| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/GridBagLayout.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/awt/GridBagConstraints.html)   [**NEXT CLASS**](http://docs.google.com/java/awt/GridBagLayoutInfo.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/GridBagLayout.html)    [**NO FRAMES**](http://docs.google.com/GridBagLayout.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#z337ya) | [METHOD](#1y810tw) |

## **java.awt**

Class GridBagLayout

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **java.awt.GridBagLayout**

**All Implemented Interfaces:** [LayoutManager](http://docs.google.com/java/awt/LayoutManager.html), [LayoutManager2](http://docs.google.com/java/awt/LayoutManager2.html), [Serializable](http://docs.google.com/java/io/Serializable.html)

public class **GridBagLayout**extends [Object](http://docs.google.com/java/lang/Object.html)implements [LayoutManager2](http://docs.google.com/java/awt/LayoutManager2.html), [Serializable](http://docs.google.com/java/io/Serializable.html)

The GridBagLayout class is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its *display area*.

Each component managed by a GridBagLayout is associated with an instance of [GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html). The constraints object specifies where a component's display area should be located on the grid and how the component should be positioned within its display area. In addition to its constraints object, the GridBagLayout also considers each component's minimum and preferred sizes in order to determine a component's size.

The overall orientation of the grid depends on the container's [ComponentOrientation](http://docs.google.com/java/awt/ComponentOrientation.html) property. For horizontal left-to-right orientations, grid coordinate (0,0) is in the upper left corner of the container with x increasing to the right and y increasing downward. For horizontal right-to-left orientations, grid coordinate (0,0) is in the upper right corner of the container with x increasing to the left and y increasing downward.

To use a grid bag layout effectively, you must customize one or more of the GridBagConstraints objects that are associated with its components. You customize a GridBagConstraints object by setting one or more of its instance variables:

[GridBagConstraints.gridx](http://docs.google.com/java/awt/GridBagConstraints.html#gridx), [GridBagConstraints.gridy](http://docs.google.com/java/awt/GridBagConstraints.html#gridy) Specifies the cell containing the leading corner of the component's display area, where the cell at the origin of the grid has address gridx = 0, gridy = 0. For horizontal left-to-right layout, a component's leading corner is its upper left. For horizontal right-to-left layout, a component's leading corner is its upper right. Use GridBagConstraints.RELATIVE (the default value) to specify that the component be placed immediately following (along the x axis for gridx or the y axis for gridy) the component that was added to the container just before this component was added. [GridBagConstraints.gridwidth](http://docs.google.com/java/awt/GridBagConstraints.html#gridwidth), [GridBagConstraints.gridheight](http://docs.google.com/java/awt/GridBagConstraints.html#gridheight) Specifies the number of cells in a row (for gridwidth) or column (for gridheight) in the component's display area. The default value is 1. Use GridBagConstraints.REMAINDER to specify that the component's display area will be from gridx to the last cell in the row (for gridwidth) or from gridy to the last cell in the column (for gridheight). Use GridBagConstraints.RELATIVE to specify that the component's display area will be from gridx to the next to the last cell in its row (for gridwidth or from gridy to the next to the last cell in its column (for gridheight). [GridBagConstraints.fill](http://docs.google.com/java/awt/GridBagConstraints.html#fill) Used when the component's display area is larger than the component's requested size to determine whether (and how) to resize the component. Possible values are GridBagConstraints.NONE (the default), GridBagConstraints.HORIZONTAL (make the component wide enough to fill its display area horizontally, but don't change its height), GridBagConstraints.VERTICAL (make the component tall enough to fill its display area vertically, but don't change its width), and GridBagConstraints.BOTH (make the component fill its display area entirely). [GridBagConstraints.ipadx](http://docs.google.com/java/awt/GridBagConstraints.html#ipadx), [GridBagConstraints.ipady](http://docs.google.com/java/awt/GridBagConstraints.html#ipady) Specifies the component's internal padding within the layout, how much to add to the minimum size of the component. The width of the component will be at least its minimum width plus ipadx pixels. Similarly, the height of the component will be at least the minimum height plus ipady pixels. [GridBagConstraints.insets](http://docs.google.com/java/awt/GridBagConstraints.html#insets) Specifies the component's external padding, the minimum amount of space between the component and the edges of its display area. [GridBagConstraints.anchor](http://docs.google.com/java/awt/GridBagConstraints.html#anchor) Specifies where the component should be positioned in its display area. There are three kinds of possible values: absolute, orientation-relative, and baseline-relative Orientation relative values are interpreted relative to the container's ComponentOrientation property while absolute values are not. Baseline relative values are calculated relative to the baseline. Valid values are:

| Absolute Values | Orientation Relative Values | Baseline Relative Values |
| --- | --- | --- |
| * GridBagConstraints.NORTH * GridBagConstraints.SOUTH * GridBagConstraints.WEST * GridBagConstraints.EAST * GridBagConstraints.NORTHWEST * GridBagConstraints.NORTHEAST * GridBagConstraints.SOUTHWEST * GridBagConstraints.SOUTHEAST * GridBagConstraints.CENTER (the default) | * GridBagConstraints.PAGE\_START * GridBagConstraints.PAGE\_END * GridBagConstraints.LINE\_START * GridBagConstraints.LINE\_END * GridBagConstraints.FIRST\_LINE\_START * GridBagConstraints.FIRST\_LINE\_END * GridBagConstraints.LAST\_LINE\_START * GridBagConstraints.LAST\_LINE\_END | * GridBagConstraints.BASELINE * GridBagConstraints.BASELINE\_LEADING * GridBagConstraints.BASELINE\_TRAILING * GridBagConstraints.ABOVE\_BASELINE * GridBagConstraints.ABOVE\_BASELINE\_LEADING * GridBagConstraints.ABOVE\_BASELINE\_TRAILING * GridBagConstraints.BELOW\_BASELINE * GridBagConstraints.BELOW\_BASELINE\_LEADING * GridBagConstraints.BELOW\_BASELINE\_TRAILING |

[GridBagConstraints.weightx](http://docs.google.com/java/awt/GridBagConstraints.html#weightx), [GridBagConstraints.weighty](http://docs.google.com/java/awt/GridBagConstraints.html#weighty) Used to determine how to distribute space, which is important for specifying resizing behavior. Unless you specify a weight for at least one component in a row (weightx) and column (weighty), all the components clump together in the center of their container. This is because when the weight is zero (the default), the GridBagLayout object puts any extra space between its grid of cells and the edges of the container.

Each row may have a baseline; the baseline is determined by the components in that row that have a valid baseline and are aligned along the baseline (the component's anchor value is one of BASELINE, BASELINE\_LEADING or BASELINE\_TRAILING). If none of the components in the row has a valid baseline, the row does not have a baseline.

If a component spans rows it is aligned either to the baseline of the start row (if the baseline-resize behavior is CONSTANT\_ASCENT) or the end row (if the baseline-resize behavior is CONSTANT\_DESCENT). The row that the component is aligned to is called the *prevailing row*.

The following figure shows a baseline layout and includes a component that spans rows:

|  |
| --- |

This layout consists of three components:

* A panel that starts in row 0 and ends in row 1. The panel has a baseline-resize behavior of CONSTANT\_DESCENT and has an anchor of BASELINE. As the baseline-resize behavior is CONSTANT\_DESCENT the prevailing row for the panel is row 1.
* Two buttons, each with a baseline-resize behavior of CENTER\_OFFSET and an anchor of BASELINE.

Because the second button and the panel share the same prevailing row, they are both aligned along their baseline.

Components positioned using one of the baseline-relative values resize differently than when positioned using an absolute or orientation-relative value. How components change is dictated by how the baseline of the prevailing row changes. The baseline is anchored to the bottom of the display area if any components with the same prevailing row have a baseline-resize behavior of CONSTANT\_DESCENT, otherwise the baseline is anchored to the top of the display area. The following rules dictate the resize behavior:

* Resizable components positioned above the baseline can only grow as tall as the baseline. For example, if the baseline is at 100 and anchored at the top, a resizable component positioned above the baseline can never grow more than 100 units.
* Similarly, resizable components positioned below the baseline can only grow as high as the difference between the display height and the baseline.
* Resizable components positioned on the baseline with a baseline-resize behavior of OTHER are only resized if the baseline at the resized size fits within the display area. If the baseline is such that it does not fit within the display area the component is not resized.
* Components positioned on the baseline that do not have a baseline-resize behavior of OTHER can only grow as tall as display height - baseline + baseline of component.

If you position a component along the baseline, but the component does not have a valid baseline, it will be vertically centered in its space. Similarly if you have positioned a component relative to the baseline and none of the components in the row have a valid baseline the component is vertically centered.

The following figures show ten components (all buttons) managed by a grid bag layout. Figure 2 shows the layout for a horizontal, left-to-right container and Figure 3 shows the layout for a horizontal, right-to-left container.

|  |  |
| --- | --- |
| Figure 2: Horizontal, Left-to-Right | Figure 3: Horizontal, Right-to-Left |

Each of the ten components has the fill field of its associated GridBagConstraints object set to GridBagConstraints.BOTH. In addition, the components have the following non-default constraints:

* Button1, Button2, Button3: weightx = 1.0
* Button4: weightx = 1.0, gridwidth = GridBagConstraints.REMAINDER
* Button5: gridwidth = GridBagConstraints.REMAINDER
* Button6: gridwidth = GridBagConstraints.RELATIVE
* Button7: gridwidth = GridBagConstraints.REMAINDER
* Button8: gridheight = 2, weighty = 1.0
* Button9, Button 10: gridwidth = GridBagConstraints.REMAINDER

Here is the code that implements the example shown above:

import java.awt.\*;  
 import java.util.\*;  
 import java.applet.Applet;  
  
 public class GridBagEx1 extends Applet {  
  
 protected void makebutton(String name,  
 GridBagLayout gridbag,  
 GridBagConstraints c) {  
 Button button = new Button(name);  
 gridbag.setConstraints(button, c);  
 add(button);  
 }  
  
 public void init() {  
 GridBagLayout gridbag = new GridBagLayout();  
 GridBagConstraints c = new GridBagConstraints();  
  
 setFont(new Font("SansSerif", Font.PLAIN, 14));  
 setLayout(gridbag);  
  
 c.fill = GridBagConstraints.BOTH;  
 c.weightx = 1.0;  
 makebutton("Button1", gridbag, c);  
 makebutton("Button2", gridbag, c);  
 makebutton("Button3", gridbag, c);  
  
 c.gridwidth = GridBagConstraints.REMAINDER; //end row  
 makebutton("Button4", gridbag, c);  
  
 c.weightx = 0.0; //reset to the default  
 makebutton("Button5", gridbag, c); //another row  
  
 c.gridwidth = GridBagConstraints.RELATIVE; //next-to-last in row  
 makebutton("Button6", gridbag, c);  
  
 c.gridwidth = GridBagConstraints.REMAINDER; //end row  
 makebutton("Button7", gridbag, c);  
  
 c.gridwidth = 1; //reset to the default  
 c.gridheight = 2;  
 c.weighty = 1.0;  
 makebutton("Button8", gridbag, c);  
  
 c.weighty = 0.0; //reset to the default  
 c.gridwidth = GridBagConstraints.REMAINDER; //end row  
 c.gridheight = 1; //reset to the default  
 makebutton("Button9", gridbag, c);  
 makebutton("Button10", gridbag, c);  
  
 setSize(300, 100);  
 }  
  
 public static void main(String args[]) {  
 Frame f = new Frame("GridBag Layout Example");  
 GridBagEx1 ex1 = new GridBagEx1();  
  
 ex1.init();  
  
 f.add("Center", ex1);  
 f.pack();  
 f.setSize(f.getPreferredSize());  
 f.show();  
 }  
 }

**Since:** JDK1.0 **See Also:**[GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html), [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html), [ComponentOrientation](http://docs.google.com/java/awt/ComponentOrientation.html), [Serialized Form](http://docs.google.com/serialized-form.html#java.awt.GridBagLayout)

| **Field Summary** | |
| --- | --- |
| double[] | [**columnWeights**](http://docs.google.com/java/awt/GridBagLayout.html#columnWeights)            This field holds the overrides to the column weights. |
| int[] | [**columnWidths**](http://docs.google.com/java/awt/GridBagLayout.html#columnWidths)            This field holds the overrides to the column minimum width. |
| protected  [Hashtable](http://docs.google.com/java/util/Hashtable.html)<[Component](http://docs.google.com/java/awt/Component.html),[GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html)> | [**comptable**](http://docs.google.com/java/awt/GridBagLayout.html#comptable)            This hashtable maintains the association between a component and its gridbag constraints. |
| protected  [GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) | [**defaultConstraints**](http://docs.google.com/java/awt/GridBagLayout.html#defaultConstraints)            This field holds a gridbag constraints instance containing the default values, so if a component does not have gridbag constraints associated with it, then the component will be assigned a copy of the defaultConstraints. |
| protected  [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) | [**layoutInfo**](http://docs.google.com/java/awt/GridBagLayout.html#layoutInfo)            This field holds the layout information for the gridbag. |
| protected static int | [**MAXGRIDSIZE**](http://docs.google.com/java/awt/GridBagLayout.html#MAXGRIDSIZE)            This field is no longer used to reserve arrays and keeped for backward compatibility. |
| protected static int | [**MINSIZE**](http://docs.google.com/java/awt/GridBagLayout.html#MINSIZE)            The smallest grid that can be laid out by the grid bag layout. |
| protected static int | [**PREFERREDSIZE**](http://docs.google.com/java/awt/GridBagLayout.html#PREFERREDSIZE)            The preferred grid size that can be laid out by the grid bag layout. |
| int[] | [**rowHeights**](http://docs.google.com/java/awt/GridBagLayout.html#rowHeights)            This field holds the overrides to the row minimum heights. |
| double[] | [**rowWeights**](http://docs.google.com/java/awt/GridBagLayout.html#rowWeights)            This field holds the overrides to the row weights. |

| **Constructor Summary** | |
| --- | --- |
| [**GridBagLayout**](http://docs.google.com/java/awt/GridBagLayout.html#GridBagLayout())()            Creates a grid bag layout manager. |

| **Method Summary** | |
| --- | --- |
| void | [**addLayoutComponent**](http://docs.google.com/java/awt/GridBagLayout.html#addLayoutComponent(java.awt.Component,%20java.lang.Object))([Component](http://docs.google.com/java/awt/Component.html) comp, [Object](http://docs.google.com/java/lang/Object.html) constraints)            Adds the specified component to the layout, using the specified constraints object. |
| void | [**addLayoutComponent**](http://docs.google.com/java/awt/GridBagLayout.html#addLayoutComponent(java.lang.String,%20java.awt.Component))([String](http://docs.google.com/java/lang/String.html) name, [Component](http://docs.google.com/java/awt/Component.html) comp)            Has no effect, since this layout manager does not use a per-component string. |
| protected  void | [**adjustForGravity**](http://docs.google.com/java/awt/GridBagLayout.html#adjustForGravity(java.awt.GridBagConstraints,%20java.awt.Rectangle))([GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) constraints, [Rectangle](http://docs.google.com/java/awt/Rectangle.html) r)            Adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads. |
| protected  void | [**AdjustForGravity**](http://docs.google.com/java/awt/GridBagLayout.html#AdjustForGravity(java.awt.GridBagConstraints,%20java.awt.Rectangle))([GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) constraints, [Rectangle](http://docs.google.com/java/awt/Rectangle.html) r)            This method is obsolete and supplied for backwards compatability only; new code should call [adjustForGravity](http://docs.google.com/java/awt/GridBagLayout.html#adjustForGravity(java.awt.GridBagConstraints,%20java.awt.Rectangle)) instead. |
| protected  void | [**arrangeGrid**](http://docs.google.com/java/awt/GridBagLayout.html#arrangeGrid(java.awt.Container))([Container](http://docs.google.com/java/awt/Container.html) parent)            Lays out the grid. |
| protected  void | [**ArrangeGrid**](http://docs.google.com/java/awt/GridBagLayout.html#ArrangeGrid(java.awt.Container))([Container](http://docs.google.com/java/awt/Container.html) parent)            This method is obsolete and supplied for backwards compatability only; new code should call [arrangeGrid](http://docs.google.com/java/awt/GridBagLayout.html#arrangeGrid(java.awt.Container)) instead. |
| [GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) | [**getConstraints**](http://docs.google.com/java/awt/GridBagLayout.html#getConstraints(java.awt.Component))([Component](http://docs.google.com/java/awt/Component.html) comp)            Gets the constraints for the specified component. |
| float | [**getLayoutAlignmentX**](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutAlignmentX(java.awt.Container))([Container](http://docs.google.com/java/awt/Container.html) parent)            Returns the alignment along the x axis. |
| float | [**getLayoutAlignmentY**](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutAlignmentY(java.awt.Container))([Container](http://docs.google.com/java/awt/Container.html) parent)            Returns the alignment along the y axis. |
| int[][] | [**getLayoutDimensions**](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutDimensions())()            Determines column widths and row heights for the layout grid. |
| protected  [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) | [**getLayoutInfo**](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutInfo(java.awt.Container,%20int))([Container](http://docs.google.com/java/awt/Container.html) parent, int sizeflag)            Fills in an instance of GridBagLayoutInfo for the current set of managed children. |
| protected  [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) | [**GetLayoutInfo**](http://docs.google.com/java/awt/GridBagLayout.html#GetLayoutInfo(java.awt.Container,%20int))([Container](http://docs.google.com/java/awt/Container.html) parent, int sizeflag)            This method is obsolete and supplied for backwards compatability only; new code should call [getLayoutInfo](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutInfo(java.awt.Container,%20int)) instead. |
| [Point](http://docs.google.com/java/awt/Point.html) | [**getLayoutOrigin**](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutOrigin())()            Determines the origin of the layout area, in the graphics coordinate space of the target container. |
| double[][] | [**getLayoutWeights**](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutWeights())()            Determines the weights of the layout grid's columns and rows. |
| protected  [Dimension](http://docs.google.com/java/awt/Dimension.html) | [**getMinSize**](http://docs.google.com/java/awt/GridBagLayout.html#getMinSize(java.awt.Container,%20java.awt.GridBagLayoutInfo))([Container](http://docs.google.com/java/awt/Container.html) parent, [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) info)            Figures out the minimum size of the master based on the information from getLayoutInfo. |
| protected  [Dimension](http://docs.google.com/java/awt/Dimension.html) | [**GetMinSize**](http://docs.google.com/java/awt/GridBagLayout.html#GetMinSize(java.awt.Container,%20java.awt.GridBagLayoutInfo))([Container](http://docs.google.com/java/awt/Container.html) parent, [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) info)            This method is obsolete and supplied for backwards compatability only; new code should call [getMinSize](http://docs.google.com/java/awt/GridBagLayout.html#getMinSize(java.awt.Container,%20java.awt.GridBagLayoutInfo)) instead. |
| void | [**invalidateLayout**](http://docs.google.com/java/awt/GridBagLayout.html#invalidateLayout(java.awt.Container))([Container](http://docs.google.com/java/awt/Container.html) target)            Invalidates the layout, indicating that if the layout manager has cached information it should be discarded. |
| void | [**layoutContainer**](http://docs.google.com/java/awt/GridBagLayout.html#layoutContainer(java.awt.Container))([Container](http://docs.google.com/java/awt/Container.html) parent)            Lays out the specified container using this grid bag layout. |
| [Point](http://docs.google.com/java/awt/Point.html) | [**location**](http://docs.google.com/java/awt/GridBagLayout.html#location(int,%20int))(int x, int y)            Determines which cell in the layout grid contains the point specified by (x, y). |
| protected  [GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) | [**lookupConstraints**](http://docs.google.com/java/awt/GridBagLayout.html#lookupConstraints(java.awt.Component))([Component](http://docs.google.com/java/awt/Component.html) comp)            Retrieves the constraints for the specified component. |
| [Dimension](http://docs.google.com/java/awt/Dimension.html) | [**maximumLayoutSize**](http://docs.google.com/java/awt/GridBagLayout.html#maximumLayoutSize(java.awt.Container))([Container](http://docs.google.com/java/awt/Container.html) target)            Returns the maximum dimensions for this layout given the components in the specified target container. |
| [Dimension](http://docs.google.com/java/awt/Dimension.html) | [**minimumLayoutSize**](http://docs.google.com/java/awt/GridBagLayout.html#minimumLayoutSize(java.awt.Container))([Container](http://docs.google.com/java/awt/Container.html) parent)            Determines the minimum size of the parent container using this grid bag layout. |
| [Dimension](http://docs.google.com/java/awt/Dimension.html) | [**preferredLayoutSize**](http://docs.google.com/java/awt/GridBagLayout.html#preferredLayoutSize(java.awt.Container))([Container](http://docs.google.com/java/awt/Container.html) parent)            Determines the preferred size of the parent container using this grid bag layout. |
| void | [**removeLayoutComponent**](http://docs.google.com/java/awt/GridBagLayout.html#removeLayoutComponent(java.awt.Component))([Component](http://docs.google.com/java/awt/Component.html) comp)            Removes the specified component from this layout. |
| void | [**setConstraints**](http://docs.google.com/java/awt/GridBagLayout.html#setConstraints(java.awt.Component,%20java.awt.GridBagConstraints))([Component](http://docs.google.com/java/awt/Component.html) comp, [GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) constraints)            Sets the constraints for the specified component in this layout. |
| [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/awt/GridBagLayout.html#toString())()            Returns a string representation of this grid bag layout's values. |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [clone](http://docs.google.com/java/lang/Object.html#clone()), [equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [hashCode](http://docs.google.com/java/lang/Object.html#hashCode()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Field Detail** |
| --- |

### MAXGRIDSIZE

protected static final int **MAXGRIDSIZE**

This field is no longer used to reserve arrays and keeped for backward compatibility. Previously, this was the maximum number of grid positions (both horizontal and vertical) that could be laid out by the grid bag layout. Current implementation doesn't impose any limits on the size of a grid.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.GridBagLayout.MAXGRIDSIZE)

### MINSIZE

protected static final int **MINSIZE**

The smallest grid that can be laid out by the grid bag layout.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.GridBagLayout.MINSIZE)

### PREFERREDSIZE

protected static final int **PREFERREDSIZE**

The preferred grid size that can be laid out by the grid bag layout.

**See Also:**[Constant Field Values](http://docs.google.com/constant-values.html#java.awt.GridBagLayout.PREFERREDSIZE)

### comptable

protected [Hashtable](http://docs.google.com/java/util/Hashtable.html)<[Component](http://docs.google.com/java/awt/Component.html),[GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html)> **comptable**

This hashtable maintains the association between a component and its gridbag constraints. The Keys in comptable are the components and the values are the instances of GridBagConstraints.

**See Also:**[GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html)

### defaultConstraints

protected [GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) **defaultConstraints**

This field holds a gridbag constraints instance containing the default values, so if a component does not have gridbag constraints associated with it, then the component will be assigned a copy of the defaultConstraints.

**See Also:**[getConstraints(Component)](http://docs.google.com/java/awt/GridBagLayout.html#getConstraints(java.awt.Component)), [setConstraints(Component, GridBagConstraints)](http://docs.google.com/java/awt/GridBagLayout.html#setConstraints(java.awt.Component,%20java.awt.GridBagConstraints)), [lookupConstraints(Component)](http://docs.google.com/java/awt/GridBagLayout.html#lookupConstraints(java.awt.Component))

### layoutInfo

protected [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) **layoutInfo**

This field holds the layout information for the gridbag. The information in this field is based on the most recent validation of the gridbag. If layoutInfo is null this indicates that there are no components in the gridbag or if there are components, they have not yet been validated.

**See Also:**[getLayoutInfo(Container, int)](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutInfo(java.awt.Container,%20int))

### columnWidths

public int[] **columnWidths**

This field holds the overrides to the column minimum width. If this field is non-null the values are applied to the gridbag after all of the minimum columns widths have been calculated. If columnWidths has more elements than the number of columns, columns are added to the gridbag to match the number of elements in columnWidth.

**See Also:**[getLayoutDimensions()](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutDimensions())

### rowHeights

public int[] **rowHeights**

This field holds the overrides to the row minimum heights. If this field is non-null the values are applied to the gridbag after all of the minimum row heights have been calculated. If rowHeights has more elements than the number of rows, rowa are added to the gridbag to match the number of elements in rowHeights.

**See Also:**[getLayoutDimensions()](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutDimensions())

### columnWeights

public double[] **columnWeights**

This field holds the overrides to the column weights. If this field is non-null the values are applied to the gridbag after all of the columns weights have been calculated. If columnWeights[i] > weight for column i, then column i is assigned the weight in columnWeights[i]. If columnWeights has more elements than the number of columns, the excess elements are ignored - they do not cause more columns to be created.

### rowWeights

public double[] **rowWeights**

This field holds the overrides to the row weights. If this field is non-null the values are applied to the gridbag after all of the rows weights have been calculated. If rowWeights[i] > weight for row i, then row i is assigned the weight in rowWeights[i]. If rowWeights has more elements than the number of rows, the excess elements are ignored - they do not cause more rows to be created.

| **Constructor Detail** |
| --- |

### GridBagLayout

public **GridBagLayout**()

Creates a grid bag layout manager.

| **Method Detail** |
| --- |

### setConstraints

public void **setConstraints**([Component](http://docs.google.com/java/awt/Component.html) comp,  
 [GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) constraints)

Sets the constraints for the specified component in this layout.

**Parameters:**comp - the component to be modifiedconstraints - the constraints to be applied

### getConstraints

public [GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) **getConstraints**([Component](http://docs.google.com/java/awt/Component.html) comp)

Gets the constraints for the specified component. A copy of the actual GridBagConstraints object is returned.

**Parameters:**comp - the component to be queried **Returns:**the constraint for the specified component in this grid bag layout; a copy of the actual constraint object is returned

### lookupConstraints

protected [GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) **lookupConstraints**([Component](http://docs.google.com/java/awt/Component.html) comp)

Retrieves the constraints for the specified component. The return value is not a copy, but is the actual GridBagConstraints object used by the layout mechanism.

If comp is not in the GridBagLayout, a set of default GridBagConstraints are returned. A comp value of null is invalid and returns null.

**Parameters:**comp - the component to be queried **Returns:**the contraints for the specified component

### getLayoutOrigin

public [Point](http://docs.google.com/java/awt/Point.html) **getLayoutOrigin**()

Determines the origin of the layout area, in the graphics coordinate space of the target container. This value represents the pixel coordinates of the top-left corner of the layout area regardless of the ComponentOrientation value of the container. This is distinct from the grid origin given by the cell coordinates (0,0). Most applications do not call this method directly.

**Returns:**the graphics origin of the cell in the top-left corner of the layout grid**Since:** JDK1.1 **See Also:**[ComponentOrientation](http://docs.google.com/java/awt/ComponentOrientation.html)

### getLayoutDimensions

public int[][] **getLayoutDimensions**()

Determines column widths and row heights for the layout grid.

Most applications do not call this method directly.

**Returns:**an array of two arrays, containing the widths of the layout columns and the heights of the layout rows**Since:** JDK1.1

### getLayoutWeights

public double[][] **getLayoutWeights**()

Determines the weights of the layout grid's columns and rows. Weights are used to calculate how much a given column or row stretches beyond its preferred size, if the layout has extra room to fill.

Most applications do not call this method directly.

**Returns:**an array of two arrays, representing the horizontal weights of the layout columns and the vertical weights of the layout rows**Since:** JDK1.1

### location

public [Point](http://docs.google.com/java/awt/Point.html) **location**(int x,  
 int y)

Determines which cell in the layout grid contains the point specified by (x, y). Each cell is identified by its column index (ranging from 0 to the number of columns minus 1) and its row index (ranging from 0 to the number of rows minus 1).

If the (x, y) point lies outside the grid, the following rules are used. The column index is returned as zero if x lies to the left of the layout for a left-to-right container or to the right of the layout for a right-to-left container. The column index is returned as the number of columns if x lies to the right of the layout in a left-to-right container or to the left in a right-to-left container. The row index is returned as zero if y lies above the layout, and as the number of rows if y lies below the layout. The orientation of a container is determined by its ComponentOrientation property.

**Parameters:**x - the *x* coordinate of a pointy - the *y* coordinate of a point **Returns:**an ordered pair of indexes that indicate which cell in the layout grid contains the point (*x*, *y*).**Since:** JDK1.1 **See Also:**[ComponentOrientation](http://docs.google.com/java/awt/ComponentOrientation.html)

### addLayoutComponent

public void **addLayoutComponent**([String](http://docs.google.com/java/lang/String.html) name,  
 [Component](http://docs.google.com/java/awt/Component.html) comp)

Has no effect, since this layout manager does not use a per-component string.

**Specified by:**[addLayoutComponent](http://docs.google.com/java/awt/LayoutManager.html#addLayoutComponent(java.lang.String,%20java.awt.Component)) in interface [LayoutManager](http://docs.google.com/java/awt/LayoutManager.html) **Parameters:**name - the string to be associated with the componentcomp - the component to be added

### addLayoutComponent

public void **addLayoutComponent**([Component](http://docs.google.com/java/awt/Component.html) comp,  
 [Object](http://docs.google.com/java/lang/Object.html) constraints)

Adds the specified component to the layout, using the specified constraints object. Note that constraints are mutable and are, therefore, cloned when cached.

**Specified by:**[addLayoutComponent](http://docs.google.com/java/awt/LayoutManager2.html#addLayoutComponent(java.awt.Component,%20java.lang.Object)) in interface [LayoutManager2](http://docs.google.com/java/awt/LayoutManager2.html) **Parameters:**comp - the component to be addedconstraints - an object that determines how the component is added to the layout **Throws:** [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if constraints is not a GridBagConstraint

### removeLayoutComponent

public void **removeLayoutComponent**([Component](http://docs.google.com/java/awt/Component.html) comp)

Removes the specified component from this layout.

Most applications do not call this method directly.

**Specified by:**[removeLayoutComponent](http://docs.google.com/java/awt/LayoutManager.html#removeLayoutComponent(java.awt.Component)) in interface [LayoutManager](http://docs.google.com/java/awt/LayoutManager.html) **Parameters:**comp - the component to be removed.**See Also:**[Container.remove(java.awt.Component)](http://docs.google.com/java/awt/Container.html#remove(java.awt.Component)), [Container.removeAll()](http://docs.google.com/java/awt/Container.html#removeAll())

### preferredLayoutSize

public [Dimension](http://docs.google.com/java/awt/Dimension.html) **preferredLayoutSize**([Container](http://docs.google.com/java/awt/Container.html) parent)

Determines the preferred size of the parent container using this grid bag layout.

Most applications do not call this method directly.

**Specified by:**[preferredLayoutSize](http://docs.google.com/java/awt/LayoutManager.html#preferredLayoutSize(java.awt.Container)) in interface [LayoutManager](http://docs.google.com/java/awt/LayoutManager.html) **Parameters:**parent - the container in which to do the layout **Returns:**the preferred size of the parent container**See Also:**[Container.getPreferredSize()](http://docs.google.com/java/awt/Container.html#getPreferredSize())

### minimumLayoutSize

public [Dimension](http://docs.google.com/java/awt/Dimension.html) **minimumLayoutSize**([Container](http://docs.google.com/java/awt/Container.html) parent)

Determines the minimum size of the parent container using this grid bag layout.

Most applications do not call this method directly.

**Specified by:**[minimumLayoutSize](http://docs.google.com/java/awt/LayoutManager.html#minimumLayoutSize(java.awt.Container)) in interface [LayoutManager](http://docs.google.com/java/awt/LayoutManager.html) **Parameters:**parent - the container in which to do the layout **Returns:**the minimum size of the parent container**See Also:**[Container.doLayout()](http://docs.google.com/java/awt/Container.html#doLayout())

### maximumLayoutSize

public [Dimension](http://docs.google.com/java/awt/Dimension.html) **maximumLayoutSize**([Container](http://docs.google.com/java/awt/Container.html) target)

Returns the maximum dimensions for this layout given the components in the specified target container.

**Specified by:**[maximumLayoutSize](http://docs.google.com/java/awt/LayoutManager2.html#maximumLayoutSize(java.awt.Container)) in interface [LayoutManager2](http://docs.google.com/java/awt/LayoutManager2.html) **Parameters:**target - the container which needs to be laid out **Returns:**the maximum dimensions for this layout**See Also:**[Container](http://docs.google.com/java/awt/Container.html), [minimumLayoutSize(Container)](http://docs.google.com/java/awt/GridBagLayout.html#minimumLayoutSize(java.awt.Container)), [preferredLayoutSize(Container)](http://docs.google.com/java/awt/GridBagLayout.html#preferredLayoutSize(java.awt.Container))

### getLayoutAlignmentX

public float **getLayoutAlignmentX**([Container](http://docs.google.com/java/awt/Container.html) parent)

Returns the alignment along the x axis. This specifies how the component would like to be aligned relative to other components. The value should be a number between 0 and 1 where 0 represents alignment along the origin, 1 is aligned the furthest away from the origin, 0.5 is centered, etc.

**Specified by:**[getLayoutAlignmentX](http://docs.google.com/java/awt/LayoutManager2.html#getLayoutAlignmentX(java.awt.Container)) in interface [LayoutManager2](http://docs.google.com/java/awt/LayoutManager2.html) **Returns:**the value 0.5f to indicate centered

### getLayoutAlignmentY

public float **getLayoutAlignmentY**([Container](http://docs.google.com/java/awt/Container.html) parent)

Returns the alignment along the y axis. This specifies how the component would like to be aligned relative to other components. The value should be a number between 0 and 1 where 0 represents alignment along the origin, 1 is aligned the furthest away from the origin, 0.5 is centered, etc.

**Specified by:**[getLayoutAlignmentY](http://docs.google.com/java/awt/LayoutManager2.html#getLayoutAlignmentY(java.awt.Container)) in interface [LayoutManager2](http://docs.google.com/java/awt/LayoutManager2.html) **Returns:**the value 0.5f to indicate centered

### invalidateLayout

public void **invalidateLayout**([Container](http://docs.google.com/java/awt/Container.html) target)

Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.

**Specified by:**[invalidateLayout](http://docs.google.com/java/awt/LayoutManager2.html#invalidateLayout(java.awt.Container)) in interface [LayoutManager2](http://docs.google.com/java/awt/LayoutManager2.html)

### layoutContainer

public void **layoutContainer**([Container](http://docs.google.com/java/awt/Container.html) parent)

Lays out the specified container using this grid bag layout. This method reshapes components in the specified container in order to satisfy the contraints of this GridBagLayout object.

Most applications do not call this method directly.

**Specified by:**[layoutContainer](http://docs.google.com/java/awt/LayoutManager.html#layoutContainer(java.awt.Container)) in interface [LayoutManager](http://docs.google.com/java/awt/LayoutManager.html) **Parameters:**parent - the container in which to do the layout**See Also:**[Container](http://docs.google.com/java/awt/Container.html), [Container.doLayout()](http://docs.google.com/java/awt/Container.html#doLayout())

### toString

public [String](http://docs.google.com/java/lang/String.html) **toString**()

Returns a string representation of this grid bag layout's values.

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a string representation of this grid bag layout.

### getLayoutInfo

protected [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) **getLayoutInfo**([Container](http://docs.google.com/java/awt/Container.html) parent,  
 int sizeflag)

Fills in an instance of GridBagLayoutInfo for the current set of managed children. This requires three passes through the set of children:

1. Figure out the dimensions of the layout grid.
2. Determine which cells the components occupy.
3. Distribute the weights and min sizes amoung the rows/columns.

This also caches the minsizes for all the children when they are first encountered (so subsequent loops don't need to ask again).

This method should only be used internally by GridBagLayout.

**Parameters:**parent - the layout containersizeflag - either PREFERREDSIZE or MINSIZE **Returns:**the GridBagLayoutInfo for the set of children**Since:** 1.4

### GetLayoutInfo

protected [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) **GetLayoutInfo**([Container](http://docs.google.com/java/awt/Container.html) parent,  
 int sizeflag)

This method is obsolete and supplied for backwards compatability only; new code should call [getLayoutInfo](http://docs.google.com/java/awt/GridBagLayout.html#getLayoutInfo(java.awt.Container,%20int)) instead. This method is the same as getLayoutInfo; refer to getLayoutInfo for details on parameters and return value.

### adjustForGravity

protected void **adjustForGravity**([GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) constraints,  
 [Rectangle](http://docs.google.com/java/awt/Rectangle.html) r)

Adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads. This method should only be used internally by GridBagLayout.

**Parameters:**constraints - the constraints to be appliedr - the Rectangle to be adjusted**Since:** 1.4

### AdjustForGravity

protected void **AdjustForGravity**([GridBagConstraints](http://docs.google.com/java/awt/GridBagConstraints.html) constraints,  
 [Rectangle](http://docs.google.com/java/awt/Rectangle.html) r)

This method is obsolete and supplied for backwards compatability only; new code should call [adjustForGravity](http://docs.google.com/java/awt/GridBagLayout.html#adjustForGravity(java.awt.GridBagConstraints,%20java.awt.Rectangle)) instead. This method is the same as adjustForGravity; refer to adjustForGravity for details on parameters.

### getMinSize

protected [Dimension](http://docs.google.com/java/awt/Dimension.html) **getMinSize**([Container](http://docs.google.com/java/awt/Container.html) parent,  
 [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) info)

Figures out the minimum size of the master based on the information from getLayoutInfo. This method should only be used internally by GridBagLayout.

**Parameters:**parent - the layout containerinfo - the layout info for this parent **Returns:**a Dimension object containing the minimum size**Since:** 1.4

### GetMinSize

protected [Dimension](http://docs.google.com/java/awt/Dimension.html) **GetMinSize**([Container](http://docs.google.com/java/awt/Container.html) parent,  
 [GridBagLayoutInfo](http://docs.google.com/java/awt/GridBagLayoutInfo.html) info)

This method is obsolete and supplied for backwards compatability only; new code should call [getMinSize](http://docs.google.com/java/awt/GridBagLayout.html#getMinSize(java.awt.Container,%20java.awt.GridBagLayoutInfo)) instead. This method is the same as getMinSize; refer to getMinSize for details on parameters and return value.

### arrangeGrid

protected void **arrangeGrid**([Container](http://docs.google.com/java/awt/Container.html) parent)

Lays out the grid. This method should only be used internally by GridBagLayout.

**Parameters:**parent - the layout container**Since:** 1.4

### ArrangeGrid

protected void **ArrangeGrid**([Container](http://docs.google.com/java/awt/Container.html) parent)

This method is obsolete and supplied for backwards compatability only; new code should call [arrangeGrid](http://docs.google.com/java/awt/GridBagLayout.html#arrangeGrid(java.awt.Container)) instead. This method is the same as arrangeGrid; refer to arrangeGrid for details on the parameter.

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/GridBagLayout.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/awt/GridBagConstraints.html)   [**NEXT CLASS**](http://docs.google.com/java/awt/GridBagLayoutInfo.html) | [**FRAMES**](http://docs.google.com/index.html?java/awt/GridBagLayout.html)    [**NO FRAMES**](http://docs.google.com/GridBagLayout.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | [FIELD](#3znysh7) | [CONSTR](#2et92p0) | [METHOD](#tyjcwt) | DETAIL: [FIELD](#1t3h5sf) | [CONSTR](#z337ya) | [METHOD](#1y810tw) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).