DESIGN ISSUES TO BE SHARED WITH UI DESIGNER AND ALSO WHILE CREATING TEMPLATES

Grid Based Layout

<http://www.telerik.com/help/aspnet-ajax/page-layout-responsive-design-grid-based-layout.html>

This article provides information on the **Grid Based Layout** that we used as a base for the **RadPageLayout** control. You can find details on the structural elements of a grid system as well as how they are used in web development.

**Collapse imageWhy are Grids Needed in Web Development?**

When web designed emerged as a profession, screen resolutions were varying from 640 by 480 to 1024 by 768 pixels, with 800 by 600 being the predominant. Sites were made mostly with tables and for that reason were either fluid or "optimized" for the common resolution (read broken for the rest). With the rapid development of technology the common resolution in the early 2000’s was 1024 and the full width approach exhibited a flaw – no longer was it comfortable to have full width sites. With some adjustments for vertical scrollbar, the horizontal space was trimmed down to 1000px. However, 1000 is not an easy number. True it’s divisible by 2, 4, 5, 10, but that’s about it. Placing content in three columns has proven difficult and yielded different results in different browsers.

**Enter the Grid Systems**

Since grids are in essence a series of guides (both vertical and horizontal) that define proportional placeholders for content all it was needed was to find a close to 1000 that provided a variety of factors. The closest number is 960: it can be divided without a remainder by 2, 3, 4, 5, 6, 8, 10, 12 and 16

**Collapse imageElements of a Grid System**

Most of the terms used in grid systems are borrowed from typography. Still the way grid frameworks are presented in HTML resembles tables and can be explained with table terms below:

Container – the outer most container (TABLE element)

Rows – intermediate containers used for contain columns (TR elements)

Columns – the actual content placeholders (TD elements)

In addition to the above structural elements, a grid may define a couple of other properties:

Column count – the width in grid units

Gutters – outer space between columns

Padding – inner space within columns

Displacement – shifting the columns left or right

**Column Count and Column Width**

A grid is defined by the number of columns it can have. Referring to the magic number above – 960 – columns can be any number from 12, 16, 20 or their product when multiplying by 2 or 3. There are also extreme cases of grid system which have up to 48 columns (20 pixel column width). The columns are sized in abstract units e.g. 1 grid unit is the width of the narrowest column. It’s an abstract unit, because it depends on the width of the container and the number of columns it has:

Column width = Container width / column count

80 = 960 / 12

60 = 960 / 16

Columns can be one grid unit wide or span multiple units. They can also be pushed (displaced to the right) or pulled / flushed (displaced to the left). You can have as many columns on a single row, as long as their width adds up to exactly or less of the container columns count e.g. 12 \* 1 = 3 \* 4 = 2 \* 3 + 1 \* 6 = 12 and so on. In case the sum exceeds the width of the grid, the columns will simply fall on the next line.

**HTML Structure**

Having in mind that all settings or properties of a grid are applied with class names, if we are to present a typical grid with HTML it would look something like the code below:

# Fluid Grids

Fixed grids, such as base 960, are useful for a single resolution or resolution not greatly exceeding the grid base. However, they are not suitable for resolutions smaller than the base or for nesting. Reason being – pixels are used. If we substitute the pixels for percent, while maintaining the same ratio between columns and between columns and the container, we get fluid grids. That allows us to have a single grid suitable for multiple resolutions and even infinite nesting.

Column width = Container width / column count

8.3333 = 100 / 12

There is one major drawback to fluid grids – due to their percentage nature, fluid grids can not have fixed width gutters. Instead either percentage gutters or any type of column padding must be used. Another interesting detail is that grid containers must be given explicit maximum width for larger displays, otherwise the grid will stretch in the entire available space.

Responsive Layout

<http://www.telerik.com/help/aspnet-ajax/page-layout-responsive-design-overview.html>

# Responsive Layout

The concept behind **responsive layout** is to adapt your application to the capabilities of the client (the end user) device (browser). This is usually done with two very simple transformations: resizing the content placeholders or hiding them. For example, showing the entire content of your web-site may not be quite user-friendly, in cases when it is browsed via mobile device. In that case you may need to hide portions of the content.

With the **RadPageLayout** control, you could easily do so, by using the responsive attributes: **HiddenXl, HiddenLg, HiddenMd, HiddenSm, and HiddenXs**. These manage the visibility of respective content place holders when viewed under various devices. In other cases you may need to resize a certain container. Again, that's easy to do with **RadPageLayout** and responsive attributes: **SpanXs, SpanSm, SpanMd, SpanLg or SpanXl**. Those attributes, as the name suggests, control the Span of the columns under the various breakpoints. In other words once you have created your initial design, adding responsive capabilities is just a matter of configuration and setting a couple of properties.

Viewport Breakpoints

<http://www.telerik.com/help/aspnet-ajax/page-layout-responsive-design-viewport-breakpoints.html>

In responsive web design, **viewport breakpoints** are browser dimensions (usually just widths) that set the active range of a given media query. Once the browser dimensions are within that range, the styles associated with that media query will apply. In other words, without the breakpoints (and media queries) sites could be just fluid, but with breakpoints (and media queries), we can change the actual layout.

**RadPageLayout** uses the following 5 breakpoints:

* extra small resolutions (or xs)
* small resolutions (or sm)
* medium resolutions (or md)
* large resolutions (or lg)
* extra large resolutions (or xl)

The notion of **viewport breakpoints** is not just within the media queries, but is used throughout the control. All responsive properties have a suffix that's actually the media query it applies for: **SpanXS** means the span for extra small resolutions; **HiddenLG** means hidden on large resolutions and so on.

The following table describes the resolutions and the corresponding viewport breakpoints.

|  |
| --- |
| RadPageLayout Grid Types  <http://www.telerik.com/help/aspnet-ajax/page-layout-responsive-design-grid-types.html> |
|  |

|  |
| --- |
|  |

The **RadPageLayout** control has three available types of [Grid Based Layout](http://www.telerik.com/help/aspnet-ajax/page-layout-responsive-design-grid-based-layout.html) that specify the behavior of the control:

* **Fixed** - there are five predefined width values for the grid columns and the container responds to viewport size changes by setting a grid column width that corresponds to the current size of the viewport. More information about these size ranges can be found in the article [Viewport Breakpoints](http://www.telerik.com/help/aspnet-ajax/page-layout-responsive-design-viewport-breakpoints.html).
* **Fluid** - fluidly reacts to changes in the viewport size by setting the size of the grid rows so that it fits the available space.
* **Static** - the grid is static and does not respond to changes in the size of the viewport.

You can check how **RadPageLayout** looks with each of the three layout types enabled in the following screenshots. The green grid columns in the background, shown by setting the property **ShowGrid** to **true**, visualize how every page layout type responds to changes in the size of the viewport:

**Example 1**: Configuring Grid Based Layout in a **RadPageLayout** control. You will check the result in **Figure 1**, **Figure 2** and **Figure 3** by changing the property**GridType** with the values in the list above.

RESPONSIVE LAYOUT

<http://www.telerik.com/help/aspnet-ajax/page-layout-scenarios-react-to-viewport-change.html>

This article will show you how to use the **RadPageLayout** control and its viewport breakpoints to create a responsive layout that changes according to the browser viewport size.

The breakpoints represents the predefined viewport sizes at which a **LayoutColumn** could be set to be hidden (e.g. **HiddenSm** attribute) or to fit into different **Span** size (e.g. **SpanSm** attribute).

More about the breakpoints and the available attributes is explained in the [Viewport Breakpoints](http://www.telerik.com/help/aspnet-ajax/page-layout-responsive-design-viewport-breakpoints.html) article.

# Collapse imageCreating a Page that Reacts to the Viewport Size

**Example 1** shows how to hide the first **LayoutColumn** and change the **Span** size of the second and third one when the viewport size appear to be less than 768px. The results of this example can be examined in **Figure 1** and **Figure 2**.

Figure 1: Result from the markup in Example 1 when the viewport size is greater than 768px.

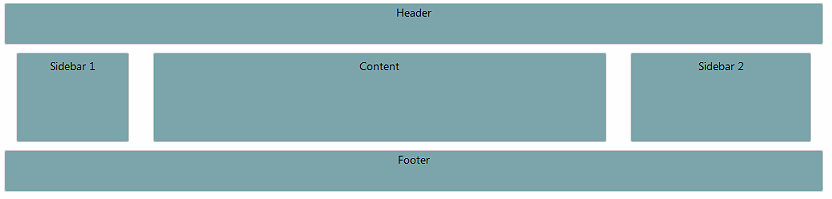
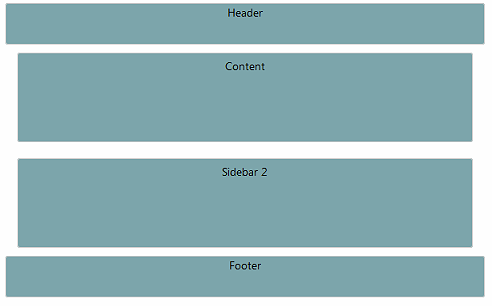


Figure 2: Result from the markup in Example 1 when the viewport size is less than 768px.



**Example 1**: Using Breakpoints to create a responsive layout.

|  |
| --- |
| Nesting RadPageLayout Elements  <http://www.telerik.com/help/aspnet-ajax/page-layout-scenarios-nest-page-layout-elements.html> |
| [See Also](http://www.telerik.com/help/aspnet-ajax/page-layout-scenarios-nest-page-layout-elements.html#seeAlsoToggle) [Send Feedback](http://www.telerik.com/help/aspnet-ajax/page-layout-scenarios-nest-page-layout-elements.html) |

|  |
| --- |
|  |

This article explains how to leverage the CompositeLayoutColumn functionality of **RadPageLayout**. In the example bellow you will see how to define such a column, allowing you to have nested rows in your **RadPageLayout** declaration.

# Collapse imageNesting RadPageLayout Elements

As a variation of the standard layout columns, the composite columns have a content element for the column's own content and a rows collection for nested rows. This allows you to easily alternate the orientation of the content in order to get the desired layout. Composite columns have the same attributes as the [standard RadPageLayout columns](http://www.telerik.com/help/aspnet-ajax/page-layout-structure-column-compositecolumn.html).

**Example 1** shows how a composite column is used to nest different **RadPageLayout** elements, would it be a custom content or nested rows. The results of this example can be examined in **Figure 1**.

|  |
| --- |
| How to make sticky footer and header  <http://www.telerik.com/help/aspnet-ajax/page-layout-sticky-footer-header.html> |
| [Send Feedback](http://www.telerik.com/help/aspnet-ajax/page-layout-sticky-footer-header.html) |

|  |
| --- |
|  |

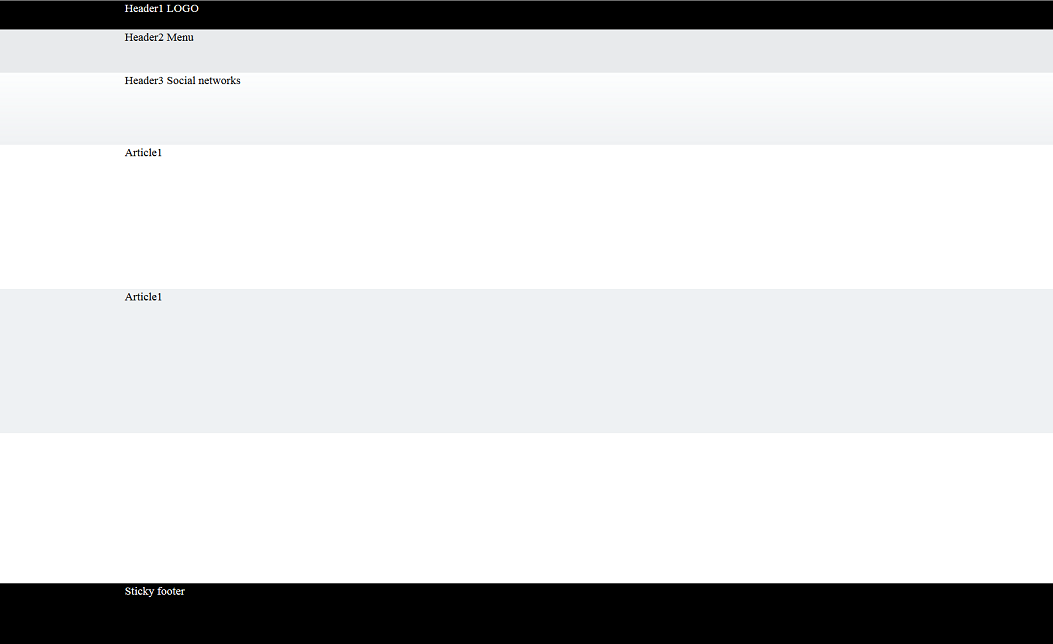
This article demonstrates how to create a scenario with sticky footer and header.

# Collapse imageSticky footer and header

In the example bellow you will see how to make the footer to be always positioned at the bottom of the page even if the content does not fill the entire gap between the header and the footer. Assuming the fact that in real scenario the header and footer are placed in master page this example shows how to use two nested RadPageLayouts to accomplish that. The inner one can go to your content page to define the content structure. The outer RadPageLayout will take care of positioning the header sections and the footer. In order to have the rows content to be centered they should have **RowType** property set to **"Container"**.

|  |
| --- |
| **http://www.telerik.com/help/aspnet-ajax/icons/alert_note.gif Note** |
| The "sticky" effect is accomplished by css and it does comes out of the box when using the RadPageLayout. In this scenario the RadPageLayout gives you the ability to structure your main elements and the content. |

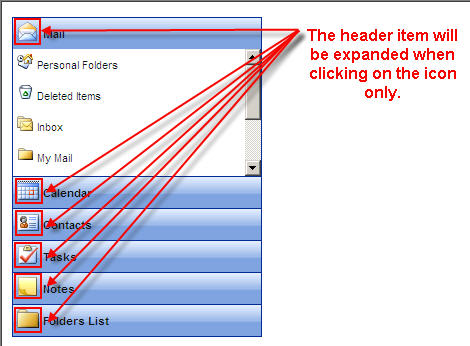
Figure 1: Result of nesting RadpageLayout and applying styles in order to accomplish sticky footer and header.



**Example Code**

|  |
| --- |
| Expand a header item while clicking on the ImageUrl only  <http://www.telerik.com/help/aspnet-ajax/panelbar-expand-header-item-while-clicking-imageurl-only.html> |
| [Send Feedback](http://www.telerik.com/help/aspnet-ajax/panelbar-expand-header-item-while-clicking-imageurl-only.html) |

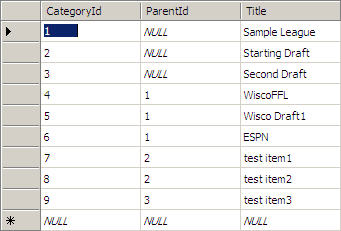
|  |
| --- |
|  |



http://www.telerik.com/help/aspnet-ajax/icons/CopyCode.gifCopy**ASPX**

<http://www.telerik.com/help/aspnet-ajax/panelbar-dynamically-created-radmenu.html>

myMenuLinkis SQL Table:



This is the result:

