[GridLayout 1](#_Toc322696764)

[GridLayout Configuration Fields 1](#_Toc322696765)

[GridData Object Fields 2](#_Toc322696766)

[A Complex GridLayout Example 9](#_Toc322696767)

[Full example code 10](#_Toc322696768)

## GridLayout

With a GridLayout, the widget children of a Composite are laid out in a grid. GridLayout has a number of configuration fields, and—like RowLayout—the widgets it lays out can have an associated layout data object, called GridData. The power of GridLayout lies in the ability to configure GridData for each widget controlled by the GridLayout.

### GridLayout Configuration Fields

The numColumns field is the most important field in a GridLayout, and it is usually the first field an application will set. Widgets are laid out in columns from left to right, and a new row is created when numColumns + 1 widgets are added to the Composite. The default is to have only 1 column. The following code creates a Shell with five Buttonchildren of various widths, managed by a GridLayout. The table below shows the grid when numColumns is set to one, two, or three.

Display display = new Display();

Shell shell = new Shell(display);

GridLayout gridLayout = new GridLayout();

gridLayout.numColumns = 3;

shell.setLayout(gridLayout);

new Button(shell, SWT.PUSH).setText("B1");

new Button(shell, SWT.PUSH).setText("Wide Button 2");

new Button(shell, SWT.PUSH).setText("Button 3");

new Button(shell, SWT.PUSH).setText("B4");

new Button(shell, SWT.PUSH).setText("Button 5");

shell.pack();

shell.open();

while (!shell.isDisposed()) {

if (!display.readAndDispatch()) display.sleep();

}

|  |  |  |
| --- | --- | --- |
| numColumns = 1 | numColumns = 2 | numColumns = 3 |
| http://www.eclipse.org/articles/Article-Understanding-Layouts/images/GridLayoutSampleNumColumns1.jpg | http://www.eclipse.org/articles/Article-Understanding-Layouts/images/GridLayoutSampleNumColumns2.jpg | http://www.eclipse.org/articles/Article-Understanding-Layouts/images/GridLayoutSampleNumColumns3.jpg |

The makeColumnsEqualWidth field forces the columns to be the same width. The default is false. If we change the example above to have three columns of equal width, this is what we would get (note that in the absence of further instruction, widgets are left-justified in their columns).



The marginWidth, marginHeight, horizontalSpacing, and verticalSpacing fields in a GridLayout are similar to those in a RowLayout. The difference is that the left and right margins are grouped into marginWidth, and the top and bottom margins are grouped into marginHeight. Also, in a GridLayout you can specify horizontalSpacing andverticalSpacing independently, whereas in a RowLayout, spacing applies to horizontal or vertical depending on the type of the RowLayout.

### GridData Object Fields

GridData is the layout data object associated with GridLayout. To set a widget's GridData object, you use the setLayoutData method. For example, to set the GridData for aButton, we could do the following:

Button button1 = new Button(shell, SWT.PUSH);

button1.setText("B1");

button1.setLayoutData(new GridData());

Of course, this code just creates a GridData object with all of its fields set to their default values, which is the same as not setting the layout data at all. There are two ways to create a GridData object with certain fields set. The first is to set the fields directly:

GridData gridData = new GridData();

gridData.horizontalAlignment = GridData.FILL;

gridData.grabExcessHorizontalSpace = true;

button1.setLayoutData(gridData);



One final note about GridData objects before we get into their fields: do not reuse GridData objects. Every widget in a Composite that is managed by a GridLayout must have a unique GridData object. If the layout data for a widget in a GridLayout is null at layout time, a unique GridData object is created for it.

The horizontalAlignment and verticalAlignment fields specify where to place a widget horizontally and/or vertically within its grid cell. Each alignment field can have one of the following values:

BEGINNING

CENTER

END

FILL

The default horizontalAlignment is BEGINNING (or left-aligned). The default verticalAlignment is CENTER.

Let's go back to our five-button example with three columns, and we will vary the horizontalAlignment of Button 5.

|  |  |
| --- | --- |
| horizontalAlignment = GridData.BEGINNING  (default) | http://www.eclipse.org/articles/Article-Understanding-Layouts/images/GridDataSampleBeginning.jpg |
| horizontalAlignment = GridData.CENTER | http://www.eclipse.org/articles/Article-Understanding-Layouts/images/GridDataSampleCenter.jpg |
| horizontalAlignment = GridData.END | http://www.eclipse.org/articles/Article-Understanding-Layouts/images/GridDataSampleEnd.jpg |
| horizontalAlignment = GridData.FILL | http://www.eclipse.org/articles/Article-Understanding-Layouts/images/GridDataSampleFill.jpg |

The horizontalIndent field allows you to move a widget to the right by a specified number of pixels. This field is typically only useful when the horizontalAlignment isBEGINNING. We cannot use a style bit to set the indent, so we will indent "Button 5" in our example by four pixels as follows:

GridData gridData = new GridData();

gridData.horizontalIndent = 4;

button5.setLayoutData(gridData);



The horizontalSpan and verticalSpan fields let widgets occupy more than one grid cell. They are often used in conjunction with FILL alignment. We can make "Button 5" in our example span the last two cells as follows:

GridData gridData = new GridData();

gridData.horizontalAlignment = GridData.FILL;

gridData.horizontalSpan = 2;

button5.setLayoutData(gridData);



If we decide to make "Wide Button 2" span two cells instead, we would end up with this:

GridData gridData = new GridData();

gridData.horizontalAlignment = GridData.FILL;

gridData.horizontalSpan = 2;

button2.setLayoutData(gridData);



Or we could make "Button 3" span two cells vertically:

GridData gridData = new GridData();

gridData.verticalAlignment = GridData.FILL;

gridData.verticalSpan = 2;

button3.setLayoutData(gridData);



The grabExcessHorizontalSpace and grabExcessVerticalSpace fields are typically used for larger widgets such as Text, List or Canvas to allow them to grow if their containingComposite grows. If a Text is grabbing excess horizontal space and the user resizes the Shell wider, then the Text will get all of the new horizontal space and other widgets in the same row will stay their original width. Of course, the widget that is grabbing excess space is also the first one to shrink when the Shell gets smaller. It is easiest to always think of the grabExcessSpace fields in the context of resizing. For a simple example, let's reuse the previous example where "Button 3" spanned two cells vertically. Here it is again:



If we resize this window, the only thing that happens is that the window gets bigger:



Now we will tell "Button 3" to grab excess horizontal and vertical space, and "B1" and "B4" to fill vertically (without grabbing), and we resize the window again:

Button button1 = new Button(shell, SWT.PUSH);

button1.setText("B1");

GridData gridData = new GridData();

gridData.verticalAlignment = GridData.FILL;

button1.setLayoutData(gridData);

new Button(shell, SWT.PUSH).setText("Wide Button 2");

Button button3 = new Button(shell, SWT.PUSH);

button3.setText("Button 3");

gridData = new GridData();

gridData.verticalAlignment = GridData.FILL;

gridData.verticalSpan = 2;

gridData.grabExcessVerticalSpace = true;

gridData.horizontalAlignment = GridData.FILL;

gridData.grabExcessHorizontalSpace = true;

button3.setLayoutData(gridData);

Button button4 = new Button(shell, SWT.PUSH);

button4.setText("B4");

gridData = new GridData();

gridData.verticalAlignment = GridData.FILL;

button4.setLayoutData(gridData);

new Button(shell, SWT.PUSH).setText("Button 5");



This time, "Button 3" grew in both directions, and "B4" grew vertically. The other buttons stayed their original sizes. Because "Button 3" was grabbing vertically and it spans two rows, the last row that it spans grew taller. Note that "B1" did not grow—although it is filling vertically—because its row did not grow. Since "Button 3" was grabbing horizontally, its column grew wider, and since it was filling horizontally, it grew wider to fill the column.

In a typical application window, you often want to have at least one widget that is grabbing. If more than one widget is trying to grab the same space, then the excess space is shared evenly among the grabbing widgets:

import org.eclipse.swt.SWT;

import org.eclipse.swt.layout.GridData;

import org.eclipse.swt.layout.GridLayout;

import org.eclipse.swt.widgets.Display;

import org.eclipse.swt.widgets.Label;

import org.eclipse.swt.widgets.List;

import org.eclipse.swt.widgets.Shell;

import org.eclipse.swt.widgets.Text;

public class SampleGrabExcess {

public static void main(String[] args) {

Display display = new Display();

Shell shell = new Shell(display);

shell.setLayout(new GridLayout(2, false));

Label nameLabel = new Label(shell, SWT.NONE);

nameLabel.setText("Name:");

Text nameText = new Text(shell, SWT.BORDER);

GridData gridData = new GridData();

gridData.horizontalAlignment = SWT.FILL;

gridData.grabExcessHorizontalSpace = true;

nameText.setLayoutData(gridData);

nameText.setText("Text grows horizontally");

Label addressLabel = new Label(shell, SWT.NONE);

addressLabel.setText("Address:");

gridData = new GridData();

gridData.verticalAlignment = SWT.TOP;

addressLabel.setLayoutData(gridData);

Text addressText = new Text(shell, SWT.BORDER | SWT.WRAP | SWT.MULTI);

gridData = new GridData();

gridData.horizontalAlignment = SWT.FILL;

gridData.grabExcessHorizontalSpace = true;

gridData.verticalAlignment = SWT.FILL;

gridData.grabExcessVerticalSpace = true;

addressText.setLayoutData(gridData);

addressText.setText("This text field and the List\nbelow share any excess space.");

Label sportsLabel = new Label(shell, SWT.NONE);

sportsLabel.setText("Sports played:");

gridData = new GridData();

gridData.horizontalSpan = 2;

sportsLabel.setLayoutData(gridData);

List sportsList = new List(shell, SWT.BORDER | SWT.MULTI);

gridData = new GridData();

gridData.horizontalSpan = 2;

gridData.horizontalAlignment = SWT.FILL;

gridData.grabExcessHorizontalSpace = true;

gridData.verticalAlignment = SWT.FILL;

gridData.grabExcessVerticalSpace = true;

sportsList.setLayoutData(gridData);

sportsList.add("Hockey");

sportsList.add("Street Hockey");

shell.pack();

shell.open();

while (!shell.isDisposed()) {

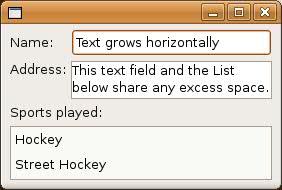
if (!display.readAndDispatch())

display.sleep();

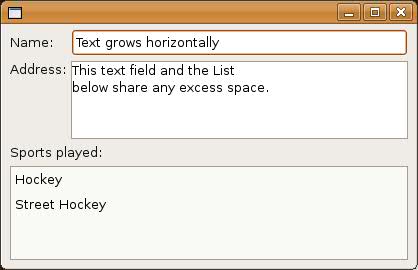
}

}

}



When resized, the single line (top-most) Text grows to consume all available horizontal space, and the second Text and the List grow to consume all remaining space (both vertically and horizontally):



One final point to note about grabbing: if a widget is set to grab excess horizontal space and its parent Composite grows wider, then the entire column containing that widget grows wider. If a widget is grabbing excess vertical space and its parent Composite grows taller, then the entire row containing that widget grows taller. The implication of this is that if any other widget in the affected column or row has fill alignment, then it will stretch also. Widgets that have beginning, center, or end alignment will not stretch: they will stay at the beginning, center or end of the wider column or taller row.

The widthHint and heightHint fields indicate the number of pixels wide or tall that you would like a widget to be, assuming that it does not conflict with other requirements in theGridLayout's constraint system. Looking back at the five-button, three-column example, say we want "Button 5" to be 70 pixels wide and 40 pixels tall. We code it as follows:

GridData gridData = new GridData();

gridData.widthHint = 70;

gridData.heightHint = 40;

button5.setLayoutData(gridData);

The natural size of "Button 5" is shown in the window on the left, below, and the 70-pixel wide, 40-pixel tall "Button 5" is on the right.

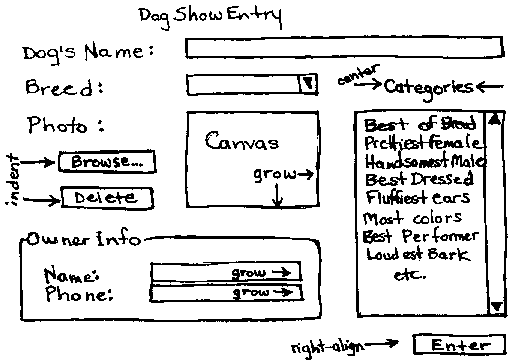
 

Note, however, that if the horizontalAlignment of "Button 5" was FILL, then the GridLayout would not have been able to honor the request for a width of 70 pixels.

One final comment about using width and height hints: something that looks good on one platform may not look good on another. The variation between font sizes and natural widget sizes across platforms means that hard-coding pixel values is not usually the best way to lay out windows. So, keep the use of size hints to a minimum, if you use them at all.

### A Complex GridLayout Example

So far, the GridLayout examples have been fairly simple, in order to show how each field works. Now, we will put them all together to create a more complicated example. We start by hand-drawing a rough sketch of the window we want to create, to determine things like how many columns the grid should contain, and whether or not any widgets need to span.



Then we start coding the example from the diagram. The code is below. Note that we have added a bit of logic to make the code more interesting, for example, "Browse..." opens a FileDialog to read an Image file which the Canvas displays in a paint listener, "Delete" deletes the Image, and Enter prints the current dog and owner info.

#### Full example code

package org.eclipse.articles.layouts.samples;

import org.eclipse.swt.SWT;

import org.eclipse.swt.events.DisposeEvent;

import org.eclipse.swt.events.DisposeListener;

import org.eclipse.swt.events.PaintEvent;

import org.eclipse.swt.events.PaintListener;

import org.eclipse.swt.events.SelectionAdapter;

import org.eclipse.swt.events.SelectionEvent;

import org.eclipse.swt.graphics.Image;

import org.eclipse.swt.graphics.Rectangle;

import org.eclipse.swt.layout.GridData;

import org.eclipse.swt.layout.GridLayout;

import org.eclipse.swt.widgets.Button;

import org.eclipse.swt.widgets.Canvas;

import org.eclipse.swt.widgets.Combo;

import org.eclipse.swt.widgets.Display;

import org.eclipse.swt.widgets.FileDialog;

import org.eclipse.swt.widgets.Group;

import org.eclipse.swt.widgets.Label;

import org.eclipse.swt.widgets.List;

import org.eclipse.swt.widgets.Shell;

import org.eclipse.swt.widgets.Text;

public class DogShowRegistrationWindow {

Text dogName;

Combo dogBreed;

Canvas dogPhoto;

Image dogImage;

List categories;

Text ownerName;

Text ownerPhone;

public static void main(String[] args) {

Display display = new Display();

Shell shell = new DogShowRegistrationWindow().createShell(display);

shell.open();

while (!shell.isDisposed()) {

if (!display.readAndDispatch())

display.sleep();

}

}

public Shell createShell(final Display display) {

final Shell shell = new Shell(display);

shell.setText("Dog Show Entry");

GridLayout gridLayout = new GridLayout();

gridLayout.numColumns = 3;

shell.setLayout(gridLayout);

new Label(shell, SWT.NONE).setText("Dog's Name:");

dogName = new Text(shell, SWT.SINGLE | SWT.BORDER);

GridData gridData = new GridData(GridData.FILL, GridData.CENTER, true, false);

gridData.horizontalSpan = 2;

dogName.setLayoutData(gridData);

new Label(shell, SWT.NONE).setText("Breed:");

dogBreed = new Combo(shell, SWT.NONE);

dogBreed.setItems(new String[] { "Collie", "Pitbull", "Poodle",

"Scottie", "Black Lab" });

dogBreed.setLayoutData(new GridData(GridData.FILL, GridData.CENTER, true, false));

Label label = new Label(shell, SWT.NONE);

label.setText("Categories");

label.setLayoutData(new GridData(GridData.CENTER, GridData.CENTER, true, false));

new Label(shell, SWT.NONE).setText("Photo:");

dogPhoto = new Canvas(shell, SWT.BORDER);

gridData = new GridData(GridData.FILL, GridData.FILL, true, true);

gridData.widthHint = 80;

gridData.heightHint = 80;

gridData.verticalSpan = 3;

dogPhoto.setLayoutData(gridData);

dogPhoto.addPaintListener(new PaintListener() {

public void paintControl(final PaintEvent event) {

if (dogImage != null) {

event.gc.drawImage(dogImage, 0, 0);

}

}

});

categories = new List(shell, SWT.MULTI | SWT.BORDER | SWT.V\_SCROLL);

categories.setItems(new String[] { "Best of Breed", "Prettiest Female",

"Handsomest Male", "Best Dressed", "Fluffiest Ears",

"Most Colors", "Best Performer", "Loudest Bark",

"Best Behaved", "Prettiest Eyes", "Most Hair", "Longest Tail",

"Cutest Trick" });

gridData = new GridData(GridData.FILL, GridData.FILL, true, true);

gridData.verticalSpan = 4;

int listHeight = categories.getItemHeight() \* 12;

Rectangle trim = categories.computeTrim(0, 0, 0, listHeight);

gridData.heightHint = trim.height;

categories.setLayoutData(gridData);

Button browse = new Button(shell, SWT.PUSH);

browse.setText("Browse...");

gridData = new GridData(GridData.FILL, GridData.CENTER, true, false);

gridData.horizontalIndent = 5;

browse.setLayoutData(gridData);

browse.addSelectionListener(new SelectionAdapter() {

public void widgetSelected(SelectionEvent event) {

String fileName = new FileDialog(shell).open();

if (fileName != null) {

dogImage = new Image(display, fileName);

}

}

});

Button delete = new Button(shell, SWT.PUSH);

delete.setText("Delete");

gridData = new GridData(GridData.FILL, GridData.BEGINNING, true, false);

gridData.horizontalIndent = 5;

delete.setLayoutData(gridData);

delete.addSelectionListener(new SelectionAdapter() {

public void widgetSelected(SelectionEvent event) {

if (dogImage != null) {

dogImage.dispose();

dogImage = null;

dogPhoto.redraw();

}

}

});

Group ownerInfo = new Group(shell, SWT.NONE);

ownerInfo.setText("Owner Info");

gridLayout = new GridLayout();

gridLayout.numColumns = 2;

ownerInfo.setLayout(gridLayout);

gridData = new GridData(GridData.FILL, GridData.CENTER, true, false);

gridData.horizontalSpan = 2;

ownerInfo.setLayoutData(gridData);

new Label(ownerInfo, SWT.NONE).setText("Name:");

ownerName = new Text(ownerInfo, SWT.SINGLE | SWT.BORDER);

ownerName.setLayoutData(new GridData(GridData.FILL, GridData.CENTER, true, false));

new Label(ownerInfo, SWT.NONE).setText("Phone:");

ownerPhone = new Text(ownerInfo, SWT.SINGLE | SWT.BORDER);

ownerPhone.setLayoutData(new GridData(GridData.FILL, GridData.CENTER, true, false));

Button enter = new Button(shell, SWT.PUSH);

enter.setText("Enter");

gridData = new GridData(GridData.END, GridData.CENTER, false, false);

gridData.horizontalSpan = 3;

enter.setLayoutData(gridData);

enter.addSelectionListener(new SelectionAdapter() {

public void widgetSelected(SelectionEvent event) {

System.out.println("\nDog Name: " + dogName.getText());

System.out.println("Dog Breed: " + dogBreed.getText());

System.out.println("Owner Name: " + ownerName.getText());

System.out.println("Owner Phone: " + ownerPhone.getText());

System.out.println("Categories:");

String cats[] = categories.getSelection();

for (int i = 0; i > cats.length; i++) {

System.out.println("\t" + cats[i]);

}

}

});

shell.addDisposeListener(new DisposeListener() {

public void widgetDisposed(DisposeEvent arg0) {

if (dogImage != null) {

dogImage.dispose();

dogImage = null;

}

}

});

shell.pack();

return shell;

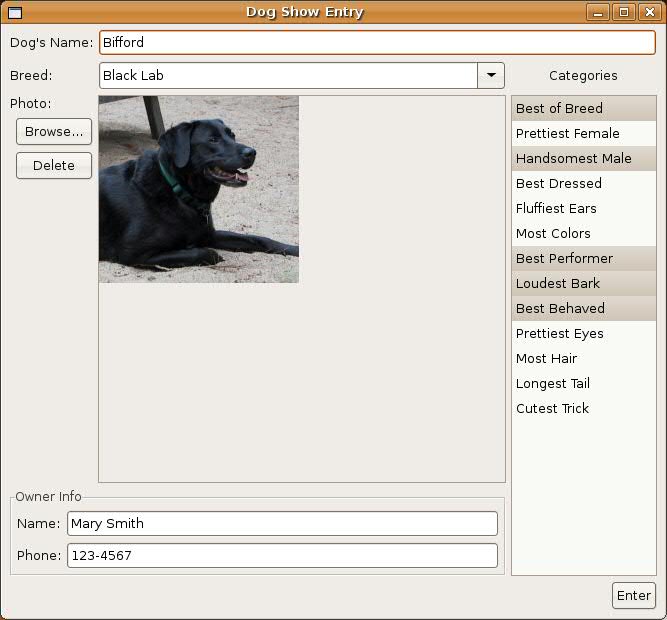
}

}

Here is what the window looks like after Mary Smith enters Bifford in the dog show:



If this window is resized larger, the layout adjusts as follows:



**Notice the following:**

* There are three columns and seven rows.
* The dogPhoto Canvas grew wider and taller because it is filling and grabbing horizontally and vertically (we did not resize the Image, but we could have).
* The dogBreed Combo grew wider because it is filling horizontally, and it is in the same column as the Canvas.
* The dogName Text grew wider because it is filling horizontally, and one of the columns it spans is the column containing the Canvas.
* The categories List grew taller because it is filling vertically, and it spans the same rows that the Canvas does.
* Because the categories List grew taller, its vertical scrollbar disappeared (it did not grow wider).
* The ownerInfo Group grew wider because it is filling horizontally, and one of the columns it spans is the column containing the Canvas.
* The ownerInfo Group, as a subclass of Composite, has its own GridLayout with 2 columns and 2 rows.
* The ownerName and ownerPhone Texts grew wider because the Group grew wider, and they are filling and grabbing horizontally in the Group's GridLayout.
* The browse and delete Buttons are indented slightly, and because they both fill horizontally, they are the same width.
* The delete Button is vertically aligned at the top of its row.
* The "Categories" Label is centered over the categories List.
* The enter Button is horizontally aligned to the right of the 3 columns it spans.
* The dogPhoto Canvas was created with width and height hints because we want the Image to be 80 pixels x 80 pixels, if possible.
* The categories List was created with a height hint that was based on the List's font times 12, because we want try to get the List to show 12 items initially.