Creating Simple to Advanced Swing and SWT Layouts Easily with MiG Layout

Mikael Grev, MiG InfoCom

TS-4928

Java

Sun microsystems

To make you actually look forward to creating a new GUI!

Agenda

- **Me, Me, Me – Why I Don't Belong Here...**
- MiG Layout Philosophy – Why Was It Created?
- Alternative Layout Managers – The Pre-emptive Strike
- Some Simple Layouts
- Sizes, Gaps & Units – Size Does Matter
- Docking and Absolute Layouts
- Platform Button Order – Not Rocket Science Anymore
- Miscellaneous Features
- Questions

Me, Me, Me...

My Day Job

My Office

Family

MiG InfoCom

migcalendar

Wing

MiG Layout Philosophy

- Fast, Small and Memory Efficient
- Simple to Use + High End = Large Range
- GUI Toolkit Independent – Easy to Port
- Resolution Independence – Automatically
- Simple to Read – Close Constraint Proximity

Alternatives – The Pre-emptive Strike

- JGoodies FormLayout
  - Inspiration for MiG Layout
  - Good for regular forms
  - Grid based, hard to do anything else
  - No automatic gaps and gaps are separate columns/rows
- TableLayout
  - Very grid based, hard to do anything else
  - Only pixel sizes, no resolution independence
  - No automatic gaps and gaps are separate columns/rows
  - Good if you are a HTML Table guru

Alternatives – Continued..

- BorderLayout
- GridLayout
- BoxLayout
- FlowLayout
- GridBagLayout
- SpringLayout
- GroupLayout (Java 6+)

Usage – String Constraints

Swing:

```
new MigLayout(
        "Layout Constr.", "Col Constr.", "Row Constr."));

JButton b = new JButton("");
p.add(b, "Constr1, Constr2, ...");
```

SWT:

```
new MigLayout(
        "Layout Constr.", "Col Constr.", "Row Constr."));

Button b = new Button(parent, SWT.WHAT_EVER);
b.setLayoutData("Constr1, Constr2, ...");
```

JavaFX Script:

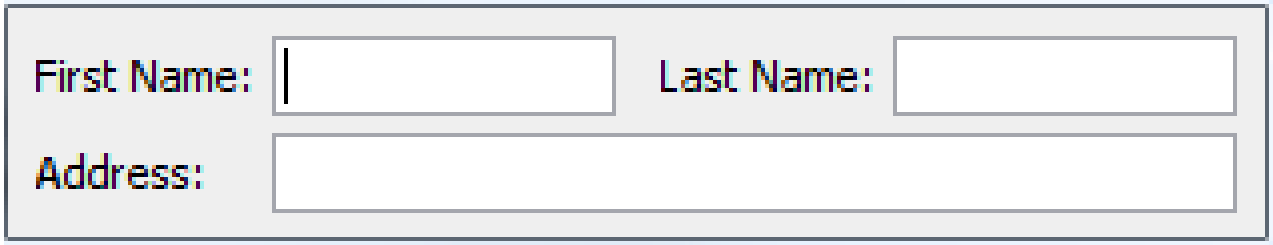
```
new MigPanel(  TBD...  )
```

Usage – API Builder Constraints

API Version (Swing):

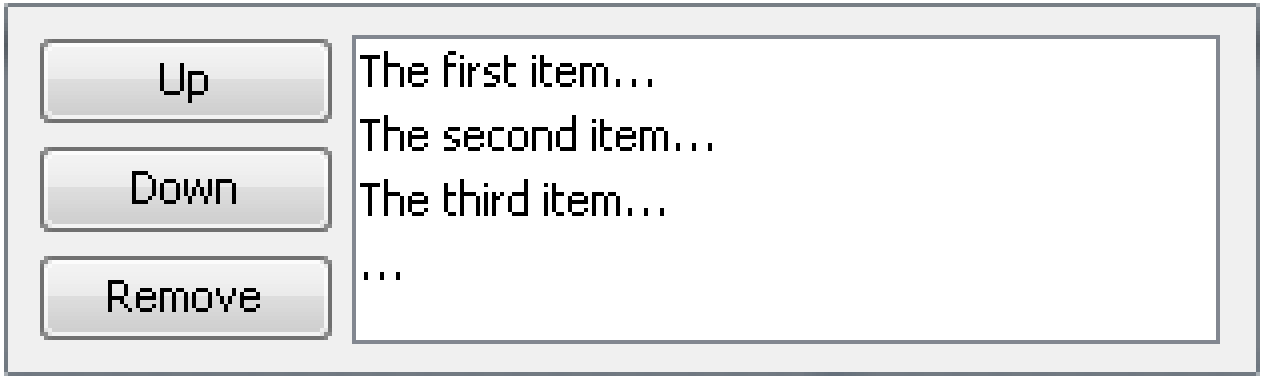
```
MigLayout layout = new MigLayout(
    new LC().wrap(3),                          // Layout Constr.
    new AC().grow(1,3,4).size("10px",1,2), // Column Constr.
    new AC().noGrid(1,4));                     // Row Constr.

panel.add(comp, new CC().grow().width("20px"));
```

A Simple Layout 1(5)



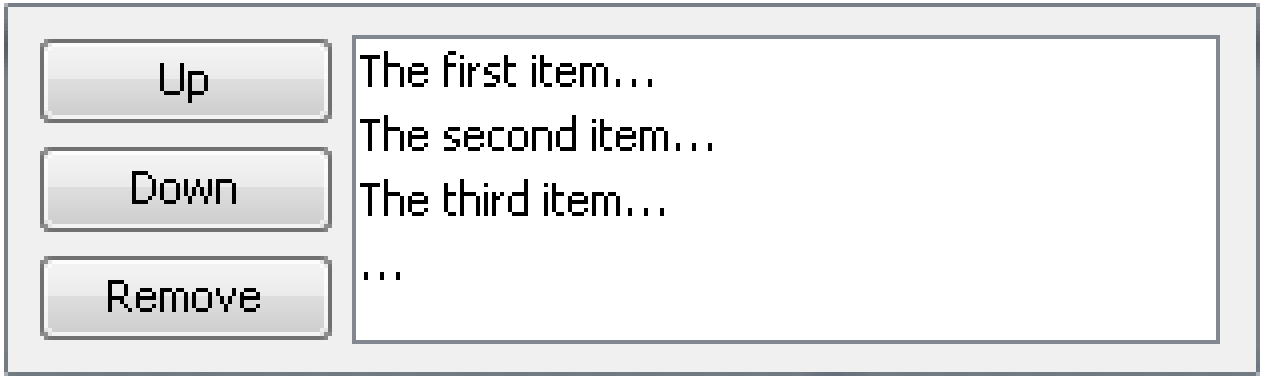

```
JPanel p = new JPanel(new MigLayout());

p.add(fNameLabel);
p.add(fNameTextF);
p.add(lNameLabel, "gap unrelated");
p.add(lNameTextF, "wrap");
p.add(addreLabel);
p.add(addreTextF, "span, growx");
```

A Simple Layout 2(5) – Spanning cells



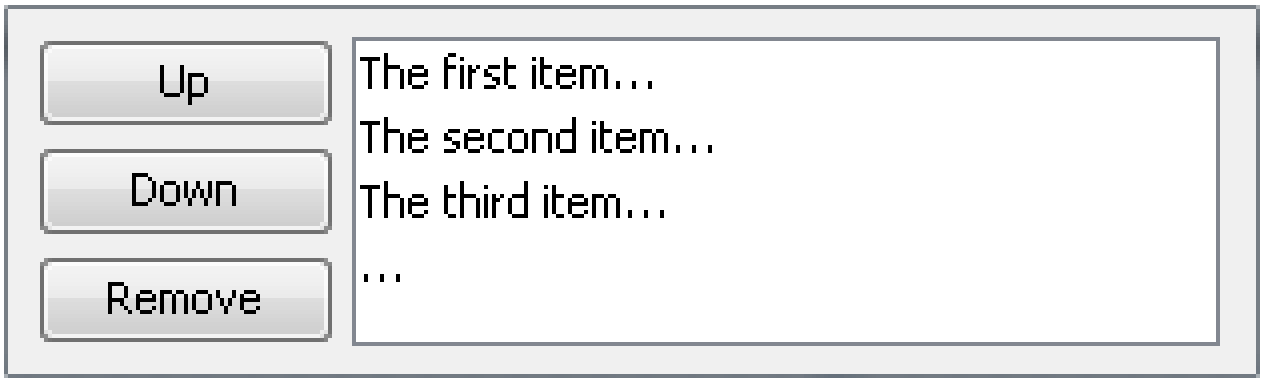

```
JPanel p = new JPanel(new MigLayout());

p.add(upButton);
p.add(itemList,   "spany 3, wrap");
p.add(downButton, "wrap");
p.add(delButton);
```

A Simple Layout 3(5) – Spanning cells




```
JPanel p = new JPanel(new MigLayout());

p.add(upButton,   "cell 0 0");
p.add(downButton, "cell 0 1");
p.add(delButton,  "cell 0 2");
p.add(itemList,   "cell 1 0 1 3"); // x, y, sx, sy
```

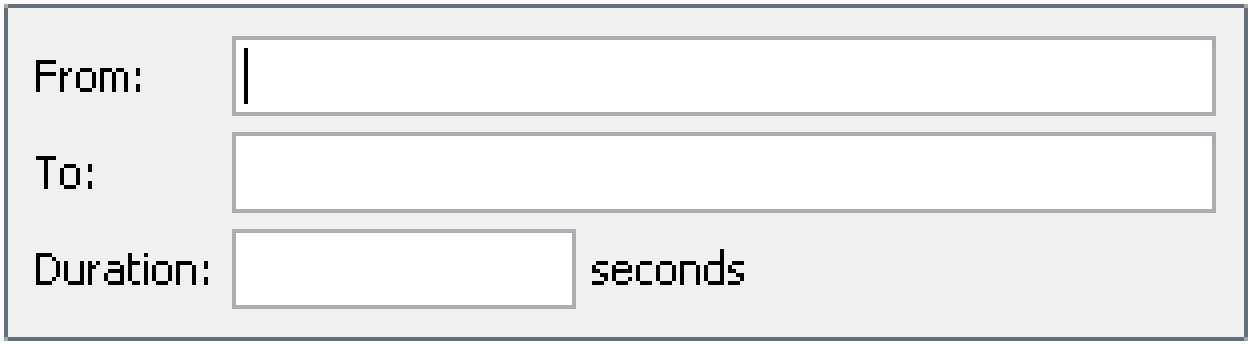
A Simple Layout 4(5) – Spanning cells




```
JPanel p = new JPanel(new MigLayout("flowy"));

p.add(upButton);
p.add(downButton);
p.add(delButton,  "wrap");
p.add(itemList,   "span 3");
```

OK, I have been cheating.. Add for buttons: `"wmin button, growx"`

Simple Layout 5(5) – Splitting Cells




```
JPanel p = new JPanel(new MigLayout());

p.add(fromLabel);
p.add(fromTextF, "wrap");
p.add(toLabel);
p.add(toTextF,   "wrap");
p.add(durLabel);
p.add(durTextF,  "split 2");
p.add(secLabel);
```

Rows & Columns - Defining the Grid

Normally not needed, but can be helpful sometimes

Syntax: "[colConstr1]gap[colConstr2]gap[...]"

```
new MigLayout("",  // Layout Constraint
              "[100]unrel[200!]10px[10:20:30]", // cols
              "[]unrel[]paragraph[]");          // rows
```

Size Matters

Swing & SWT only support pixel sizes and setting width and height at the same time

```
comp.setPreferredSize(100, comp.getPreferredSize().height);
```

- Syntax for size - "width/height [min:]preferred[:max]"
  - "width 10:20:30" // Default unit
  - "height 10cm:20inch:300mm"
  - "width 10:20" // Min/preferred
  - "width 50px" // Preferred
  - "width 50px+1cm" // Math OK. (..) if spaces
  - "width 100px!" // ! means min+pref+max
  - "height pref!" // "min/pref/max" is comp values
  - "wmin 10, wmax 20" // hmin/hmax also
  - "width (min + 10px)*2"

Gaps & Insets

White space is the key to a good looking UI

Gap sizes are specified the same way as normal sizes

- Component Gaps – Space around components
  - Syntax: "gap before [after] [top] [bottom]"
  - "gap 10sp 10px 1in 5%"
  - "gap 1cm" // Only before!
  - "gap 10:push" // "push" creates a greedy gap
  - "gap 10:20:30"
  - "gapafter rel" // *top *bottom *before *left *right
- Grid Gaps – Space between rows and columns
  - Syntax: "[]gap[]gap[]..."

```
E.g. new MigLayout("",                    // Layout Constr.
                   "[]10px[]2cm[]",       // Col Constr.
                   "[]unrel[]rel[]");     // Row Constr.
```

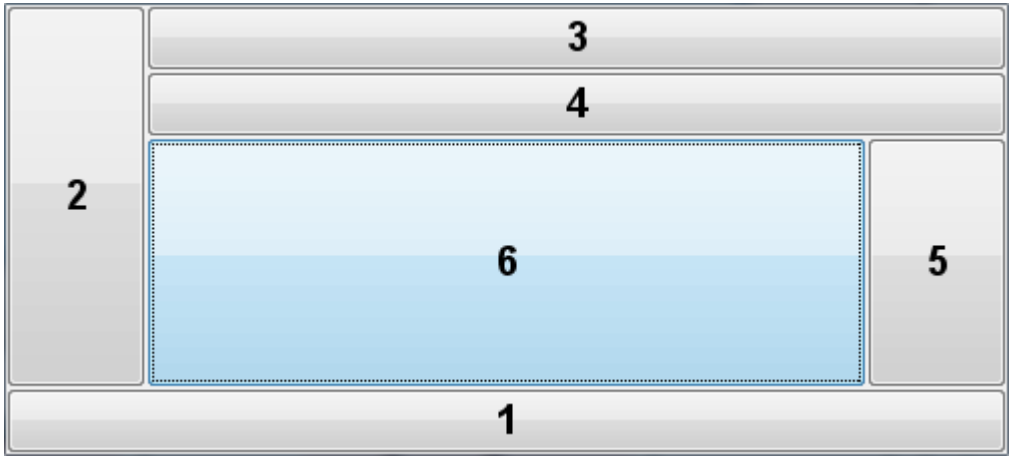
Gaps & Insets

White space is the key to a good looking UI

- Wrap Gaps – Space to next row declared inline
  - `Syntax: "wrap [gap]"`
  - `"wrap unrelated"`
  - `"wrap 20:push"  // "push" means gap turns greedy`
- Insets – Space between parent container edges and grid
  - `Syntax: "insets [all] [[top] [left] [bottom] [right]]"`
  - `new MigLayout("insets 10 20 30 40");`
  - `new MigLayout("insets 2cm");`
  - `new MigLayout("insets dialog");  // or "panel"`
  - `new MigLayout("ins 10 n 10 n");`

Docking Layout

Great for positioning top level containers




```
JPanel p = new JPanel(new MigLayout("fill"));

p.add(c1, "dock south");
p.add(c2, "dock west");
p.add(c3, "dock north");
p.add(c4, "dock north");
p.add(c5, "dock east");
p.add(c6, "dock center");
```

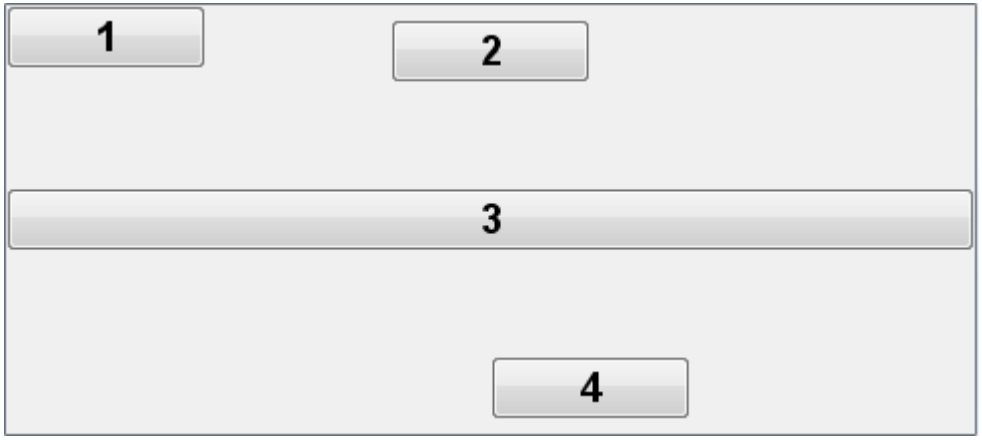
Absolute Layout

Sometimes you need the power

- Replaces "null" Layout, XYLayout and Similar
- Not "Grid" Components – But Can Link To Them
- Link to the Container, with or without it's Insets
- Link to other Components
- Flexible Math Expressions with Units – Re-evaluated at Resize

Syntax: `"pos x y [x2] [y2]"`

Absolute Layouts – Example 1(3)

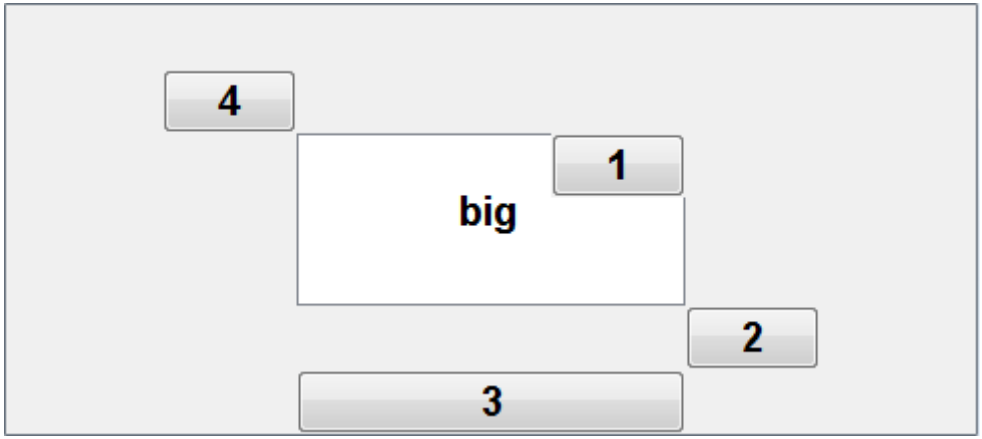



```
JPanel p = new JPanel(new MigLayout());

p.add(c1, "pos 0 0");
p.add(c2, "pos 0.5al 0al");
p.add(c3, "pos 0 0.5al 100% n");
p.add(c4, "pos 50% 1al");
```

Absolute Layouts – Example 2(3)


```
JPanel p = new JPanel(new MigLayout());

p.add(c1, "pos 100%-pref 0");
p.add(c2, "pos visual.x2-pref 0.5al");
p.add(c3, "pos container.x2-pref 1al");
p.add(c4, "pos 0 n n container.y2");
p.add(c5, "pos 50%-pref/2 50%-pref/2, pad -9 -9 9 9");
```

Absolute Layout – Example 3(3)



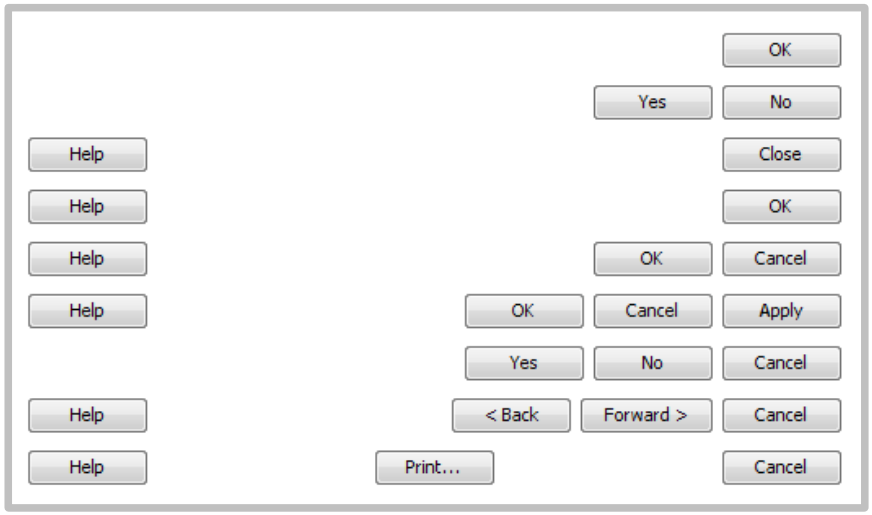

```
JPanel p = new JPanel(new MigLayout());

p.add(big,"pos 30% 30% 70% 70%, id big");
p.add(c1, "pos n big.y big.x2 n");
p.add(c2, "pos big.x2 big.y2");
p.add(c3, "pos big.x n big.x2 container.y2");
p.add(c4, "pos n n big.x big.y");
```

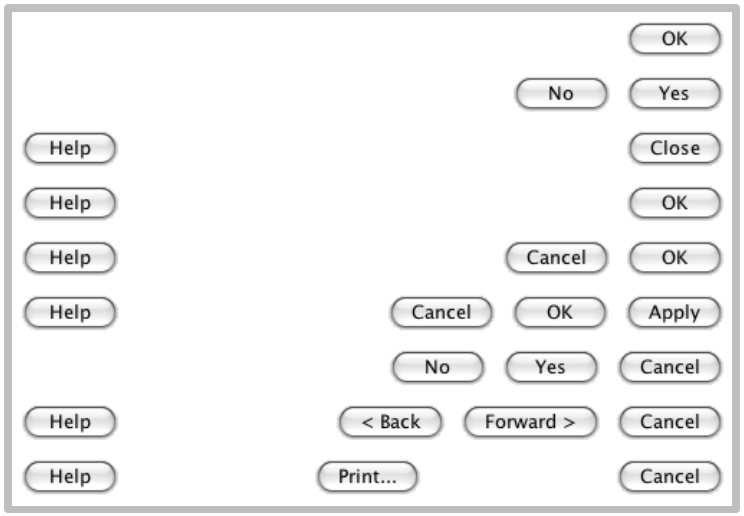
Button Order

Not exactly rocket science – any more

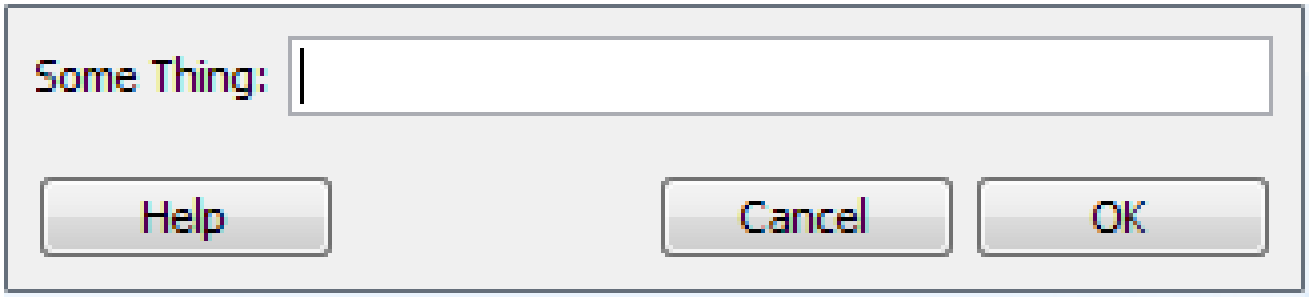
Button order definition:

```
Windows: L_E+U+YNBXOCAH_R
Mac OS X: L_HE+U+NYBXCOA_R
```





Button Order - cont.




```
JPanel p = new JPanel(new MigLayout("nogrid"));

p.add(label);
p.add(textF,       "wrap paragraph");

p.add(helpButt,    "tag help2");
p.add(okButt,      "tag ok");
p.add(cancelButt,  "tag cancel");
```

Growing & Shrinking

What to do when the components don't fit

Applies to:

- Components in the same cell
- Rows/Columns in the grid

Grow Syntax: "**growprio *prio*" and "grow *weight***"
Shrink Syntax: "**shrinkprio *prio*" and "shrink *weight***"

```
p.add(comp, "growpriox 200, growx 200);
```

```
new MigLayout("",  // Layout Constraint
              "[growprio 200][grow 200][gp 200, grow 50]",
              "");
```

Grow & Shrink

Resolution Independence

Getting ready for HiDPI displays

HiDPI / Screen Size??

Ingredients for Resolution Independence:

- Look & Feel – Fonts, Borders and Decorations
- Layout Manager – Gaps and Sizes

How do you enable Resolution Independence in MiG Layout?

[Intentionally Left Blank]

HiDPI Simulator

Miscellaneous Features 1(7)

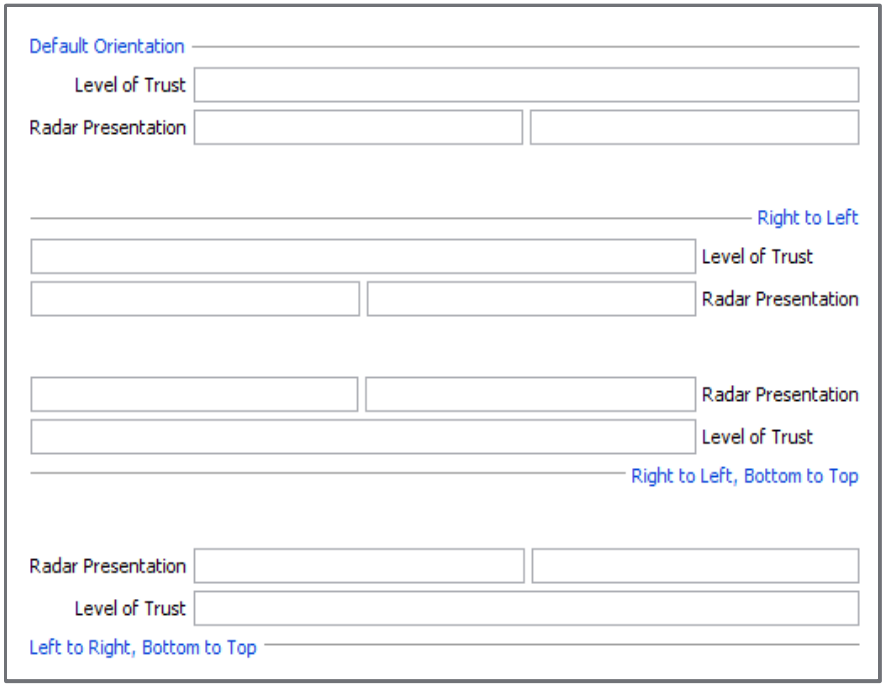
Layout Size – Overrides the Container's Calculated Size

Syntax: "**width/height [size]**"

```
new MigLayout("width 20cm, height 20sp:300px:40sp");
```

Miscellaneous Features 2(7)

Orientation - Getting ready for the rest of the world



Syntax:

```
"righttoleft" // trl
"lefttoright" // ltr
"toptobottom" // ttb
"bottomtotop" // btt
```

```
E.g.  new MigLayout("righttoleft, bottomtotop");
```

Miscellaneous Features 3(7)

External – When you need custom Java™ code to position

```
E.g. panel.add(component, "external");
```

Hidemode – What happens to invisible components?

```
panel.add(component, "hidemode 1");
new MigLayout("hidemode 4");
```

Miscellaneous Features 4(7)

Giving Components the Same Size

```
panel.add(comp1, "sizegroupx 1");  // sgx 1
panel.add(comp2, "sizegroupx 1");  // sgx 1
```

Giving Rows/Columns the Same Size

```
new MigLayout("",  // Layout Constraint
              "[sg 1][sg 1][sg 2][sg 2]",
              "");
```

Miscellaneous Features 5(7)

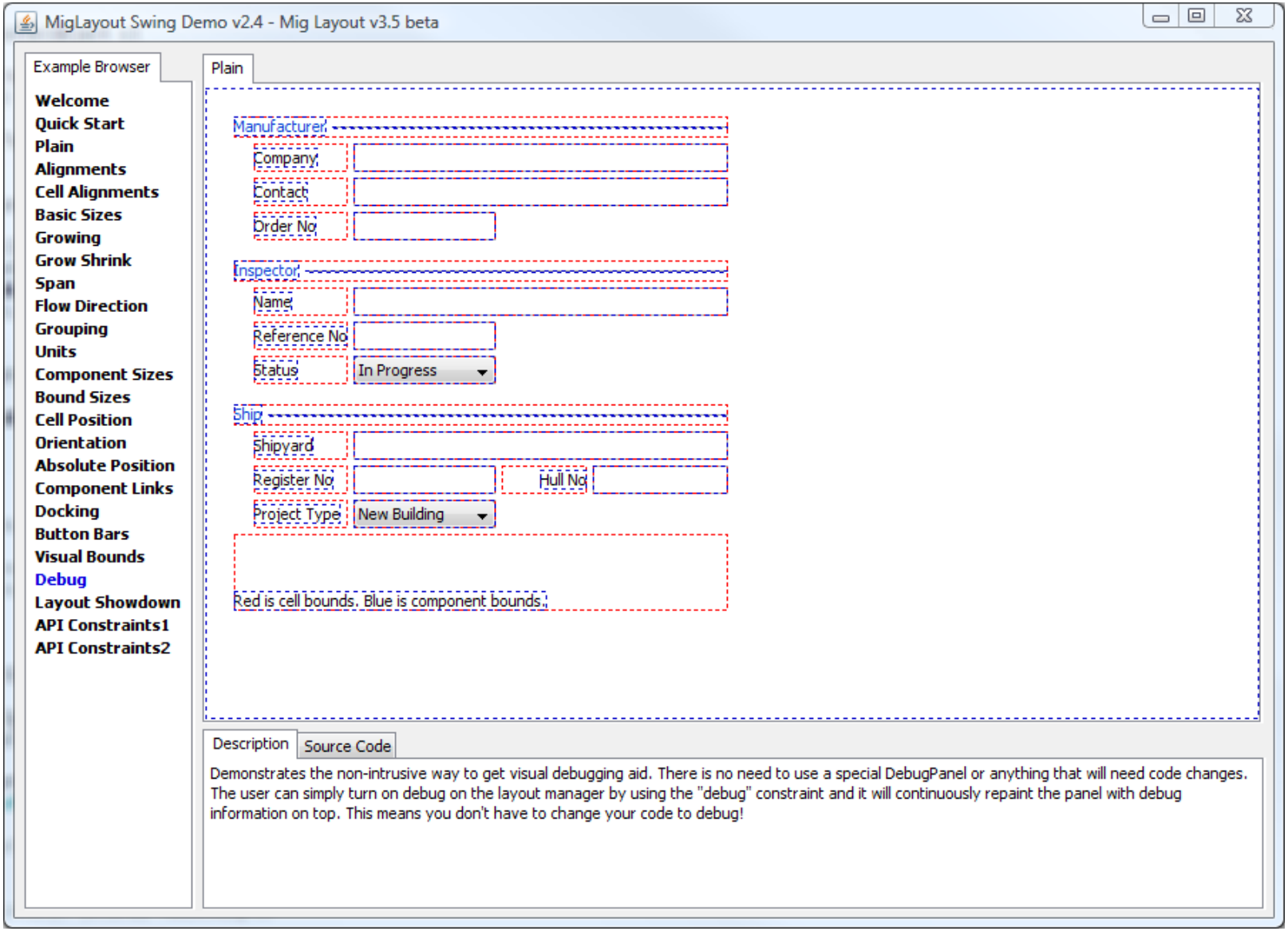
Aligning Components

```
panel.add(comp1, "align right");  // alx center

new MigLayout("",  // Layout Constraint
              "[center][right][left]",
              "[top][center][bottom][baseline]");
```

Miscellaneous Features 6(7)

Debugging Layouts – Non Intrusive

```
new MigLayout("debug");
```


MigLayout Swing Demo v2.4 - Mig Layout v3.5 beta
Example Browser
Welcome
Quick Start
Plain
Alignments
Cell Alignments
Basic Sizes
Growing
Grow Shrink
Span
Flow Direction
Grouping
Units
Component Sizes
Bound Sizes
Cell Position
Orientation
Absolute Position
Component Links
Docking
Button Bars
Visual Bounds
Debug
Layout Showdown
API Constraints1
API Constraints2
Plain
Manufacturer
Company
Contact
Order No
Inspector
Name
Reference No
Status
In Progress
Ship
Shipyard
Register No
Hull No
Project Type
New Building
Red is cell bounds. Blue is component bounds.
Description
Source Code
Demonstrates the non-intrusive way to get visual debugging aid. There is no need to use a special DebugPanel or anything that will need code changes. The user can simply turn on debug on the layout manager by using the "debug" constraint and it will continuously repaint the panel with debug information on top. This means you don't have to change your code to debug!

Miscellaneous Features 7(7)

Layout Call Back – Kindly asks for your corrections

```
migLayout.addLayoutCallback(new LayoutCallback() {
    public BoundSize[] getSize(ComponentWrapper comp)
    {
        return ...
    }

    public void correctBounds(ComponentWrapper c)
    {
        ...
    }
});
```

Layout Callback

For More Information

- `www.miglayout.com`

Please, Fill In the Report Card!

(especially if you are going to give a good grade ;-)

THANK YOU

Creating Simple to Advanced Swing and SWT Layouts Easily with MiG Layout

Mikael Grev

TS-4928