sortType:`*x*.

setRow:tag:
- (void)**setRow:**(int)*row* **tag:**(int)*tag*

Sets the tag for row *row* to *tag*.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row*` tag:`*tag*.

setRow:title:
- (void)**setRow:**(int)*row* **title:**(NSString*)*title*

Sets the title for row *row* to *title*.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row*` title:`*title*.

setRowOrder:
- (BOOL)**setRowOrder:**(NSArray*)*list*

Equivalent to: `-border:MISC_ROW_BORDER setSlotOrder:`*list*.

setRowOrderFromString:
- (BOOL)**setRowOrderFromString:**(NSString*)*s*

Equivalent to: `-border:MISC_ROW_BORDER setSlotOrderFromString:`*s*.

setRowSizes:
- (BOOL)**setRowSizes:**(NSArray*)*list*

Equivalent to: `-border:MISC_ROW_BORDER setSlotSizes:`*list*.

setRowSizesFromString:
- (BOOL)**setRowSizesFromString:**(NSString*)*s*

Equivalent to: `-border:MISC_ROW_BORDER setSlotSizesFromString:`*s*.

setRowSortVector:
- (void)**setRowSortVector:**(int const*)*v*

Equivalent to: `-border:MISC_ROW_BORDER setSlotSortVector:`*v*

setRowTitleMode:
- (void)**setRowTitleMode:**(MiscTableTitleMode)*x*

Sets the title-mode for row *row* to *x*.   Equivalent to: `-border:MISC_ROW_BORDER setSlot:`*row* `titleMode:`*x*.

setRowTitlesOn:
- (BOOL)**setRowTitlesOn:**(BOOL)*on_off*

Turns the row titles on or off.   When *on_off* is YES, row titles will be displayed.   When *on_off* is NO, row titles will not be displayed.   Row titles are not displayed by default.   Equivalent to: `-border:MISC_ROW_BORDER` `setSlotTitlesOn:`*on_off*.

setRowTitlesWidth:
-   (void)**setRowTitlesWidth:**(float)*size*

Equivalent to: `-border:MISC_ROW_BORDER setSlotTitlesSize:`*size*.

setSelectedBackgroundColor:
-   (void)**setSelectedBackgroundColor:**(NSColor*)*value*

Sets the *selectedBackgroundColor* for the MiscTableScroll object.   The *selectedBackgroundColor* is used to initialize new cells added to the table.   This information is propagated to the cells of the table as follows.   If the cells respond to the `-setOwnerSelectedBackgroundColor:` message, that message is sent, else if the cells respond to the `-setSelectedBackgroundColor:` message, that message is sent instead.   If the cells do not respond to either of these messages, no message is sent to the cell.   Finally, the display is invalidated.

See also: **-setSelectedBackgroundColor:** (MiscTableCell)**, -setOwnerSelectedBackgroundColor:** (MiscTableCell)

setSelectedTextColor:
-   (void)**setSelectedTextColor:**(NSColor*)*value*

Sets the *selectedTextColor* for the MiscTableScroll object.   The *selectedTextColor* is used to initialize new cells added to the table.   This message is also propagated to the existing cells of the table as follows.   If the cells respond to the `-setOwnerSelectedTextColor:` message, that message is sent, else if the cells respond to the `-setSelectedTextColor:` message, that message is sent.   If the cells do not respond to either of these messages, no

message is sent to the cell.   Finally, the display is invalidated.

setSelectionMode:
-   (void)**setSelectionMode:**(MiscSelectionMode)*x*

Sets the selection mode for the MiscTableScroll object.   The selection mode, *x*, can be any of the following:

```
MISC_LIST_MODE,
MISC_RADIO_MODE,
MISC_HIGHLIGHT_MODE
```

The modes each correspond to the similarly named selection modes declared in the NSMatrix class.   The MiscTableScroll object extends the highlight mode selection by implementing the Alternate-key modifier in the same fashion that it works in list mode.

setSizeableColumns:
-   (void)**setSizeableColumns:**(BOOL)*flag*

Enables or disables user-sizing of columns.   Equivalent to: `-border:MISC_COL_BORDER setSizeableSlots:`*flag*.

setSizeableRows:
-   (void)**setSizeableRows:**(BOOL)*flag*

Enables or disables user-sizing of rows. Equivalent to: `-border:MISC_ROW_BORDER setSizeableSlots:`*flag*.

setTag:
- (void)**setTag:**(int)*x*

Sets the **tag** of the MiscTableScroll object to *x*.

See also: -border:setSlot:tag:, -setColumn:tag:, -setRow:tag:, -tag

setTarget:
- (void)**setTarget:**(id)*obj*

Makes *obj* the object which will receive the *action* message whenever there is a single mouse-click on the body of the table.   Does not retain *obj*.

setTextColor:
- (void)**setTextColor:**(NSColor*)*value*

Sets the *textColor* for the MiscTableScroll object.   The *textColor* is used to initialize new cells added to the table. The message is propagated to existing cells as follows.   If the cell responds to the `-setOwnerTextColor:` message, that message is sent, else if the cell responds to the `-setTextColor:` message, that message is sent.   If the cell does not respond to either of these messages, no message is sent to the cell.   Finally, the display is invalidated.

setUniformSizeColumns:
-   (void)**setUniformSizeColumns:**(float)*uniform_size*

Sets or clears the *uniform-size* property for columns.   When *uniform_size* is a non-zero value, all columns will have the same, fixed (uniform) size.   When *uniform_size* is zero, each column can be assigned sizes individually.   By default, columns are not uniformly sized.   Equivalent to: `-border:MISC_COL_BORDER setUniformSizeSlots:`*uniform_size*.

setUniformSizeRows:
-   (void)**setUniformSizeRows:**(float)*uniform_size*

Sets or clears the *uniform-size* property for rows.   When *uniform_size* is a non-zero value, all rows will have the same, fixed (uniform) size.   When *uniform_size* is zero, each row can be assigned sizes individually.   By default, row are uniformly sized.   Equivalent to: `-border:MISC_ROW_BORDER setUniformSizeSlots:`*uniform_size*.

sizeableColumns
-   (BOOL)**sizeableColumns**

Indicates whether or not columns can be resized by the user.   Equivalent to: `-sizeableSlots:MISC_COL_BORDER`.

sizeableRows
- (BOOL)**sizeableRows**

Indicates whether or not rows can be resized by the user.   Equivalent to: `-sizeableSlots:MISC_ROW_BORDER`.

sizeableSlots:
- (BOOL)**sizeableSlots:**(MiscBorderType)*b*

Indicates whether or not the user can resize the slots on border *b*.

sizeToCells
- (void)**sizeToCells**

Instructs the MiscTableScroll object to adjust the frames of its subviews.

See also: **-addRow**

sizeToFit
- (void)**sizeToFit**

Calculates the size of every cell in the MiscTableScroll object using `-cellSize:`.   Then uses the maximum size for each slot to set the size of the slot.   For uniform-size borders, the size is set to the maximum size of all slots.   Finally, this method calls `-sizeToCells` to finish the process of updating the frames of all the subviews.

slotsAreSorted:
- (BOOL)**slotsAreSorted:**(MiscBorderType)*b*

Returns YES if the slots are sorted, NO otherwise.

See also: -border:slotIsSorted:, -border:sortSlot:, -sortSlots:

slotOrder:
- (NSArray*)**slotOrder:**(MiscBorderType)*b*

Returns an array of NSNumber objects containing the current slot order.   The list is organized in *physical* (original) slot order.   Each value in the list is the current *visual* position of the corresponding slot.   In other words, returns the physical to visual mapping.   This method is useful for saving the user's slot order preference.

List also encodes the sort direction.   Negative values indicate slots that are sorted in descending order.   The negative value is computed by using the 'C' bitwise complement operator (~).

See also: -border:setSlotOrder:, -border:setSlotSizes:, -slotOrderAsString:

slotOrderAsString:
- (NSString*)**slotOrderAsString:**(MiscBorderType)*b*

Returns a string representation of the array returned by `-slotOrder:`. This is useful for saving and restoring user slot

order preferences.

See also: -border:setSlotOrder:, -slotOrder:

slotSizes:
  - (NSArray*)**slotSizes:**(MiscBorderType)*b*

Returns an array of NSNumber objects containing the sizes of all slots.   The list is organized in *physical* (original) slot order.   The values are the sizes of the correspond slot.   This method is useful for saving the user's slot size preferences.

See also: -border:setSlotSizes:, -slotOrder:, -slotSizesAsString:

slotSizesAsString:
  - (NSString*)**slotSizesAsString:**(MiscBorderType)*b*

Returns a string representation of the array returned by `-slotSizes:`.   This is useful for saving and restoring user slot size preferences.

See also: -slotSizes:

slotSortVector
  - (NSArray*)**slotSortVector:**(MiscBorderType)*b*

Returns the current *slotSortVector* for border *b*, which is an array of NSNumber objects.

See also: -border:setSlotSortVector:

slotTitleMode:
- (MiscTableTitleMode)**slotTitleMode:**(MiscBorderType)*b*

Returns the title-mode for *slot*.

slotTitlesOn:
- (BOOL)**slotTitlesOn:**(MiscBorderType)*b*

Indicates whether or not the titles for border *b* are displayed.

slotTitlesSize:
- (float)**slotTitlesSize:**(MiscBorderType)*b*

Returns either the height of column titles or the width of row titles, based upon *b*.

sortColumn:
- (void)**sortColumn:**(int)*n*

Re-sorts a single column.   Equivalent to: `-border:MISC_COL_BORDER sortSlot:`*n*.

See also: -border:sortSlot:, -sortSlots:

sortColumns
  - (void)**sortColumns**

Equivalent to: `-sortSlots:MISC_COL_BORDER`.

See also: -sortSlots:

sortInfoDone:
  - (void)**sortInfoDone:**(MiscSlotSortInfo*)*sortInfo*

This method reclaims temporary storage held in the *sortInfo* structure.   You must call this method whenever you are finished using a *sortInfo* object.

See also: -sortInfoInit:border:

sortInfoInit:border:
  - (void)**sortInfoInit:**(MiscSlotSortInfo*)*sortInfo* **border:**(MiscBorderType)*b*

This method precomputes the sorting information needed by the sorting methods.   If you call any of the sorting methods that accept an `info:` argument, you must initialize the *sortInfo* structure by calling this method first.   After

you have finished using the *sortInfo* structure, you must reclaim the storage by passing the *sortInfo* structure to `-sortInfoDone:`.   NOTE: The *sortInfo* structure stores the current sorting information for the table.   Any changes made to the sorting environment after the *sortInfo* structure has been initialized will not affect the contents of the *sortInfo* structure, and therefore will not affect comparisons made using the *sortInfo* structure.   Actions that affect the sorting environment include: rearranging columns/rows, installing a slotSortVector, installing a custom slot comparison function, changing the sort-type or sort-direction of a slot.   Actions which alter the structure of the "other" border (like removing columns/rows) can potentially cause catastrophic failures.

See also: -border:compareSlots::info:

sortRow:
- (void)**sortRow:**(int)*n*

Re-sorts a single row.   Equivalent to: `-border:MISC_ROW_BORDER sortSlot:`*n*.

See also: -border:sortSlot:, -sortSlots:

sortRows
- (void)**sortRows**

Equivalent to: `-sortSlots:MISC_ROW_BORDER`.

See also: -sortSlots:

sortSlots:
- (void)**sortSlots:**(MiscBorderType)*b*

Sorts the slots in border *b*.

See also: **-border:setSlot:sortDirection:, -border:setSlot:sortFunction:, -border:setSlot:sortType:,**
-border:setSlotSortVector:, -setCompareSlotFunction:

stateAtRow:column:
- (int)**stateAtRow:**(int)*row* **column:**(int)*col*

Returns the value of sending a `-state` message to the cell at *row*, *col*.   If the table is lazy, the *delegate*, and then the *dataDelegate* are given the opportunity to reply to the `-tableScroll:stateAtRow:column:` message.   This gives lazy tables an opportunity to return this information directly, without the overhead of preparing and formatting a cell.   If the table is not lazy, or the *delegate* and *dataDelegate* do not respond to the `-tableScroll:doubleValueAtRow:column:` message, then the cell is retrieved via `-cellAtRow:column:.`   If the cell responds to the `-state` message, that value is returned; otherwise, zero is returned.

See also: **-cellAtRow:column:, -isLazy, -setLazy:, -state** (NSButtonCell)**, -tableScroll:stateAtRow:column:**
(delegate method)

stringForNSStringPboardType
- (NSString*)**stringForNSStringPboardType**

Returns a string of the selected cells as ASCII text.   Columns are separated by tab characters (ASCII decimal 9).

Rows are terminated with newline characters (ASCII decimal 10).   The text is retrieved from the cells by first trying the `-title` message.   If the cell does not respond to the `-title` message, then the `-stringValue` message is tried. Each tab character in the text retrieved from the cell is replaced with a single space character (ASCII decimal 32) before the text is written to string.   The selection is written in the current (*visual*) ordering.   Called from `-builtinStringForPboardType:`, and `-stringForNSTabularTextPBoardType`.   Override this method in your subclass if you want different behavior.

See also: -builtinStringForPboardType:, -stringForNSTabularTextPBoardType

stringForNSTabularTextPBoardType
  - (NSString*)**stringForNSTabularTextPBoardType**

Calls `[self stringForNSStringPboardType]`.   Called from `-builtinStringForPboardType:`.   Override this method in your subclass if you want different behavior.

See also: -builtinStringForPboardType:, -stringForNSStringPboardType

stringValueAtRow:column:
  - (NSString*)**stringValueAtRow:**(int)*row* **column:**(int)*col*

Returns the value of sending a `-stringValue` message to the cell at *row*, *col*.   If the table is lazy, the *delegate*, and then the *dataDelegate* are given the opportunity to reply to the `-tableScroll:stringValueAtRow:column:` message. This gives lazy tables an opportunity to return this information directly, without the overhead of preparing and formatting a cell.   If the table is not lazy, or the *delegate* and *dataDelegate* do not respond to the

`-tableScroll:doubleValueAtRow:column:` message, then the cell is retrieved via `-cellAtRow:column:`. If the cell responds to the `-stringValue` message, that value is returned; otherwise, zero (a NULL pointer) is returned. NOTE: If you are using NSButtonCells, you probably want `-titleAtRow:column:`, not this method.

See also: **-cellAtRow:column:, -isLazy, -setLazy:, -tableScroll:stringValueAtRow:column:** (delegate method)**, -titleAtRow:column:**

suspendEditing
-   (void)**suspendEditing**

Internal method that temporarily suspends the current cell editing session (if any), while slots are being resized or rearranged.   The editing session is resumed by `-resumeEditing`. These calls nest. These methods do nothing if cell editing is not in progress.

tag
-   (int)**tag**

Returns the *tag* of the MiscTableScroll object.

See also: **-border:slotTag:, -columnTag:, -rowTag:, -setTag:**

tagAtRow:column:
-   (int)**tagAtRow:**(int)*row* **column:**(int)*col*

Returns the value of sending a `-tag` message to the cell at *row*, *col*.   If the table is lazy, the *delegate*, and then the *dataDelegate* are given the opportunity to reply to the `-tableScroll:tagAtRow:column:` message.   This gives lazy tables an opportunity to return this information directly, without the overhead of preparing and formatting a cell.   If the table is not lazy, or the *delegate* and *dataDelegate* do not respond to the `-tableScroll:doubleValueAtRow:column:` message, then the cell is retrieved via `-cellAtRow:column:`.   If the cell responds to the `-tag` message, that value is returned; otherwise, zero is returned.

See also: **-cellAtRow:column:, -isLazy, -setLazy:, -tableScroll:tagAtRow:column:** (delegate method)

target
- (id)**target**

Returns a pointer to the object which receives the *action* message on a single mouse-click event.

textColor
- (NSColor*)**textColor**

Returns the current textColor.

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

Sends the notification `NSControlTextDidBeginEditingNotification` which the delegates are automatically registered

to receive.

See also: **± NSControlTextDidBeginEditing** (NSControl notification)

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

Sends the notification `NSControlTextDidChangeNotification` which the delegates are automatically registered to receive.

See also: **± NSControlTextDidChangeNotification** (NSControl notification)

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

Invoked by the NSText object, when text editing ends.   If the text changed then attempts to set the cell's new value by sending `-tableScroll:setStringValue:atRow:column:` to the *delegate* or *dataDelegate*.   If neither responds, then sends `-setStringValue:` to the cell instead.   It then sends the delegate notification `-tableScroll:didEdit:atRow:column:`, sorts the data if auto-sorting is enabled, and finally sends the notification `NSControlTextDidEndEditingNotification`, which the delegate is automatically registered to receive.   Lastly, it checks to see if one of *return*, *tab*, or *shift-tab* caused the editing to end.   If the cause was return, then the action is sent to the target, else if it was tab or shift-tab, then `-getNext:editRow:column:` is invoked to determine which cell should be edited next and editing is initiated for that cell using `-editCellAtRow:column:`.   If no cell is eligible for editing, then `-selectNextKeyView:` or `-selectPreviousKeyView:` is sent to the window as appropriate.

You should never need to call this method manually, though subclasses may want to override it.

See also: **± NSControlTextDidEndEditingNotification** (NSControl notification)

textShouldBeginEditing:
 - (BOOL)**textShouldBeingEditing:**(NSText*)*sender*

Invoked automatically during editing to determine if it is okay to edit the cell.   Sends `-control:textShouldBeginEditing:` to the *delegate* or *dataDelegate*.   If neither responds then returns YES.

See also: **± control:textShouldBeginEditing:** (NSControl delegate), **± textShouldBeginEditing:** (NSText delegate)

textShouldEndEditing:
 - (BOOL)**textShouldEndEditing:**(NSText*)*sender*

Invoked automatically during editing to determine if it is okay to end editing.   Sends `-control:textShouldEndEditing:` to the *delegate* or *dataDelegate*.   If neither responds then returns YES.

See also: **± control:textShouldEndEditing:** (NSControl delegate), **± textShouldEndEditing:** (NSText delegate)

titleAtRow:column:
 - (NSString*)**titleAtRow:**(int)*row* **column:**(int)*col*

Returns the value of sending a `-title` message to the cell at *row*, *col*.   If the table is lazy, the *delegate*, and then the

dataDelegate are given the opportunity to reply to the `-tableScroll:titleAtRow:column:` message.   This gives lazy tables an opportunity to return this information directly, without the overhead of preparing and formatting a cell.   If the table is not lazy, or the *delegate* and *dataDelegate* do not respond to the `-tableScroll:titleAtRow:column:` message, then the cell is retrieved via `-cellAtRow:column:`.   If the cell responds to the `-title` message, that value is returned; otherwise, zero (a NULL pointer) is returned.   NOTE: NSButtonCell implements the `-stringValue` message by formatting the integer value of its state as a string.   To retrieve the text label displayed on the button, you must use the `-title` method.

See also: **-cellAtRow:column:, -isLazy, -setLazy:, -tableScroll:titleAtRow:column:** (delegate method)**, -title** (NSButtonCell)

totalHeight
- (float)**totalHeight**

Equivalent to: `-totalSize:MISC_ROW_BORDER`.

totalSize:
- (float)**totalSize:**(MiscBorderType)*b*

Returns the total display size.   The sum of `-border:`*b* `slotAdjustedSize:` for all slots on the border.

See also: **-border:slotAdjustedSize:**

totalWidth

- (float)**totalWidth**

Equivalent to: `-totalSize:MISC_COL_BORDER`.

trackBy:

- (void)**trackBy:**(MiscBorderType)*b*

Sets the orientation by which the mouse is tracked for selection.   If *b* is MISC_ROW_BORDER then selection is performed on a row-wise basis.   If *b* is MISC_COL_BORDER then selection is performed on a column-wise basis.

See also: ± trackingBy

tracking

- (BOOL)**tracking**

Returns YES if the mouse is currently being tracked by a cell, else NO.   Technically, this method returns YES after the cell which will track the mouse has been highlighted, and NO after it has been unhighlighted.   The cell which is tracking the mouse can be accessed via `-clickedCell`, `-clickedColumn`, or `-clickedRow`.   Although setting a Cell's *highlight* flag is sufficient during mouse tracking in eager-mode, it is not sufficient in lazy-mode.   Therefore MiscTableScroll uses this method in lazy-mode to determine when a cell should be drawn highlighted.   You should rarely need to call this method, though it might be useful in subclasses.

See also: ± clickedCell, ± clickedColumn, ± clickedRow, ± clickedSlot:

trackingBy
- (MiscBorderType)**trackingBy**

Returns the current orientation of keyboard tracking.

See also: ± trackBy:

uniformSizeColumns
- (float)**uniformSizeColumns**

Returns the uniform size for columns.   If columns are not being sized uniformly, this method will return zero.   Any non-zero value indicates the size that all columns have the same size as the value returned by this method.   By default, columns are not uniformly sized.   Equivalent to: `-uniformSizeSlots:MISC_COL_BORDER`.

uniformSizeRows
- (float)**uniformSizeRows**

Returns the uniform size for rows.   If rows are not being sized uniformly, this method will return zero.   Any non-zero value indicates the size that all rows have the same size as the value returned by this method.   Rows are uniformly sized by default.   Equivalent to: `-uniformSizeSlots:MISC_ROW_BORDER`.

uniformSizeSlots:

- (float)**uniformSizeSlots:**(MiscBorderType)*b*

Returns the uniform size for slots in border *b*.   If slots are not being sized uniformly, this method will return zero.   Any non-zero value indicates the size that all slots have the same size as the value returned by this method.

validRequestorForSendType:returnType:
- (id)**validRequestorForSendType:**(NSString*)*t_send* **returnType:**(NSString*)*t_return*

This method is called by the services system to update the services menu.   If the *delegate* responds to the `-tableScroll:validRequestorForSendType:returnType:` message, it is sent to the *delegate*.   If not, the *dataDelegate* is checked.   If neither object responds,   `-builtinValidRequestorForSendType:returnType:` is called. Override this method in your subclass if you need different behavior.   Returns the result of the called method.

See also: **-builtinValidRequestorForSendType:returnType:,**
            -tableScroll:validRequestorForSendType:returnType: (delegate method)

writeSelectionToPasteboard:types:
- (void)**writeSelectionToPasteboard:**(NSPasteboard*)*pboard* **type:**(NSString*)*type*

This method is responsible for writing data to the pasteboard.   If the *delegate* responds to `-tableScroll:writeSelectionToPasteboard:types:`, the message is sent to the *delegate*.   If not, the *dataDelegate* is tried.   If neither object responds to the message, the default method, `-builtinStringForPboardType:`, is called. Called from `-builtinWriteSelectionToPasteboard:types:`.   Override this method in your subclass if you want different behavior.

Methods Implemented by Cell Subclasses

tableScroll:retireAtRow:column:
-    (id)**tableScroll:**(MiscTableScroll*)*scroll* **retireAtRow:**(int)*row* **column:**(int)*col*

If neither the *delegate* nor the *dataDelegate* respond to `-tableScroll:retireCell:atRow:column:` then the MiscTableScroll tries sending this message to the cell itself to give it the opportunity to perform special handling when it is being retired from active use and returned to the cache.   Must return **self**, or a suitable replacement object for storage in the cache.

tableScroll:reviveAtRow:column:
-    (id)**tableScroll:**(MiscTableScroll*)*scroll* **reviveAtRow:**(int)*row* **column:**(int)*col*

If neither the *delegate* nor the *dataDelegate* respond to `-tableScroll:reviveCell:atRow:column:` then the MiscTableScroll tries sending this message to the cell itself to give it the opportunity to perform special handling when it is being brought into use for the first time, or is being retrieved from the cache for reuse.

Methods Implemented by the Delegate

tableScroll:abortEditAtRow:column:
- (void)**tableScroll:**(MiscTableScroll*)*scroll* **abortEditAtRow:**(int)*row* **column:**(int)*col*

Notifies the delegate that a cell editing session has been aborted.   This means that the normal `-control:textShouldEndEditing:` validation did not take place.

See also: **Cell Editing** (Introduction)

tableScroll:backgroundColorChangedTo:
- (void)**tableScroll:**(MiscTableScroll*)*scroll* **backgroundColorChangedTo:**(NSColor*)*newColor*

This message is sent to the *delegate* and then the *dataDelegate* when the MiscTableScroll receives a `-setBackgroundColor:` message that actually changes the background color.

tableScroll:border:slotDraggedFrom:to:
- (void)**tableScroll:**(MiscTableScroll*)*scroll* **border:**(MiscBorderType)*b* **slotDraggedFrom:**(int)*from_pos* **to:** (int)*to_pos*

Notifies the *delegate* whenever the user drags a slot to a new position.

tableScroll:border:slotPrototype:

- (void)**tableScroll:**(MiscTableScroll*)*scroll* **border:**(MiscBorderType)*b* **slotPrototype:**(int)*slot*

Sent to the *delegate* whenever the MiscTableScroll object needs the prototype cell for a column which has the `MISC_TABLE_CELL_CALLBACK` cell type. If the *delegate* does not respond to the message, the *dataDelegate* is tried.

See also: **-border:setSlot:cellType:**

tableScroll:border:slotResized:

- (void)**tableScroll:**(MiscTableScroll*)*scroll* **border:**(MiscBorderType)*b* **slotResized:**(int)*n*

Notifies the *delegate* whenever the user resizes a slot.

tableScroll:border:slotSortReversed:

- (void)**tableScroll:**(MiscTableScroll*)*scroll* **border:**(MiscBorderType)*b* **slotSortReversed:**(int)*n*

Notifies the *delegate* whenever the user reverses the sort direction of a slot.

tableScroll:border:slotTitle:

- (NSString*)**tableScroll:**(MiscTableScroll*)*scroll* **border:**(MiscBorderType)*b* **slotTitle:**(int)*slot*

Sent to the *delegate* whenever the MiscTableScroll object needs a title for a border which has the `MISC_DELEGATE_TITLE` title mode. If the *delegate* does not respond to the message, the *dataDelegate* is tried.

See also: **-border:setSlotTitleMode:**

tableScrollBufferCount:
    - (int)**tableScrollBufferCount:**(MiscTableScroll*)*scroll*

Gives the *delegate* and *dataDelegate* of a lazy-mode MiscTableScroll the opportunity to report the number of buffers used for each slot.   This information can be used to optimize-away string copying during sorting.   This is a micro-optimization for sorting lazy-mode tables based on string values.

Sorting retrieves the values of two cells from the same slot to compare them.   In eager mode, it is sufficient to perform the comparison in a manner similar to this:

    [[cell1 stringValue] isEqualToString:[cell2 stringValue]]

However, it is common for lazy-mode delegates to simply recycle a single cell to handle the `-tableScroll:cellAtRow:column:` message.   In that case, the first value is no longer valid when the second value is retrieved, so the MiscTableScroll object needs to copy the first value before retrieving the second value.   By default, the MiscTableScroll object makes the pessimistic (but safe) assumption that it needs to copy the first string value before retrieving the second string value.   Sophisticated delegates can eliminate this copy operation if they provide at least two buffers for each slot, or if they implement the string retrieval methods (`-tableScroll:stringValueAtRow:column:` and/or `-tableScroll:titleAtRow:column:`) in a way that makes it possible to retrieve a second string value from the same slot without invalidating the previous string value retrieved from that slot.

If the value returned is greater than or equal to two (2), the string values will not be copied.

tableScroll:canEdit:atRow:column:
- (BOOL)**tableScroll:**(MiscTableScroll*)*scroll* **canEdit:**(NSEvent*)*event* **atRow:**(int)*row* **column:**(int)*col*

Gives the *delegate* the opportunity to decide whether or not an editing session should be started for the cell at *row*, *col*. This method is invoked from both a -mouseDown: event, in which case *event* is the mouse-down event itself, and also from keyboard events in which case *event* is 0. This message is sent both on single-click events and multi-click events. It is the responsibility of the *delegate* to decide whether or not editing should begin. If the *delegate* returns YES, an editing session will begin for the cell. If the *delegate* returns NO, no editing session will begin. This message is tested with the *delegate*, *dataDelegate* and the cell itself. The first object that responds to the message decides the outcome. If none of those objects respond to the message, the default behavior will allow editing only on a double-click or keyboard event, and only if the cell is both editable and enabled.

tableScroll:canWritePboardType:
- (BOOL)**tableScroll:**(MiscTableScroll*)*scroll* **canWritePboardType:**(NSString*)*type*

If the *delegate* responds to this message, the *delegate* has the opportunity to select which datatypes will be written to the pasteboard. If the *delegate* does not respond, the *dataDelegate* is given the opportunity.

tableScroll:cellAtRow:column:
- (id)**tableScroll:**(MiscTableScroll*)*scroll* **cellAtRow:**(int)*row* **column:**(int)*col*

If the table scroll is in *lazy* mode this message is sent first to the *delegate* and then to the *dataDelegate* (if *delegate* does not respond) whenever the cell at *row*, *col* is needed.   You must implement this method in either the *delegate* or the *dataDelegate* whenever you use a MiscTableScroll in lazy mode.   The table scroll does not manage the cells for itself in lazy mode; the *delegate* or the *dataDelegate* must.

See also: ± setLazy:, ± isLazy

tableScroll:changeFont:to:
- (void)**tableScroll:**(MiscTableScroll*)*scroll* **changeFont:**(NSFont*)*oldFont* **to:**(NSFont*)*newFont*

This message is sent to the *delegate* and then the *dataDelegate* whenever a -changeFont: message is received and the new font is different than the current font.   The NSFontManager sends the -changeFont: message whenever the user changes the font using either the Font Panel or the Font menu.   This is distinguished from programmatic changes via the -setFont: method so that you can record user preferences.   This notification message is sent after the font change has been applied, but before the new font is displayed.

See also: -changeFont:, -setFont:, -tableScroll:fontChangedFrom:to: (delegate method)

tableScroll:didEdit:atRow:column:
- (void)**tableScroll:**(MiscTableScroll*)*scroll* **didEdit:**(BOOL)*changed* **atRow:**(int)*row* **column:**(int)*col*

Notifies the *delegate* that a cell-editing session terminated, and whether or not the value in the cell changed.   This is a notification-only message.   The *delegate* is not given any veto power at this point.   Veto power is available in the -control:textShouldEndEditing: (NSControl) delegate message.   This message is sent immediately after the -tableScroll:abortEdit:atRow:column: messages and the -tableScroll:setStringValue:atRow:column: messages, so this message is always sent in all cases when a cell editing session terminates.

See also: -tableScroll:willEditAtRow:column: (delegate method)**, Cell Editing** (Introduction)

tableScrollDidPrint:
  - (void)**tableScrollDidPrint:**(MiscTableScroll*)*scroll*

  Notifies the *delegate* that a printing session terminated.   This method is called from `-print:` after printing has completed.   This gives the *delegate* the opportunity to peform post-print cleanup, such as restoring global NSPrintInfo values.

See also: -tableScrollWillPrint: (delegate method)

tableScroll:doubleValueAtRow:column:
  - (double)**tableScroll:**(MiscTableScroll*)*scroll* **doubleValueAtRow:**(int)*row* **column:**(int)*col*

Lazy tables send this message to the *delegate* and then the *dataDelegate* to retrieve the value for -doubleValueAtRow:column:.   If the *delegate* or *dataDelegate* respond to this message, that value is returned.   If neither responds to the message, or if the table is not lazy, the cell is retrieved via -cellAtRow:column:.   If the cell responds to the -doubleValue message, that value is returned; otherwise zero is returned.   This method gives lazy tables the opportunity

to provide the information content of cells without the overhead of preparing and formatting a cell.   You should implement this method in your *delegate* or *dataDelegate* if you have any slots that contain `double` values.

See also: **-cellAtRow:column:, -doubleValueAtRow:column:, -isLazy, -setLazy:**

tableScroll:floatValueAtRow:column:
   - (float)**tableScroll:**(MiscTableScroll*)*scroll* **floatValueAtRow:**(int)*row* **column:**(int)*col*

Lazy tables send this message to the *delegate* and then the *dataDelegate* to retrieve the value for _{-floatValueAtRow:column:}. If the *delegate* or *dataDelegate* respond to this message, that value is returned.   If neither responds to the message, or if the table is not lazy, the cell is retrieved via _{-cellAtRow:column:}.   If the cell responds to the _{-floatValue} message, that value is returned; otherwise zero is returned.   This method gives lazy tables the opportunity to provide the information content of cells without the overhead of preparing and formatting a cell.   You should implement this method in your *delegate* or *dataDelegate* if you have any slots that contain `float` values.

See also: **-cellAtRow:column:, -floatValueAtRow:column:, -isLazy, -setLazy:**

tableScroll:fontChangedFrom:to:
   - (void)**tableScroll:**(MiscTableScroll*)*scroll* **fontChangedFrom:**(NSFont*)*oldFont* **to:**(NSFont*)*newFont*

This message is sent to the *delegate* and then the *dataDelegate* whenever a _{-setFont:} message is received and the new font is different than the current font.   This notification message is sent after all font changes, both user-initiated and programmatic.   This message is sent after the font change has been applied, but before the new font is displayed.

See also: **-changeFont:, -setFont:, -tableScroll:changeFont:to:** (delegate method)

tableScroll:getISearchColumn:
- (BOOL)**tableScroll:**(MiscTableScroll*)*scroll* **getISearchColumn:**(int*)*col*

First the *delegate* and then the *dataDelegate* is tested for response to this message whenever a keystroke is received that could start incremental searching.   If the *delegate* responds, the message is sent to the *delegate*.   Otherwise, if the *dataDelegate* responds, the message is sent to the *dataDelegate*.   Return YES if incremental searching should be enabled, and set **col* to the physical index of the column that the table is sorted on, otherwise return NO.   The delegates have the opportunity to decide whether or not incremental searching should be enabled, and indicate which column the table is sorted on.   If you want to enable incremental searching and you do not use the auto-sort facilities, then you must implement this method to tell the MiscTableScroll object which column the table is sorted on.   The table must be sorted in *col* order (ascending or descending).   -doIncrementalSearch:column: calls `[self border:MISC_COL_BORDER slotSortType:`*col*`]` to determine the sort-type.   The sort-type for *col* must be one of the string-based sort-types: `MISC_SORT_STRING_CASE_INSENSITIVE`, `MISC_SORT_STRING_CASE_SENSITIVE`, `MISC_SORT_TITLE_CASE_INSENSITIVE` or `MISC_SORT_TITLE_CASE_SENSITIVE`.   *Col* must not have a custom sort function.   You are responsible for ensuring that the table is sorted in *col* order.   -doIncrementalSearch:column: calls `[self border:MISC_COL_BORDER slotSortDirection:`*col*`]` to determine the sort-direction.   If the table is sorted in the other direction, use the complement of the column's physical index (~*col*).

See also: -doGetISearchColumn:, -doIncrementalSearch:column:, -getISearchColumn:, -incrementalSearch:

tableScroll:intValueAtRow:column:
- (int)**tableScroll:**(MiscTableScroll*)*scroll* **intValueAtRow:**(int)*row* **column:**(int)*col*

Lazy tables send this message to the *delegate* and then the *dataDelegate* to retrieve the value for -intValueAtRow:column:. If the *delegate* or *dataDelegate* respond to this message, that value is returned.   If neither responds to the message, or if the table is not lazy, the cell is retrieved via -cellAtRow:column:.   If the cell responds to the -intValue message, that value is returned; otherwise zero is returned.   This method gives lazy tables the opportunity to provide the information content of cells without the overhead of preparing and formatting a cell.   You should implement this method in your *delegate* or *dataDelegate* if you have any slots that contain int values.

See also: **-cellAtRow:column:, -intValueAtRow:column:, -isLazy, -setLazy:**

tableScroll:readSelectionFromPasteboard:
- (BOOL)**tableScroll:**(MiscTableScroll*)*scroll* **readSelectionFromPasteboard:**(id)*pboard*

If the *delegate* responds to this message, the *delegate* has the opportunity to take over the process of reading data from the pasteboard.   If the *delegate* does not respond to this message, the dataDelegate is tried.   The delegate should return YES if the data was successfully read from the pasteboard, else it should return NO.

See also: **-readSelectionFromPasteboard:**

tableScrollRegisterServicesTypes:
- (void)**tableScrollRegisterServicesTypes:**(MiscTableScroll*)*scroll*

If the *delegate* responds to this message, the delegate has the opportunity to register different datatypes with the services system.   If the *delegate* does not respond, the *dataDelegate* is tried.

See also: **-registerServicesTypes:**

tableScroll:retireCell:atRow:column:

- (id)**tableScroll:**(MiscTableScroll*)*scroll* **retireCell:**(id)*cell* **atRow:**(int)*row* **column:**(int)*col*

If the *delegate* responds to this message, the *delegate* has the opportunity to perform special handling of cells that are being retired to the cache.   If the *delegate* does not respond, the *dataDelegate* is tried.   If the *dataDelegate* does not respond either, the cell itself is tried (with -tableScroll:retireAtRow:column:).   Must return *cell*, or a suitable replacement object for storage in the cache.

See also: -retireCell:atRow:column:

tableScroll:reviveCell:atRow:column:

- (id)**tableScroll:**(MiscTableScroll*)*scroll* **reviveCell:**(id)*cell* **atRow:**(int)*row* **column:**(int)*col*

If the *delegate* responds to this message, the *delegate* has the opportunity to perform special handling of cells that are being brought into use for the first time, or are being retrieved from the cache for reuse.   If the *delegate* does not respond, the *dataDelegate* is tried.   If the *dataDelegate* does not respond either, the cell itself is tried (with -tableScroll:reviveAtRow:column:).

See also: -reviveCell:atRow:column:

tableScroll:selectedBackgroundColorChangedTo:

- (void)**tableScroll:**(MiscTableScroll*)*scroll* **selectedBackgroundColorChangedTo:**(NSColor*)*newColor*

This message is sent to the *delegate* and then the *dataDelegate* when the MiscTableScroll receives a -setSelectedBackgroundColor: message that actually changes the background color.

tableScroll:selectedTextColorChangedTo:
  - (void)**tableScroll:**(MiscTableScroll*)*scroll* **selectedTextColorChangedTo:**(NSColor*)*newColor*

This message is sent to the *delegate* and then the *dataDelegate* when the MiscTableScroll receives a -setSelectedTextColor: message that actually changes the background color.

tableScroll:setStringValue:atRow:column:
  - (BOOL)**tableScroll:**(MiscTableScroll*)*scroll* **setStringValue:**(NSString*)*s* **atRow:**(int)*row* **column:**(int)*col*

This message is sent from within -textDidEndEditing: when a cell editing session has successfully finished (not aborted) and the string value for the cell was actually changed.   This method provides the delegate with an after-the-fact veto option.   If the *delegate* returns NO, then the MiscTableScroll object assumes that the change was rejected, and that the cell retains its previous contents.   If the *delegate* returns YES, then the MiscTableScroll object assumes that the delegate stored the new string value into the appropriate cell, and that slot needs to have its sort position reevaluated. This method is optional for delegates of eager MiscTableScroll objects.   This method is mandatory for delegates of lazy MiscTableScroll objects that allow cell editing.   If the *delegate* implements this method, the *delegate* is responsible for setting the string value in the appropriate cell in the MiscTableScroll object.

See also: **Cell Editing** (Introduction)

tableScroll:stateAtRow:column:
   - (int)**tableScroll:**(MiscTableScroll*)*scroll* **stateAtRow:**(int)*row* **column:**(int)*col*

Lazy tables send this message to the *delegate* and then the *dataDelegate* to retrieve the value for -stateAtRow:column:.   If the *delegate* or *dataDelegate* respond to this message, that value is returned.   If neither responds to the message, or if the table is not lazy, the cell is retrieved via -cellAtRow:column:.   If the cell responds to the -state message, that value is returned; otherwise zero is returned.   This method gives lazy tables the opportunity to provide the information content of cells without the overhead of preparing and formatting a cell.   You should implement this method in your *delegate* or *dataDelegate* if you have any slots that contain state values.

See also: **-cellAtRow:column:, -stateAtRow:column:, -isLazy, -setLazy:, -state** (NSButtonCell)

tableScroll:stringValueAtRow:column:
   - (NSString*)**tableScroll:**(MiscTableScroll*)*scroll* **stringValueAtRow:**(int)*row* **column:**(int)*col*

Lazy tables send this message to the *delegate* and then the *dataDelegate* to retrieve the value for -stringValueAtRow:column:. If the *delegate* or *dataDelegate* respond to this message, that value is returned.   If neither responds to the message, or if the table is not lazy, the cell is retrieved via -cellAtRow:column:.   If the cell responds to the -stringValue message, that value is returned; otherwise zero is returned.   This method gives lazy tables the opportunity to provide the information content of cells without the overhead of preparing and formatting a cell.   You should implement this method in your *delegate* or *dataDelegate* if you have any slots that contain string values.

See also: **-cellAtRow:column:, -stringValueAtRow:column:, -isLazy, -setLazy:**

tableScroll:tagAtRow:column:
   - (int)**tableScroll:**(MiscTableScroll*)*scroll* **tagAtRow:**(int)*row* **column:**(int)*col*

Lazy tables send this message to the *delegate* and then the *dataDelegate* to retrieve the value for -tagAtRow:column:.   If the *delegate* or *dataDelegate* respond to this message, that value is returned.   If neither responds to the message, or if the table is not lazy, the cell is retrieved via -cellAtRow:column:.   If the cell responds to the -tag message, that value is returned; otherwise zero is returned.   This method gives lazy tables the opportunity to provide the information content of cells without the overhead of preparing and formatting a cell.   You should implement this method in your *delegate* or *dataDelegate* if you have any slots that contain tag values.

See also: -cellAtRow:column:, -tagAtRow:column:, -isLazy, -setLazy:

tableScroll:textColorChangedTo:
   - (void)**tableScroll:**(MiscTableScroll*)*scroll* **textColorChangedTo:**(NSColor*)*newColor*

This message is sent to the *delegate* and then the *dataDelegate* when the MiscTableScroll receives a -setTextColor: message that actually changes the background color.

tableScroll:titleAtRow:column:
   - (NSString*)**tableScroll:**(MiscTableScroll*)*scroll* **titleAtRow:**(int)*row* **column:**(int)*col*

Lazy tables send this message to the *delegate* and then the *dataDelegate* to retrieve the value for -titleAtRow:column:.   If the *delegate* or *dataDelegate* respond to this message, that value is returned.   If neither responds to the message, or if the table is not lazy, the cell is retrieved via -cellAtRow:column:.   If the cell responds to the -title message, that value is

returned; otherwise zero is returned.   This method gives lazy tables the opportunity to provide the information content of cells without the overhead of preparing and formatting a cell.   You should implement this method in your *delegate* or *dataDelegate* if you have any slots that contain `title` values.

See also: **-cellAtRow:column:, -stringValueAtRow:column:, -isLazy, -setLazy:, -title** (NSButtonCell)

tableScroll:validRequestorForSendType:returnType:
- (id)**tableScroll:**(MiscTableScroll*)*scroll* **validRequestorForSendType:**(NSString*)*t_send* **returnType:** (NSString*)*t_return*

If the *delegate* responds to this message, the *delegate* has the opportunity to interact with the services system using different combinations of send and return types than the MiscTableScroll object alone normally does.   If the *delegate* does not respond, the *dataDelegate* is tried.

See also: **-validRequestorForSendType:returnType:**

tableScroll:willEditAtRow:column:
- (void)**tableScroll:**(MiscTableScroll*)*scroll* **willEditAtRow:**(int)*row* **column:**(int)*col*

Notifies the *delegate* that a cell editing session is about to start.   This is a notification-only message.   There is no veto option associated with this message.   A veto option is provided with the -tableScroll:canEdit:atRow:column: message. This message, and its counterpart, -tableScroll:didEdit:atRow:column:, bracket cell editing sessions for delegates that want to perform extra operations before and after cell editing sessions without interfering with any of the begin/end decisions. This message is offered to the *delegate*, and then the *dataDelegate* if the *delegate* does not respond.

See also: **-tableScroll:didEdit:atRow:column:, Cell Editing** (Introduction)

tableScrollWillPrint:
- (void)**tableScrollWillPrint:**(MiscTableScroll*)*scroll*

Notifies the *delegate* that a printing session is about to start.   This is a notification-only message.   There is no veto option associated with this message.   This message is sent from `-print:` before the NSPrintPanel is presented to the user.   This message, and its counterpart, -tableScrollDidPrint:, bracket printing sessions for delegates that want to perform extra operations before and after printing sessions.   This method is useful for pre-print setup operations like overriding global NSPrintInfo settings.   This message is offered to the *delegate*, and then the *dataDelegate* if the delegate does not respond.

See also: **-tableScrollDidPrint:** (delegate method)

tableScroll:writeSelectionToPasteboard:types:
- (BOOL)**tableScroll:**(MiscTableScroll*)*scroll* **writeSelectionToPasteboard:**(NSPasteboard*)*pboard* **types:** (NSArray*)*types*

If the *delegate* responds to this message, the *delegate* has the opportunity to completely take over the writing of data to the pasteboard.   If the *delegate* does not respond, the *dataDelegate* is tried.

See also: **-writeSelectionToPasteboard:types:**

Delegate Methods For Dragging Source Operations

tableScroll:allowDragOperationAtRow:column:
- (BOOL)**tableScroll:**(MiscTableScroll*)*scroll* **allowDragOperationAtRow:**(int)*row* **column:**(int)*col*

This message is sent to the *delegate* or the *dataDelegate* if necessary by the MiscTableScroll to determine whether or not dragging an image from a cell should be allowed.   The appropriate delegate should return YES if dragging the image from the cell at *row* and *col* is allowed, or NO if it is not.

tableScroll:draggingSourceOperationMaskForLocal:
- (NSDraggingOperation)**tableScroll:**(MiscTableScroll*)*scroll* **draggingSourceOperationMaskForLocal:**
    (BOOL)*isLocal*

This message is sent to the *delegate* or the *dataDelegate* if necessary in order to give the delegate a chance to respond to NSDraggingSource's -draggingSourceOperationMaskForLocal: method.   If the delegate does not implement this method, then MiscTableScroll returns NSDragOperationGeneric by default.

tableScrollIgnoreModifierKeysWhileDragging:
- (BOOL)**tableScrollIgnoreModifierKeysWhileDragging:**(MiscTableScroll*)*scroll*

This message is sent to the *delegate* or the *dataDelegate* if necessary in order to give the delegate a chance to respond to NSDraggingSource's -ignoreModifierKeysWhileDragging method.   If the delegate does not implement this method, then MiscTableScroll returns YES by default.

tableScroll:imageForDragOperationAtRow:column:
- (NSImage*)**tableScroll:**(MiscTableScroll*)*scroll* **imageForDragOperationAtRow:**(int)*row* **column:**(int)*col*

This message is sent to the *delegate* or the *dataDelegate* if necessary to give the delegate a chance to provide the image used   for the dragging operation.   This method is required for non-image cells.   It is optional for image cells, in which case, if the *delegate* does not implement it or if it returns 0, then the image stored in the cell is used for dragging instead.

tableScroll:preparePasteboard:forDragOperationAtRow:column:
- (void)**tableScroll:**(MiscTableScroll*)*scroll* **preparePasteboard:**(NSPasteboard*)*pb* **forDragOperationAtRow:** (int)*row* **column:**(int)*col*

This message is sent to the *delegate* or the *dataDelegate* if necessary in order to have the NSPasteboard *pb* prepared for the dragging operation.   The appropriate delegate *must* send a -declareTypes:num:owner: message to the NSPasteboard followed by appropriate -set…:forType: or -write… messages if needed.   Extra precaution should be taken when declaring a non-null pasteboard *owner* as discussed in the **Image Dragging** section of this document.

tableScroll:shouldDelayWindowOrderingForEvent:
- (BOOL)**tableScroll:**(MiscTableScroll*)*scroll* **shouldDelayWindowOrderingForEvent:**(NSEvent*)*event*

This message is sent to the *delegate* or the *dataDelegate* if necessary in order to give the delegate a chance to respond to NSView's -shouldDelayWindowOrderingForEvent: method.   This method is only invoked if the delegate's -tableScroll:allowDragOperationAtRow:column: method indicates that the cell under the mouse is a potential a dragging source.   If the delegate does not implement this method, then this method returns YES by default.

Constants and Defined Types

```
typedef int MiscPixels;
typedef int MiscCoord_V;   // Visual coordinate.
typedef int MiscCoord_P;   // Physical coordinate.

#define MISC_MIN_PIXELS_SIZE ((MiscPixels) 10)
#define MISC_MAX_PIXELS_SIZE ((MiscPixels) 0x7FFF0000)

typedef enum
    {
    MISC_COL_BORDER,
    MISC_ROW_BORDER
    } MiscBorderType;

#define MISC_MAX_BORDER     MISC_ROW_BORDER
#defineMISC_OTHER_BORDER(B) \
    (B == MISC_ROW_BORDER ? MISC_COL_BORDER : MISC_ROW_BORDER)


typedef struct
    {
    NSSize      page_size;      // [NSPrintInfo paperSize]
    NSRect      print_rect;     // MiscTableView rect.
    MiscCoord_V first_print_row;// one's comp if started on prev page.
    MiscCoord_V last_print_row; // one's comp if ends on later page.
    MiscCoord_V first_print_col;// one's comp if started on prev page.
```

```c
    MiscCoord_V last_print_col; // one's comp if ends on later page.
    int         print_page;     // 1 <= print_page <= num_print_pages
    int         print_row;      // 1 <= print_row <= num_print_rows
    int         print_col;      // 1 <= print_col <= num_print_cols
    int         num_print_pages;
    int         num_print_rows;
    int         num_print_cols;
    double      scale_factor;
    BOOL        is_scaled;
    } MiscTablePrintInfo;


typedef enum
    {
    MISC_NO_TITLE,          // No titles on row/col cells.
    MISC_NUMBER_TITLE,      // Titles are sequential numbers.
    MISC_ALPHA_TITLE,       // Titles are sequential alphabetics...
    MISC_CUSTOM_TITLE,      // Titles are user-supplied strings...
    MISC_DELEGATE_TITLE     // Ask the delegate for titles.
    } MiscTableTitleMode;

#define MISC_MAX_TITLE      MISC_DELEGATE_TITLE


typedef enum
    {
    MISC_LIST_MODE,
    MISC_RADIO_MODE,
    MISC_HIGHLIGHT_MODE
    } MiscSelectionMode;
```

```c
#define MISC_MAX_MODE       MISC_HIGHLIGHT_MODE


typedef enum
    {
    MISC_TABLE_CELL_TEXT,
    MISC_TABLE_CELL_IMAGE,
    MISC_TABLE_CELL_BUTTON,
    MISC_TABLE_CELL_CALLBACK
    } MiscTableCellStyle;

#define MISC_TABLE_CELL_MAXMISC_TABLE_CELL_CALLBACK


#define MISC_SIZING_SPRINGY_BIT (1 << 0) // Adjusts for global limits.
#define MISC_SIZING_USER_BIT    (1 << 1) // User can resize.


typedef enum
    {
    MISC_NUSER_NSPRINGY_SIZING,
    MISC_NUSER_SPRINGY_SIZING,
    MISC_USER_NSPRINGY_SIZING,
    MISC_USER_SPRINGY_SIZING,
    } MiscTableSizing;

#define MISC_MAX_SIZING    MISC_USER_SPRINGY_SIZING
```

```c
typedef enum
    {
    MISC_SORT_ASCENDING,
    MISC_SORT_DESCENDING
    } MiscSortDirection;

#define MISC_SORT_DIR_MAX   MISC_SORT_DESCENDING
#defineMISC_OTHER_DIRECTION(D) \
    ((D) == MISC_SORT_DESCENDING ? \
    MISC_SORT_ASCENDING : MISC_SORT_DESCENDING)


typedef enum                            // Selector used to get data:
    {
    MISC_SORT_STRING_CASE_INSENSITIVE,  // -stringValue
    MISC_SORT_STRING_CASE_SENSITIVE,    // -stringValue
    MISC_SORT_INT,                      // -intValue
    MISC_SORT_UNSIGNED_INT,             // -intValue
    MISC_SORT_TAG,                      // -tag
    MISC_SORT_UNSIGNED_TAG,             // -tag
    MISC_SORT_FLOAT,                    // -floatValue
    MISC_SORT_DOUBLE,                   // -doubleValue
    MISC_SORT_SKIP,                     // Don't compare cells in this slot.
    MISC_SORT_TITLE_CASE_INSENSITIVE,   // -title
    MISC_SORT_TITLE_CASE_SENSITIVE,     // -title
    MISC_SORT_STATE,                    // -state
    MISC_SORT_UNSIGNED_STATE,           // -state
    } MiscSortType;

#define MISC_SORT_TYPE_MAX    MISC_SORT_UNSIGNED_STATE
```

```c
#define MISC_SORT_CUSTOM        ((MiscSortType)(int(MISC_SORT_TYPE_MAX) + 1))


@class MiscTableScroll;

typedef struct MiscEntrySortInfo MiscEntrySortInfo;
typedef struct MiscSlotSortInfo MiscSlotSortInfo;

typedef int (*MiscCompareEntryFunc)
            ( int r1, int c1, int r2, int c2,
              MiscEntrySortInfo const* entry_info,
              MiscSlotSortInfo* sort_info );

typedef int (*MiscCompareSlotFunc)
            ( int slot1, int slot2, MiscSlotSortInfo* );

extern int MiscDefaultCompareSlotFunc
            ( int slot1, int slot2, MiscSlotSortInfo* );

#define MISC_TS_TYPE_AT( TYPE, NAME ) \
    typedef TYPE (*MISC_TS_##NAME##_AT)(id,SEL,id,int r,int c, ...);

MISC_TS_TYPE_AT( int, INT )             // MISC_TS_INT_AT
MISC_TS_TYPE_AT( float, FLOAT )         // MISC_TS_FLOAT_AT
MISC_TS_TYPE_AT( double, DOUBLE )       // MISC_TS_DOUBLE_AT
MISC_TS_TYPE_AT( char const*, STRING ) // MISC_TS_STRING_AT

typedef union
    {
    MISC_TS_INT_AT      i;
```

```
    MISC_TS_FLOAT_AT     f;
    MISC_TS_DOUBLE_AT    d;
    MISC_TS_STRING_AT    s;
    } MISC_TS_VAL_AT_FUNC;

#define MISC_TS_TYPE_VAL( TYPE, NAME ) \
typedef TYPE (*MISC_TS_##NAME##_VAL)(id,SEL);

MISC_TS_TYPE_VAL( int, INT )            // MISC_TS_INT_VAL
MISC_TS_TYPE_VAL( float, FLOAT )        // MISC_TS_FLOAT_VAL
MISC_TS_TYPE_VAL( double, DOUBLE )      // MISC_TS_DOUBLE_VAL
MISC_TS_TYPE_VAL( char const*, STRING ) // MISC_TS_STRING_VAL

typedef union
    {
    MISC_TS_INT_VAL      i;
    MISC_TS_FLOAT_VAL    f;
    MISC_TS_DOUBLE_VAL   d;
    MISC_TS_STRING_VAL   s;
    } MISC_TS_VAL_FUNC;


// *** WARNING ***
// The sizes of these structures are likely to change between versions.
// *** WARNING ***

struct MiscEntrySortInfo
    {
    int slot;
    int ascending;
```

```
    MISC_TS_VAL_AT_FUNC value_func;
    id  value_target;
    SEL value_sel;
    id  value_obj;
    IMP cell_at_func;
    id  cell_class;
    SEL cell_sel;
    MISC_TS_VAL_FUNC cell_func;
    MiscSortType sort_type;
    MiscCompareEntryFunc compare_func;
    };


struct MiscSlotSortInfo
    {
    MiscTableScroll* table_scroll;
    NSZone* zone;
    MiscBorderType border_type;
    int num_entries;
    MiscEntrySortInfo const* entry_info;
    BOOL need_copy;
    };
```