

OculusLayout API Tutorial

How to Use the OculusLayout API

User interfaces built with OculusLayout tend to flow together. Understanding how OculusLayout works on a conceptual level will help you understand why. Once you get the behind-the-scenes concept, you'll see how easy it is to build quality user interfaces quickly.

How OculusLayout Works

OculusLayout is basically a series of nested boxes and grids that tries to be smart about alignment and stretching. (There are of course ways to override the smartness if it's too smart or not quite smart enough for you in a particular circumstance.)

Boxes (Java class *OculusBox*) can have either horizontal or vertical orientation. These containers lay out components in a left to right or top to bottom configuration, respectively. Additionally, components can be aligned or justified along the auxiliary axis of the box (the auxiliary axis is the opposite of the box's orientation; i.e., a vertical box has a horizontal auxiliary axis).

Grids (Java class *OculusGrid*) are two-dimensional arrays of components with a specified number of rows and columns. Each row and column completely encompasses its components, but individual grid cells can specify horizontal and vertical justifications for their contained component. Rows and columns will automatically stretch to accommodate the sizes of their components.

In the code, user interfaces are built with an *OculusLayoutHelper* instance. OculusLayoutHelper uses a cursor-based system to assist you in laying out complicated, nested structures. As you add each component to your layout, the cursor advances to the next position in the layout; when you add containers (like boxes or grids), the cursor moves into the nested container, and subsequently added components will be placed into the new container. Then, after you have filled out the nested container, you can pop the cursor back out to its parent container at the position immediately following the newly added container. You can then continue adding components to the parent container, and so on. The cursor will move from left to right in a horizontal box; from top to bottom in a vertical box; and from left to right and then top to bottom in a grid.

In order to avoid having regular warning dialogs pop up when you run your program, you will have to specify a valid OculusLayout license key in your code. See the first example for details.

Examples

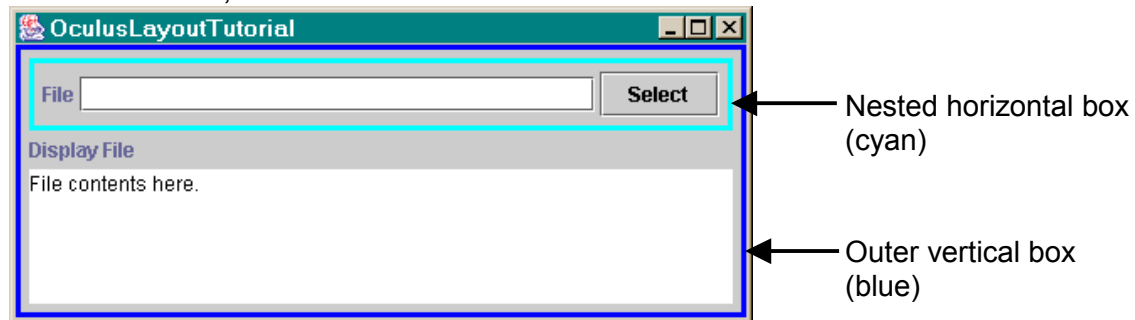
Here we'll present five examples of using OculusLayout that cover the basic concepts through the finer nuances.

Nesting Boxes

In this example, you will see:

- how to specify a license key;
- how to nest boxes;
- how vertical and horizontal boxes are laid out;
- how the parent method is used; and
- how to get the top-level box of the layout.

In the UI below, we've nested a horizontal box within a vertical box.



Here's the code for the UI:

```
// Set the license key
OculusLayout.setLicenseNumber("Your license key here");

// Create some components
JLabel fileLabel = new JLabel("File");
JTextField file = new JTextField(10);
JButton fileButton = new JButton("Select");
JLabel displayLabel = new JLabel("Display File");
JTextArea display = new JTextArea("File contents here.");

// Lay the components out. The top-level box will be vertically
// oriented.
OculusLayoutHelper layout =
    new OculusLayoutHelper(OculusLayout.VERTICAL);

// Add an outer border to the layout
layout.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(Color.blue, 3),
    BorderFactory.createEmptyBorder(5,5,5,5)));

// Nest a horizontal box in the current cursor position as the first
// component
layout.nestBox(OculusLayout.HORIZONTAL);
{
```

```

        layout.addBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createLineBorder(Color.cyan, 3),
            BorderFactory.createEmptyBorder(5,5,5,5)));
        // Now components are added to the nested horizontal box
        layout.add(fileLabel);
        layout.add(file);
        layout.add(fileButton);
        layout.parent(); // Pop out to the vertical box
    }
    // Now components are added to the vertical box following the nested
    // horizontal box
    layout.add(displayLabel);
    layout.add(display);

    // Put our layout contents into a JFrame.
    // getRoot() returns a reference to the top-most box in the layout.
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(layout.getRoot(), BorderLayout.CENTER);

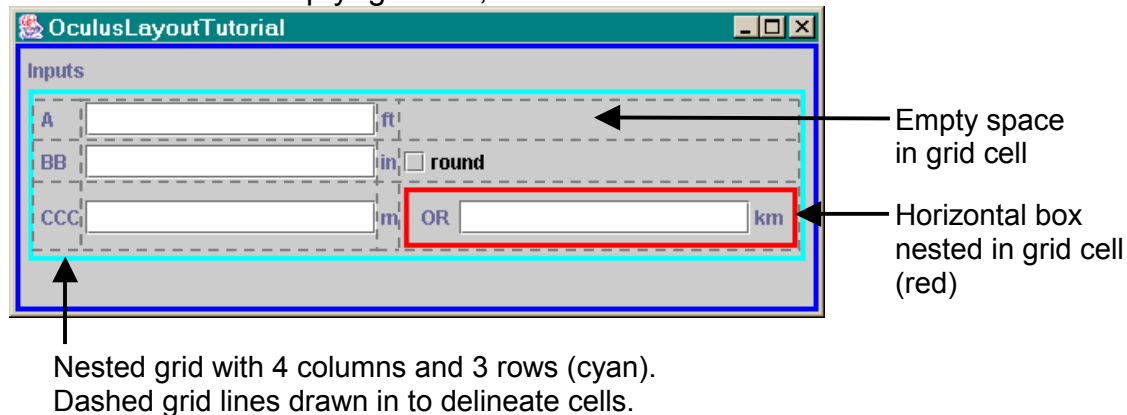
```

Nesting Grids and Using Spaces as Placeholders

In this example, you will see:

- how components are easily added to grids left to right, top to bottom;
- how to use spaces as placeholders; and
- how grids and boxes can be nested together.

In the UI below, we've used a grid to align labels and fields nicely. A space is used to fill in the "empty" grid cell, and a box has been nested in one of the cells.



Here's the code for the UI:

```

// Create some components
JLabel mainLabel = new JLabel("Inputs");
JLabel labelA = new JLabel("A");
JLabel labelB = new JLabel("BB");
JLabel labelC = new JLabel("CCC");
JLabel postLabelA = new JLabel("ft");
JLabel postLabelB = new JLabel("in");
JLabel postLabelC = new JLabel("m");
JLabel postLabelC2 = new JLabel("km");
JLabel orLabel = new JLabel(" OR ");

```

```

JTextField a = new JTextField(10);
JTextField b = new JTextField(10);
JTextField c = new JTextField(10);
JTextField c2 = new JTextField(10);
JCheckBox round = new JCheckBox("round");

// Lay the components out. The top-level box will be vertically
// oriented.
OculusLayoutHelper layout = new
    OculusLayoutHelper(OculusLayout.VERTICAL);

// Add an outer border to the layout
layout.addBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(Color.blue, 3),
    BorderFactory.createEmptyBorder(5,5,5,5)));
layout.add(mainLabel);
layout.nestGrid(4, 3); // Nest a grid with 4 columns and 3 rows
{
    layout.addBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(Color.cyan, 3),
        BorderFactory.createEmptyBorder(5,5,5,5)));
    // Now components are added to the nested grid, starting with the
    // top row, from left to right.
    layout.add(labelA);
    layout.add(a);
    layout.add(postLabelA);
    // A Space is added to the fourth cell in the top row just to
    // fill the spot and move the cursor along.
    layout.addSpace(5);
    // After the four cells in the top row are filled, the cursor
    // position automatically shifts to the beginning of the second
    // row down.
    layout.add(labelB);
    layout.add(b);
    layout.add(postLabelB);
    layout.add(round);
    // And now the third row...
    layout.add(labelC);
    layout.add(c);
    layout.add(postLabelC);
    // The fourth component in the bottom row is itself a nested box.
    layout.nestBox(OculusLayout.HORIZONTAL);
    {
        layout.addBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createLineBorder(Color.red, 3),
            BorderFactory.createEmptyBorder(5,5,5,5)));
        layout.add(orLabel);
        layout.add(c2);
        layout.add(postLabelC2);
    }
}

// Put our layout contents into a JFrame.
// getRoot() returns a reference to the top-most box in the layout.
getContentPane().setLayout(new BorderLayout());
getContentPane().add(layout.getRoot(), BorderLayout.CENTER);

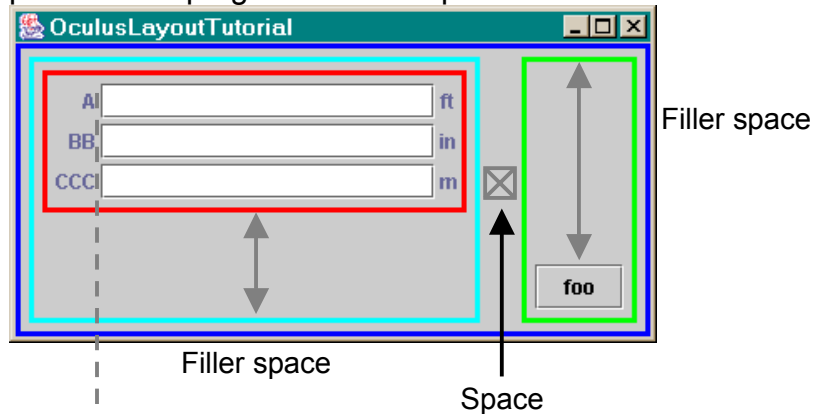
```

Using Simple Alignment Techniques

In this example, you will learn about:

- how to use spaces and fillers to achieve alignment;
- how to customize grid cell alignment; and
- how to customize justification.

In the UI below, we've used a space to separate the two vertical boxes (cyan and green ones). Fillers are used to push the grid and button to the top and bottom of the pane, respectively. Also, the labels *A*, *BB*, and *CCC* are right-justified to push them up against their respective text fields.



Here's the code for the UI:

```
// Create some components
JLabel labelA = new JLabel("A");
JLabel labelB = new JLabel("BB");
JLabel labelC = new JLabel("CCC");
JLabel postLabelA = new JLabel("ft");
JLabel postLabelB = new JLabel("in");
JLabel postLabelC = new JLabel("m");
JTextField a = new JTextField(10);
JTextField b = new JTextField(10);
JTextField c = new JTextField(10);
JButton foo = new JButton("foo");

// Lay the components out. The top-level box will be horizontally
// oriented.
OculusLayoutHelper layout = new
    OculusLayoutHelper(OculusLayout.HORIZONTAL);

// Add an outer border to the layout
layout.addBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(Color.blue, 3),
    BorderFactory.createEmptyBorder(5, 5, 5, 5)));
layout.nestBox(OculusLayout.VERTICAL);
{
```

```

layout.addBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(Color.cyan, 3),
    BorderFactory.createEmptyBorder(5,5,5,5)));
layout.nestGrid(3, 3); // Nest a grid
{
    layout.addBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(Color.red, 3),
        BorderFactory.createEmptyBorder(5,5,5,5)));
    // Right-justify the label so it is up against the text
    // field. This is the first component in the top row.
    layout.setGridCellJustification(OculusGrid.RIGHT_JUSTIFY,
        OculusGrid.DEFAULT_JUSTIFICATION);
    layout.add(labelA);
    layout.add(a);
    layout.add(postLabelA);
    // Right-justify the label so it is up against the text
    // field. This is the first component in the middle row.
    layout.setGridCellJustification(OculusGrid.RIGHT_JUSTIFY,
        OculusGrid.DEFAULT_JUSTIFICATION);
    layout.add(labelB);
    layout.add(b);
    layout.add(postLabelB);
    // Right-justify the label so it is up against the text
    // field. This is the first component in the bottom row.
    layout.setGridCellJustification(OculusGrid.RIGHT_JUSTIFY,
        OculusGrid.DEFAULT_JUSTIFICATION);
    layout.add(labelC);
    layout.add(c);
    layout.add(postLabelC);
    layout.parent(); // Pop out to the vertical box
}
// Add filler to the vertical box after the grid to push the grid
// to the top. By default, the grid would be centered in the
// vertical axis.
layout.addFiller();
layout.parent(); // Pop out to the horizontal box
}
// Separate the preceding vertical box and the following vertical box
// by 25 pixels
layout.addSpace(25);
layout.nestBox(OculusLayout.VERTICAL);
{
    layout.addBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(Color.green, 3),
        BorderFactory.createEmptyBorder(5,5,5,5)));
    // Add filler to the vertical box first to push the button to the
    // bottom
    layout.addFiller();
    layout.add(foo);
}

// Put our layout contents into a JFrame.
// getRoot() returns a reference to the top-most box in the layout.
getContentPane().setLayout(new BorderLayout());
getContentPane().add(layout.getRoot(), BorderLayout.CENTER);

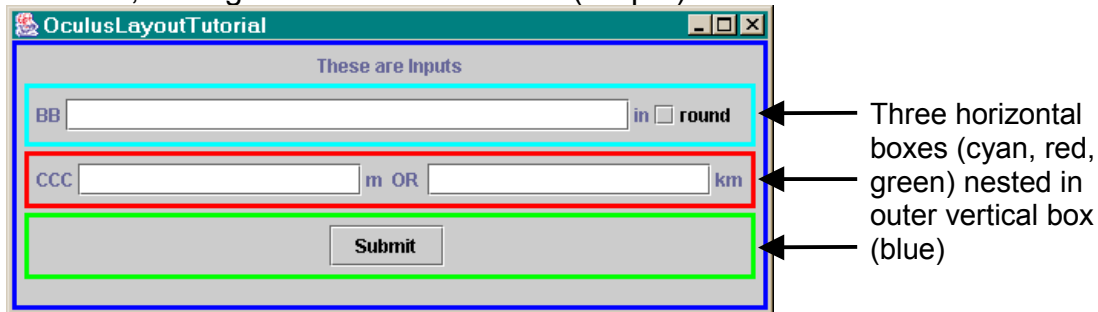
```

Aligning Components Across Boxes

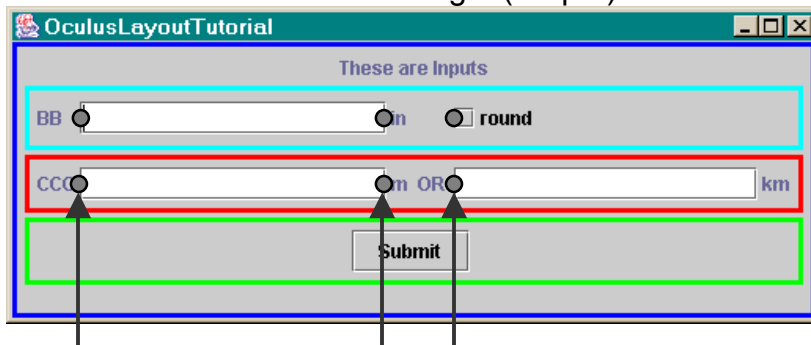
In this example, you will see:

- how to align components with alignment points and
- how to align components by edges.

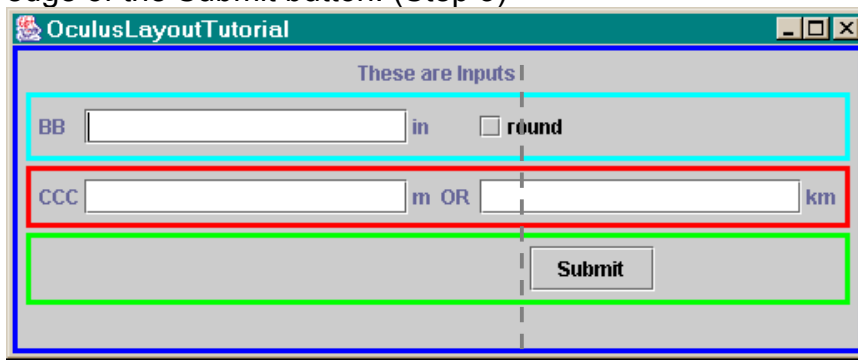
We'll do this in stages so you can see how alignment affects layout. First, in the UI below, no alignment has been done. (Step 1)



Next, we've used alignment points to line up the text fields. The first alignment points in each horizontal box (which are at the same box level) align, then the second alignment points in each of the boxes align, and so on. Alignment points at different box levels do not align. (Step 2)



Next, we've used edge alignment to line up components at different box levels. Here, the trailing edge of the *These are Inputs* label lines up with the leading edge of the Submit button. (Step 3)



Edge alignment of components

Here's the code for the final UI:

```
// Create some components
JLabel mainLabel = new JLabel("These are Inputs");
JLabel labelB = new JLabel("BB");
JLabel labelC = new JLabel("CCC");
JLabel postLabelB = new JLabel("in");
JLabel postLabelC = new JLabel("m");
JLabel postLabelC2 = new JLabel("km");
JLabel orLabel = new JLabel(" OR ");
JTextField bField = new JTextField(10);
JTextField cField = new JTextField(10);
JTextField c2Field = new JTextField(10);
JCheckBox round = new JCheckBox("round");
JButton submitButton = new JButton("Submit");

// Lay the components out. The top-level box will be vertically
// oriented.
OculusLayoutHelper layout = new
    OculusLayoutHelper(OculusLayout.VERTICAL);

layout.addBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(Color.blue, 3),
    BorderFactory.createEmptyBorder(5,5,5,5)));
// By default, the main vertical box would be left-justified. This
// will center its components.
layout.setJustification(OculusLayout.JUSTIFY_CENTER);
layout.add(mainLabel);
layout.nestBox(OculusLayout.HORIZONTAL);
{
    layout.addBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(Color.cyan, 3),
        BorderFactory.createEmptyBorder(5,5,5,5)));
    layout.add(labelB);
    // Step 2: add alignment points before and after the text field,
    // and before the check box. These points will only affect the
    // layout in conjunction with alignment points in another box.
    layout.addAlignmentPoint(); // Step 2: 1st point in cyan box
    layout.add(bField);
    layout.addAlignmentPoint(); // Step 2: 2nd point in cyan box
    layout.add(postLabelB);
    layout.addAlignmentPoint(); // Step 2: 3rd point in cyan box
    layout.add(round);
    layout.parent();
}
layout.nestBox(OculusLayout.HORIZONTAL);
{
    layout.addBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(Color.red, 3),
        BorderFactory.createEmptyBorder(5,5,5,5)));
    layout.add(labelC);
    // Step 2: These first two alignment points will align the
    // beginnings and ends, respectively, of the text fields in this
    // box with those in the previous horizontal box.
    layout.addAlignmentPoint(); // Step 2: 1st point in red box
    layout.add(cField);
    layout.addAlignmentPoint(); // Step 2: 2nd point in red box
```

```

        layout.add(postLabelC);
        layout.add(orLabel);
        // Step 2: This third alignment point will align the check box
        // with the right-most text field.
        layout.addAlignmentPoint(); // Step 2: 3rd point in red box
        layout.add(c2Field);
        layout.add(postLabelC2);
        layout.parent();
    }
    layout.nestBox(OculusLayout.HORIZONTAL);
    {
        layout.addBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createLineBorder(Color.green, 3),
            BorderFactory.createEmptyBorder(5,5,5,5)));
        layout.addFiller();
        // Step 3: Align the beginning (leading edge) of the submit
        // button with the end (trailing edge) of mainLabel
        layout.alignNextComponentTo(mainLabel,
            AlignedComponentSpacing.TRAILING_EDGE,
            AlignedComponentSpacing.LEADING_EDGE);

        layout.add(submitButton);
        layout.addFiller();
    }

    // Put our layout contents into a JFrame.
    // getRoot() returns a reference to the top-most box in the layout.
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(layout.getRoot(), BorderLayout.CENTER);

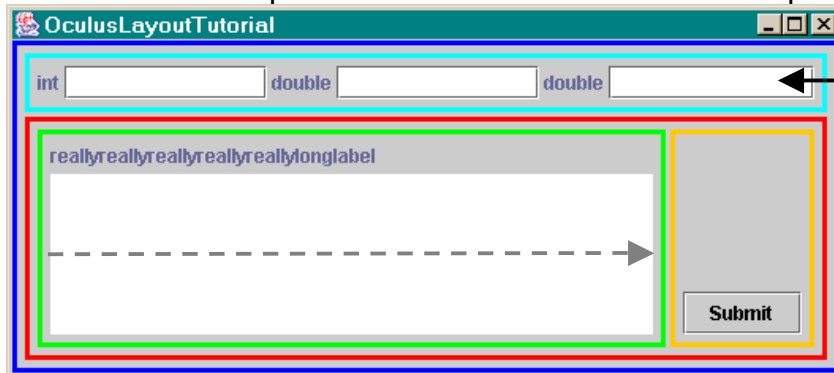
```

Customizing Stretching

In this example, you will learn how to customize the stretching options.

See the *OculusLayoutConstraints* Javadocs for a list of default stretching preferences for common swing components.

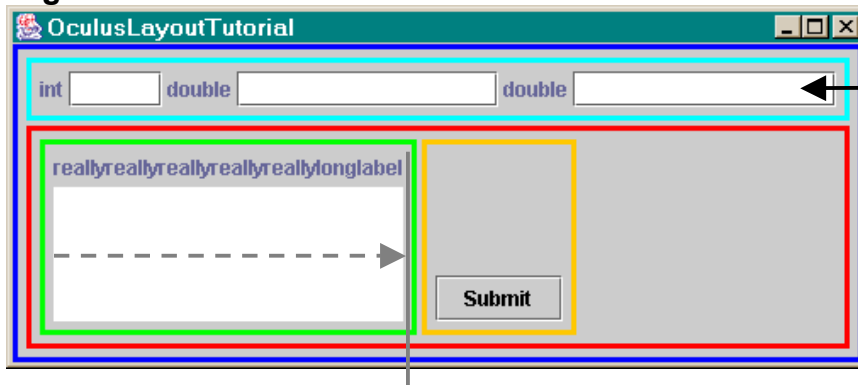
In the UI below, the default stretching preferences are used. Notice that the three text fields across the top take up equal space and that the text area stretches to take up all leftover vertical and horizontal space.



All three text fields stretch equally in the horizontal direction.

The text area stretches as wide as it can, consuming all the available width.

Now, we've told the two double fields that they **want to be stretched**, while the integer field only **can be stretched**. Since it will only take and display integers, it doesn't need to be as wide as the other fields. Also, we've specified that the text area should only be as wide as its label – it should only horizontally **stretch to align**.



The two double fields stretch equally to consume all the available width. The integer field does not stretch because it has a lower stretching preference.

The text area stretches only wide enough to align with other components in the vertical box (green). In this case, the label is defining the width of the vertical box.

Here's the code for the final UI:

```
// Create some components
JLabel intLabel = new JLabel("int");
JLabel doubleLabel = new JLabel("double");
JLabel doubleLabel2 = new JLabel("double");
JLabel longLengthLabel = new JLabel(
    "reallyreallyreallyreallyreallyreallylonglabel");
JTextField intField = new JTextField(5);
JTextField doubleField = new JTextField(5);
JTextField doubleField2 = new JTextField(5);
JTextArea textArea = new JTextArea();
JButton submitButton = new JButton("Submit");

// Lay the components out. The top-level box will be vertically
// oriented.
OculusLayoutHelper layout = new
    OculusLayoutHelper(OculusLayout.VERTICAL);

layout.addBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createLineBorder(Color.blue, 3),
    BorderFactory.createEmptyBorder(5,5,5,5)));
layout.nestBox(OculusLayout.HORIZONTAL);
{
    layout.addBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(Color.cyan, 3),
        BorderFactory.createEmptyBorder(5,5,5,5)));
    layout.add(intLabel);
    // The first text field is added with the default stretching
    // preferences - CAN_BE_STRETCHED horizontally and NO_STRETCH
    // vertically.
    layout.add(intField);
    layout.add(doubleLabel);
```

```

// The second and third text field are given higher stretching
// preferences - they want to be stretched horizontally.
layout.add(doubleField, OculusLayoutInfo.WANT_STRETCHED,
           OculusLayoutInfo.NO_STRETCH);
layout.add(doubleLabel2);
layout.add(doubleField2, OculusLayoutInfo.WANT_STRETCHED,
           OculusLayoutInfo.NO_STRETCH);
layout.parent();
}
layout.nestBox(OculusLayout.HORIZONTAL);
{
    layout.addBorder(BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(Color.red, 3),
        BorderFactory.createEmptyBorder(5,5,5,5)));
    layout.nestBox(OculusLayout.VERTICAL);
    {
        layout.addBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createLineBorder(Color.green, 3),
            BorderFactory.createEmptyBorder(5,5,5,5)));
        layout.add(longLengthLabel);
        // The text area is given a lower horizontal stretching
        // preference (STRETCH_ONLY_TO_ALIGN) than its default
        // (CAN_BE_STRETCHED). Note that STRETCH_ONLY_TO_ALIGN is
        // applied to the auxiliary axis of the vertically-oriented
        // parent box.
        layout.add(textArea,
            OculusLayoutInfo.STRETCH_ONLY_TO_ALIGN,
            OculusLayoutInfo.CAN_BE_STRETCHED);
        layout.parent();
    }
    nestBox(OculusLayout.VERTICAL);
    {
        layout.addBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createLineBorder(Color.orange, 3),
            BorderFactory.createEmptyBorder(5,5,5,5)));
        layout.addFiller();
        layout.add(submitButton);
    }
}

// Put our layout contents into a JFrame.
// getRoot() returns a reference to the top-most box in the layout.
getContentPane().setLayout(new BorderLayout());
getContentPane().add(layout.getRoot(), BorderLayout.CENTER);

```