THE ARRANGEMENT OF THE SCREENS: INTRODUCTION TO LAYOUT IN ADF FACES

Peter Koletzke, Quovera

Yes ... the arrangement of the screens:
... Beyond the screens
That glide aside
Are further screens
That open wide
With scenes of screens like the ones that glide.

—Stephen Sondheim (1930-), *Pacific Overtures* (1976)

Web page layout has always been a challenge; the items you place on the page can move around when the user resizes the window or when the page is displayed in a different screen resolution. Part of this challenge results from the limitations of Hypertext Markup Language (HTML), the main vehicle for web content. In the not-so-distant past, constructing web pages with complex layouts usually meant relying on the alignment, spacing, and padding properties of HTML tag elements like tables and cells. It also meant using extensions to HTML such as images, JavaScript, and Cascading Style Sheet (CSS) code.

Modern solutions to the layout challenge have evolved into the form of web component frameworks that shift development work from the internals of squeezing and stretching HTML to relying on prebuilt layout features of a component set, such as Application Development Framework Faces Rich Client (ADF Faces RC or just ADF Faces). *ADF Faces* is a JavaServer Faces (JSF) framework that provides a large collection of rich user interface controls. As with other ADF frameworks, JDeveloper 11*g* supports ADF Faces development work better than any other Java tool, although the Oracle Enterprise Pack for Eclipse now supplies ADF Essentials to the Eclipse Java tool.

This white paper discusses how you would approach the task of achieving a desired web page layout using ADF Faces components. ADF Faces offers a wide variety of choices for placing elements on the page using layout components. It also provides properties that allow you to program in a declarative way rather than by writing lines of code to create the final layout. The white paper introduces the essentials for working in this declarative way with these components and their properties. It divides the information about ADF Faces layout features and techniques into the following subjects:

- **The basics**
- **The major components**
- **Hands-on practices for layout techniques**
- **Learning more**

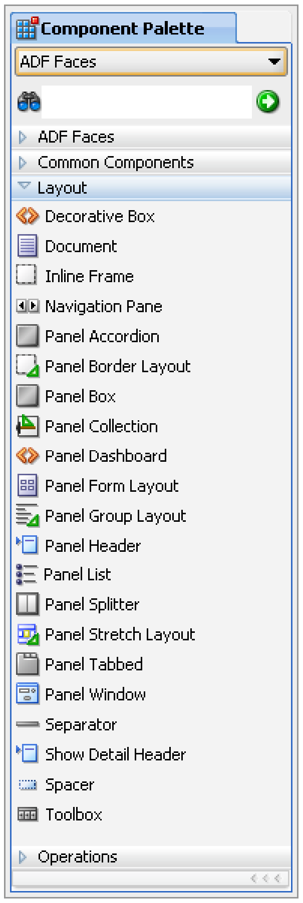
Note

The screenshots and practices in this white paper were developed with JDeveloper 11*g*, version 11.1.1.1.0. If you use a later release, the appearance may be slightly and you may need to adjust steps to changes introduced with that later release. However, the principles and major steps should remain the same regardless of the version.

The Basics

Just as you paint from a palette of colors, so too will you lay out web pages using a palette of ADF Faces components. Success in laying out a page using ADF Faces components requires at least a modest comfort level with the range and features of this palette. The palette consists of a large range of layout components and their facets and properties.

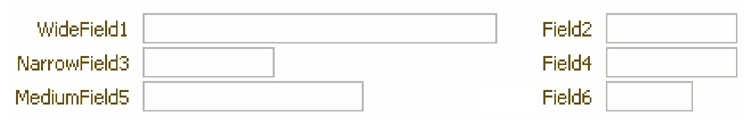
Layout Components



ADF Faces consists of several categories of controls such as *atomic components*, for rendering and user interfaces items such as fields and buttons, and *layout components*, which surround atomic components and other layout components and offer automatic layout features like field alignment. Knowledge of the ADF Faces layout components is a critical success factor in achieving the desired layout. Fortunately, the list of layout components is relatively small as shown in the illustration on the right, from the Layout panel of the ADF Faces Component Palette page in JDeveloper, where you can access most of the layout components.

The Component Palette shows available items using their common names, such as Panel Form Layout. When working with the component in JSF code, you use the programmatic name such as `af:panelFormLayout`. The prefix of the programmatic name starts with "af," which identifies the component as being located in the ADF Faces library. The next part of the designator is the component (tag) name; ADF Faces layout component names often start with "panel."

Each layout component supplies a certain behavior for the components within it. For example, the aforementioned `af:panelFormLayout`. allows you to lay out fields in columns without requiring any other embedded components. The fields are left-aligned, and their prompts are right-aligned. All fields are stacked on top of one another, but you can specify the number of columns and rows for a columnar arrangement as shown here.



This arrangement provides a familiar appearance for data entry needs. The fields are rendered down one column and then down the next column. This ordering implements a *tab order* (the order in which the cursor navigates fields when the TAB key is pressed) of down and across. (The section "Nesting Layout Components" later in this white paper discusses tab order.) Layout components are the main tool you will need to master for screen layout.

Properties

Each layout component offers default behavior that you can modify by assigning property values. For example, as discussed in the preceding description of `af:panelFormLayout`, layout component properties declare the behavior of items within the component. Another example, is the `af:panelGroupLayout` component, whose *Layout* property determines whether components within the layout component are stacked horizontally or vertically. Custom property settings contribute greatly to achieving the desired layout.

Facets

Layout components offer *facets*, components into which you place other components. If the facet is responsible for a visual effect, at runtime, the facet maintain relative positioning of its contents in relationship to other components. For example, the `af:panelFormLayout` layout component offers a footer facet that appears under all fields and is not bound by the alignment rules of the main area of the component.

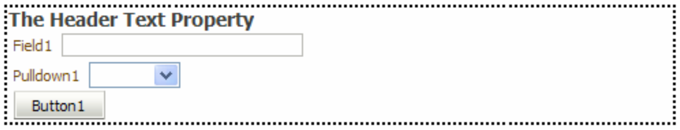
Note

Facets can contain only one component. However, the one component in a facet could be a layout component, which, in turn, can hold more than one component. JDeveloper leads by example in this case—if you drop a second component into a facet, instead of raising an error, JDeveloper will automatically add an `af:group` component around the two components. An `af:group` component is just used to group related components into a single unit but it offers no visual properties of its own.

The Major Components

You can get a sense of the depth and breadth of functionality and flexibility available in the ADF Faces layout components by knowing a bit about their features. The following is a summary of the functionality provided by the most commonly used layout components.

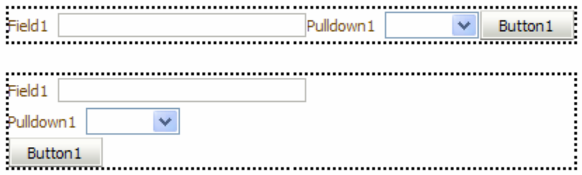
- **`af:panelHeaderLayout`** This component provides a title to denote a grouping of elements on the page. It offers properties for *Size* to set the height of the heading text, *MessageType* to automatically add an icon (information or warning, for example) to the left of the heading text, *Text* for the heading text, and others. Nesting components within this component will stack them vertically on the page as shown next. The *InlineStyle* property has been set to "border-color:Black; border-style:dotted; border-width:thin;" to make the edges of the component visible.



Tip

Setting the background color as well as the border style of layout components in this way is a useful debugging technique because it allows you to see the edges of each layout component.

- **`af:panelGroupLayout`** This component lays out components in a single row (horizontal layout) or column (vertical layout). The *Layout* property sets the orientation by which components in the component will be arranged: horizontal, vertical, or scroll (vertical with scrollbars). The following illustration shows an **`af:panelGroupLayout`** component with a *Layout* of "horizontal" and under it, another with the same components but with a *Layout* of "vertical." As with the preceding illustration, *InlineStyle* defines a dotted line to show the edge of the components.

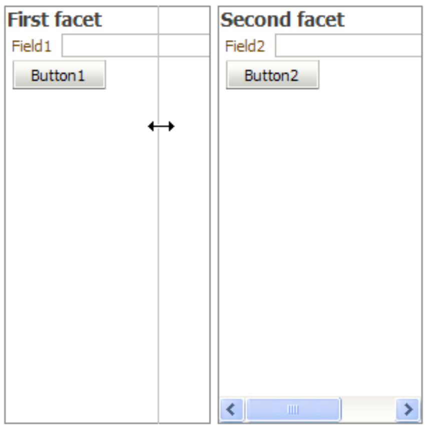


Tip

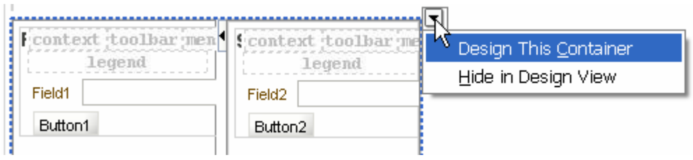
`af:panelGroupLayout` works best when you explicitly assign a *Layout* property value other than "default" rather than relying on the default.

- **`af:panelSplitter`** This component divides an area horizontally or vertically into two panels and optionally allows the user to move the dividing line between the panels to change the relative sizes of the panels. If the *Disabled* property is set to "true," the user will not be able to move the splitter bar. You drop components into the first and second facets in this component to assign them to one panel or the other. For example, if you define a horizontal splitter, the user can move the splitter bar (divider) between panels to the left or right to make the first panel narrower or wider. The illustration on the right shows this effect as the splitter bar is in motion (the mouse cursor is the double-headed arrow).

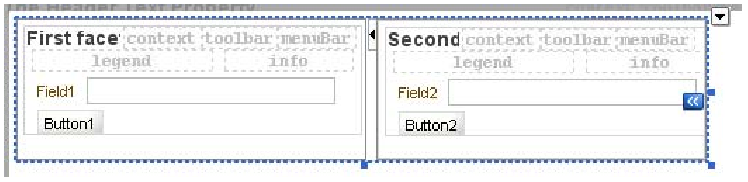
  The user can click the arrow icon on the splitter bar (not shown when the splitter bar is in motion, as it is in the preceding illustration) to collapse the first panel completely. Notice that the components in this display are truncated.



To help you solve layout issues such as this, JDeveloper offers a pulldown menu in the top-right corner of the component after selecting it in the visual editor as shown here:



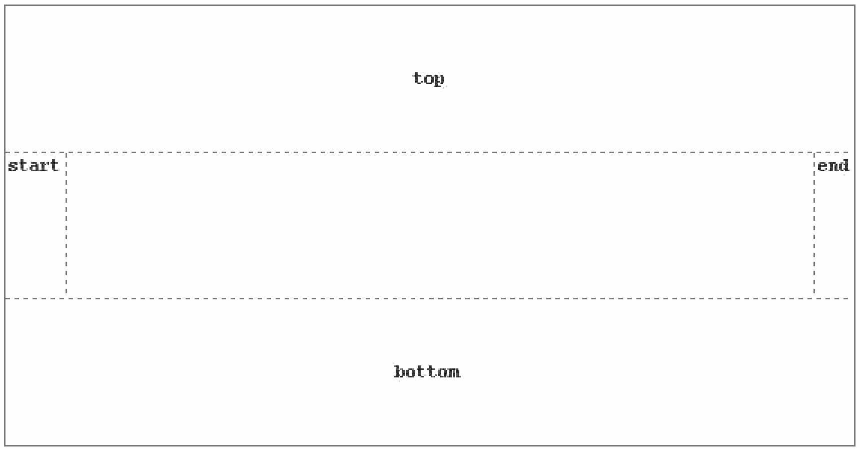
Selecting **Design This Container** opens an editor area that allows you to resize the component borders so that you could create a layout such as the following, where all components fit within the panels.



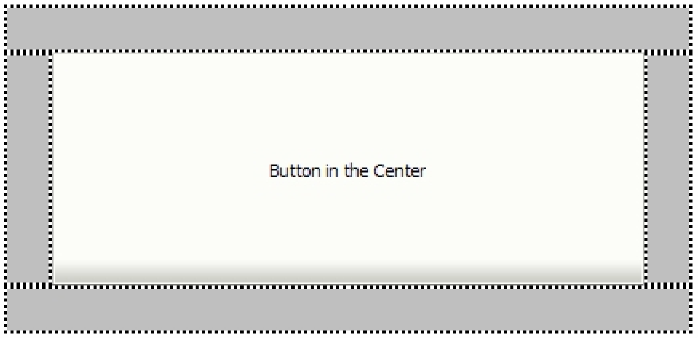
Note

This editor just sets the *InlineStyle* and *SplitterPositions* properties, but it allows you to control the size visually instead of by experimenting with property values.

- **`af:panelBorderLayout`** This layout component provides facets such as top, bottom, start, and end. Components dropped into one of these facets will maintain their relative positions at runtime. You can use this layout component for pages that require a logo in the header area (top facet) and copyright and informational text in the footer area (bottom facet). The facets are easiest to see in the visual editor as shown in the next screenshot.



- **`af:panelStretchLayout`** This component also offers facets for bottom, start, top, and end. It also provides a center area, which stretches the contained component to fill the available width. You would use this component if another component cuts off the display of a layout component or atomic component. This illustration shows this component with *StartWidth*, *EndWidth*, *TopWidth*, and *BottomWidth* properties set to "30." Each facet contains an `af:panelGroupLayout` component with a background color of "Silver" and dotted border to better display the facet areas. The center area contains a button with no properties set except *Text* (the label). Notice how the center facet stretches the button to fill the available space.



The "Design This Container" editor described earlier allows you to set the size of the facets surrounding the center facet of the `af:panelStretchLayout` component as well its overall size.

> **Tip**
>
> The `af:panelGroupLayout` component is only as large as its child components. Therefore, if you have trouble aligning components within this layout component, it could be because the `af:panelGroupLayout` component is not wide enough. Try surrounding it with an `af:panelStretchLayout` component, which will expand the `af:panelGroupLayout` component and align its child components properly.

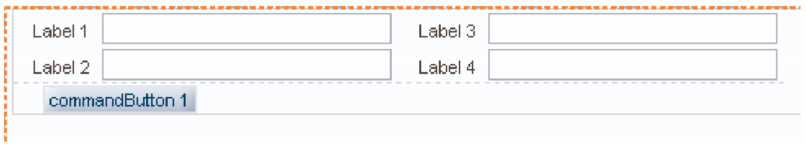
Follow these steps for a short demonstration of layout component properties and alignment capabilities:

1. Create a JDeveloper application using the Fusion Web Application template;
2. Create a JSF page and drop in a Panel Form Layout.
3. Within the layout component, add an Input Text component.

> **Tip**
>
> You can add quickly components into the proper layout component by selecting the component, then clicking the Component Palette icon for the atomic component. For example, to add Input Text items under the `af:panelFormLayout` component, select `af:panelFormLayout` in the Structure window and click Input Text in the Component Palette. If you need to add more components, repeat the selection and click operations.

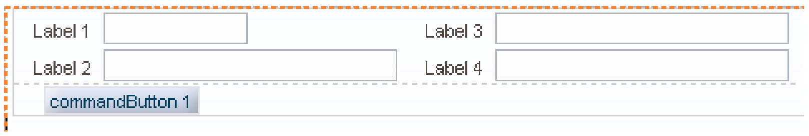
4. Add three more Input Text items under the layout component.
5. Select the footer facet and click Button in the Component Palette.
6. Navigate to the Property Inspector for the `af:panelFormLayout` component and set *MaxColumns* (in the Common region) as "2" and *Rows* as "2."

   Additional Information: Notice that the fields rearrange into two columns as shown next. The button retains its position at the bottom of the layout because it is placed in the footer facet.



7. On the first field, set the *Columns* property (Appearance region) as "10" to set the width of the field.

   Additional Information: Notice that the fields in the second column retain their horizontal position as shown next.

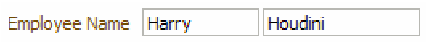


   This alignment occurs because the Label 3 field is in the second column, which is sized according to the widest field in the first column (Label 2).

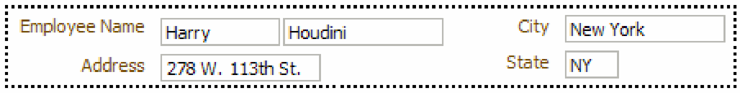
Other Common Layout Components

Although most of the layout components are available in the Layout panel of the Component Palette displayed earlier, you will find some additional layout components in the Common Components panel as in the following examples:

- **`af:form`** This component renders a submit form (in HTML, `<form>`) used to pass content to the server for processing. It is technically not a layout component because it does not provide layout characteristics, but it is a component that holds other components.
- **`af:menuBar`** This component provides an area into which you place pulldown menu components. This component is called "Panel Menu Bar" in the Component Palette.
- **`af:menu`** This component displays a pulldown menu and can be placed into a popup or menubar.
- **`af:popup`** This component has no visual aspect; it displays whatever you place in it on top of the page.
- **`af:dialog`** This component displays a window containing a border, one or more buttons, and a content area. This component must be placed inside an `af:popup` component.
- **`af:toolbar`** This component displays an area into which you place command components such as buttons. This component is usually placed inside another layout component—`af:toolbox` (in the Layout panel), which contains toolbars and menubars.
- **`af:panelLabelAndMessage`** This layout component supplies a label to its contents. For example, the `af:outputText` component does not have a *Label* property. If you want to use that component to display read-only data values in a form, you can surround the component with `af:panelLabelAndMessage` and set the layout component's *Label* property. Since you can place more than one component in a component, you could remove the labels on employee First Name and Last Name fields, surround those fields with `af:panelLabelAndMessage`, and fill in the component's Label as "Employee Name" as shown here:



If the Employee Name structure is contained in an `af:panelFormLayout`, the component will treat it as a single field and apply the label and field layout alignments as shown next. The `af:panelFormLayout` is assigned a dotted border for visibility in this illustration.

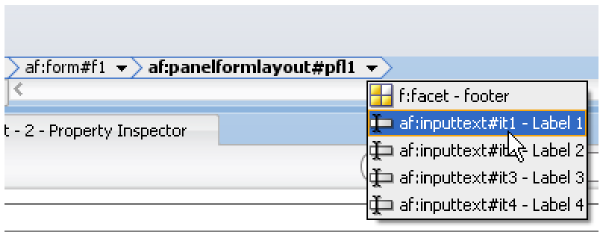


Tip

Set Simple as "true" for `af:inputText` field components inside the `af:panelLabelAndMessage` component so that the field labels are not displayed. This setting also causes the fields to lay out horizontally as shown in the preceding illustration.

Nesting Layout Components

As with other tag languages, such as HTML, you can embed layout components within layout components to take advantage of more than one type of arrangement. This is a useful and necessary tool for achieving the desired layout. For example, the next illustration shows an `af:panelGroupLayout` component (defined with *Layout* as "vertical") that surrounds three more `af:panelGroupLayout` components (defined with *Layout* as "horizontal"), each of which contains two `af:inputText` components. The *Columns* properties of the fields have been set differently to show the somewhat ragged alignment that will result.


af:form#f1
af:panelformlayout#pfl1
t - 2 - Property Inspector
f:facet - footer
af:inputtext#it1 - Label 1
af:inputtext#it2 - Label 2
af:inputtext#it3 - Label 3
af:inputtext#it4 - Label 4

Tip

Use the `af:spacer` component to add a fixed horizontal or vertical space between components. Enter values in the `af:spacer` properties "Width" and "Height" to designate a pixel width or height for the spacer, for example, "10px."

The tab order for an `af:panelGroupLayout` with Layout set to "horizontal" is across. The tab order for an `af:panelGroupLayout` set to "vertical" is down. Therefore, the preceding arrangement implements a tab order of across and down—within an inner horizontal `af:panelGroupLayout` component and between inner horizontal `af:panelGroupLayout` components. This type of layout provides an alternative to the `af:panelFormLayout` tab order of down and across. Techniques for implementing a tab order of across and down are explored in the following hands-on practice.

Tip

You can use the Structure window to navigate to a specific component. You can also use the editor's breadcrumbs (status bar) area to navigate to quickly select a component, regardless of its parent. Select the child component from the pulldown on the parent component as shown next.

Hands-on Practices for Layout Techniques

Now that you've gotten an idea of ADF Faces layout component features and functions, you can work with them to solve a specific layout problem. The following hands-on practices address an issue mentioned before (tab order of the `af:panelFormLayout` component). Although you may be happy with (or at least not bothered by) the default down-across tab order, these practices will show several techniques for creating an across-down tab order. The main objective of these practices is to show the types of development techniques you will use with the ADF Faces components and demonstrate how properties and nested components can help you create a specific layout design.

Note

Many of the techniques to make layout work faster and easier are worked into the discussions in this white paper, but some additional techniques appear in the presentation slides that accompany this white paper.

Hands-on Practice: Implement an Across and Down Tab Order

If you wanted to emulate the alignment characteristics of an `af:panelFormLayout` layout component in addition to implementing an across-and-down tab order, you could use these two alternative techniques:

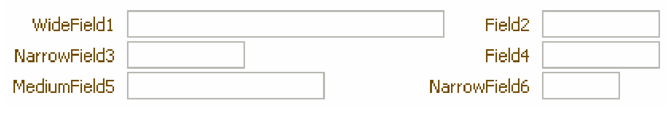
I. Use spacers to align the fields

II. Use properties to align the fields

This hands-on practice demonstrates both of these alternative techniques and mentions some of the benefits and drawbacks of each.

I. Use Spacers to Align the Fields

This technique consists of adding `af:spacer` components before and between fields to achieve the following layout:

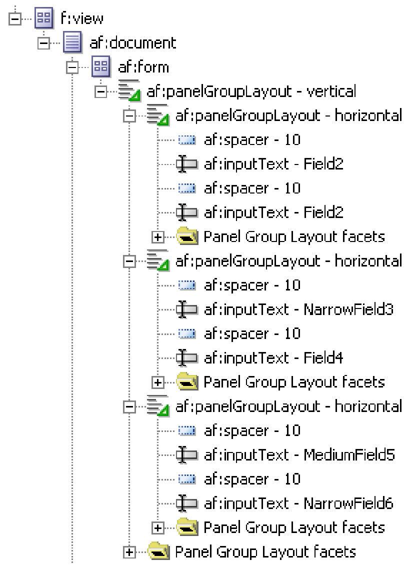


This arrangement looks similar to the arrangement of components within an `af:panelFormLayout` component. It has a more complicated setup (finding the widths required for each spacer takes experimentation) but allows you to provide a tab order that some users prefer.

8. In the ADFFacesTest application you used for the earlier hands-on practice, create a JSF page called tabOrder1.jspx. Specify that JDeveloper will not create a backing bean file.
9. Drop a Panel Group Layout component into the `af:form`. Set *Layout* to "vertical." Drop three more Panel Group Layout components on top of the first af:`panelGroupLayout` and set *Layout* for the three components to "horizontal." This will result in three rows inside the outer layout component.
10. In the first row, drag a Spacer component on top of the `af:panelGroupLayout` and set its *Width* property to "22"; then add an Input Text component and set *Columns* to "35"; add a Spacer and set its *Width* to "25"; and add a final Input Text and set *Columns* to "10."

    Repeat this process (using copy and paste if you want to) for the other two rows using the settings in the following table.

Row	**Field (*Columns*) or Spacer (*Width*)**
2	Spacer (11); Input Text (10); Spacer (150); Input Text (10)
3	Spacer (10); Input Text (20); Spacer (65); Input Text (5)



```
f:view
  af:document
    af:form
      af:panelGroupLayout - vertical
        af:panelGroupLayout - horizontal
          af:spacer - 10
          af:inputText - Field2
          af:spacer - 10
          af:inputText - Field2
          Panel Group Layout facets
        af:panelGroupLayout - horizontal
          af:spacer - 10
          af:inputText - NarrowField3
          af:spacer - 10
          af:inputText - Field4
          Panel Group Layout facets
        af:panelGroupLayout - horizontal
          af:spacer - 10
          af:inputText - MediumField5
          af:spacer - 10
          af:inputText - NarrowField6
          Panel Group Layout facets
      Panel Group Layout facets
```

Additional Information: The Structure window should appear as shown on the right (although the Structure window labels for your fields may be different at this point). Notice that the `af:spacer` nodes show the *Height* property (all defaulting to 10), not the *Width* property that you set. The *Columns* property represents character columns, not pixels.

11. Set the field labels as shown in the preceding illustration. The layout editor will not show the final runtime alignment although the **Preview in Browser** right-click menu option will.
12. Run the file to test the field alignments. Proper alignment will depend upon the skin you are using (discussed in the section "Working with Skins" later on). If needed, so adjust the properties so the field alignment matches the earlier illustration. Resize the window width to test variations on the window size.

    Additional Information: Since the field labels are single words, the alignment will be maintained. If you change the labels to multiple words, they would wrap if the window width became too small; this would disrupt the field alignment. Since the field labels are used to maintain field alignment, if the field labels need to be changed dynamically based on the user's language, the work required for this technique becomes nearly unmanageable.
13. Close the browser.

What Did You Just Do?

You set up nested layout components with specific alignments to implement an across and down tab order. The problem with this layout arrangement is that it requires experimentation to find the correct size for the spacers. Another problem mentioned before is the dependency on the width of field labels, which may change based on the user's language and which are subject to wrapping if the window is narrowed sufficiently.

Caution

This alignment technique may not produce consistent results on all versions of all browsers. It is always a good idea to try your application with different browsers to verify the visual effects.

II. Use Properties to Align the Fields

An alternative technique for implementing an across and down tab order, but still maintaining the alignment of fields between rows, is to use multiple `af:panelFormLayout` components, each of which displays a single row. If you set each row's layout component with the same space allocated to the field width and label width, you can produce a vertical alignment of fields, even if the fields and field labels are different widths. The following steps use this technique to create the layout shown in Figure 1.

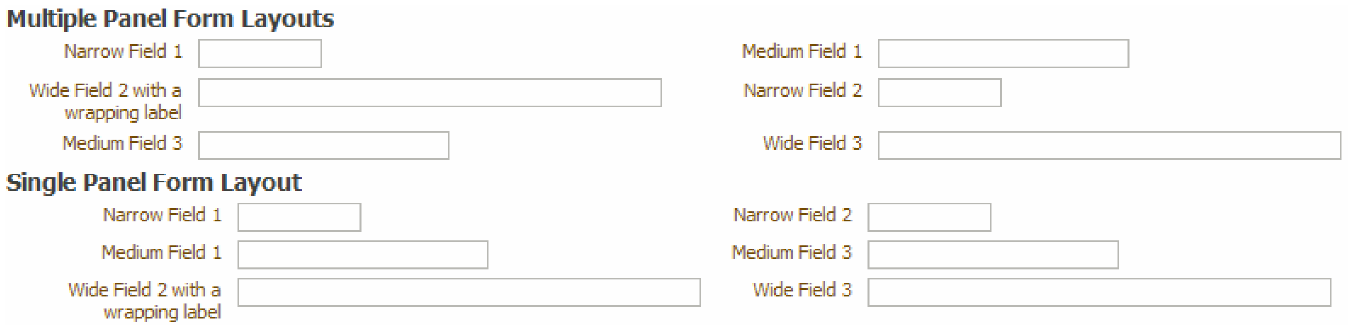


Figure 1. af:panelFormLayout component tab order demonstration

Caution

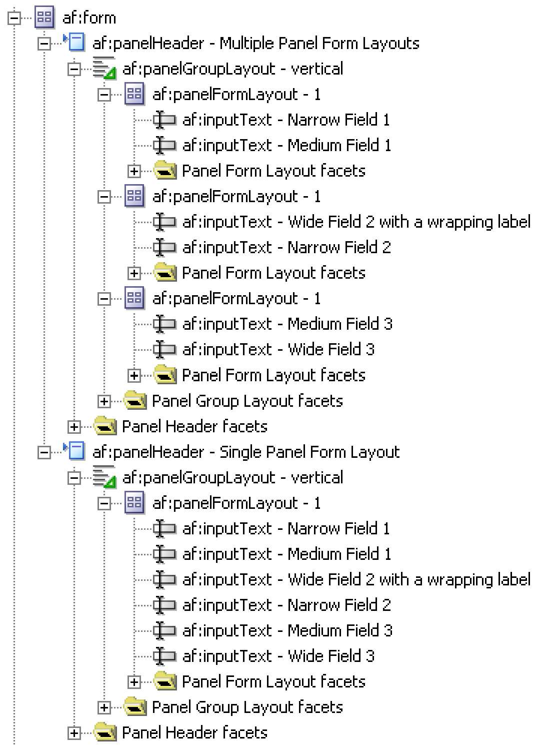
This is not a complete functional equivalent to the single `af:panelFormLayout` solution. We explain the drawbacks of this technique after the hands-on practice.

1. Create a JSF page called tabOrder2.jspx (no managed bean). You will be adding components to create the structure shown in the Structure window snippet in Figure 2.
2. Drop a Panel Header on the page and set *Text* to "Multiple Panel Form Layouts."
3. Drop a Panel Group Layout into `af:panelHeader` and set *Layout* as "vertical." This component will supply the mechanism to stack rows of fields.
4. Drop a Panel Form Layout component onto the `af:panelGroupLayout` component and set *MaxColumns* as "2" and *Rows* as "1."
5. Select the `af:panelHeader` and copy it. Paste it onto the `af:form` component to create another region. Change *Text* of the copy to "Single Panel Form Layout."
6. In the Multiple Panel Form Layouts region, drop two Input Text components into the `af:panelFormLayout` component.
7. Copy the `af:panelFormLayout` component and paste on top of the `af:panelGroupLayout` component to create the second row of fields.
8. Repeat the preceding step to create the third row of fields.

9. Group the three `af:panelFormLayout` components and set the following properties:
   FieldWidth as "300px"
   LabelWidth as "100px"
   Inline Style as "width:800.0px;"

   Additional Information: The *FieldWidth* property designates how much space is available for each input item in the layout component. The *LabelWidth* property sets a fixed space for each input item's prompt. The *Inline Style* width sets the available space for the entire layout component. The amount of space allocated to each column is half of that (400 pixels in this example) because we are specifying two columns.

10. Change the `af:inputText` *Label* properties to match those shown in Figure 1.
11. Group the two components with "Narrow" labels (using CTRL-click) and set *Columns* as "10."
12. Group the two components with "Medium" labels and set *Columns* as "25."
13. Group the two components with "Wide" labels and set *Columns* as "50."



```
af:form
  af:panelHeader - Multiple Panel Form Layouts
    af:panelGroupLayout - vertical
      af:panelFormLayout - 1
        af:inputText - Narrow Field 1
        af:inputText - Medium Field 1
        Panel Form Layout facets
      af:panelFormLayout - 1
        af:inputText - Wide Field 2 with a wrapping label
        af:inputText - Narrow Field 2
        Panel Form Layout facets
      af:panelFormLayout - 1
        af:inputText - Medium Field 3
        af:inputText - Wide Field 3
        Panel Form Layout facets
      Panel Group Layout facets
    Panel Header facets
  af:panelHeader - Single Panel Form Layout
    af:panelGroupLayout - vertical
      af:panelFormLayout - 1
        af:inputText - Narrow Field 1
        af:inputText - Medium Field 1
        af:inputText - Wide Field 2 with a wrapping label
        af:inputText - Narrow Field 2
        af:inputText - Medium Field 3
        af:inputText - Wide Field 3
        Panel Form Layout facets
      Panel Group Layout facets
    Panel Header facets
```

Figure 2. Structure window view of the tab order demonstration

14. Under the Single Panel Form Layout `af:panelGroupLayout`, set the `af:panelFormLayout` *Rows* as "3."

    Additional Information: *MaxColumns* is the important setting, but it is good practice to set *Rows* as well. Rows will be formed by the layout component placing the MaxColumns number of items on each line. If the number of items exceeds *MaxColumns* times *Rows*, the number of rows will be extended.

> **Note**
>
> Be sure the *FieldWidth*, *LabelWidth* and *InlineStyle* properties are blank for the Single Panel Form Layout `af:panelFormLayout` component.

15. Group all six fields from the three `af:panelFormLayout` components in the multiple panel region above and copy them, then paste them onto the `af:panelFormLayout` in the Single Panel Form Layout region. Compare your Structure window nodes with the illustration shown in Step 1.

16. Click Save All. Run the page.
17. Click in the first field and tab through the six fields in the top region to verify that the tab order is across and down.
18. Continue tabbing through the fields in the bottom region to verify that the tab order is down and across.
19. Resize the window width. Notice that the fields in the top region maintain their relative positions but they do not move based on the window width. The fields in the bottom region react to the window resizing and relocate so that more of the content is visible.
20. Close the browser. Stop the application.

What Did You Just Do?

You created a tab across-and-down region and a tab down-and-across region each with six fields. The former layout technique uses multiple `af:panelGroupLayout` and `af:panelFormLayout` components to emulate the alignment effect of a single `af:panelFormLayout` component. The resulting layout is not as fragile as the technique demonstrated in the preceding practice and it does not require as much experimentation. However, it has some limitations: mainly that the component widths need to be set explicitly. This takes a small amount of calculation and experimentation. Also, if the user increases the browser font size or uses a different skin, the width of one of the fields may exceed the width allotted and one or more rows may lose their alignment with the other rows. A summary of the differences in features between the single `af:panelFormLayout` component technique and the multiple `af:panelFormLayout` technique appears in Table 1.

Feature	**Single**	**Multiple**
Tab order	Down and across	Across and down
Fields left-align and labels right-align	Yes	Yes
Width of the browser window determines width of the component	Yes	No
Width of a column within the component is based on the width of the widest field in that column	Yes	No
Calculations and experimentation are needed to assign widths	No	Yes
Technique is immune to the user increasing the browser font size	Yes	No

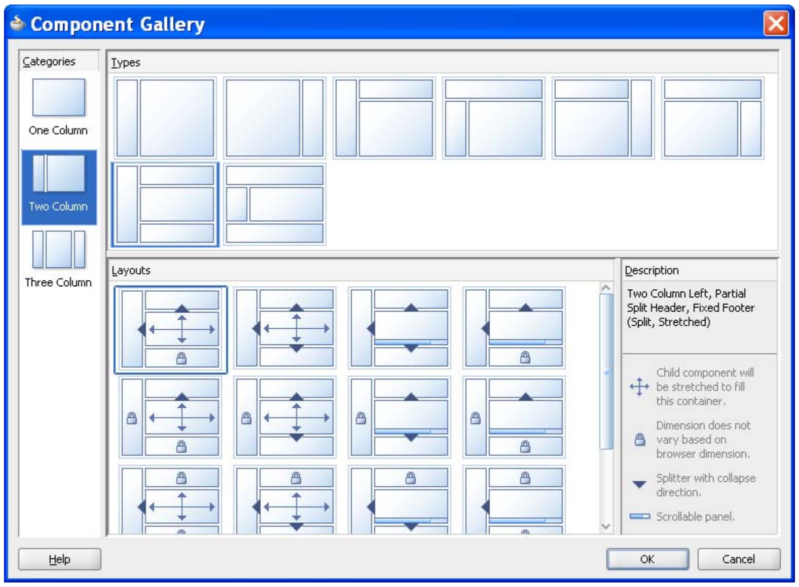
Table 1. Comparison of Features for Single Panel Form Layout and Multiple Panel Form Layout Components

Should you decide to use this technique, you will need to experiment with setting the *FieldWidth* property (in pixels) to accommodate the widest field. Fields are sized in character columns, which do not translate to pixels, so you need to experiment a bit. Remember that you need to multiply the sum of the largest field width plus a reasonable label width by the number of fields per line to reach the *InlineStyle* width for the `af:panelFormLayout` component. In other words, the formula to use (after finding the optimal field width and label width) follows (all widths use the unit pixels):

```
(FieldWidth + LabelWidth) = (Inline Style:Width / Number of Fields per Line)
```

Another Technique: Use the Quick Start Layouts

This resource is in the category of "why reinvent the wheel?" *Quick Start Layouts* are pre-built sets of layout components that are designed for specific functionality. You access this feature from the Create JSF Page dialog (started from the New Gallery's JSF Page item). Selecting Click Start Layout in the dialog and then clicking Browse will open a dialog such as that shown in Figure 3.



Two Column Left, Partial Split Header, Fixed Footer (Split, Stretched)

Child component will be stretched to fill this container.

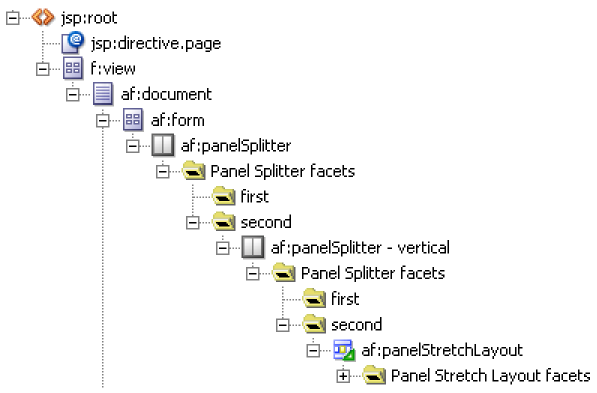
Dimension does not vary based on browser dimension.

Splitter with collapse direction.

Scrollable panel.

Figure 3. Quick Start Layout Component Gallery window

Selecting a category in the Categories panel and a type in the Types panel will display a number of options in the Layouts panel. Clicking a layout will display a short description about its layout. After selecting a layout, you click OK. JDeveloper will create the JSF page with a set of layout components based on the layout you selected. For example, selecting "Two Column Left, Partial Split Header, Fixed Footer (Split, Stretched)" will preload the following structure onto the new JSF page:



```
jsp:root
  jsp:directive.page
  f:view
    af:document
      af:form
        af:panelSplitter
          Panel Splitter facets
            first
            second
              af:panelSplitter - vertical
                Panel Splitter facets
                  first
                  second
                    af:panelStretchLayout
                      Panel Stretch Layout facets
```

You then drop components into the pre-built layout components.

Becoming familiar with this feature can save you some time in laying out the proper layout components because it is likely that the layout arrangement you require has already been created. It will save you time in nesting components and setting properties. As a bonus, studying the resulting layout from a Quick Start Layout selection will likely teach you a bit about how to properly nest layout components.

Tip

If need to embed a layout available in the Quick Start Layouts inside one of your pages, create a page from a Quick Start Layout, then copy the components in the Structure window or Source code editor window and paste them into your page.

Learning More

By now, you have probably realized that the ADF Faces components have enough flexibility to allow you to achieve any layout you can think of. This white paper is really just a start for learning about the ADF Faces components. You can rely on the other resources that follow as references and to continue your learning.

Tip

Be sure to allocate design time into your project for planning how you will use the layout components in your application.

Visual Component Guide

The JDeveloper help system contains a visual representation of ADF Faces component (as well as all other) components. Enter “enhanced tag doc” in the help search field and click the link with the same name in the search results. Alternatively, navigate in the Contents tree to Javadoc and Tag Library Reference\JDeveloper Tag Library Reference, Oracle ADF Faces Tag Library link.

Other Sources of Information

The key principle for further study is the same as always and was best expressed by Samuel Johnson (1820–1784) who said, “Knowledge is of two kinds: we know a subject ourselves, or we know where we can find information upon it.” The following list several useful sources for finding the ADF Faces information you do not know yourself.

- **Oracle Fusion Developer Guide** This is a McGraw-Hill Professional, Oracle Press book written by Frank Nimphius and Lynn Munsinger that explains intermediate-level details about a number of ADF techniques. This book includes a must-read chapter on building layouts with ADF Faces components.
- **ADF Faces RC website** This Oracle Technology Network (OTN) page (www.oracle.com/technetwork/developer-tools/adf/, click “Oracle ADF Faces” under Related Technologies) is the jumping off point for many resources related to ADF Faces. The following OTN pages are accessible from links on this site.
    - **The Web UI Developer’s Guide for Oracle ADF** This document (over 1200 PDF pages) is available in a downloadable PDF format from a Documentation link at the ADF Faces RC website. This book is also part of the JDeveloper help system. Look for it under the Contents tree node “Designing and Developing Applications\Developing Oracle ADF Applications.” The website version may be more up-to-date than the version in the JDeveloper help system.
    - **Tutorials, demos, and downloads** The ADF Faces RC website contains links to these information sources. In addition, you can search for “Oracle ADF Faces” on youtube.com for links to various tutorial videos.
    - **Cheat sheets** Look for the “ADF Faces Cheat Sheets” link in the Documentation section of the ADF Faces RC website. The cheat sheets are intended to assist you selecting the correct ADF Faces component; they also provide details about some of the features for various components.
    - **ADF Faces Rich Client Demonstration** This application is available from a link (“Oracle ADF Faces Hosted Components Demo”) at the ADF Faces RC website. You can run it on an Oracle-hosted server or download the application and run it locally. This demonstration contains a demonstration of all ADF Faces components; a separate demo application exists for DVT components. It also contains tips and advice on using ADF Faces components.
- **Martin Deh’s Blog** At martindeh.blogspot.com, this blog offers an article “ADF Layout Overview and Best Practices” that describes many useful layout techniques.
- **AMIS ADF Blog** This consulting company, based in The Netherlands, is a frequent contributor to the ADF knowledgebase. Start searching for a topic at their ADF & JHeadstart home page (technology.amis.nl/category/oracle/adf-jheadstart).
- **ADF EMG** The ADF Enterprise Methodology Group (groups.google.com/group/adf-methodology) discusses best practices for techniques and methodologies related to ADF. The prerequisite of experts contributing to this effort is that they know what “ADF” stands for.

Tip

After you drop a component onto a page, press F1 to display the tag help page for that component. This reference page is helpful for learning more about the component because it explains the purpose of the tag as well as its properties. For layout components, it also lists the supported facets.

Summary

This white paper described the basics of ADF Faces layout components that you can use to achieve a desired web page layout. It described the behavior and properties of some of the commonly used layout components. It also offered demonstrations in the form of tips and hands-on practices for exploring the layout components and how you work with their properties. Lastly, it gave some direction for further study and research. This discussion should help you be as productive as possible when using ADF Faces RC to create the layout your applications require.

All parts should go together without forcing...
By all means, do not use a hammer.

—IBM Maintenance Manual (1925)

About the Author

Peter Koletzke is a technical director and principal instructor for Quovera, in Mountain View, California, and has 29 years of industry experience, 24 of which is in the Oracle arena. Peter has presented at various Oracle users group conferences more than 300 times and has won awards such as Pinnacle Publishing's Technical Achievement, Oracle Development Tools Users Group (ODTUG) Editor's Choice (three times), ODTUG Best Speaker, ODTUG Volunteer of the Year, NYOUG Editor’s Choice (three times), and ECO/SEOUC Oracle Designer Award. He is an Oracle Certified Master, Oracle ACE Director, and coauthor (variously with Dr. Paul Dorsey, Avrom Roy-Faderman, and Duncan Mills) of eight Oracle Press development tools books including *Oracle JDeveloper 11g Handbook* (on which some of the material in this white paper is based).