ITI's GridBLT Learning Tool: Exposing Java's GridBagLayout Class

by Steve Johnston


Comment on this article

Part One: Introduction

What follows is a straightforward tutorial on using Java's notoriously difficult GridBagLayout class to create GUI screens in Java applications. But in addition to that, this is an introduction to GridBLT. Never heard of it? Well, GridBLT is a developer learning tool built specially with GridBagLayout in mind. GridBLT lets you easily visualize the behavior of GridBagLayout, allowing you to more easily learn its peculiarities and idiosyncrasies, since it lets you see them right on the screen.

Java comes with several layout managers to help developers create GUI screens that maintain a consistent appearance across platforms. Although difficult to learn, GridBagLayout is the most powerful and flexible of the layout managers included with the JDK. This tutorial, along with ITI's GridBLT, will help you understand the GridBagLayout class and have you developing professional-looking Java-based GUI screens in short order.

You can download and use GridBLT free of charge. In fact, you will need to download the GridBLT application before working through these tutorials. There are two versions available - a web demo version and a full application.

[Download the GridBLT Application]

Why Use GridBagLayout?

The first reason is to create a truly platform-independent GUI. Java is a platform-independent language and has to be able to display GUI screens on multiple systems that have varying capabilities and properties. With the introduction of the Swing toolkit, you can now create applications that look nearly identical on all operating systems. Despite that, fonts are still a problem, since Java maps logical fonts to the native platform's available fonts. For example, the "Dialog" font in Java maps to "MS Sans Serif" on Microsoft Windows, "b&h-lucida" on X Windows, and "Geneva" on Macintosh. The AWT tries to map logical font names to scalable native fonts but is not always successful. When it fails, the point size you want may be mapped to a larger or smaller font. When this occurs, a string rendered in the same Java font on different operating systems or different virtual machines may have varying widths or heights.

In Java, it is possible to specify the exact coordinates for the location and size of your components, just as in traditional GUI programming. You call setLayout(null) for the container, then call setBounds for each component. However, a problem with setting absolute limits on component size is that, if a font renders unpredictably, your component may not have enough room to display the full text. The following examples show this. Both use the same font ("Dialog", Font.PLAIN, 14), but they render differently. Both images are from the same applet, both are running on Windows 95, but each uses a different Internet browser:

This button (above) looks fine in Internet Explorer 4.0.

The same button rendered in Netscape 4.05 cuts off part of the text.

The second reason to use GridBagLayout concerns Windows resize events. Not using a layout manager for a Java GUI can cause problems with resize events. For example, when you resize a window, components will stay in the top-left corner of the window. But if you use GridBagLayout, the components center in the window. Furthermore, you can set the weights of components to resize exactly as you want them. For example, you can assign a weight value to a component so it will take up the entire width or height of its container as the user resizes the window.

How GridBagLayout Works

Layout managers encapsulate an algorithm that arranges components within a Container. The algorithm for GridBagLayout arranges components in a grid of variable-sized rows and columns. The height of a row is equal to the vertical dimension of the "tallest" component in that row. Likewise, the width of a column is equal to the horizontal dimension of the "widest" component in that column. The number of rows and columns changes dynamically to accommodate the maximum row and column number of all components currently in the layout. Components can be constrained to more than one column wide and more than one row high. There are several other ways to constrain components, which I will describe later in this section.

The variable size of rows and columns is a powerful feature. It means that column 1 can be wide and contain large components, while column 2 can be narrow and contain small components. Column 2's narrower width means there is no wasted space around its smaller components. At the same time, this feature can be confusing for beginners. If a container is empty and a component is added at row 5, column 5, a novice might expect the component to appear in the lower-right corner of the container. This is not the case. Instead, the component appears in the center of the container because rows 0-4 and columns 0-4 do not contain any components and, therefore, do not have any height or width.

A GridBagConstraints object determines a component's location and size. When a container needs to rearrange its components, it calls upon its GridBagLayout object, which queries each of its constraint objects to determine where each component should appear. Note that GridBagLayout constrains components relative to each other, not to absolute coordinates of the container. So, if you change the constraints on one component, it is likely to affect the location and/or size of other components.

To position components relative to other components in a container, the GridBagConstraints class lets you modify any of several fields. To set a component's constraints using GridBagLayout, you first instantiate a GridBagConstraints object and set the field values. All of the constraint fields have default values, so you do not have to specify a value for each constraint field. After the fields are set, you associate the constraints with the component by calling setConstraints() on the GridBagLayout object. Finally, you add the component to the container. Here's a simple example that places a button in a panel:

```
import java.awt.*;
class MyPanel extends Panel
{
   MyPanel( )
   {
      //Create the GridBagLayout object

      GridBagLayout gridBagLayout = new GridBagLayout();
      setLayout(gridBagLayout);

      //Create a button component

      Button button1 = new Button();

      //Create a constraints object, and
      //set the desired field values

      GridBagConstraints gbc = new GridBagConstraints();
      gbc.weightx = 1.0;
      gbc.fill = GridBagConstraints.HORIZONTAL;

      //Associate the constraints and the component

      gridBagLayout.setConstraints(button1, gbc);

      //Finally, add the component to its container

      add(button1);
   }

}
```

Using the constraints gridwidth and gridheight, you can make a component span multiple cells. A component's display area is the rectangular

area whose top left corner is at cell (gridx, gridy), covering gridwidth number of cells in the horizontal direction and gridheight number of cells in the vertical direction. The following table summarizes the remaining fields of GridBagConstraints.

Field	Specifies
gridx, gridy	the x- and y-coordinate of the top-left cell of the component's display area
anchor	where to anchor the component within its display area
fill	how the component fills the grid cells in its display area
gridwidth, gridheight	the number of cells in the horizontal and vertical directions that compose the component's display area
weightx, weighty	how extra space is distributed among cells in the horizontal and vertical directions
insets	padding *outside* the component; affects the amount of space around a component, within its display area
ipadx, ipady	padding inside the component; affects the component size in the horizontal and vertical directions

Previous Page | Return to beginning of article | Next Page

ITI's GridBLT Learning Tool: Exposing Java's GridBagLayout Class

by Steve Johnston

Part Two: How Constraints Affect a GridBagLayout

The first part of the tutorial discusses each constraint in detail. We'll use GridBLT to experiment with constraints and learn how they behave. You may want to download the PDF version of this article, so you will not have to switch back and forth between your web browser and the application while you are working on this tutorial.

Part Two of the tutorial is divided into three segments: A, B, and C. If you perform a step incorrectly and cannot figure out how to get the components and constraints back in sync with this tutorial, you can return to the beginning of the segment you are currently on. This way, if you make a mistake near the end of section C, for example, you don't have to start over at the beginning of the tutorial.

The tables list default values for each constraint, as well as its valid range of values for use with GridBagLayout. Also listed are defaults and valid values for GridBLT. In some cases, GridBLT is more restrictive than GridBagLayout.

gridx, gridy

Default value for GridBagLayout	RELATIVE (which has a value of -1).
Valid values for GridBagLayout	RELATIVE, or any integer greater than or equal to 0.
Default value for GridBLT	The first component is added to cell (0,0). A new component is added to the selected cell if that cell is empty. Otherwise, a new component is added to a new column at the end of row 0.
Valid values for GridBLT	Any integer greater than or equal to 0 is valid. RELATIVE is not allowed when using GridBLT. RELATIVE is not necessary when you are using a visual editor that allows you to easily place components where you want them.

The gridx and gridy constraints specify the x- and y-coordinates of the component's top-left cell. The row and column numbers in a GridBagLayout are zero-based (i.e., the top-left cell of the grid is (0,0)). Note that coordinates are always given as (gridx, gridy), which is the same as (column, row). For example, (1,2) represents column 1, row 2.

Section A

1. Once you have the GridBLT started, go to the Layout menu and click Add component.... A Select Component dialog will appear. Select java.awt.List from the list of components. A List component will appear on the editor at cell (0,0). The white area of the main window is the Editing Window where you will lay out components. Notice that cell (0,0) shows up in the center of the Editing Window, rather than in the top-left corner. GridBagLayout always centers the cells in the rectangle from (0,0) to (largest gridx, largest gridy) within their container.

2. While the List is still selected, change gridx to 2 and gridy to 1 on the Constraints dialog. There are two ways to apply changes you make in the Constraints dialog. Either press the Enter key while the text field still has the focus, or click the Apply button. Apply the new constraints for the List using one of these two methods. The List's row and column headers change to reflect the new values, but the List does not move. This is because the first and second columns and the first row are empty. Rows and columns do not show up in GridBagLayout unless they contain a component; therefore, the List stays centered.

3. In the Layout menu, click Spaced Mode. This places GridBLT in Spaced Mode, wherein rows and columns show up even if they contain no components. You can see that the List is in cell (2,1). Spaced Mode helps you lay out components, but it does not affect the way components will appear in the GridBagLayout for your finished GUI screen.

4. Click in cell (1,0). The cell is highlighted. Add a java.awt.Checkbox component, just as in Step A1. A Checkbox component appears in the selected cell.

5. With cell (1,0) still selected, add a java.awt.Label component. The Label is added to cell (3,0), although you may not be able to see it because of the Constraints dialog. Because cell (1,0) already contains a component, GridBLT places the new component at the end of the first row. Although GridBagLayout allows you to have more than one component per cell, this is a programming error. GridBLT protects you from doing this when you add a new component, but it is still possible to move two components into the same cell using the Constraints dialog.

6. On the Constraints dialog, click the Hide button. The dialog disappears. Now, click in cell (1,0) which contains the Checkbox. The Constraints dialog will reappear and will show the constraints for the Checkbox.

anchor

Default value for GridBagLayout	CENTER
Valid values for GridBagLayout	CENTER, EAST, NORTH, NORTHEAST, NORTHWEST, SOUTH, SOUTHEAST, SOUTHWEST, WEST
Default value for GridBLT	identical to GridBagLayout
Valid values for GridBLT	identical to GridBagLayout

The anchor constraint specifies where to anchor the component within its display area when the display area is larger than the component's preferred size. Components have a getPreferredSize method that returns the size that the component would like to be at any given time. GridBagLayout queries each component for both its preferred size and minimum size and determines the location and size of each component.

GridBLT abbreviates the anchor values on the Constraints dialog as follows:

```
C  = CENTER
E  = EAST
N  = NORTH
NE = NORTHEAST
NW = NORTHWEST
S  = SOUTH
SE = SOUTHEAST
SW = SOUTHWEST
W  = WEST.
```

7. Move the Constraints dialog off to the right and select the Label component. Set the Label's gridx and gridy to (1,1) and click Apply. Notice that after being relocated to cell (1,1), the Label has extra space between itself and its cell borders. Column 1 is wider than the Label because it is sized to the widest component in the column, which is the Checkbox. Row 1 is taller than the Label because it is sized to the tallest component in the row, which is the List. Also, notice that the Label is centered in its cell. This is the default anchor location, and you can see that "C" (which stands for "Center") is the selected anchor in the Constraints dialog.

8. With the Label still selected, click on the NW radio button on the Constraints dialog. The Label moves to the northwest corner of cell (1,1). Notice that you do not have to press the Apply button for your changes to take effect. Experiment by clicking on other anchor values.

9. Click on the List component. Specify an anchor of NE. This has no visible effect on the location of the List because the List is already filling its entire grid cell.

fill

Default value for GridBagLayout	NONE
Valid values for GridBagLayout	NONE, BOTH, HORIZONTAL, VERTICAL
Default value for GridBLT	identical to GridBagLayout
Valid values for GridBLT	identical to GridBagLayout

The fill constraint specifies how the component fills the grid cells in its display area when the display area is larger than the component's preferred size. Setting a component's fill to NONE causes the component to appear at its preferred width and height. A HORIZONTAL or VERTICAL fill makes the component fill its entire display area horizontally and vertically, respectively. BOTH causes a component to fill its

entire display area.

10. Select the Label component and click on the BOTH radio button on the Constraints dialog. The Label component fills its entire display area.

11. Notice that there is still space around the component; however, this is because you are still in Spaced Mode. Go to the Layout menu and click Spaced Mode to turn it off. You will see the Label fill its entire cell. Now turn Spaced Mode back on.

12. Select VERTICAL fill and EAST anchor for the Label. You will see that the Label occupies the entire vertical space of the cell, but not the entire horizontal space. Horizontally, the Label appears anchored to the east.

13. Click on the List component. Select the BOTH fill. The size of the List does not change, because there are no components in row 1 that are taller and no components in column 2 that are wider than the List's preferred size.

gridwidth, gridheight

Default value for GridBagLayout	1
Valid values for GridBagLayout	RELATIVE, REMAINDER, or any integer greater than 0
Default value for GridBLT	identical to GridBagLayout
Valid values for GridBLT	any integer greater than 0 (RELATIVE and REMAINDER are not allowed in GridBLT because they are not necessary when using a visual editor)

The gridwidth and gridheight constraints specify the component's display area in horizontal and vertical cells.

14. Select the Label component. Enter the following values for these constraints: gridx = 0, gridwidth = 2, gridheight = 3, fill = NONE, anchor = CENTER. You will see the cells that belong to the display area of the Label. The display area is highlighted in light blue.

15. Turn off Spaced Mode. Notice that the Spaced Mode view was deceiving. In Spaced Mode, the label occupies all of the cells shown in light blue, even though there are no other components in rows 2 and 3, or in column 0. However, when you turn Spaced Mode off and show the layout as it would normally appear, the empty rows and columns vanish, making it appear that the label only occupies cell (1,1).

16. Add a java.awt.Button component, then click on it and set its gridx = 3 and gridy = 3. Now you can see that the Label component spans more than one cell vertically because row 3 contains a component that gives it height. The Label has a CENTER fill, so it appears exactly in the center of its display area. It does not appear in the center of its (gridx, gridy) cell because that cell is now only the top-left corner of the display area.

17. Set the Label's anchor = SOUTH. It now appears that the Label is in row 3, but it only appears that way because it is anchored to the bottom of a display area that spans multiple cells.

18. Set the Label's fill = VERTICAL. Until now, increasing the gridwidth and gridheight of the Label only increased the size of the display area and changed the Label's location and did not affect the size of the Label itself. Now the Label's size has increased to fill the entire vertical area of its display area.

19. Click on the Checkbox component and set its gridwidth = 3. Notice that the Label no longer has any extra horizontal space. This is a subtle part of GridBagLayout that can be confusing. Turn on Spaced Mode to try to get a better understanding of what is happening. The width of the Checkbox has been distributed among three columns. Previously, all of its width was in column 1. Set the Checkbox's gridwidth back to 1. The Checkbox now has all of its width in column 1. Now the column has a greater width than the Label component, and the Label again has extra space to its left and right.

20. Click on cell (2,3). Add a java.awt.TextField component. Set the TextField's gridwidth = 2. You can see that the TextField and the Button have overlapping display areas. This is a programming error. Currently, GridBLT does not check the constraints you enter to make sure this does not happen. Be careful to avoid this situation.

21. Click on the TextField and delete it. You can do this by pressing the Delete key or by selecting the Delete Selected Component in the Layout menu.

22. Turn Spaced Mode off. Set the Button's gridwidth = 2. Notice that when Spaced Mode is off, the Button appears to be in column 4. Turn Spaced Mode on. Now, you can see that the Button really starts in column 3, but it has a gridwidth of 2. Cases like this make Spaced Mode an invaluable feature for quickly determining a component's actual constraints.

weightx, weighty

Default value for GridBagLayout	0.0
Valid values for GridBagLayout	Any double value greater than or equal to 0.0
Default value for GridBLT	Identical to GridBagLayout
Valid values for GridBLT	Identical to GridBagLayout

The constraints weightx and weighty specify how extra space is distributed among cells horizontally and vertically. This occurs when a container is larger than the combined preferred size of all its components, thereby creating extra space. These are probably the two most confusing constraints, but they are also extremely powerful. They give GridBagLayout its ability to react to resize events.

GridBagLayout applies weights only to whole rows and columns, not to individual components. The component in a row or column with the greatest weight determines the weight of its respective row and column. If column 0 has three components with weightx values of 0.2, 0.4, and 0.0, then column 0 will have a weightx of 0.4. Even though the third component has a weightx of 0.0, it is still given the extra space since the second component's weight of 0.4 sets the weightx for the whole column.

Section B

1. Click New in the File menu to clear the editor. Add three Button components, and assign them the following constraints.

```
Button1: gridx = 0, gridy = 0, weightx = 0.2
Button2: gridx = 0, gridy = 1, weightx = 0.4
Button3: gridx = 0, gridy = 2, weightx = 0.0
```

Note that all Buttons in column 0 take up all of the extra horizontal space in the Editing Window.

A container calculates extra space as follows. First, the preferred size of each column is determined by the largest preferred width of any component in the column. Second, the preferred column widths are summed and subtracted from the container's current width to find the extra horizontal space in the container. Extra vertical space is determined similarly by examining the rows. The amount of extra space in a row or column depends on its weight in relation to the total weight for all rows and columns of the GridBagLayout.

Continuing the example above, where column 0 had a weight of 0.4, let us also say that columns 1, 2 and 3 have a weightx of 0.0, 0.1, and 0.5, respectively. In this example, the total weightx of all columns is 0.4 + 0.0 + 0.1 + 0.5 = 1.0, so column 0 will take up 0.4 out of 1.0 (or 40%) of the available extra space. It is not necessary for the sum of all row and column weights to equal 1.0. However, an easy way to determine the percentage of extra space for each row and column is to provide for their total weights to be 1.0.

Also, in this example, column 1 will take up 0% of the extra space. Rows and columns with a weight of 0.0 will never receive extra space. When all components in a container still have their default weights of 0.0, GridBagLayout groups all of them in the center of the container. Extra space is placed around the whole group.

2. Add three more Buttons with the following constraints:

```
Button1: gridx = 1, gridy = 0, weightx = 0.0
Button2: gridx = 2, gridy = 0, weightx = 0.1
Button3: gridx = 3, gridy = 0, weightx = 0.5
```

You see that column 0 takes up 40% of the extra space, column 1 takes up 0% of the extra space, and columns 2 and 3 take up 10% and 50%, respectively. The calculation of row and column weights becomes more complicated when components span multiple cells. Here are the rules that GridBagLayout uses to determine the weights for each column (and similarly, for each row).

- Columns that have no components have a weightx of 0.
- Examine columns that only contain components with a gridwidth of 1. For each of these columns, the weight of the column equals the greatest weightx of any component in the column.
- Examine components that have a gridwidth greater than 1, in the order in which they were added to the container. If there is a component X that spans three columns, and the weight of component X is less than the combined weight of the three columns that are spanned, nothing happens. However, if X's weightx is greater than the combined weight of the three columns, then the extra weight is distributed among the three columns. The extra weight is distributed in proportion to the current weights of the columns. If all of the columns have a zero weight, all of the weight will be given to the last column.
- The GridBagLayout class has member variables rowWeights and columnWeights. These are arrays of double values that can add additional weight to rows and columns. If the specified columnWeights value is greater than the currently calculated weight of the column, the columnWeights value becomes the weight of the column; otherwise, columnWeights is ignored.

3. Click on the Button at cell (0,2). Change its gridwidth to 3. This does not change the weighting of the columns because this Button's

weightx of 0 is less than the combined weight of columns 0, 1, and 2, which is 0.4 + 0.0 + 0.1 = 0.5.

4. Change the weightx of the Button at cell (0,2) to 2.5. Now, the extra weight of 2.0 is distributed over columns 0, 1, and 2. Column 0 gets 0.4/0.5 (80%) of the extra weight, giving it a total weight of 0.4 + (0.8 * 2.0) = 2.0.

- Column 1 gets 0% of the extra weight, still leaving it with a total weight of 0.0.
- Column 2 gets 0.1/0.5 (20%) of the extra weight, making it have a total weight of 0.1 + (0.2 * 2.0) = 0.5.
- Column 3 remains unaffected and still has a weight of 0.5.

You can see that columns 2 and 3 receive the same weight and take up the same amount of space. Column 0 now takes up 2/3 of the extra space.

5. Delete the Button at cell (0,1). Change the weightx of the Buttons at cells (0,0) and (2, 0) to 0. Since columns 0, 1, and 2 now have zero weight, all of the weight of the Button at cell (0,2) is distributed to column 2.

6. Change the weightx of the Button at cell (0,2) to 0.5. Notice that columns 2 and 3 are the same width because they both have a weightx of 0.5. The Button now gives a weight of 0.5 to column 2.

7. Add a List component to cell (4,0). Give it a weightx of 0.5. Column 4 now has a different width than columns 2 and 3, even though they all have the same weight. This is because the preferred size of the List is wider than the preferred size of the Buttons. The Buttons and the List both have the same amount of extra space on each side, but the List component itself is wider, making the column wider.

8. Select the List component and set its weighty = 1.0. The List's cell now fills the entire Editing Window vertically, but the List itself is still the same size. Row 0 now has a non-zero weight causing the display area of the List to change.

9. Change the List's fill to BOTH. Now, the List fills its entire display area.

10. Resize the main window, making it larger in the vertical direction. You will see that as the size of the window changes, the height of row 0 and the height of the List changes.

If the container's size is smaller than the GridBagLayout's preferred size, the GridBagLayout will either shrink rows or columns smaller than their preferred size, or it will clip components so that they do not appear in the container. There is no easy way to control how GridBagLayout arranges your components when this occurs.

11. Now, resize the main window so that it is very small horizontally. Observe the way the columns resize when there is not enough room for them to display at their preferred width.

insets

Default value for GridBagLayout	(0,0,0,0)
Valid values for GridBagLayout	An Insets object with positive or negative values
Default value for GridBLT	Identical to GridBagLayout
Valid values for GridBLT	An Insets object with values greater than or equal to 0

The insets constraint specifies padding outside the component. It affects the amount of space around a component, within its display area, setting the minimum amount of space between the component and the edges of its display area.

Section C

1. Click New in the File menu to clear the editor. Add the following components:

```
Button: gridx = 0, gridy = 0
List: gridx = 1, gridy = 0
```

2. Change the left inset of the List to 15. There are now 15 pixels of space between the Button and the List.

3. Change the bottom inset of the List to 30. The height of row 0 is increased by 30 pixels. Since the List was already taller than the Button, the List determines the height of row 0. Increasing the bottom inset of the List only increases its display area, not the visible size of the List itself.

4. Now give the Button a BOTH fill. The Button fills its entire display area.

5. Change the top inset of the Button to 10. Since the height of row 0 is currently determined by the larger List component, the Button has extra space. Therefore, changing the top or bottom insets does not change the height of the Button's display area. Instead, it changes the size of the Button within its display area.

6. Increase the top inset of the Button to 150. Now, the preferred size of the Button plus this large top inset should increase the vertical space requested by the Button. It is now taller than the List's current display area. The Button reverts back to its preferred size with the large top inset making the Button appear at the bottom of its display area.

ipadx, ipady

Default value for GridBagLayout	0
Valid values for GridBagLayout	Any integer
Default value for GridBLT	Identical to GridBagLayout
Valid values for GridBLT	Identical to GridBagLayout

The ipadx and ipady constraints specify padding inside the component. Ipadx increases the preferred and minimum width of the component by (ipadx * 2) pixels. A negative ipadx will decrease the preferred and minimum size of the component. The ipady constraint behaves in a similar way. These two constraints are rarely used because components generally know what size they should be and no external forces should change these sizes.

7. Change the ipady of the Button to -5. This will shrink the Button so there is less vertical space inside the Button.

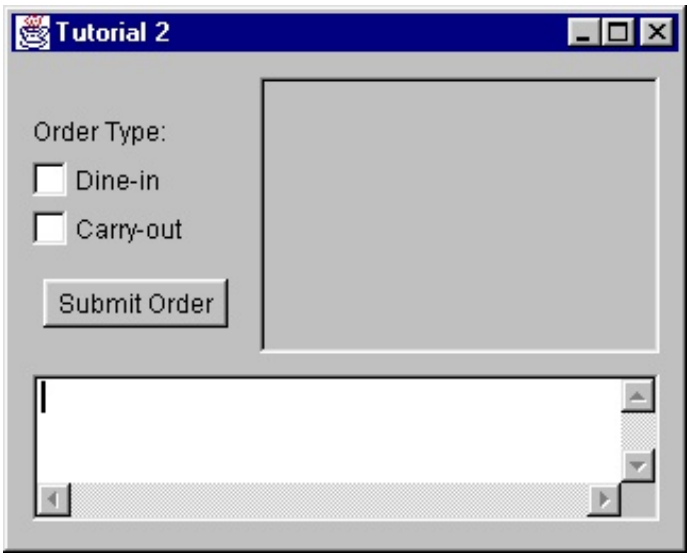
Previous Page Return to beginning of article Next Page

ITI's GridBLT Learning Tool: Exposing Java's GridBagLayout Class

by Steve Johnston

Part Three: Create a Basic GUI with GridBagLayout

This part of the tutorial uses GridBLT to produce a real (although small-scale) example of what GridBagLayout can do for you. You will use GridBagLayout to produce your layout code, then place that code in an application frame, so that the code can be executed. When you resize the Frame, the List and the TextArea will resize to fill the entire Frame. The picture below shows the completed frame.



1. Add all components, and set gridwidths and gridheights

- Click File New to clear the Editing Window.
- Turn Spaced Mode on
- Add a TextArea
- Set the TextArea's gridx = 0 and gridy = 5
- Add a List
- Set the List's gridx = 1 and gridy = 1. From this point on you can click in an empty cell before adding a component, and the component will be added to that cell.
- Set the TextArea's gridwidth = 2
- Set the List's fill = BOTH
- Add a Label at cell (0,1)
- Add a Checkbox at cell (0,2)
- Add a Checkbox at cell (0,3)
- Add a Button at cell (0,4)
- Set the List's gridheight = 4

2. Set weights and fills to handle resizing

- Now that all the components have been added, turn Spaced Mode off so you will get a more accurate picture of what the GridBagLayout really looks like.
- Set the Label's weightx = 0.3
- Set the List's weightx = 0.7
- Set the TextArea's fill = BOTH
- Set the TextArea's weighty = 0.5
- Set the Label's weighty = 0.25

- Set the Button's weighty = 0.25

3. Fine-tune constraints to make components appear exactly where you want them

- Set the Label's anchor = SOUTHWEST
- Set both Checkboxes' anchor = WEST
- Set the TextArea's insets: top = 10, left = 10, bottom = 10, right = 10
- Set the List's insets: top = 10, right = 10
- Set the Label's insets: left = 10
- Set both Checkboxes' insets: left = 10
- Set the TextArea's ipady = -150 (Java.awt.TextArea has a large preferred size. Use a negative internal padding if you want to make it smaller)

4. Set properties of the components (Skip this step if you are using the GridBLT Web Demo version.) Now the components are laid out in the desired locations. Use the property sheet to set the following properties:

- Set the Label's text = "Order Type". Press the Apply button.
- Set one Checkbox's label = "Dine-in". Press the Apply button.
- Set one Checkbox's label = "Carry-out". Press the Apply button.
- Set the Button's label = "Submit Order". Press the Apply button.

You can experiment by changing any of the other properties you wish.

5. Use the layout in a Frame

a. Create a Java file containing a Frame class that displays the layout created in steps 1-4.

If you are using the Web Demo version of GridBLT, you can copy and paste the code from the link below. Save the file with the name "TutorialFrame.java".

[View the Code]

If you are using the full GridBLT application:

- Click Save in the File menu. Save the layout with a file name of "TutorialFrame.java".
- Open TutorialFrame.java in a text editor, and place the following code in the file, replacing the two comments with the actual code that was created by GridBLT.

```
import java.awt.*;
public class TutorialFrame extends Frame
{
   //GridBLT variable declaration section goes here
   //This section begins with: //<<GBLT_VARDEC and
   // ends with: //>>GBLT_VARDEC


   public TutorialFrame()
   {
      this.addWindowListener(new CWindowListener());
      setBackground(Color.lightGray);
      setTitle("GridBagLayout Tutorial");

      //GridBLT component initialization section goes here
      //This section begins with: //<<GBLT_INIT with: ends
          // and>GBLT_INIT

   }

   static public void main(String args[])
   {
      (new TutorialFrame()).setVisible(true);
   }

   class CWindowListener extends java.awt.event.WindowAdapter
   {
      public void windowClosing(java.awt.event.WindowEvent event)
      {
         setVisible(false);
         dispose();
```

```
            System.exit(0);
        }
    }
}
```

b. Run the application. (You must have the JDK 1.1.5 installed)

- Compile the application by going to a command prompt in the directory where you saved the file and typing javac TutorialFrame.java
- Run the application. When the Frame appears, you will be able to resize it and observe the components' behavior.

c. More about GridBLT's file capabilities - (Not available on the Web Demo version)

- To Open a file in GridBLT, the file must contain the GBLT tags mentioned in Step 5. The GBLT_VARDEC section should come before the GBLT_INIT section. Place the GBLT_VARDEC tags around all your component member variable declarations. Place the GBLT_INIT tags around all the code that initializes components by calling "set" methods on them. If you place any other code or comments in this section, you may be able to open the file, but the extra code will not be saved when you use GridBLT to save the file.
- Save will overwrite the contents of the existing tagged sections with the updated code for the components that you have edited in GridBLT.
- Save As will create a new file and will write the tagged sections to it. If a Java file is currently open in the editor, Save As will also copy code that is outside of the tagged sections to the new file. If you are editing a layout from scratch and use Save As, only the tagged sections will be written to the file. You will need to wrap the generated code inside a Container-derived class.
- Reload current file lets you make modifications to the file using a text editor and then update the GridBLT with those changes. Try deleting one of the Checkboxes from TutorialFrame.java from the GBLT_VARDEC and GBLT_INIT sections. Reload the file. The component will disappear from the Editing Window.
- The Select Compiler menu item in the Options menu lets you specify what compiler GridBLT uses when opening files. The faster the compiler you select, the faster you will be able to open files.

d. Using GridBLT-generated code in other container classes - (Not for use with the Web Demo version)

The previous steps showed you how to save the output of GridBLT to a file that you can compile and execute. Note that the file does not have to be a Frame, as in the above example. You can wrap GridBLT-generated code in any Container-derived class, such as a Panel, Dialog, etc. In addition, you can use GridBLT to view a file that contains existing GridBagLayout code. Just be sure to use the GBLT_VARDEC and GBLT_INIT tags.

You can use GridBLT on files created with Visual Cafe by changing the tagged sections. Be aware that the layout will not be visible in Visual Cafe unless you change the tags in the file back to Visual Café's tags. If you are using Visual Cafe, try inserting a Dialog into a project by setting its layout to GridBagLayout, adding some components to it, and opening it with GridBLT using the following procedures:

- Replace tags for the component initialization section. //{{INIT_CONTROLS and //}} should be replaced by //<<GBLT_INIT and //>>GBLT_INIT
- Replace tags for the variable declaration section. //{{DECLARE_CONTROLS and //}} should be replaced by //<<GBLT_VARDEC and //>>GBLT_VARDEC
- Move the GBLT_VARDEC section to the top of the class file before the GBLT_INIT section
- Each component must have its gridx and gridy constraints explicitly set to non-negative values. When you try to open a file that has components with no gridx/gridy or a negative gridx/gridy, the components will be rejected by GridBLT
- Now, you can click Open on the GridBLT File menu to open the file.

Previous Page | Return to beginning of article | Next Page

```
import java.awt.*;
public class TutorialFrame extends Frame
{
    //&lt;&lt;GBLT_VARDEC
    java.awt.TextArea m_textArea;
    java.awt.List m_list;
    java.awt.Label m_label;
    java.awt.Checkbox m_checkbox1;
    java.awt.Checkbox m_checkbox2;
    java.awt.Button m_button;
    //GBLT_VARDEC

    public TutorialFrame()
    {
        this.addWindowListener(new CWindowListener());
        setBackground(Color.lightGray);
        setTitle("GridBagLayout Tutorial");

        //&lt;&lt;GBLT_INIT
        GridBagLayout gridBagLayout;
        gridBagLayout = new GridBagLayout();
        setLayout(gridBagLayout);
        setSize(305,274);
        GridBagConstraints gbc;
        //Component: m_textArea (java.awt.TextArea)
        m_textArea = new java.awt.TextArea();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 5;
        gbc.gridwidth = 2;
        gbc.ipady = -150;
        gbc.weighty = 0.5;
        gbc.insets.bottom = 10;
        gbc.insets.left = 10;
        gbc.insets.top = 10;
        gbc.insets.right = 10;
        gbc.fill = 1;
        ((GridBagLayout)getLayout()).setConstraints(m_textArea, gbc);
        m_textArea.setBackground(new java.awt.Color(192,192,192));
        m_textArea.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_textArea.setForeground(new java.awt.Color(0,0,0));
        add(m_textArea);
        //Component: m_list (java.awt.List)
        m_list = new java.awt.List();
        gbc = new GridBagConstraints();
        gbc.gridx = 1;
        gbc.gridy = 1;
        gbc.gridheight = 4;
        gbc.weightx = 0.7;
        gbc.insets.top = 10;
        gbc.insets.right = 10;
        gbc.fill = 1;
        ((GridBagLayout)getLayout()).setConstraints(m_list, gbc);
        m_list.setBackground(new java.awt.Color(192,192,192));
        m_list.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_list.setForeground(new java.awt.Color(0,0,0));
        add(m_list);
        //Component: m_label (java.awt.Label)
        m_label = new java.awt.Label();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
```

```
        gbc.gridy = 1;
        gbc.weightx = 0.3;
        gbc.weighty = 0.25;
        gbc.insets.left = 10;
        gbc.anchor = 16;
        ((GridBagLayout)getLayout()).setConstraints(m_label, gbc);
        m_label.setBackground(new java.awt.Color(192,192,192));
        m_label.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_label.setForeground(new java.awt.Color(0,0,0));
        m_label.setText("Order Type:");
        add(m_label);
        //Component: m_checkbox1 (java.awt.Checkbox)
        m_checkbox1 = new java.awt.Checkbox();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.insets.left = 10;
        gbc.anchor = 17;
        ((GridBagLayout)getLayout()).setConstraints(m_checkbox1, gbc);
        m_checkbox1.setBackground(new java.awt.Color(192,192,192));
        m_checkbox1.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_checkbox1.setForeground(new java.awt.Color(0,0,0));
        m_checkbox1.setLabel("Dine-in");
        add(m_checkbox1);
        //Component: m_checkbox2 (java.awt.Checkbox)
        m_checkbox2 = new java.awt.Checkbox();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 3;
        gbc.insets.left = 10;
        gbc.anchor = 17;
        ((GridBagLayout)getLayout()).setConstraints(m_checkbox2, gbc);
        m_checkbox2.setBackground(new java.awt.Color(192,192,192));
        m_checkbox2.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_checkbox2.setForeground(new java.awt.Color(0,0,0));
        m_checkbox2.setLabel("Carry-out");
        add(m_checkbox2);
        //Component: m_button (java.awt.Button)
        m_button = new java.awt.Button();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 4;
        gbc.weighty = 0.25;
        ((GridBagLayout)getLayout()).setConstraints(m_button, gbc);
        m_button.setBackground(new java.awt.Color(192,192,192));
        m_button.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_button.setForeground(new java.awt.Color(0,0,0));
        m_button.setLabel("Submit Order");
        add(m_button);
        //GBLT_INIT
    }

    static public void main(String args[])
    {
        (new TutorialFrame()).setVisible(true);
    }

    class CWindowListener extends java.awt.event.WindowAdapter
    {
        public void windowClosing(java.awt.event.WindowEvent event)
        {
```

```
            setVisible(false);
            dispose();
            System.exit(0);
        }
    }
}
```

ITI's GridBLT Learning Tool: Exposing Java's GridBagLayout Class

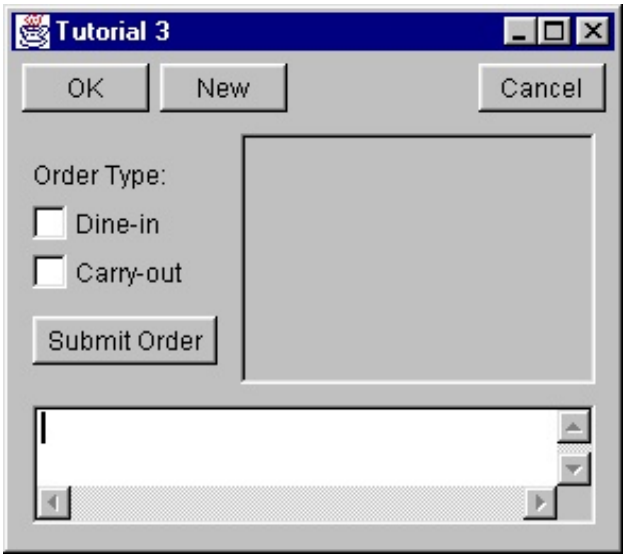
by Steve Johnston

Part Four: An Advanced GridBagLayout Example

The goal in this section of the tutorial is to walk you through some advanced features of the GridBagLayout that are not yet supported by GridBLT. These features are:

- Creating nested panels
- Using rowHeights, columnWidths, rowWeights, and columnWeights

The picture below shows the result of the exercise in this part of the tutorial. This is the same panel we created in Part Three, except for the addition of 3 buttons: OK, New, and Cancel.



All of the following code should appear at the end of the constructor in the file you created for the Tutorial in Part Three.

1. Declare new member variables. Add the following lines to TutorialFrame.java:

```
Button m_okButton;
Button m_newButton;
Button m_cancelButton;
```

You should not place these member variable declarations in the GBLT_VARDEC section. If you do, they will be removed when you save the file using GridBLT. A good location for these variables is immediately following the GBLT_VARDEC section.

2. Create the panel and its GridBagLayout object. The rest of the code in this part of the tutorial should be placed immediately following the GBLT_INIT section. If it is placed inside the GBLT_INIT section, the changes will not be saved.

[View the Code]

```
Panel buttonPanel = new Panel();
GridBagLayout buttonPanelGBL = new GridBagLayout();
buttonPanel.setLayout(buttonPanelGBL);
```

3. Add each of the three buttons to the button panel. Set the constraints and properties for each button.

```
m_okButton = new Button();
gbc = new GridBagConstraints();
gbc.gridx = 0;
gbc.gridy = 0;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.insets.left = 5;
buttonPanelGBL.setConstraints(m_okButton, gbc);
m_okButton.setLabel("OK");
buttonPanel.add(m_okButton);

m_newButton = new Button();
gbc = new GridBagConstraints();
gbc.gridx = 1;
gbc.gridy = 0;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.insets.left = 5;
buttonPanelGBL.setConstraints(m_newButton, gbc);
m_newButton.setLabel("New");
buttonPanel.add(m_newButton);

m_cancelButton = new Button();
gbc = new GridBagConstraints();
gbc.gridx = 3;
gbc.gridy = 0;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.insets.right = 5;
buttonPanelGBL.setConstraints(m_cancelButton, gbc);
m_cancelButton.setLabel("Cancel");
buttonPanel.add(m_cancelButton);
```

4. Add the nested button panel to the main panel. Set the button panel's constraints just like any other component you would add to the main panel. Note that it is given a gridwidth of 2, because components have been placed in columns 0 and 1 in the main panel. Give it a fill of BOTH, so the panel will take up its whole display area.

```
//Set the constraints on the button panel itself and
//add the button panel to the main panel
gbc = new GridBagConstraints();
gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 2;
gbc.fill = GridBagConstraints.BOTH;
gbc.insets.top = 5;
((GridBagLayout)getLayout()).setConstraints(buttonPanel, gbc);
add(buttonPanel);
```

Creating nested panels is a simple and powerful feature of containers and layout managers. Creating complex GUI screens is difficult without this feature. Anytime you have difficulty placing a large number of components in a GridBagLayout because the components do not line up in discrete rows and columns, it is a good idea to use one or more nested panels. You can use nested panels to group together a subset of the components that do line up into discrete rows and columns. For example, the three buttons we added above do not line up very well with the two columns of components that were in the main panel. They need to be spaced so they are independent of the main panel's columns.

5. Set minimum row heights and column widths.

GridBagLayout allows you to set the minimum height for each of its rows and the minimum width for each of its columns. In this example, the columnWidths variable is set to specify the minimum width for each column in the button panel. All of the buttons will have the same minimum width. Note that the buttons added to the button panel have a HORIZONTAL fill so that they will fill up the minimum width and will have the same visible size. You do not have to specify a value for each column or row. Each array element that has not been assigned a value will simply be ignored when GridBagLayout is arranging its components. If the array has more elements than there are rows or columns, the extra values will be ignored.

```
buttonPanelGBL.columnWidths = new int[4];
buttonPanelGBL.columnWidths[0] = 65; //OK button
buttonPanelGBL.columnWidths[1] = 65; //New button
buttonPanelGBL.columnWidths[3] = 65; //Cancel button
```

6. Set minimum row and column weights.
GridBagLayout also allows you to specify the minimum weights for each of its columns and rows. Setting column 2 to have a non-zero weight will consume all the extra horizontal space in the button panel. This forces the OK and New buttons to the left edge of the panel and the Cancel button to the right of the panel.

```
buttonPanelGBL.columnWeights = new double[4];
buttonPanelGBL.columnWeights[2] = 1.0;
```

Conclusion

You can see that GridBagLayout clearly takes some time and effort to learn. Hopefully, though, you can also see from this tutorial that its benefits are worth the extra effort. It's a great way to create platform-independent GUI screens that will handle window resize events just the way you want. Once you get used to it, you probably will not want to use any other layout manager. GridBagLayout is powerful enough to handle any layout situation you might have.

Developed Under:
Visual Cafe 3.0

Previous Page Return to beginning of article Next Page

```
import java.awt.*;
public class TutorialFrame extends Frame
{
    //GBLT_VARDEC
    java.awt.TextArea m_textArea;
    java.awt.List m_list;
    java.awt.Label m_label;
    java.awt.Checkbox m_checkbox1;
    java.awt.Checkbox m_checkbox2;
    java.awt.Button m_button;
    //GBLT_VARDEC

    Button m_okButton;
    Button m_newButton;
    Button m_cancelButton;

    public TutorialFrame()
    {
        this.addWindowListener(new CWindowListener());
        setBackground(Color.lightGray);
        setTitle("GridBagLayout Tutorial");

        //&lt;&lt;GBLT_INIT
        GridBagLayout gridBagLayout;
        gridBagLayout = new GridBagLayout();
        setLayout(gridBagLayout);
        setSize(305,274);
        GridBagConstraints gbc;

        //Component: m_textArea (java.awt.TextArea)
        m_textArea = new java.awt.TextArea();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 5;
        gbc.gridwidth = 2;
        gbc.ipady = -150;
        gbc.weighty = 0.5;
        gbc.insets.bottom = 10;
        gbc.insets.left = 10;
        gbc.insets.top = 10;
        gbc.insets.right = 10;
        gbc.fill = 1;
        ((GridBagLayout)getLayout()).setConstraints(m_textArea, gbc);
        m_textArea.setBackground(new java.awt.Color(192,192,192));
        m_textArea.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_textArea.setForeground(new java.awt.Color(0,0,0));
        add(m_textArea);

        //Component: m_list (java.awt.List)
        m_list = new java.awt.List();
        gbc = new GridBagConstraints();
        gbc.gridx = 1;
        gbc.gridy = 1;
        gbc.gridheight = 4;
        gbc.weightx = 0.7;
        gbc.insets.top = 10;
        gbc.insets.right = 10;
        gbc.fill = 1;
        ((GridBagLayout)getLayout()).setConstraints(m_list, gbc);
        m_list.setBackground(new java.awt.Color(192,192,192));
        m_list.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
```

```
        m_list.setForeground(new java.awt.Color(0,0,0));
        add(m_list);

        //Component: m_label (java.awt.Label)
        m_label = new java.awt.Label();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 1;
        gbc.weightx = 0.3;
        gbc.weighty = 0.25;
        gbc.insets.left = 10;
        gbc.anchor = 16;
        ((GridBagLayout)getLayout()).setConstraints(m_label, gbc);
        m_label.setBackground(new java.awt.Color(192,192,192));
        m_label.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_label.setForeground(new java.awt.Color(0,0,0));
        m_label.setText("Order Type:");
        add(m_label);

        //Component: m_checkbox1 (java.awt.Checkbox)
        m_checkbox1 = new java.awt.Checkbox();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.insets.left = 10;
        gbc.anchor = 17;
        ((GridBagLayout)getLayout()).setConstraints(m_checkbox1, gbc);
        m_checkbox1.setBackground(new java.awt.Color(192,192,192));
        m_checkbox1.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_checkbox1.setForeground(new java.awt.Color(0,0,0));
        m_checkbox1.setLabel("Dine-in");
        add(m_checkbox1);

        //Component: m_checkbox2 (java.awt.Checkbox)
        m_checkbox2 = new java.awt.Checkbox();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 3;
        gbc.insets.left = 10;
        gbc.anchor = 17;
        ((GridBagLayout)getLayout()).setConstraints(m_checkbox2, gbc);
        m_checkbox2.setBackground(new java.awt.Color(192,192,192));
        m_checkbox2.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_checkbox2.setForeground(new java.awt.Color(0,0,0));
        m_checkbox2.setLabel("Carry-out");
        add(m_checkbox2);

        //Component: m_button (java.awt.Button)
        m_button = new java.awt.Button();
        gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 4;
        gbc.weighty = 0.25;
        ((GridBagLayout)getLayout()).setConstraints(m_button, gbc);
        m_button.setBackground(new java.awt.Color(192,192,192));
        m_button.setFont(new java.awt.Font("Dialog", Font.PLAIN, 12));
        m_button.setForeground(new java.awt.Color(0,0,0));
        m_button.setLabel("Submit Order");
        add(m_button);
        //GBLT_INIT
```

```
//Create the panel and its GridBagLayout object
Panel buttonPanel = new Panel();
GridBagLayout buttonPanelGBL = new GridBagLayout();
buttonPanel.setLayout(buttonPanelGBL);



//Add the OK button to the button panel
Button m_okButton = new Button();
gbc = new GridBagConstraints();
gbc.gridx = 0;
gbc.gridy = 0;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.insets.left = 5;
buttonPanelGBL.setConstraints(m_okButton, gbc);
m_okButton.setLabel("OK");
buttonPanel.add(m_okButton);


//Add the New button to the button panel
Button m_newButton = new Button();
gbc = new GridBagConstraints();
gbc.gridx = 1;
gbc.gridy = 0;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.insets.left = 5;
buttonPanelGBL.setConstraints(m_newButton, gbc);
m_newButton.setLabel("New");
buttonPanel.add(m_newButton);

//Add the Cancel button to the button panel
Button m_cancelButton = new Button();
gbc = new GridBagConstraints();
gbc.gridx = 3;
gbc.gridy = 0;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.insets.right = 5;
buttonPanelGBL.setConstraints(m_cancelButton, gbc);
m_cancelButton.setLabel("Cancel");
buttonPanel.add(m_cancelButton);

//Set the constraints on the button panel itself and
//add the button panel to the main panel
gbc = new GridBagConstraints();
gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 2;
gbc.fill = GridBagConstraints.BOTH;
gbc.insets.top = 5;
((GridBagLayout)getLayout()).setConstraints(buttonPanel, gbc);
add(buttonPanel);

//Set column widths, so each button will have the same
//minimum width
buttonPanelGBL.columnWidths = new int[4];
buttonPanelGBL.columnWidths[0] = 65;
buttonPanelGBL.columnWidths[1] = 65;
buttonPanelGBL.columnWidths[3] = 65;

//Set column weights, so that extra space will be taken
//up by column 2, and the buttons will be pushed to
//the left and right edge of the panel
```

```
        buttonPanelGBL.columnWeights = new double[4];
        buttonPanelGBL.columnWeights[2] = 1.0;
    }

    static public void main(String args[])
    {
        (new TutorialFrame()).setVisible(true);
    }

    class CWindowListener extends java.awt.event.WindowAdapter
    {
        public void windowClosing(java.awt.event.WindowEvent event)
        {
            setVisible(false);
            dispose();
            System.exit(0);
        }
    }
}
```

ITI's GridBLT Learning Tool: Exposing Java's GridBagLayout Class

Please read the Public License before downloading the GridBLT Application.

GRIDBLT PUBLIC LICENSE

READ THE FOLLOWING TERMS AND CONDITIONS CAREFULLY BEFORE USING THIS PRODUCT. USING THIS PRODUCT INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THE TERMS AND CONDITIONS OF THIS AGREEMENT, DO NOT INSTALL OR USE THIS PRODUCT.

1. GRANT OF LICENSE
    1. **Authorized Use.** GridBLT Version 1.0 is licensed under the terms and conditions of the GridBLT Public License ("Agreement"). Interface Technologies, Inc. ("Licensor") hereby grants to any individual or legal entity ("Licensee") a world-wide, non-exclusive, royalty-free license to use, copy, display, publish and/or distribute the GridBLT Version 1.0 computer software program ("Software") and documentation ("Documentation"), and to permit others to whom the Software and Documentation are furnished to do so, subject to the terms and conditions of this Agreement. In this Agreement, the Software and Documentation and any and all copies are referred to as the "Licensed Product."
    2. **Ownership.** Licensor owns the Licensed Product and all right, title and interest in and to the Licensed Product, including but not limited to all copyrights, patents, trademarks, trade secrets, and all other intellectual property rights of whatever nature in and to the Licensed Product. Nothing in this Agreement should be construed as transferring any aspect of such rights to Licensee or any third party.
    3. **Restrictions.** Licensee shall not modify or create derivative works based on the Licensed Product. Licensee agrees not to rent, lease or sublicense all or any portion of the Software. Licensee may not redistribute the Licensed Product itself as a single product for profit. Licensee may not repackage the Licensed Product in such a way that any new product derived from the Licensed Product has primarily the same functions and/or features of the Licensed Product as originally distributed by Licensor. Redistributions of the Licensed Product must include a copy of this Agreement.
    4. **Reverse Engineering.** The Software is delivered in object code only. Licensee agrees not to reverse engineer, decompile or disassemble the Software.
2. WARRANTY DISCLAIMER

    THE LICENSED PRODUCT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. LICENSOR AND ITS SUPPLIERS DO NOT WARRANT THAT THE SOFTWARE WILL MEET LICENSEE'S REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT ANY ERRORS IN THE SOFTWARE WILL BE CORRECTED. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LICENSED PRODUCT IS WITH THE LICENSEE.

3. LIMITATION OF LIABILITY

    IN NO EVENT SHALL LICENSOR OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO, DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, OR OTHERWISE ARISING IN ANY WAY OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR PERFORMANCE OF THE SOFTWARE, EVEN IF THE DAMAGES ARE FORESEEABLE OR LICENSOR OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. LICENSEE HEREBY ASSUMES ALL RISK AS TO THE SELECTION, USE, PERFORMANCE AND QUALITY OF THE SOFTWARE AND DOCUMENTATION.

4. TERMINATION

    This License and the rights granted hereunder will terminate automatically if the Licensee fails to comply with the terms and conditions of this Agreement and fails to cure such breach within 30 days of becoming aware of the breach. All provisions of this Agreement relating to warranty disclaimers, limitation of liability, remedies or damages, and Licensor's proprietary rights shall survive termination.

5. GENERAL PROVISIONS
    a. **Waiver.** The waiver or failure of Licensor to exercise any right or remedy under this Agreement does not signify acceptance of the event giving rise to such right or remedy and shall not be deemed a waiver of any right or remedy Licensor may have.
    b. **Governing Law.** This Agreement shall be deemed to have been executed in the State of North Carolina and will be governed by and construed in accordance with the laws of the State of North Carolina without regard to conflict of law principles. Licensee hereby consents to the jurisdiction of the courts of the State of North Carolina or the United States District Court for

the Eastern District of North Carolina for the purpose of any action or proceeding brought by either party in connection with this Agreement. Any litigation or other dispute resolution will take place in Wake County, North Carolina.

c. **Resolution of Disputes.** Any controversy or claim arising out of or relating to this Agreement (excepting any dispute relating to Licensor's proprietary rights) which cannot be settled by the parties shall be resolved by binding arbitration in accordance with the Commercial Arbitration Rules of the American Arbitration Association. The arbitration shall interpret this Agreement in accordance with the ordinary meaning of language and commercial customs, usage and practices as they may be applicable. Any award entered by an arbitrator shall be final and judgment thereon may be entered in any court having jurisdiction. The prevailing party shall be entitled to recovery of costs, fees (including reasonable attorneys' fees) and/or taxes paid or incurred in obtaining the award. Furthermore, any costs, fees or taxes involved in enforcing the award shall be fully assessed against and paid by the party resisting enforcement of the award. Notwithstanding the foregoing, nothing in this section shall prevent Licensor from seeking, obtaining, and enforcing any equitable, legal and/or any other relief in a court of competent jurisdiction to protect its proprietary rights to the Software.

d. **Severability.** If any provision of this Agreement is invalid, illegal or unenforceable under the applicable statute or rule of law, it is to that extent to be deemed omitted. The remainder of the Agreement shall be valid and enforceable to the maximum extent possible.

6. U.S. GOVERNMENT RESTRICTED RIGHTS

In the case of the United States Government or an agency thereof as Licensee, the following additional terms apply: Restricted Computer Software, as defined in the Rights in Data-General clause at Federal Acquisition Regulations 52.227-14; and as applicable.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Interface Technologies, Inc., Raleigh, North Carolina

LICENSEE ACKNOWLEDGES THAT IT HAS READ AND UNDERSTANDS THIS AGREEMENT AND AGREES TO BE BOUND BY ITS TERMS. LICENSEE FURTHER AGREES THAT THIS AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN LICENSEE AND LICENSOR, AND SUPERSEDES ANY PROPOSAL OR PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

Do not download the GridBLT Application unless you have read and agree to the above Public License.

The GridBLT Download page has a Java applet embedded - the page may take a few seconds to load over slower connections. This applet is archived, and will not execute unless you click on the Web Demo button.

[I AGREE TO THE PUBLIC LICENSE]

[Return to the Tutorial]

ITI's GridBLT Learning Tool: Exposing Java's GridBagLayout Class

by Steve Johnston

Download the Files

GridBLT is a Java application that allows you to easily visualize a GridBagLayout. GridBLT is designed to be a learning tool. It doesn't support JFC (Swing) components well, so you may encounter repainting problems if you insert JFC components. However, if you are using AWT components for your applications, GridBLT works well for developing GUI screes for your projects.

The greatest advantage of using GridBLT is that you can actually see the grid as you add components. Grid lines are drawn, and rows and columns are labeled, so you can easily see the rows and columns. This makes it much easier to visualize the constraints on each object. For example, you don't have to look through your code or look in a properties sheet to determine if a component has a gridwidth greater than 1. GridBLT has a Spaced Mode that will force the rows and columns to appear even if they are empty. You can quickly get an overall picture of the current layout.

There are two versions of GridBLT available: The Web Demo version and the Full Application. Both are free to the public. The Web Demo is a slimmed-down version of GridBLT that has a much smaller download time so you can start using it quickly. It contains all the functionality necessary to learn the basics of GridBagLayout. The full application is more powerful. It has a properties sheet that allows you to change the properties of any component, similar to that found in Visual Cafe™. It also supports file use, so you can read in your java files, manipulate the components on a GridBagLayout, and save the layout back to file.

The full GridBLT application allows you to generate and edit Java code directly. There are no 3rd party classes added to your project. The code that the editor produces is optimized. Only non-default constraints and properties will be placed in your code. The bottom line is that there is no more code than if you had written it by hand.

Web Demo Version

The Web Demo version will allow you to continue through the tutorials quickly without having to do any set up. Just select the Web Demo link below, and then return to the tutorials.

You will need a browser that supports JDK1.1 to run GridBLT. See the table below:

❑

If you:

- Do not see a "Start" button in the rectangle

  OR
- You only see a gray rectangle

  OR
- There is an applet error message in your browser's status bar

you will need to download one of the browsers listed below.

Internet Explorer 4.x	Microsoft http://www.microsoft.com/ie
Netscape Communicator 4.04	Netscape http://developer.netscape.com/
Netscape 4.05 Professional Edition	Netscape http://developer.netscape.com/

The Full Application

The complete GridBLT application is approximately 315KB.

To run the full application:

1. You will need JDK 1.1.5 or later. To download the latest JDK, visit Sun at http://java.sun.com/
2. From a command prompt, go to the directory where you placed the GridBLT jar file. Then enter the following:

   java -classpath .;*libpath*\classes.zip;gridblt1_00.jar GBLTFrame

   where *libpath* is the path to your java library files in the JDK.

[Download the GridBLT Application]

[Return to the Tutorial]