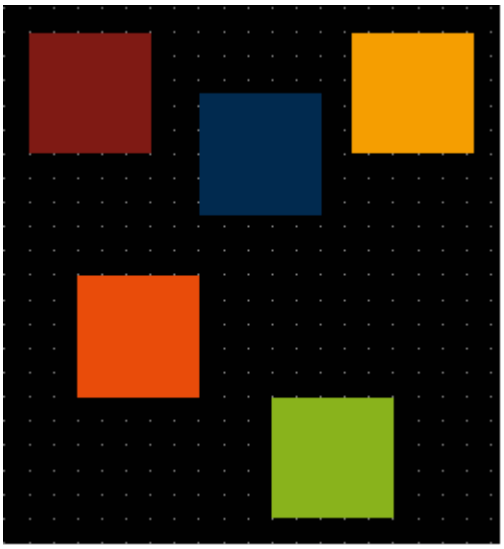
Types of Layouters

Default Layouter

When no layouter is set (Layouter Type is set to None), the default layouter is used. Absolute positioning with the default layouter is not very flexible for dynamic structures. A better layout behavior that e.g. adapts to dynamic changes of elements in their size can be achieved using the defined CGI-Studio layouters. Using the Default Layouter all nodes are positioned with absolute coordinates in pixel. In this example below, each of the five colored squares is positioned with an absolute x and y coordinate.



Similiar to WPF Canvas

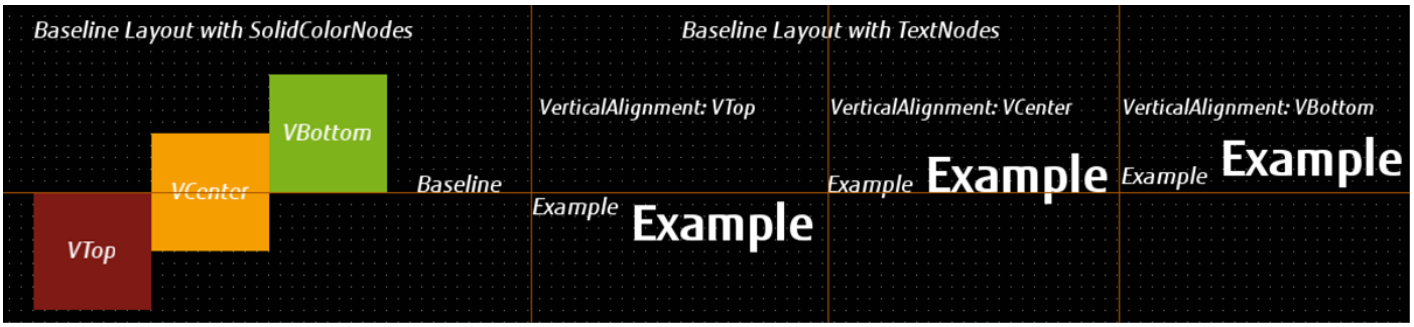
The CGI Studio Default Layouter is similar to the WPF Canvas, which also allows absolute positioning. The same layout as above can be realized with the following configuration using WPF canvas:

```
<Canvas>
    <Rectangle Height="50" Width="50" Fill="Red" Canvas.Left="0" Canvas.Top="0"></Rectangle>
    <Rectangle Height="50" Width="50" Fill
="Green" Canvas.Left="100" Canvas.Top="150"></Rectangle>
    <Rectangle Height="50" Width="50" Fill="Blue" Canvas.Left="70" Canvas.Top="25"></Rectangle>
    <Rectangle Height="50" Width="50" Fill
="DarkOrange" Canvas.Left="133" Canvas.Top="0"></Rectangle>
    <Rectangle Height="50" Width="50" Fill
="Gray" Canvas.Left="20" Canvas.Top="100"></Rectangle>
</Canvas>
```

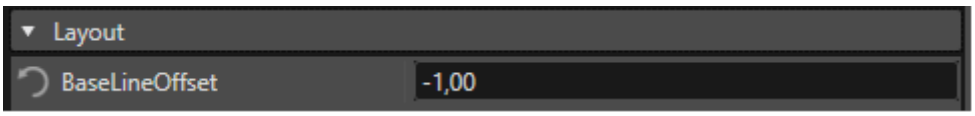
Baseline Layouter

The Baseline Layouter realizes a stacked layout in the horizontal direction. It behaves similar to the Horizontal Stack Layouter but it aligns the objects to a line instead of aligning them to the client area. This layouter offers a property Base Line Offset to configure the position of the Baseline. The main purpose of this layouter is to align texts with different sizes in various ways. Elements can be aligned along the top (VTop), the center (VCenter) or the bottom (VBottom) of their bounding boxes, which can be configured with the child elements' Vertical Alignment property.

The order of the elements is given by the order of the child nodes within the laid out group.



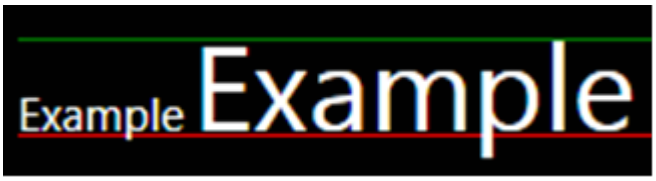
After selecting the Baseline Layouter, an additional layout property Base Line Offset becomes available. Using this property, the baseline can be set to a fixed position within the client area. Setting the baseline offset to a negative value (e.g. -1) configures the baseline for automatic, which means, the baseline is automatically set to a position where all children are positioned below the top of the client area. The Base Line Offset property is not only available to the Baseline Layouter, but also its child objects. The property can be found under the layout section in the properties panel.



WPF Implementation

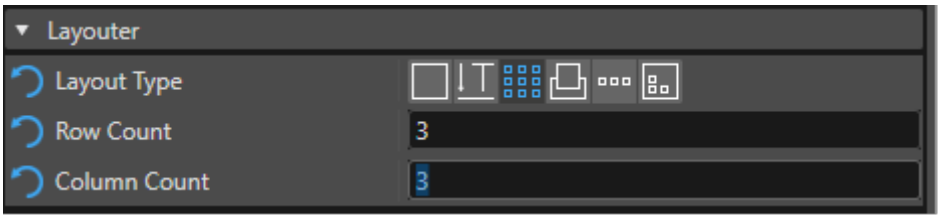
There is no implementation similar to a Baseline Layouter in WPF. However, texts with different sizes can be aligned to a common baseline using nested TextBlocks.

```
<Rectangle Height="1" Width="650" Fill="green" VerticalAlignment="Top" Margin="0,12,0,0"/>
<Rectangle Height="1" Width="650" Fill="red" VerticalAlignment="Top" Margin="0,44,0,0"/>
<StackPanel Orientation="Horizontal">
    <TextBlock Foreground="white">
        <TextBlock FontSize="15">Example </TextBlock>
        <TextBlock FontSize="40">Example</TextBlock>
    </TextBlock>
</StackPanel>
```

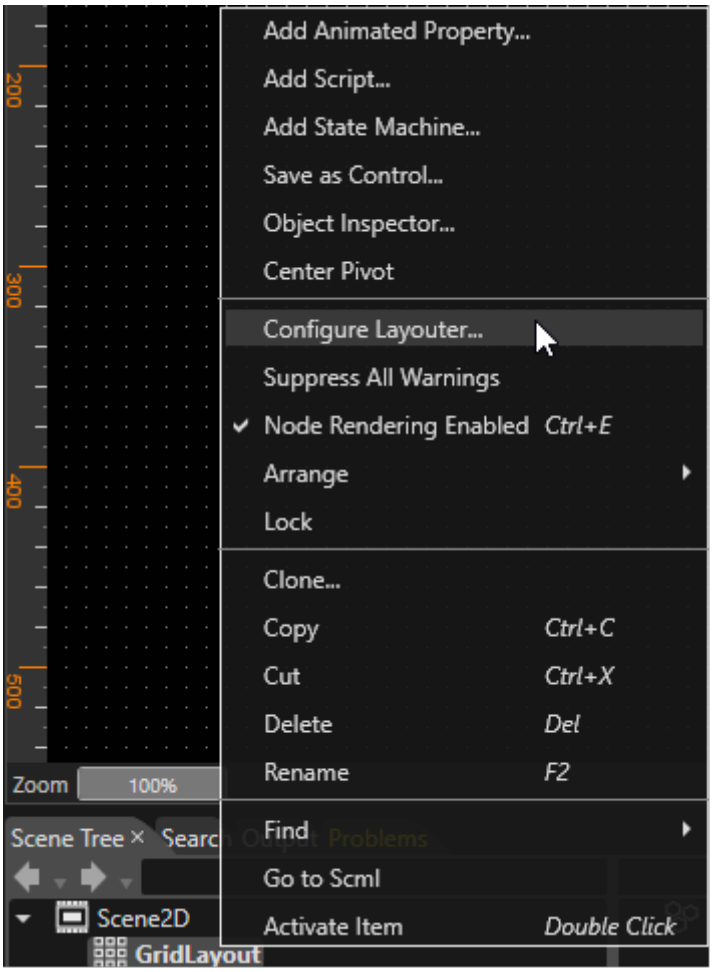


Grid Layouter

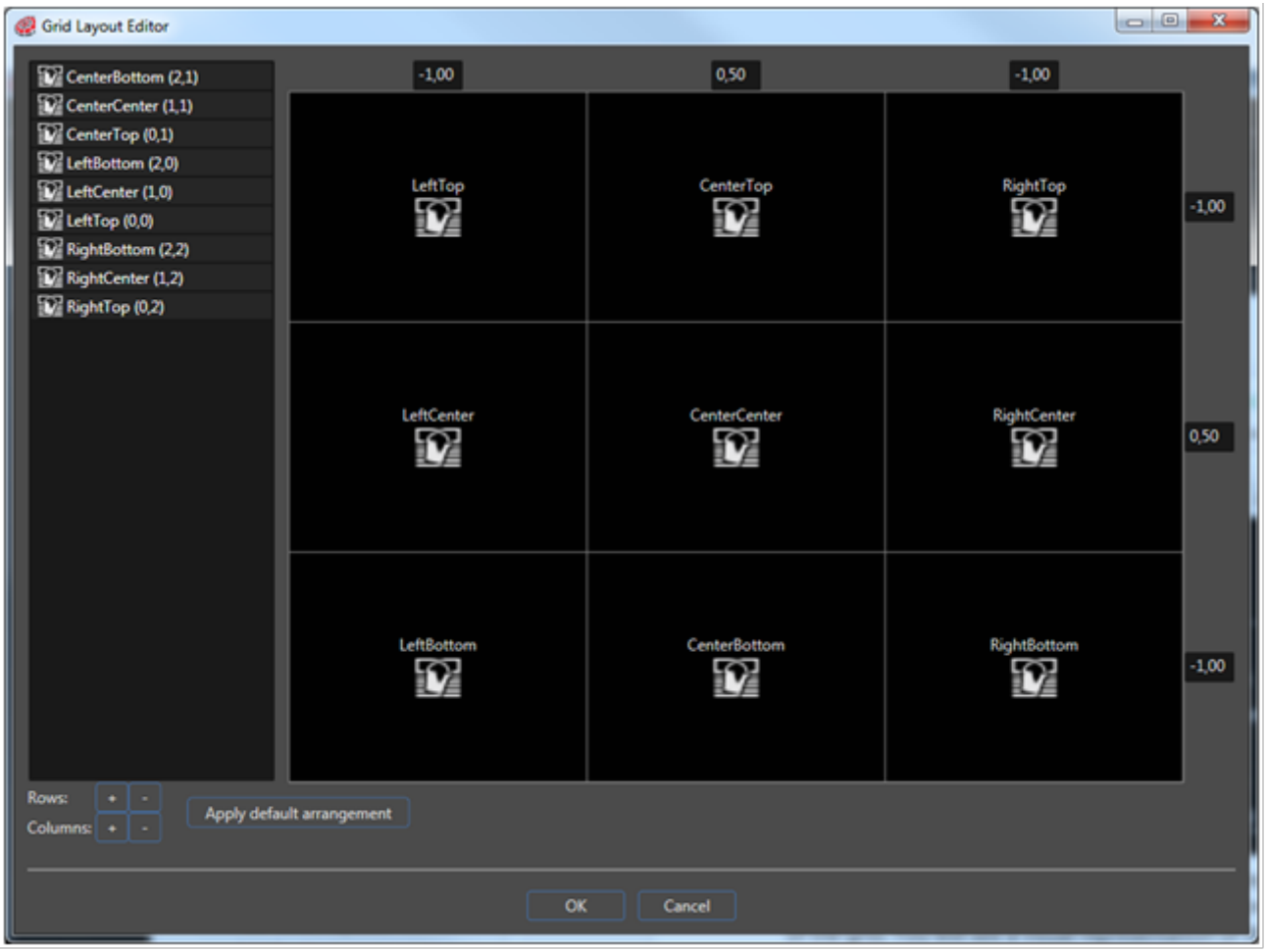
This layouter allows you to define a grid, assign nodes to the grid cells and layout these nodes to fit into the cells. If you select the grid layouter for a node, the properties for Row Count and Column Count can be configured in the properties panel.



Additionally, the context menu for the grid layout node (right-click on the specific grid layout) is extended by the entry Configure Layouter.



After choosing this context menu entry the Grid Layout Editor will open.



On the left side there is a list of nodes that can be assigned to the cells. Each cell shall contain only one node. At the bottom you can increase/decrease the number of rows/columns of the grid. You will see a visual representation of the grid and the assigned nodes in the center. The width of the columns and height of the rows can be configured at the top and on the right side. The value -1 means that it is flexible and the remaining area will be distribute by the grid layouter. Values larger than or equal to 1.0 are absolute values. The grid layouter will use exactly this size for the column/row and reduce the remaining area by this value. Values below 1.0 are relative values. In the sample above, the center row will receive 50% of the remaining height (after all absolute values have been considered) and the center column will receive 50% of the remaining height.

> Relative values will be normalized. This means, since a configuration of e.g. 0.1, 0.3 and 0.1 doesn’t sum up to 1.0, the values will be recalculated. The sum of all relative values is 0.5, so the relative value of 0.1 will be normalized with 0.1/0.5, which results in 0.2. So this part of the grid will get 20% of the available space.

A child node can be configured to span over multiple cells using its layout properties Row Span or Column Span. Here is an example on a Grid Layout configuration in CGI Studio with various cells using the column span and row span properties:

Similar to WPF Grid Layout

A similar GridLayout can be configured using a WPF GridLayout:

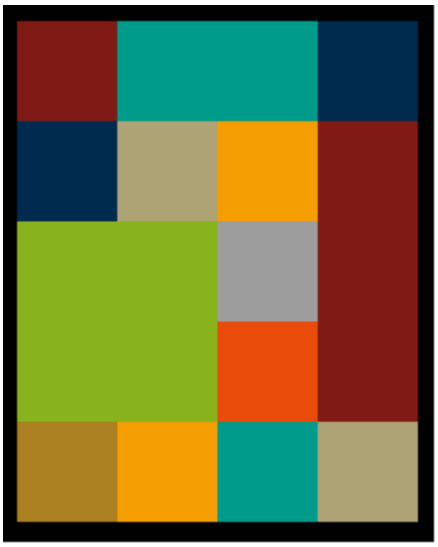
```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Rectangle Height="50" Width="50"  Fill="Red"   Grid.Row="0" Grid.Column="0"/>
    <Rectangle Height="50" Width="100" Fill
="Green" Grid.Row="0" Grid.Column="1" Grid.ColumnSpan="2" />
    <Rectangle Height="50" Width="50"  Fill="Blue"  Grid.Row="0" Grid.Column="3" />

    <Rectangle Height="50" Width="50"  Fill="Aqua"   Grid.Row="1" Grid.Column="0" />
    <Rectangle Height="50" Width="50"  Fill="Yellow" Grid.Row="1" Grid.Column="1" />
    <Rectangle Height="50" Width="50"  Fill="Coral"  Grid.Row="1" Grid.Column="2" />
    <Rectangle Height="150" Width="50" Fill
="Purple" Grid.Row="1" Grid.Column="3" Grid.RowSpan="3"/>

    <Rectangle Height="100" Width="100" Fill
="LimeGreen"  Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="2" Grid.RowSpan="2"/>
    <Rectangle Height="50"  Width="50"  Fill="Chartreuse" Grid.Row="2" Grid.Column="2" />

    <Rectangle Height="50"  Width="50"  Fill="Crimson"  Grid.Row="3" Grid.Column="2" />
```

```
    <Rectangle Height="50" Width="50"  Fill="RosyBrown"   Grid.Row="4" Grid.Column="0"/>
    <Rectangle Height="50" Width="50"  Fill="GreenYellow" Grid.Row="4" Grid.Column="1" />
    <Rectangle Height="50" Width="50"  Fill="DarkCyan"    Grid.Row="4" Grid.Column="2" />
    <Rectangle Height="50" Width="50"  Fill="Salmon"      Grid.Row="4" Grid.Column="3" />
</Grid>
```



Overlay Layouter

This is the simplest layouter. It delegates the same layout area that it has received from its parent layouter to its child layouters and returns the maximum size back to its parent layouter. The Overlay Layouter can be used to layout the same area for a foreground and a background part.

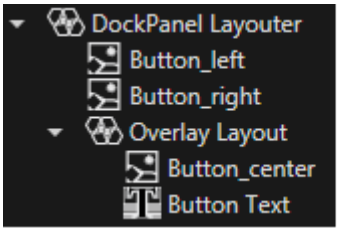
WPF Implementation

There is no separate layout panel available in WPF like the CGI Studio Overlay Layouter. However, the same layout can be achieved using a grid without column and row information.

```
<Grid>
    <Image Source="..\Images\button.png"/>
    <Label FontSize="18" FontStyle="Italic" VerticalAlignment="Center" HorizontalAlignment="Cent
       Simple Button
    </Label>
</Grid>
```

Use Cases

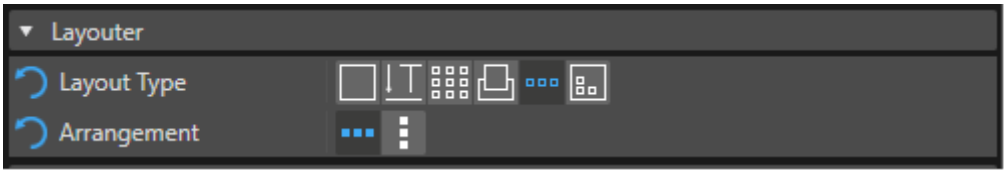
A button that can grow with the length of its text can be configured as follows with CGI Studio Layouters:



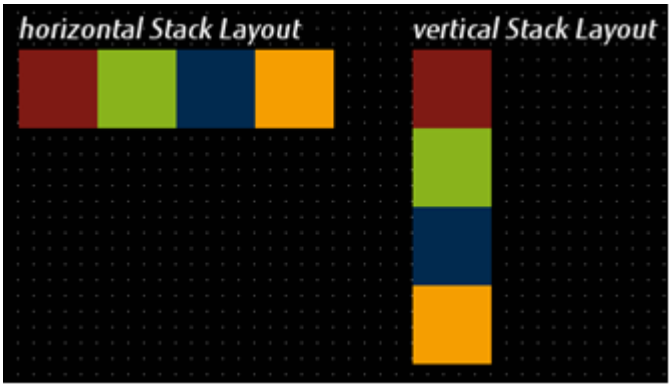
The DockPanel Layouter contains a node for the left of the button (docked left), a node for the right of the button (docked right) and an Overlay Layouter for the background of the button and the button's label. The Button_center node's stretch behavior is set to Fill, so that the image is resized automatically when the button's label is changed. The text node as foreground should be aligned to vertical and horizontal center.

Stack Layouter

The Stack Layouter stacks the nodes on top of each other under consideration of their preferred size. The additional Arrangement property will be shown in the properties panel as soon as you switch to a Stack Layouter. You can choose to configure horizontal or vertical stacking (horizontal is the default). There are no additional properties at child node level for this layouter.



Here is an example of a horizontal and a vertical Stack Layout using the CGI Studio Stack Layouter.

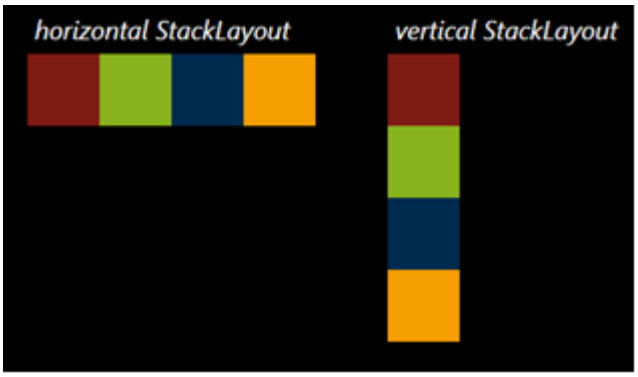


Similar to WPF StackPanel

The CGI Studio Stack Layouter is similar to the WPF StackPanel. WPF defined a vertical alignment as default. Here is how to configure a horizontal and a vertical Stack Layout like the one above using WPF:

```
<StackPanel Orientation="Horizontal">
    <Rectangle Fill="Red" Width="50" Height="50"/>
    <Rectangle Fill="Green" Width="50" Height="50"/>
    <Rectangle Fill="Blue" Width="50" Height="50"/>
    <Rectangle Fill="Gray" Width="50" Height="50"/>
</StackPanel>

<StackPanel>
    <Rectangle Fill="Red" Width="50" Height="50"/>
    <Rectangle Fill="Green" Width="50" Height="50"/>
    <Rectangle Fill="Blue" Width="50" Height="50"/>
    <Rectangle Fill="Gray" Width="50" Height="50"/>
</StackPanel>
```



Use Cases

A vertical stack layouter can be used to create any kind of lists. Horizontal stack layouters can be useful when placing ok and cancel buttons in dialog windows.

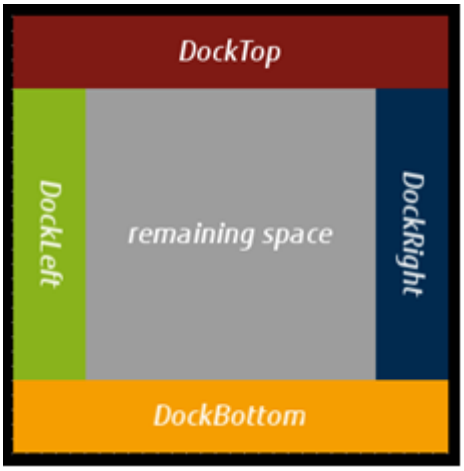
Dock Panel Layouter

The dock panel layouter is a very simple but powerful layouter. It provides an easy snapping or docking of elements to the top, bottom, left or right of the client area. One child is docked after the other, each leaving the remaining space for the rest of the elements, with regard to the order in which the child elements are added.

If a CGI-Studio Dock Panel layouter is set on a group node, an additional property Dock Side is available to all its child nodes. This property can be used to configure the side they should be docking to.

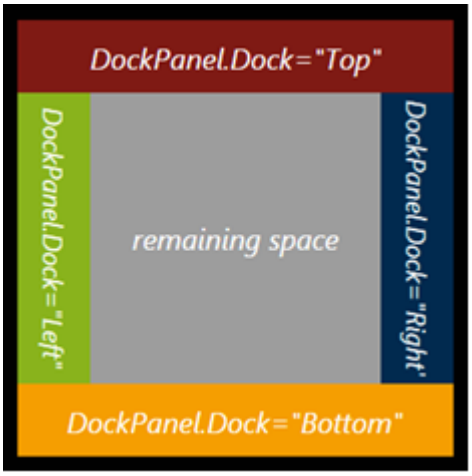
Elements can be aligned to the following four dock sides: DockLeft, DockTop, DockRight and DockBottom.

There is no center dock side because it is simply not needed and potentially confusing. The last child node takes over the complete remaining area, since there is no more node to be arranged.



WPF equivalent - DockPanel

Using WPF, a similar result can be achieved using the WPF DockPanel.




```
<DockPanel Height="300" Width="300">
    <Label DockPanel.Dock="Top" Height="50" Background="Red" Foreground="White" FontStyle="Itali
    <Label DockPanel.Dock="Bottom" Height="50" Background="DarkKhaki" Foreground="White" FontSty
    <TextBlock  DockPanel.Dock="Left" Width="50" Background="Green" Foreground="White" FontStyle
                    <LineBreak />Panel
                    <LineBreak />=
                    <LineBreak/>"Left"
    </TextBlock>
    <TextBlock DockPanel.Dock="Right" Width="50" Background="Blue" Foreground="White" FontStyle=
                    <LineBreak /> Panel
```

```
                    <LineBreak /> =
                    <LineBreak/> "Right"
    </TextBlock>
    <Label Background="Gray" Foreground="White" FontStyle="Italic" FontSize="20">remaining space
</DockPanel>
```

Use Cases

- A button that can grow with the length of its text can be configured using the Dock Laoyuter. A detailed description can be found in the chapter of the Overlay Layouter.
- If you want to create a center, make five groups, dock the first four in this order: top, bottom, left and right. The fifth group then builds the center. But this is only one of several possible center layouts (depending on the dock side order of the first four nodes).
- If you always use DockLeft you can achieve a horizontal stack like layout. If you always use DockTop you can achieve a vertical stack like layout. In difference to the stack layouter the last node receives the remaining area.

Revision #3
Created 2023-02-27 08:00:31 UTC by Tsuyoshi.Kato
Updated 2023-06-19 02:37:44 UTC by Tsuyoshi.Kato