

## OculusLayout Builder Tutorial

## About OculusLayout Builder

OculusLayout Builder is an easy-to-use program designed to help you layout Java interfaces quickly and simply. Building is intuitive with Drag and Drop, components are configurable while you are building and you can generate the corresponding Java code to use in your own applications.

### *How OculusLayout Works*

OculusLayout is the Java LayoutManager used by the Builder. OculusLayout is basically a series of nested boxes and grids that is smart about alignment and stretching. You can easily override the stretching and alignment properties for any component if the defaults are not what you had in mind, though they most frequently will be.

Components are typically placed inside an OculusBox. OculusBoxes can be either horizontal or vertical, and are built by dropping components next to (for horizontal boxes) or above or below (for vertical boxes) other components. You can also add OculusGrids, which are aligned arrays of components. Both of these containers can be arbitrarily nested within one another.

Both OculusLayout and OculusGrid assume reasonable X- and Y-stretching preferences for each component type, but you can set the size and stretching properties of any component at any time if it is necessary for your particular layout.


Additionally, OculusLayout has the ability to align arbitrary components across box boundaries. You can literally say, “Component A of my layout should line up with Component B,” regardless of which OculusBoxes A and B are in. This is an advanced feature, covered in the API documentation rather than this tutorial.

### *Example*

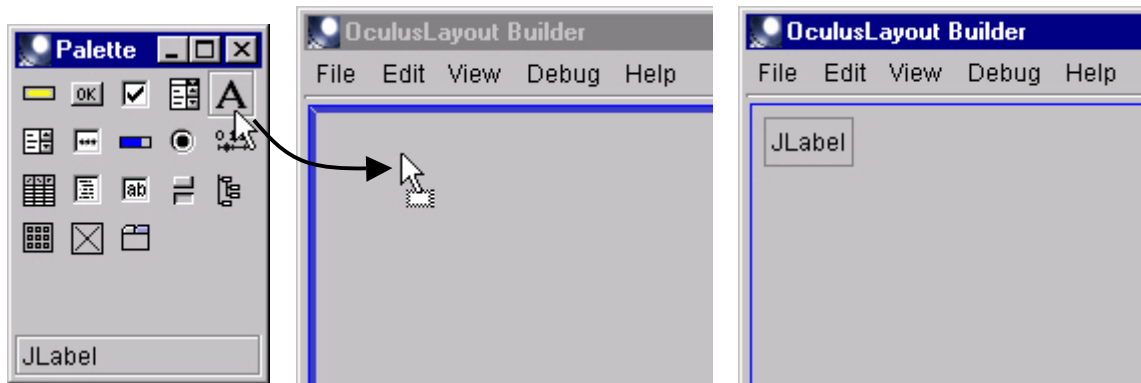
Here we’ll present an example of using OculusLayout Builder that will get you started making your own custom interfaces.


### **Laying Out Components**

When you start up the Builder, you will have three windows: the main Builder window, the component Palette and the Inspector. The Builder window is where you do all your adding and positioning of components. (You can save off these layouts, open saved layouts, and work on multiple layouts at a time.) The Palette is the drag source for all the components you can add to your design – simply drag a component off the Palette to the place you’d like it to appear in your layout. The Inspector is where you edit the properties of whatever you have selected in the active Builder window.

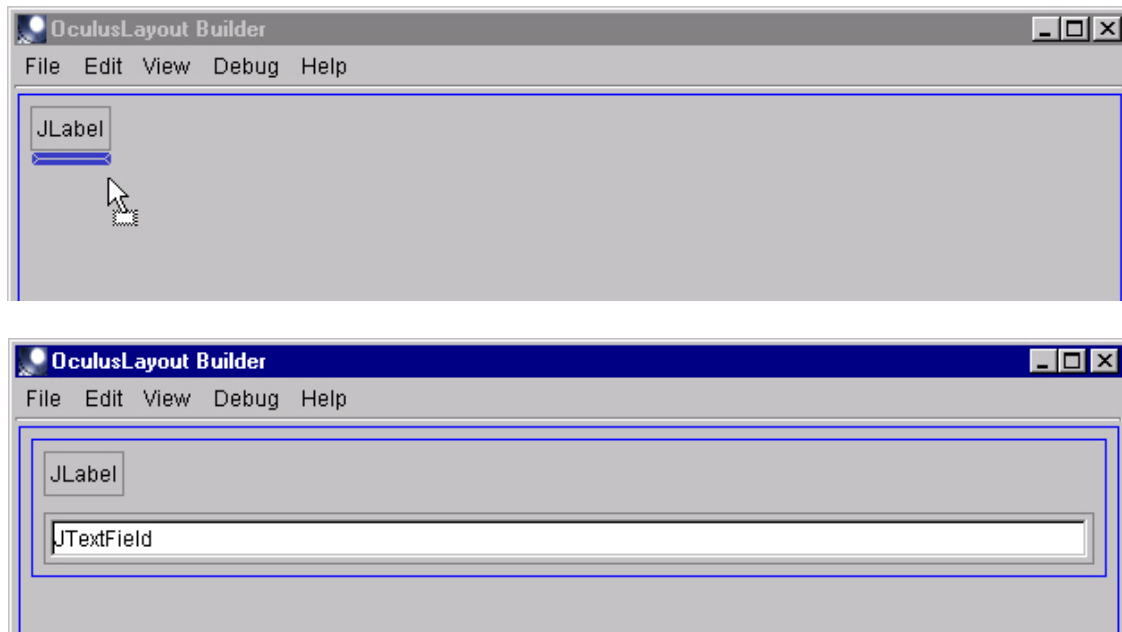
Start by dragging in a JLabel  from the **Palette** to your Builder window. You will notice that the blue box in the Builder window is highlighted when you drag something into it.

### Drag from the Palette to the Builder window



When you drop the JLabel in, it should appear in the layout. Next drag in a JTextField  and drop it below the JLabel. A blue highlight will indicate where the component will be placed.

### Adding a JTextField

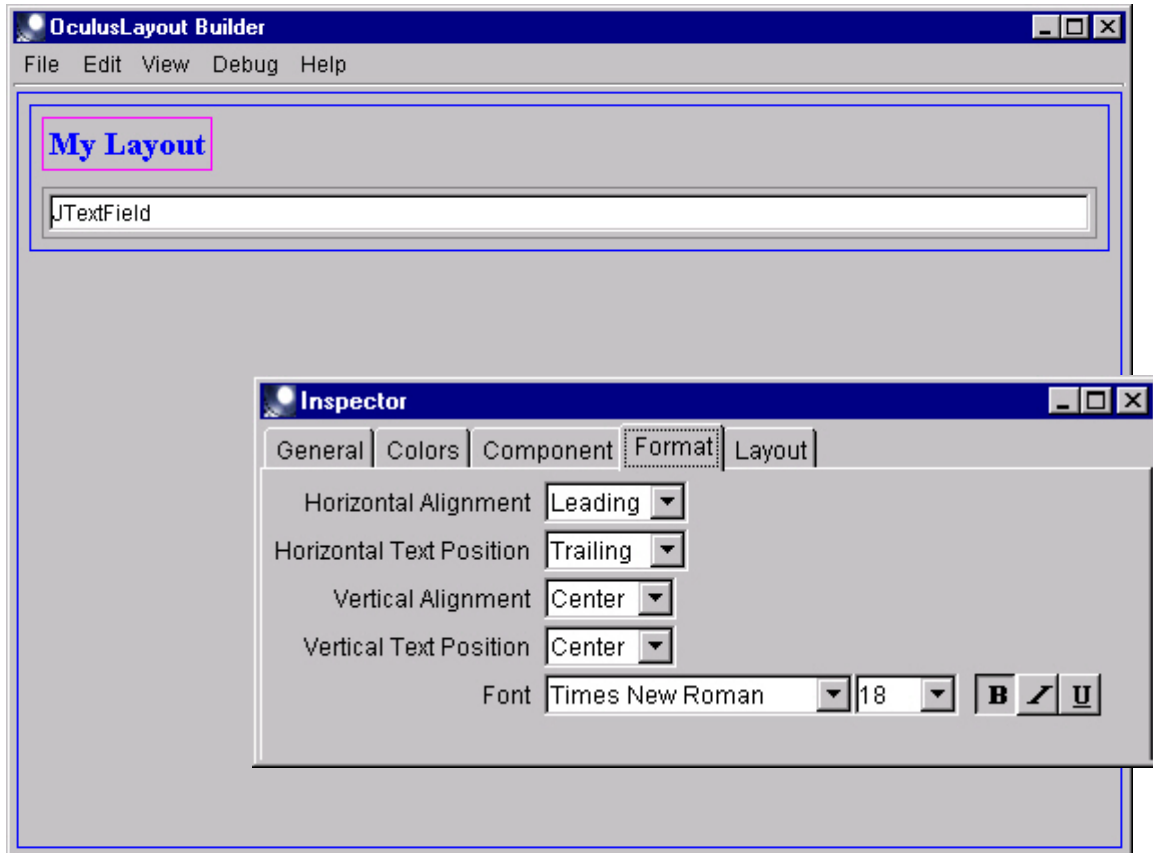


### Configuring Components

Now click on the label once to select it. It will have a magenta box around it and the **Inspector** will show its properties. In the General tab, change the **Text** to

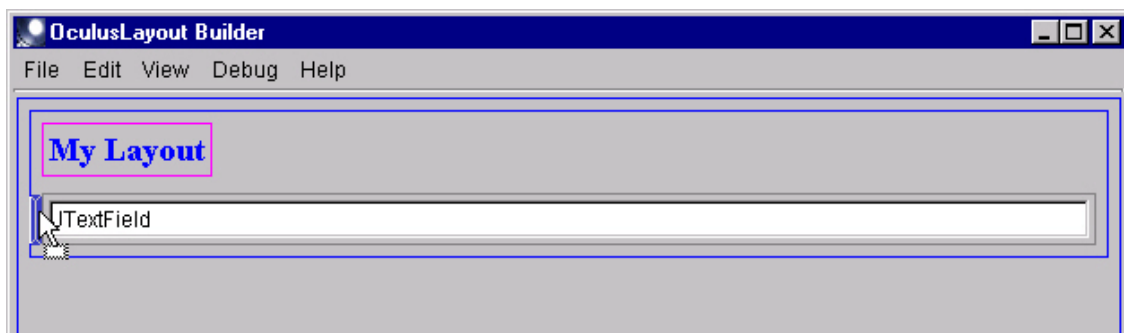
“My Layout”. In the images below, the label was also set to have a Blue **Foreground**, and Times New Roman 18 point bold **Font**.

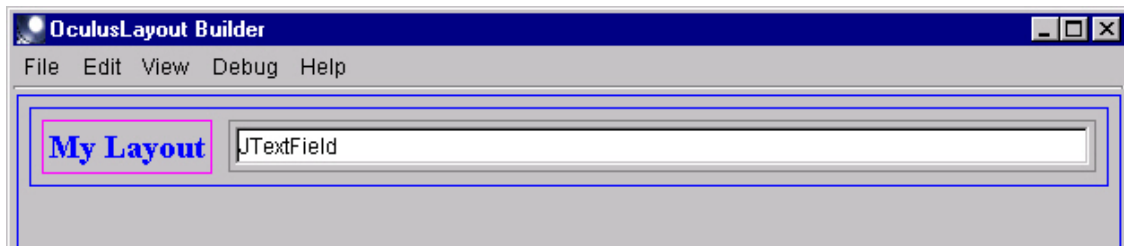
### Configuring a component in the Inspector



Let's say you wanted the label and text field on the same line. Drag the label over to the left of the text field. To begin dragging a component, you have to “pick it up” at one of its corners or edges, near it's border.

### Moving a component to relocate it



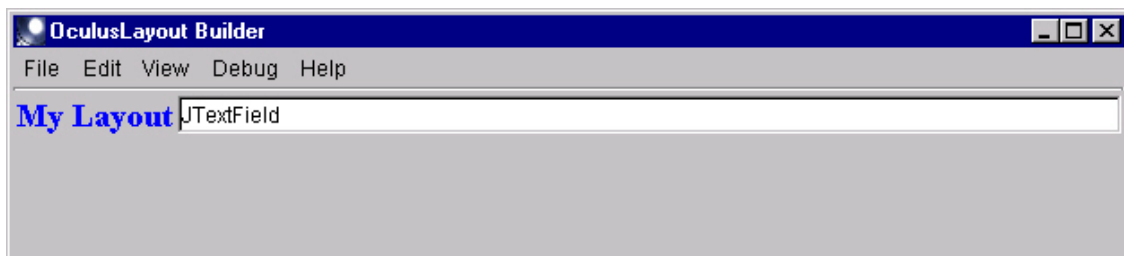


These few steps are basically all you ever have to do:


1. Drag in a component and drop it where you want it to go.
2. If you don't like its default properties, change them in the Inspector.
3. If you change your mind about where you want it, drag it somewhere else.

The borders you see around all the objects are there only for building. If you want to see how the layout would actually appear, uncheck the menu **View > Edit mode**. You can return to edit mode anytime by checking that menu item again.

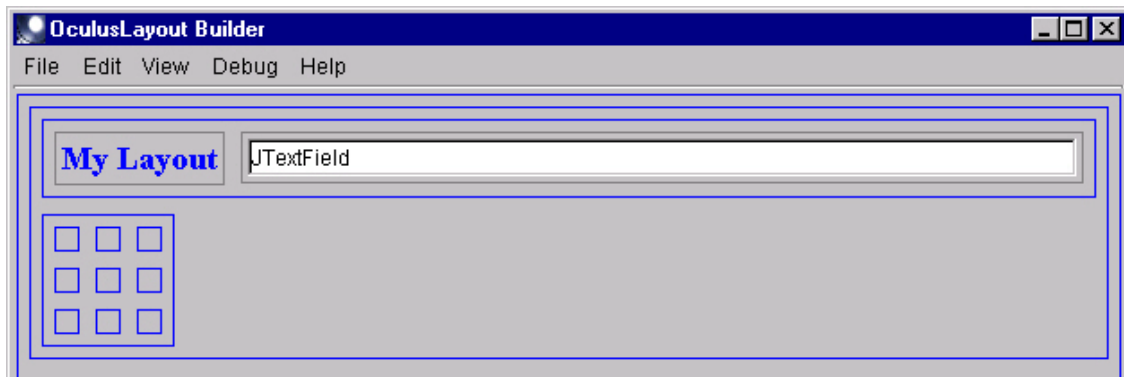
### View mode



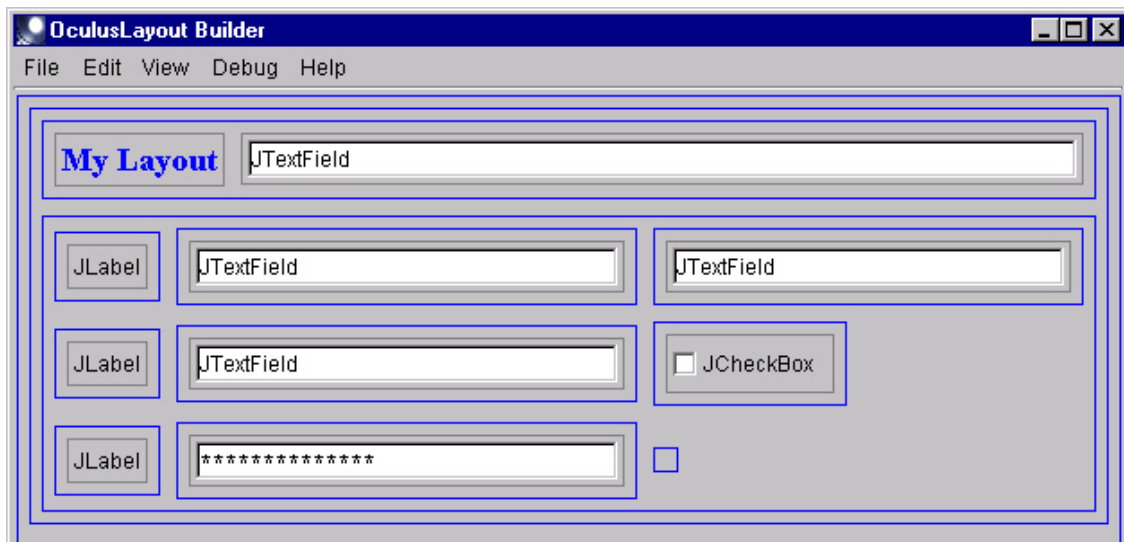
### Using Grids

Continue building the layout by adding an Oculus Grid  below the other components. You can change its numbers of **Columns** and **Rows** in the Inspector, but this example will use the default grid size (3 x 3). Drop components into the grid cells to build the design shown below. You can't move the grid cells themselves, but you can move an object out of one grid cell elsewhere, or nest boxes and grids inside a grid cell. It is also OK to leave a grid cell empty.

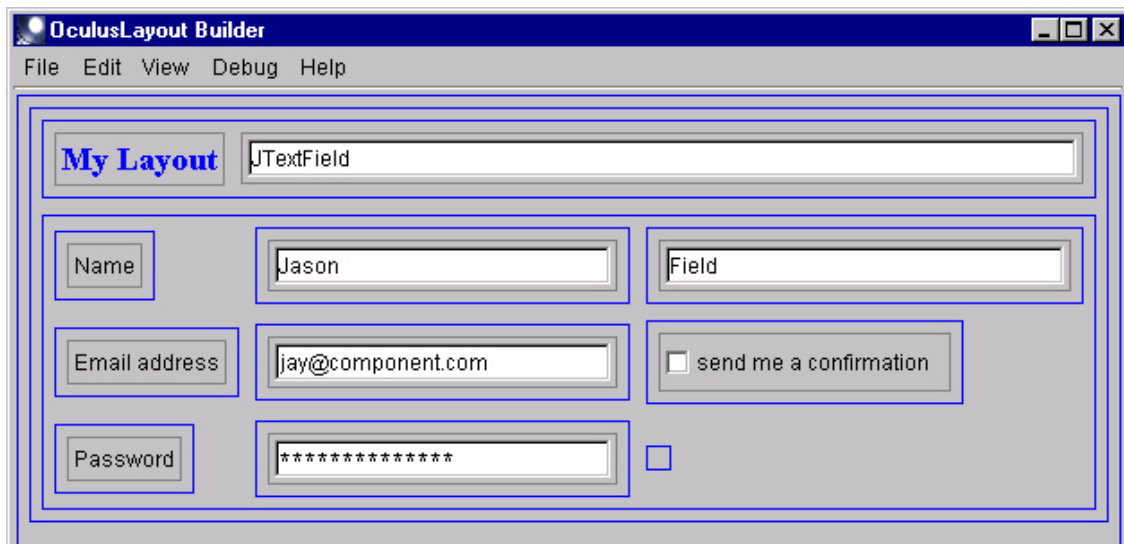
## An empty OculusGrid



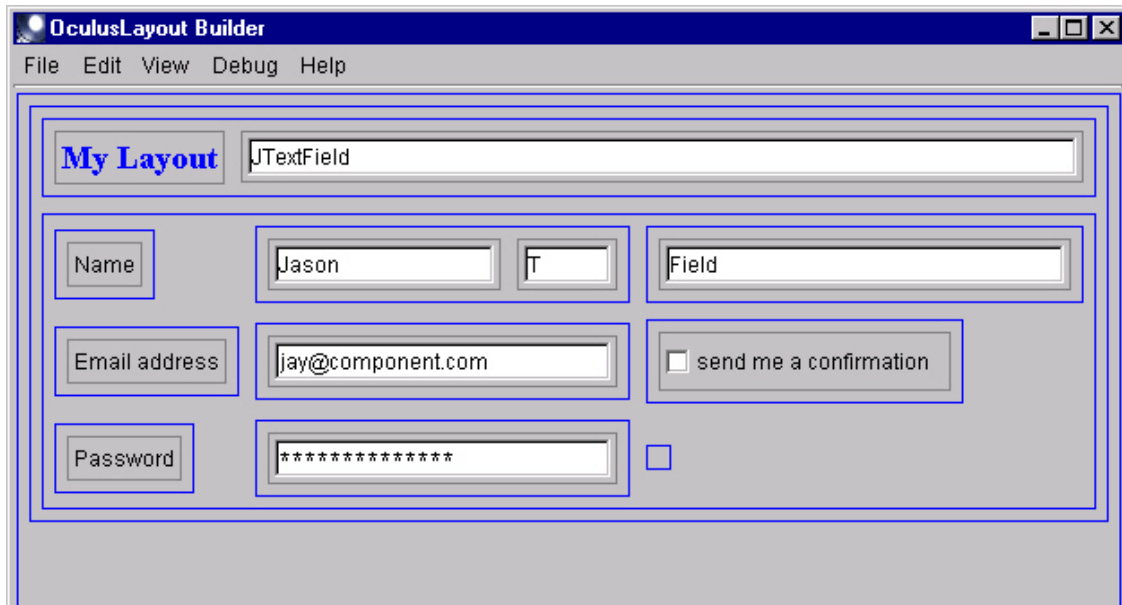
## The grid filled with components




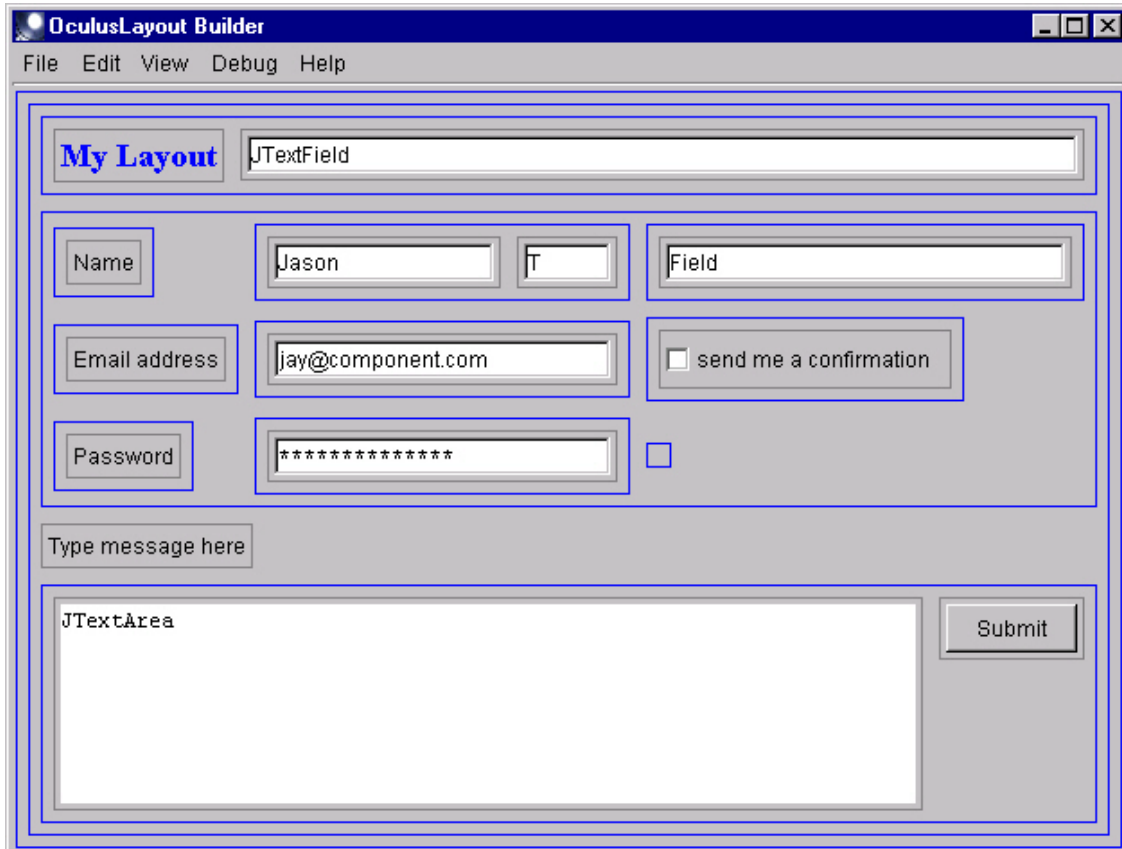
## After setting the text on all the components



Another JTextField is nested in the upper middle grid cell.

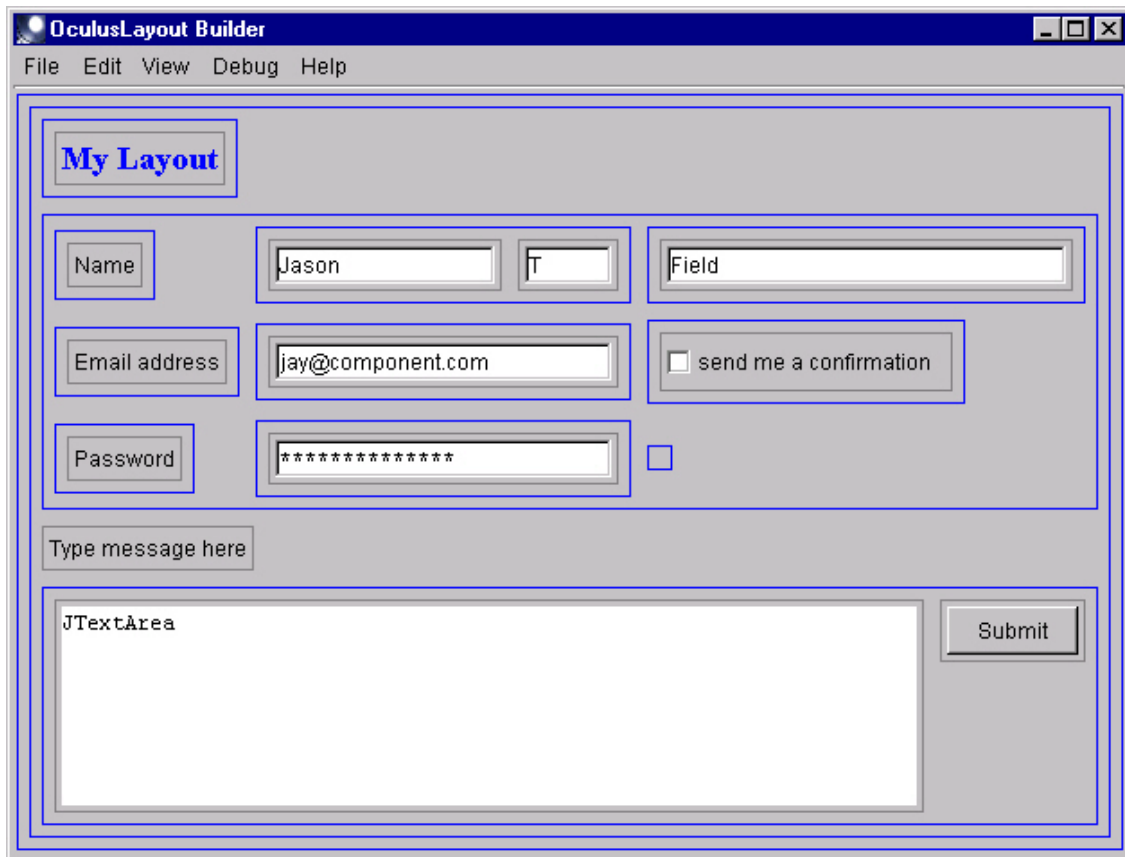


Construct the lower portion of the layout by adding a JLabel, JTextArea and JButton  .



## Removing Components

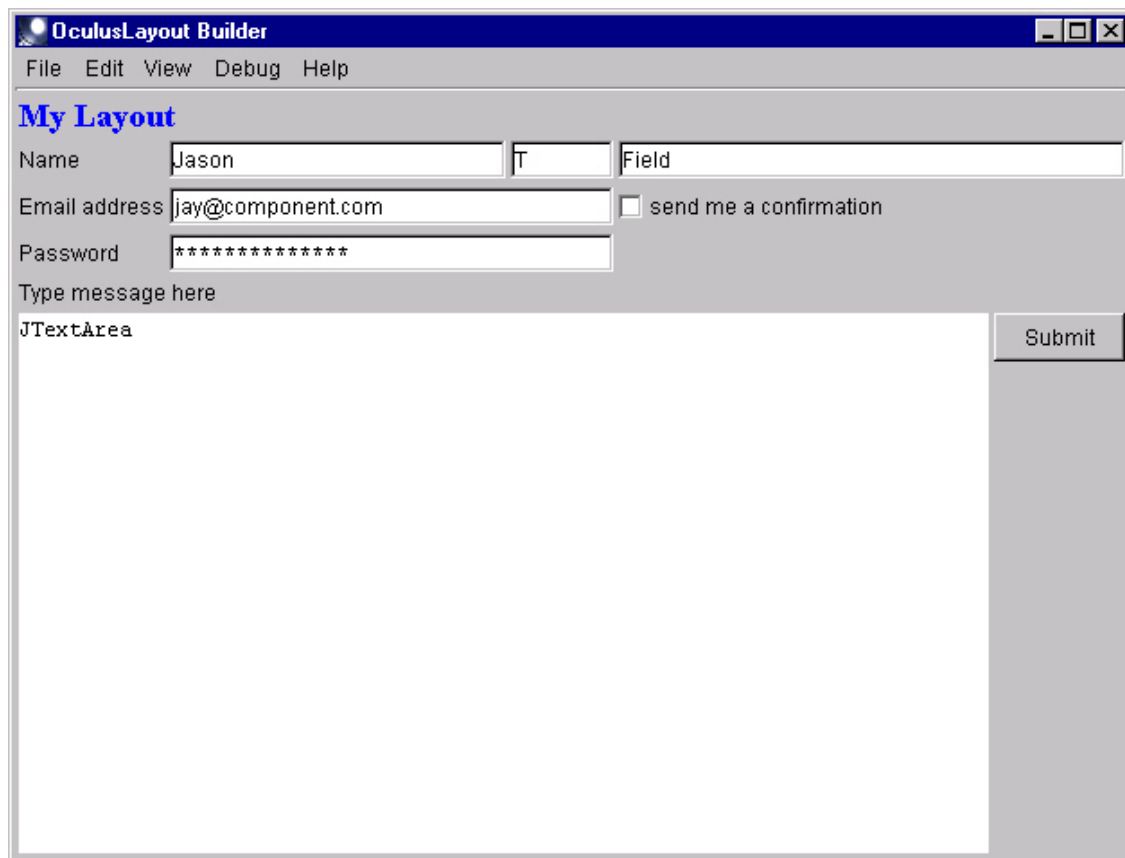
You decide that you don't need the original text field at the top. Select it and choose **Edit > Delete**.




When you are ready to use a layout in your own application, the Builder will auto-generate the Java code. Just choose **File > Generate code...**. Note that for each component, you can specify its variable name to be used in the code in the General tab of the Inspector. Take a look at the generated code for the layout you just built. It is organized and easy to read. When you run it, it will look exactly the way it does when you are out of edit mode in the Builder. Just compile the code with OculusLayout.jar in the classpath, and then run it like you would any Java program.

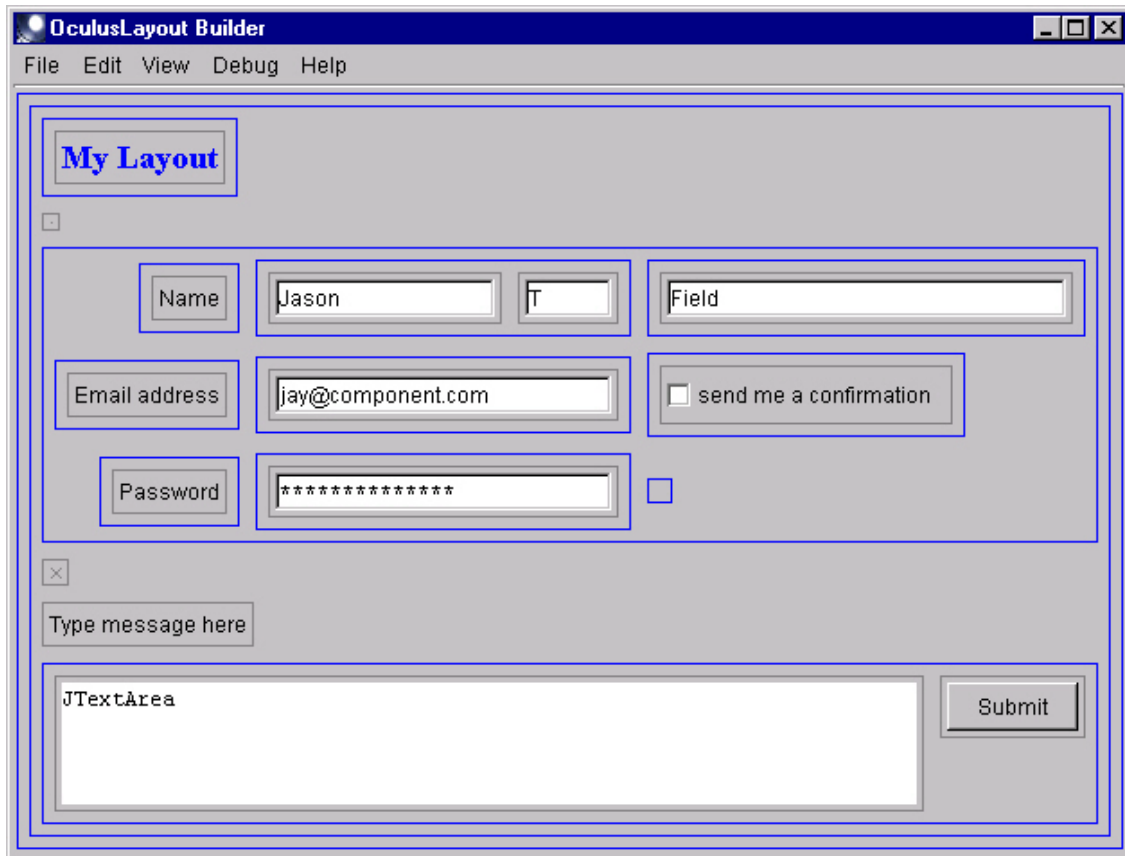



## The current example in view mode



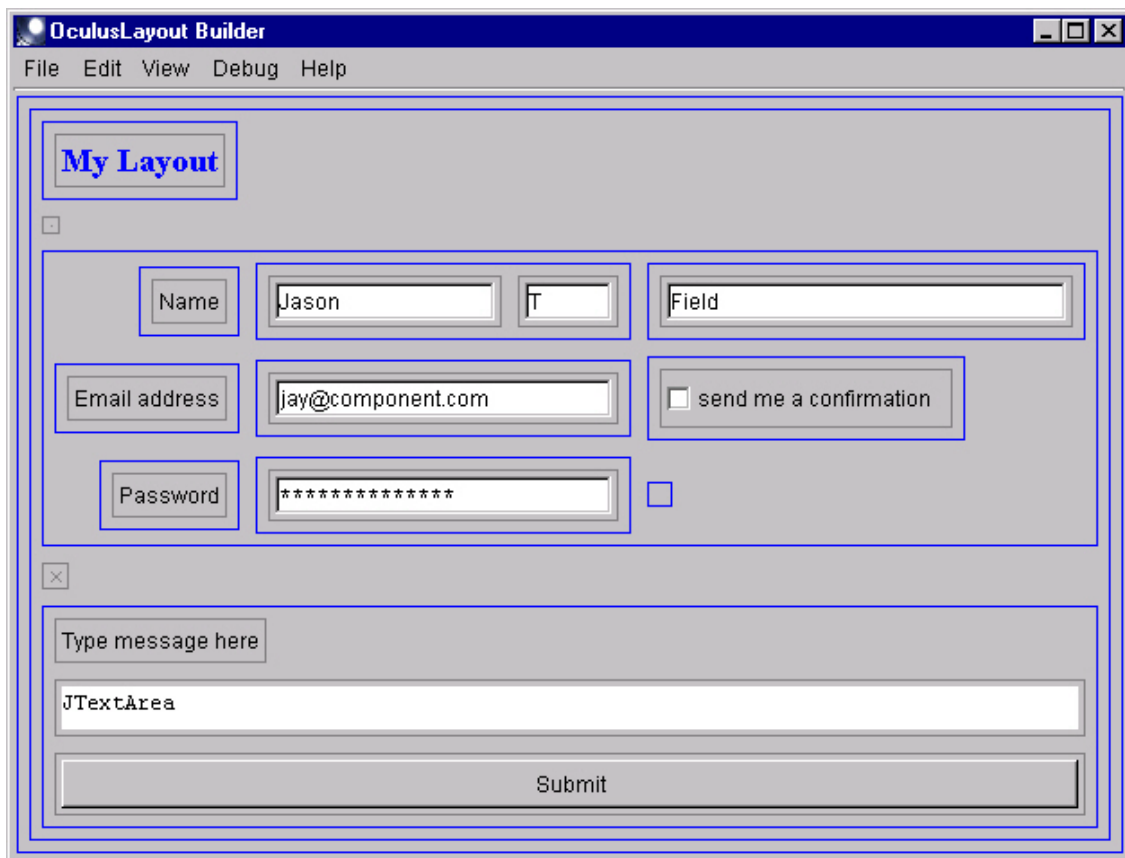
## Spaces and Filler Spaces

When out of edit mode, the layout looks a little cramped. Add one **Space**  below the title and another below the grid. Change the preferred size of the lower space to be 15 x 15 pixels (although the width doesn't matter in this case). Also, right-justify the three JLabels in the left column of the grid by selecting each appropriate grid cell and changing the **Grid cell horizontal justification** to **Right**.

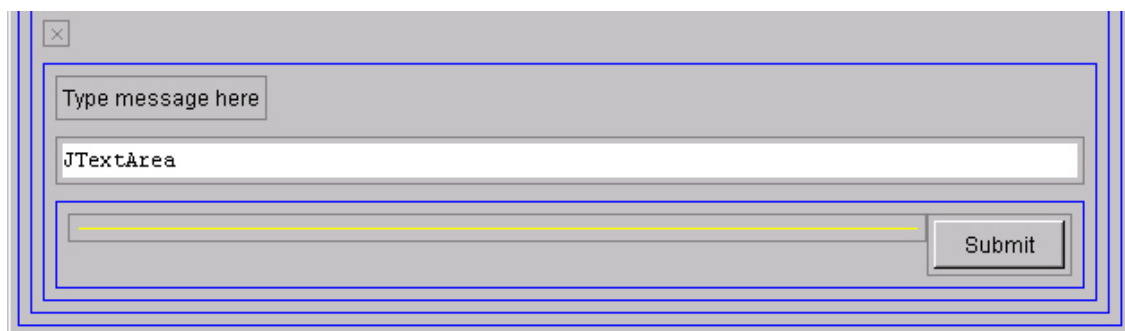


The Submit button is taking up too much space on the right side, so move it below the text area. The “Type message here” label has also been moved into that vertical box. To prevent the button from stretching horizontally, add a **Filler Space**  to the left of the button.

### The button is stretched



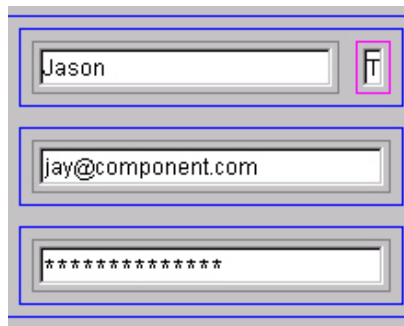
### Filler Spaces help with alignment and stretching



### Stretching and Justification

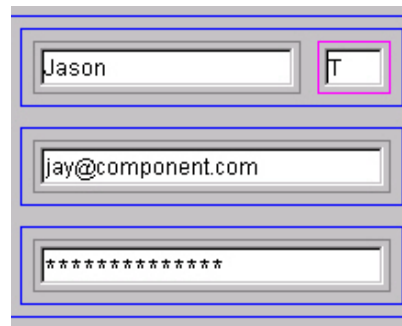
Since the middle initial field does not need to be long, set the **X Stretching** preference to **None** and the **Preferred Width** to 40 pixels. This will fix that JTextField's width.

### No X Stretching



A form layout with three input fields. The first field contains 'Jason' and has a small 'F' icon to its right. The second field contains 'jay@component.com'. The third field contains '\*\*\*\*\*'. The fields are arranged vertically and are not stretched to fit the container.

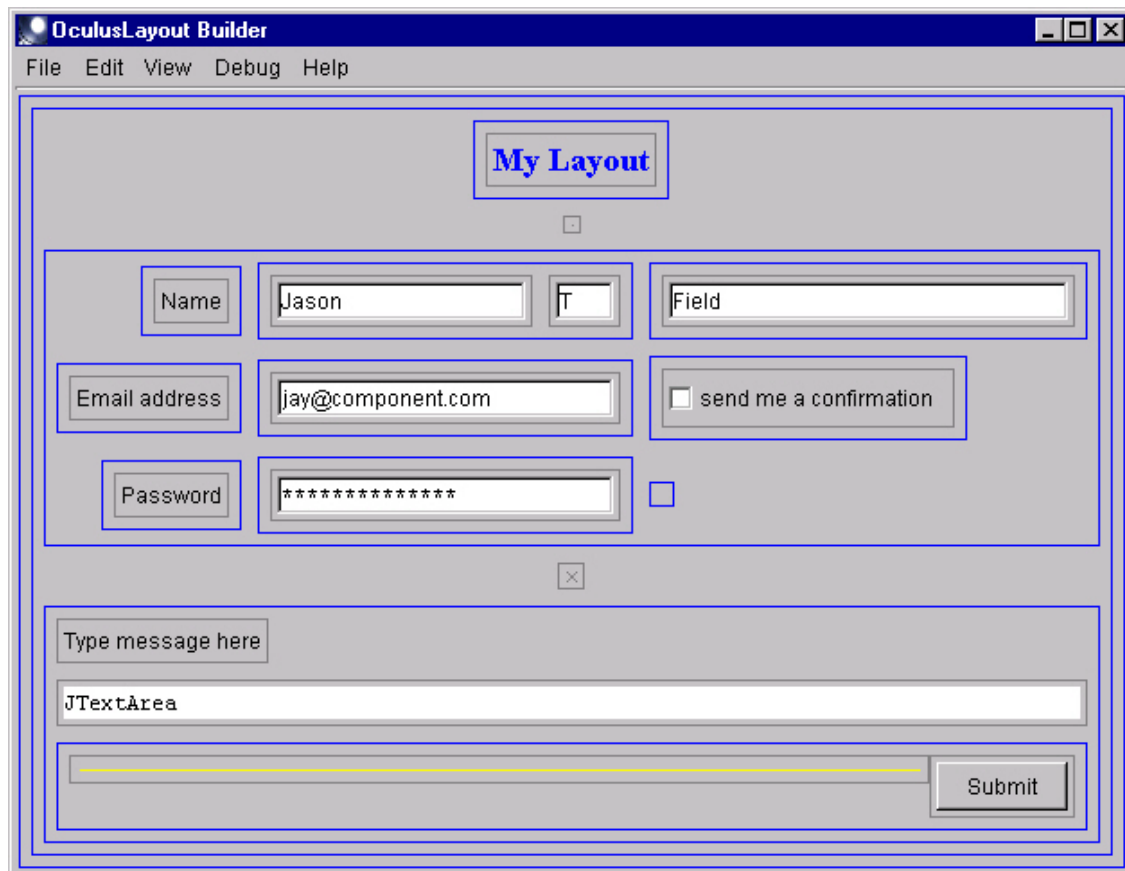
### Fixed width



A form layout with three input fields. The first field contains 'Jason' and has a small 'F' icon to its right. The second field contains 'jay@component.com'. The third field contains '\*\*\*\*\*'. The fields are arranged vertically and have a fixed width, with the 'F' icon to the right of the first field.

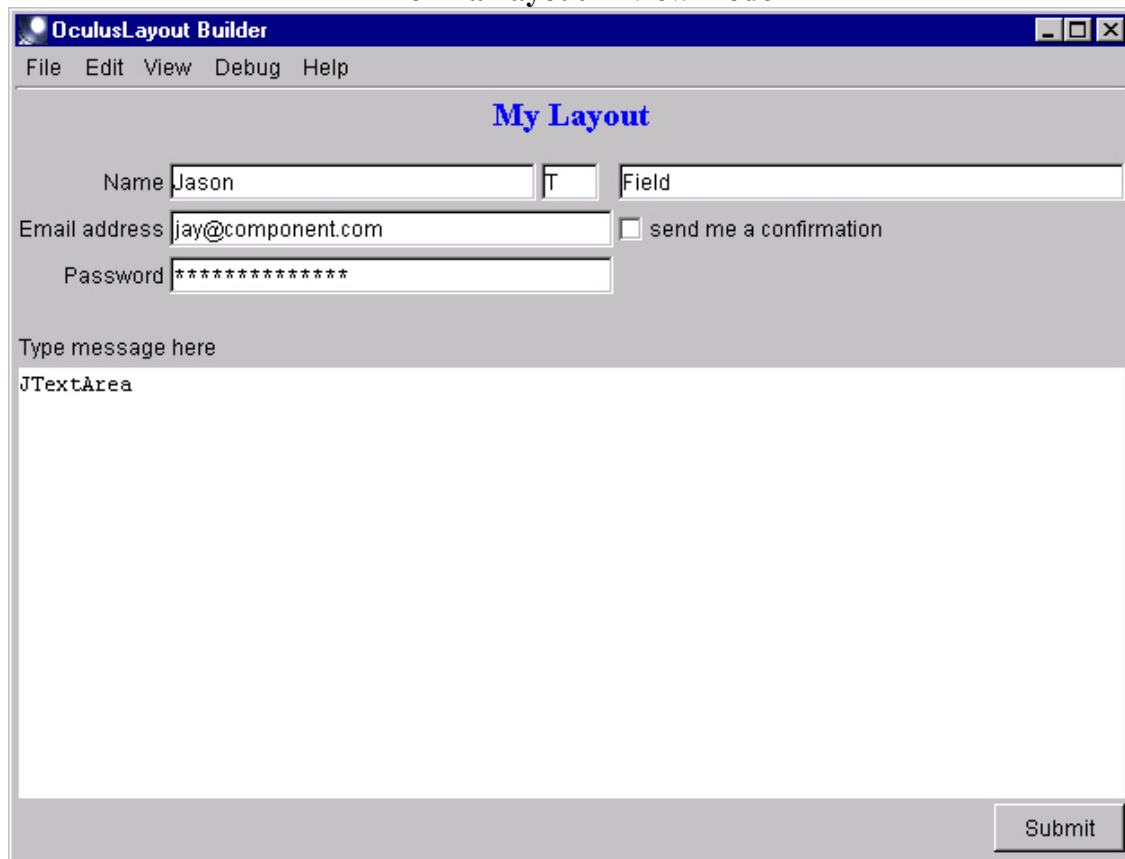
Center the main label by selecting one of the boxes around it and setting the **Box auxiliary dimension justification** to **Center**. Alternatively, putting Filler Spaces on either side of the label would have the same effect.

### The final layout



A screenshot of the OculusLayout Builder window showing a form layout titled 'My Layout'. The form contains several input fields and a submit button. The fields are arranged in a grid-like structure. The first row has a 'Name' label, a text field with 'Jason', a small 'F' icon, and a 'Field' label. The second row has an 'Email address' label, a text field with 'jay@component.com', and a checkbox labeled 'send me a confirmation'. The third row has a 'Password' label, a text field with '\*\*\*\*\*', and a checkbox. Below these fields is a 'Type message here' label and a large text area labeled 'JTextArea'. At the bottom right is a 'Submit' button.

### The final layout in view mode



### ***Other Builder Features***

You can also use the Builder's Plugin API to add your own Java components to the Palette so that you can build layouts with them. See the Builder API javadocs for more information. Also see the `examples` and `exampleplugins` directories for sample builder files and plugin code.