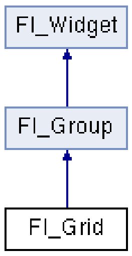
Fl_Grid Class Reference

Fl_Grid is a container (layout) widget with multiple columns and rows. More...

```
#include <Fl_Grid.H>
```

Inheritance diagram for Fl_Grid:



Classes

class	**Cell**

Public Member Functions

Fl_Grid::Cell *	**cell** (Fl_Widget *widget) Get the grid cell of widget `widget`. More...
Fl_Grid::Cell *	**cell** (int row, int col) Get the grid cell of row `row` and column `col`. More...
virtual void	**clear_layout** () Reset the layout w/o removing widgets. More...
void	**col_gap** (const int *value, size_t size)
void	**col_gap** (int col, int value)
void	**col_weight** (const int *value, size_t size)
void	**col_weight** (int col, int value) Set the weight of a column. More...
void	**col_width** (const int *value, size_t size) Set minimal widths of more than one column. More...
void	**col_width** (int col, int value) Set the minimal width of a column. More...
	Fl_Grid (int X, int Y, int W, int H, const char *L=0)
virtual void	**gap** (int row_gap, int col_gap=-1) Set default gaps for rows and columns. More...
virtual void	**layout** () Calculate the grid layout and resize and position all widgets. More...
virtual void	**layout** (int rows, int cols, int margin=-1, int gap=-1) Set the basic layout parameters of the Fl_Grid widget. More...
virtual void	**margin** (int left, int top=-1, int right=-1, int bottom=-1) Set all margins (left, top, right, bottom). More...
virtual void	**resize** (int X, int Y, int W, int H) Recalculate the layout and position and resize all widgets. More...
void	**row_gap** (const int *value, size_t size)
void	**row_gap** (int row, int value)

void	**row_height** (const int *value, size_t **size**)
void	**row_height** (int row, int value)
void	**row_weight** (const int *value, size_t **size**)
void	**row_weight** (int row, int value)
Cell *	**widget** (**Fl_Widget** *W, int row, int col, **Fl_Align align**=FL_GRID_FILL) Assign a widget to a grid cell and set its alignment. More...
Cell *	**widget** (**Fl_Widget** *W, int row, int col, int rowspan, int colspan, **Fl_Align align**=FL_GRID_FILL) Assign a widget to a grid cell and set cell spanning and alignment. More...

▸ **Public Member Functions inherited from Fl_Group**

▸ **Public Member Functions inherited from Fl_Widget**

Public Attributes

Fl_Rect	**old_size**

Protected Member Functions

virtual void	**child_moved** (**Fl_Widget** ***w**, int from, int to) Receive notifications of widget removal and deletion. More...

▸ **Protected Member Functions inherited from Fl_Group**

▸ **Protected Member Functions inherited from Fl_Widget**

Additional Inherited Members

▸ **Static Public Member Functions inherited from Fl_Group**

▸ **Static Public Member Functions inherited from Fl_Widget**

▸ **Protected Types inherited from Fl_Widget**

flags possible values enumeration. More...

Detailed Description

Fl_Grid is a container (layout) widget with multiple columns and rows.

This container widget features very flexible layouts in columns and rows w/o "counting pixels". Widgets are assigned to grid cells (column, row) with their minimal sizes in `w()` and `h()`, the position `x()` and `y()` are ignored and can be (0, 0). Fl_Grid calculates widget positions and resizes the widgets to fit into the grid. It is possible to create a single row or column of widgets with Fl_Grid.

You should design your grid with the smallest possible size of all widgets in mind. Fl_Grid will automatically assign **additional** space to cells according to some rules (described later) when resizing the Fl_Grid widget.

Hint: You should set a minimum window size to make sure the Fl_Grid is never resized below its minimal sizes. Resizing below the given widget sizes results in undefined behavior.

Fl_Grid and other container widgets (e.g. Fl_Group) can be nested. One main advantage of this usage is that widget coordinates in embedded Fl_Group widgets are relative to the group and will be positioned as expected.

Fl_Grid child widgets are handled by its base class Fl_Group but Fl_Grid stores additional data corresponding to each widget in internal grid cells.

To be continued ...

Todo:

Add Fl_Grid code example and screenshot.

Member Function Documentation

◆ cell() [1/2]

Fl_Grid::Cell * Fl_Grid::cell (**Fl_Widget** * widget)

Get the grid cell of widget `widget`.

The pointer to the cell can be used for further assignment of properties like alignment etc.

Note

See **Fl_Grid::cell(int row, int col)** for details about and the validity of cell pointers.

Parameters

`[in]`	**widget**	widget whose cell is requested

Return values

NULL	if `widget` is not assigned

◆ cell() [2/2]

```
Fl_Grid::Cell * Fl_Grid::cell ( int  row,
                                 int  col
                               )
```

Get the grid cell of row `row` and column `col`.

Widgets and other attributes are organized in cells (**Fl_Grid::Cell**).

This cell is an opaque structure (class) with some public methods. **Don't** assume anything about grid cell sizes and ordering in memory. These are implementation details that can be changed without notice. The public declaration of **Fl_Grid::Cell** does not necessarily represent the whole size of an internal cell, thus pointer arithmetic on cells **does not** work.

The validity of an **Fl_Grid::Cell** pointer is limited. It will definitely be invalidated when the overall grid layout is changed, for instance by calling layout(int, int).

Adding new cells beyond the current layout limits will also invalidate cell pointers but this is not (yet) implemented. Attempts to assign widgets to out-of-bounds cells are ignored.

A well-defined usage of cell pointers is only to set one or more properties like widget alignment of a cell after you retrieve the pointer. Don't store cell pointers in your program for later reference.

Parameters

`[in]`	**row**	row index
`[in]`	**col**	column index

Returns

pointer to cell

Return values

NULL	if `row` or `col` is out of bounds

◆ child_moved()

```
void Fl_Grid::child_moved ( Fl_Widget *  w,
                            int          from,
                            int          to
                          )                          protected  virtual
```

Receive notifications of widget removal and deletion.

Note

Work in progress, depends on unpublished work! See GitHub fork Albrecht-S/fltk, branch **Fl_Grid**.

Note: if (to < 0) the widget has been removed!

Reimplemented from **Fl_Group**.

◆ clear_layout()

void Fl_Grid::clear_layout ( ) virtual

Reset the layout w/o removing widgets.

Removes all cells and sets rows and cols to zero. Existing widgets are kept as children of the **Fl_Group** (base class) but are hidden.

This method should be rarely used. You may want to call **Fl_Grid::clear()** to remove all widgets and reset the layout to zero rows and columns.

You must call layout(int rows, int cols, ...) to set a new layout, allocate new cells, and assign widgets to new cells.

◆ col_weight()

```
void Fl_Grid::col_weight ( int col,
                           int value
                         )
```

Set the weight of a column.

Column and row weights are used to distribute additional space when the grid is resized beyond its defined (minimal) size. All weight values are relative and can be chosen freely. Suggested weights are in the range {0 .. 100}, 0 (zero) disables resizing of the column.

How does it work?

Whenever additional space (say: SPACE in pixels) is to be distributed to a set of columns the weights of all columns are added to a value SUM, then every single column width is increased by the value (in pseudo code):

```
col.width += SPACE * col.weight / SUM
```

Resulting pixel values are rounded to the next integer and rounding differences are added to or subtracted from the column with the highest weight. If more columns have the same weight one of them is chosen.

Note

If none of the columns considered for resizing have weights > 0 then **Fl_Grid** assigns the remaining space to an arbitrary column or to all considered columns evenly. This is implementation defined and can be changed without notice. You can avoid this situation by designing your grid with sensible sizes and weights.

Parameters

[in]	**col**	column number (counting from 0)
[in]	**value**	weight, must be >= 0

◆ col_width() [1/2]

```
void Fl_Grid::col_width ( const int *  value,
                          size_t       size
                        )
```

Set minimal widths of more than one column.

The values are taken from the array `value` and assigned sequentially to columns, starting from column 0. If the array `size` is too large extraneous values are ignored.

Negative values in the `array` are not assigned to their columns, i.e. the existing value for the corresponding column is not changed.

Example:

```
int widths[] = { 0, 0, 50, -1, -1, 50, 0 };
grid->col_width(widths, 7);
```

Parameters

`[in]`	**value**	an array of column widths
`[in]`	**size**	the size of the array (number of values)

◆ col_width() [2/2]

```
void Fl_Grid::col_width ( int  col,
                          int  value
                        )
```

Set the minimal width of a column.

Column widths are calculated by using the maximum of all widget widths in that column and the given column width. After calculating the width additional space is added when resizing according to the `weight` of the column.

You can set one or more column widths in one call by using **Fl_Grid::col_width(const int *value, size_t size)**.

Parameters

`[in]`	**col**	column number (counting from 0)
`[in]`	**value**	minimal column width, must be >= 0

See also

Fl_Grid::col_width(const int *value, size_t size)

◆ gap()

```
void Fl_Grid::gap ( int  row_gap,
                    int  col_gap = -1
                  )
```
virtual

Set default gaps for rows and columns.

All gaps are measured in pixels below the rows or right of the columns.

The last row and column don't have a gap, i.e. the gap for these are ignored. You can use a right or bottom margin instead.

You have to specify at least one argument, `col_gap` is optional. If you don't specify an argument or use a negative value (e.g. -1) then that margin is not affected.

You can also initialize the default gaps with **layout(int, int, int, int)**.

Parameters

`[in]`	**row_gap**	default gap for all rows
`[in]`	**col_gap**	default gap for all columns

See also

Fl_Grid::layout(int rows, int cols, int margin, int gap)

◆ layout() `[1/2]`

```
void Fl_Grid::layout ( )
```
virtual

Calculate the grid layout and resize and position all widgets.

This is called automatically when the **Fl_Grid** is resized. You need to call it once after you added widgets or moved widgets between cells.

Calling it once after all modfications are completed is enough.

See also

Fl_Grid::layout(int rows, int cols, int margin, int gap)

◆ layout() `[2/2]`

```
void Fl_Grid::layout ( int  rows,
                       int  cols,
                       int  margin = -1,
                       int  gap = -1
                     )
```
virtual

Set the basic layout parameters of the Fl_Grid widget.

You need to specify at least `rows` and `cols` to define a layout before you can add widgets to the grid.

Parameters `margin` and `gap` are optional.

You can call **layout(int rows, int cols, int margin, int gap)** again to change the layout but this is inefficient since all cells are reallocated if the layout changed.

Calling this with the same values of `rows` and `cols` is fast and can be used to change `margin` and `gap` w/o reallocating the cells.

`margin` sets all margins (left, top, right, bottom) to the same value. Negative values (e.g. -1) don't change the established margins. The default value set by the constructor is 0.

`gap` sets row and column gaps to the same value. Negative values (e.g. -1) do not affect the established gaps. The default value set by the constructor is 0.

After you added all widgets you must call **layout()** once without arguments to calculate the actual layout and to position and resize all widgets.

Parameters

`[in]`	**rows**	number of rows
`[in]`	**cols**	number of columns
`[in]`	**margin**	margin in pixels inside the Fl_Grid's border
`[in]`	**gap**	gap in pixels between cells

See also

Fl_Grid::layout()

◆ margin()

```
void Fl_Grid::margin ( int  left,
                       int  top = -1,
                       int  right = -1,
                       int  bottom = -1
                     )
```

virtual

Set all margins (left, top, right, bottom).

All margins are measured in pixels inside the box borders. You need to specify at least one argument, all other arguments are optional. If you don't specify an argument or use a negative value (e.g. -1) then that margin is not affected.

Parameters

`[in]`	**left**	left margin
`[in]`	**top**	top margin
`[in]`	**right**	right margin
`[in]`	**bottom**	margin

◆ resize()

```
void Fl_Grid::resize ( int  X,
                       int  Y,
                       int  W,
                       int  H
                     )
```

virtual

Recalculate the layout and position and resize all widgets.

This method overrides **Fl_Group::resize()** and calculates all positions and sizes of its children according to its own rules.

Parameters

`[in]`	**X,Y**	new widget position
`[in]`	**W,H**	new widget size

Reimplemented from **Fl_Widget**.

◆ widget() `[1/2]`

```
Fl_Grid::Cell * Fl_Grid::widget ( Fl_Widget *  wi,
                                  int          row,
                                  int          col,
                                  Fl_Align     align = FL_GRID_FILL
                                )
```

Assign a widget to a grid cell and set its alignment.

This sets row and column spanning to (1, 1).

For more information see Fl_Grid::widget(Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Align align)

Parameters

`[in]`	**wi**	widget to be assigned to the cell
`[in]`	**col**	column
`[in]`	**row**	row
`[in]`	**align**	widget alignment inside the cell

Returns

assigned cell

Return values

NULL if `row` or `col` is out of bounds

See also

Fl_Grid::widget(Fl_Widget *wi, int row, int col, int rowspan, int colspan, Fl_Align align)

◆ widget() `[2/2]`

```
Fl_Grid::Cell * Fl_Grid::widget ( Fl_Widget *  wi,
                                  int          row,
                                  int          col,
                                  int          rowspan,
                                  int          colspan,
                                  Fl_Align     align = FL_GRID_FILL
                                )
```

Assign a widget to a grid cell and set cell spanning and alignment.

Default alignment is `FL_GRID_FILL` which stretches the widget in horizontal and vertical directions to fill the whole cell(s) given by `colspan` and `rowspan`.

You can use this method to move a widget from one cell to another; it is automatically removed from its old cell. If the new cell is already assigned to another widget that widget is deassigned but kept as a child of the group.

Parameters

`[in]`	**wi**	widget to be assigned to the cell
`[in]`	**col**	column
`[in]`	**row**	row
`[in]`	**colspan**	horizontal span in cells, default 1
`[in]`	**rowspan**	vertical span in cells, default 1
`[in]`	**align**	widget alignment inside the cell

Returns

assigned cell

Return values

NULL if `row` or `col` is out of bounds

The documentation for this class was generated from the following files:

- **Fl_Grid.H**
- **Fl_Grid.cxx**