



the
POWER
of
JAVA™



Creating Professional Swing UIs Using the NetBeans™ GUI Builder

Tomas Pavek, Jan Stola, Scott Violet

Sun Microsystems

<http://www.netbeans.org>
<http://swinglabs.dev.java.net>

TS-4916

Goal of This Presentation

Learn how to easily create professional Swing UIs using the NetBeans™ GUI Builder (formerly code-named Matisse)

Agenda

NetBeans GUI Builder Introduction

Cross Platform UI

How to Design Layout

Internationalization

Using Custom Components

Managing Generated Code

Agenda

NetBeans GUI Builder Introduction

Cross Platform UI

How to Design Layout

Internationalization

Using Custom Components

Managing Generated Code

Introduction

Why to use a GUI builder

- Visual interaction (WYSIWYG)
- Simplify layout design
- Easy manipulation and customization of components
- Quick prototyping
- Consistency
- Generating code, binding to UI
- Ease of maintenance

NetBeans GUI Builder Introduction

Main features

- Supports AWT/Swing
- Based on JavaBeans™ architecture
- Allows designing layout in a natural way
- One-way code generator
- Using standard JDK™ software classes

NetBeans GUI Builder Introduction

What will we present?

- Special NetBeans IDE build for JavaOneSM conference (upcoming 6.0 version)
- Update for NetBeans 5.0/5.5 IDE will be available
- <http://form.netbeans.org/JavaOne>

DEMO

NetBeans GUI Builder Introduction

NetBeans GUI Builder Introduction

Basic tips

- Note the Source/Design view switch in toolbar
- Use Inspector to explore hierarchy of components
- Use preview to see live GUI quickly
- “Design This Container” action is useful for:
 - Standalone containers
 - Panels in tabbed pane
 - A panel in a scroll pane
- Note you can design UI without subclassing a visual class

Agenda

NetBeans GUI Builder Introduction

Cross Platform UI

How to Design Layout

Internationalization

Using Custom Components

Managing Generated Code

UI Design Goals

Swing GUI Specifics

- Different platforms, look and feels, localization
 - Component sizes and proportions may change
- Dynamic behavior
 - We want resizability most of GUI forms
- UI is expressed by code
 - No common resource format
 - Layout can't be modified independently (localized)

UI Design Goals

Requirements on Java Technology GUI

- Platform (look and feel) independence
- Localization independence
- Scale with size, font, and resolution
- Follow UI guidelines
- Visual consistency
- UI separated from application logic

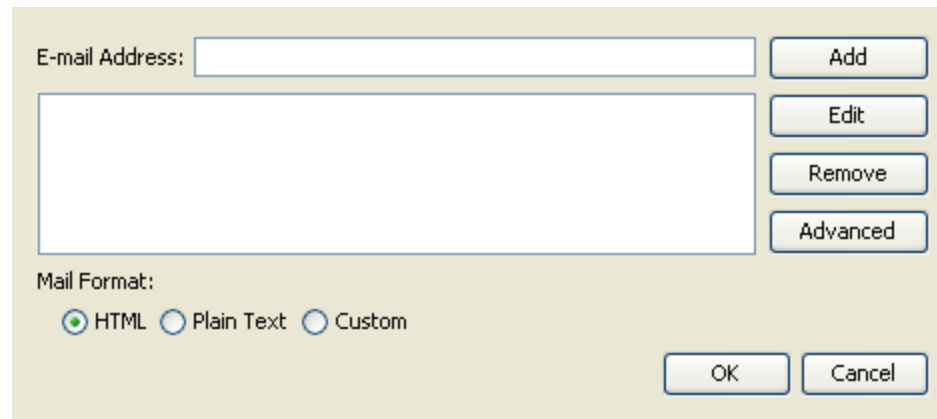
Platform Independence

Common mistakes in cross platform UI design

- Using absolute sizes or positions
- Relying on relative proportions of components
- Implicit position dependencies
- Hard coded strings
- Hard coded fonts and colors
- Hard coded left to right orientation

Cross Platform UI

Example: absolute sizes and positions

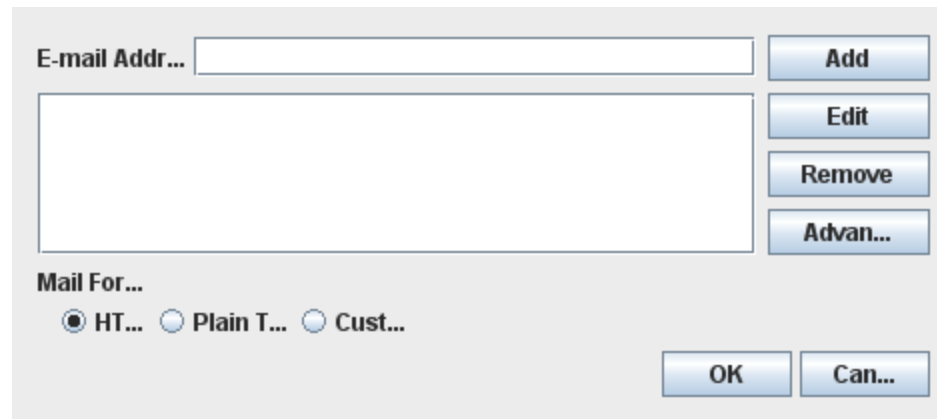


The image shows a Java Swing dialog box with a light beige background. It contains the following elements:

- E-mail Address:** A text input field followed by an **Add** button.
- A large empty rectangular area, likely a list or table.
- Edit**, **Remove**, and **Advanced** buttons stacked vertically to the right of the large area.
- Mail Format:** A label followed by three radio buttons: **HTML** (selected), **Plain Text**, and **Custom**.
- OK** and **Cancel** buttons at the bottom right.

Cross Platform UI

Example: absolute sizes and positions



E-mail Addr...

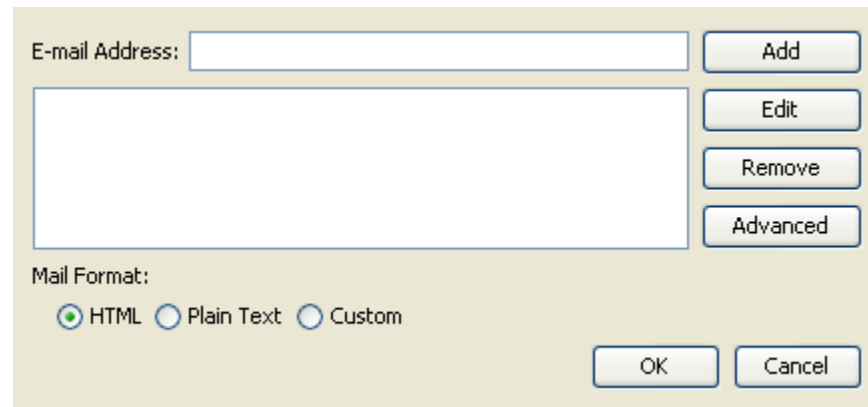
Mail For...

HT... Plain T... Cust...

OK Can...

Cross Platform UI

Example: missing resizable element



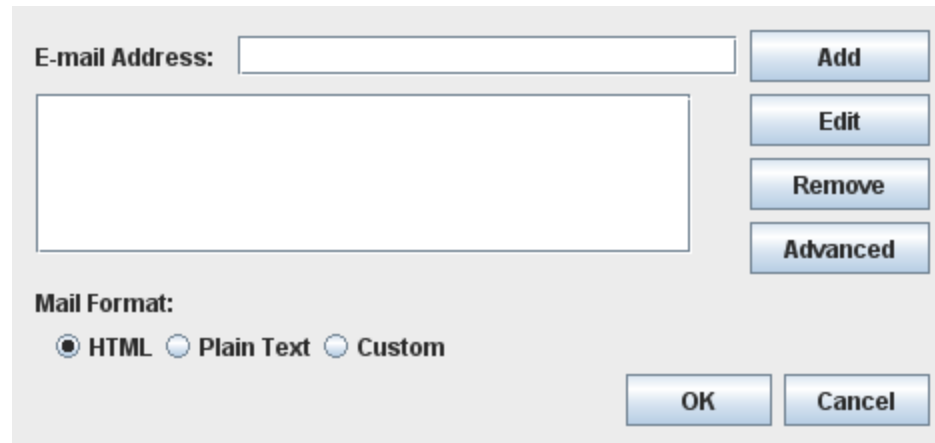
The screenshot shows a Java Swing dialog box with a light beige background. It contains the following elements:

- E-mail Address:** A text input field followed by an **Add** button.
- Empty List:** A large empty rectangular area, likely intended for a list of email addresses, with **Edit**, **Remove**, and **Advanced** buttons stacked vertically to its right.
- Mail Format:** A section with three radio buttons: **HTML** (selected), **Plain Text**, and **Custom**.
- OK** and **Cancel** buttons at the bottom right.

The dialog box is not resizable, which is the issue being highlighted in the presentation.

Cross Platform UI

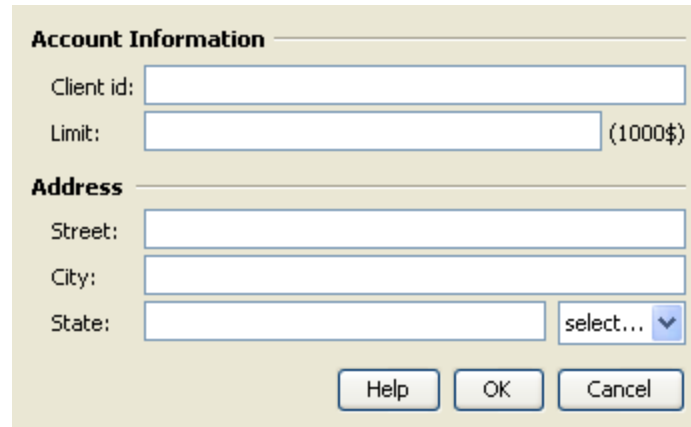
Example: missing resizable element



The screenshot shows a Java Swing dialog box with a light gray background. At the top left, the text "E-mail Address:" is followed by a single-line text input field. Below this is a large, empty rectangular area that is not resizable, as evidenced by the absence of a standard Mac OS-style title bar with a red, yellow, and green control bar. To the right of this area are four vertically stacked buttons: "Add", "Edit", "Remove", and "Advanced". Below the large area, the text "Mail Format:" is followed by three radio buttons labeled "HTML", "Plain Text", and "Custom". The "HTML" radio button is selected. At the bottom right of the dialog are two buttons: "OK" and "Cancel".

Cross Platform UI

Complex example: insidious grid



Account Information

Client id:

Limit: (1000\$)

Address

Street:

City:

State: select... ▼

Help OK Cancel

Cross Platform UI

Complex example: insidious grid

Informace o vyberovem limitu klienta

Cislo klienta:

Limit: (v tisicich korun)

Kontaktni informace

Ulice:

Mesto:

Stat:

Cross Platform UI

Complex example: insidious grid

Account Information			
Client id:	<input type="text"/>		
Limit:	<input type="text"/>		(1000\$)
Address			
Street:	<input type="text"/>		
City:	<input type="text"/>		
State:	<input type="text"/>		select... ▼
		Help	OK
			Cancel

Cross Platform UI

Complex example: insidious grid

Account Information			
Client id:	<input type="text"/>		
Limit:	<input type="text"/>		(1000\$)
Address			
Street:	<input type="text"/>		
City:	<input type="text"/>		
State:	<input type="text"/>		select... ▼
Help		OK	Cancel

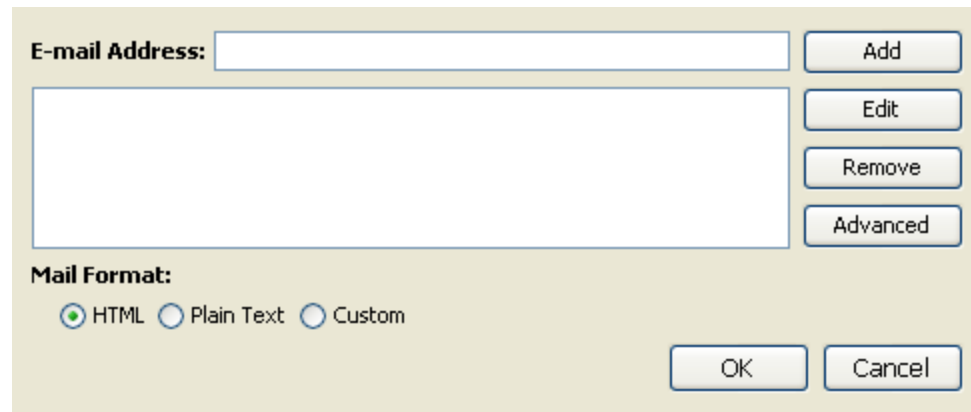
Informace o vyberovem limitu klienta			
Cislo klienta:	<input type="text"/>		
Limit:	<input type="text"/>		(v tisicich korun)
Kontaktni informace			
Ulice:	<input type="text"/>		
Mesto:	<input type="text"/>		
Stat:	<input type="text"/>		vyber... ▼
Napoveda		OK	Zrus

Mistakes:

- Relying on relative proportions of components
- Implicit position dependencies

Cross Platform UI

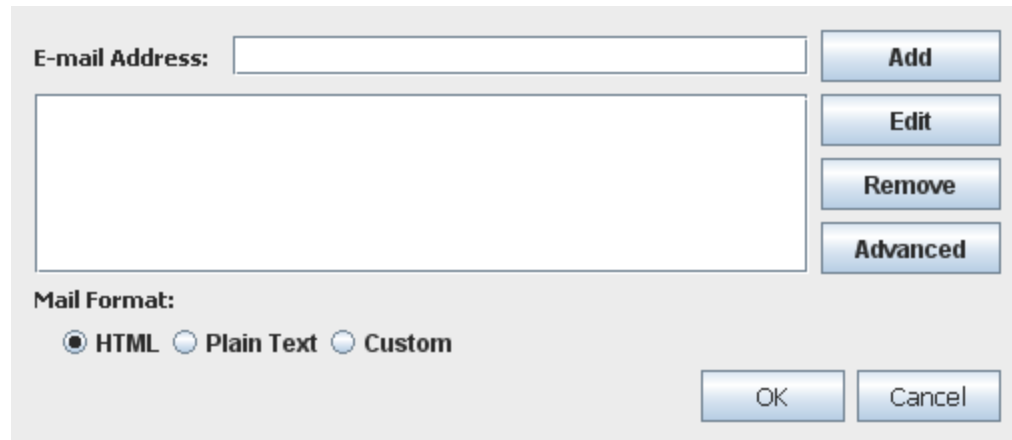
Example: hard coded font



The screenshot shows a Java Swing dialog box with a light beige background. It features a text input field labeled "E-mail Address:" with an "Add" button to its right. Below the input field is a large empty rectangular area. To the right of this area are three stacked buttons: "Edit", "Remove", and "Advanced". Below these buttons is a "Mail Format:" section with three radio buttons: "HTML" (selected), "Plain Text", and "Custom". At the bottom right of the dialog are "OK" and "Cancel" buttons. The font used for all text in the dialog is a consistent, clean sans-serif typeface, demonstrating a hard-coded font across different platform look-and-feels.

Cross Platform UI

Example: hard coded font



The image shows a Java Swing dialog box with a light gray background. At the top left, the text "E-mail Address:" is followed by a text input field. To the right of the input field are four stacked buttons: "Add", "Edit", "Remove", and "Advanced". Below the input field is a large empty rectangular area. At the bottom left, the text "Mail Format:" is followed by three radio buttons labeled "HTML", "Plain Text", and "Custom". The "HTML" radio button is selected. At the bottom right, there are two buttons: "OK" and "Cancel".

Agenda

NetBeans GUI Builder Introduction

Cross Platform UI

How to Design Layout

Internationalization

Using Custom Components

Managing Generated Code

Cross Platform UI design in NetBeans IDE

Layout designed once can run everywhere

- Matisse helps to avoid the cross platform mistakes
- Dynamic layout definition built behind the scene
 - Relative positioning
 - Adaptive spacing (gaps according to UI guidelines)
 - Aligning (also supports baseline alignment)
 - Resizing definition
 - BiDi compliant
- Built-in internationalization support

Cross Platform UI design in NetBeans IDE

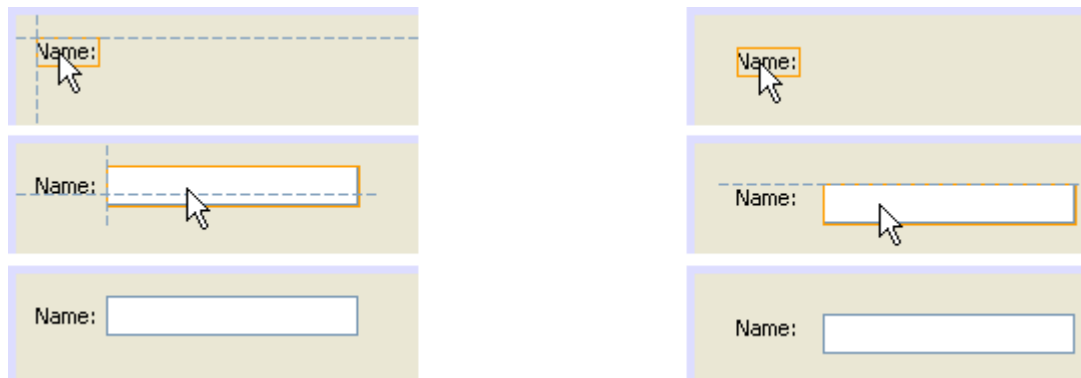
Can combine different layout managers

- Need not care about layout managers while in the default Free Design mode
- Using a layout manager still valid in some cases:
 - Free Design not convenient for all types of layout
 - May need full control over all aspects of the layout
- Invoke “Customize Layout” on a container with **GridBagLayout** for a helpful customizer
- Containers with Free Design and layout managers can be combined freely

Good Design Practices

Layout guidelines

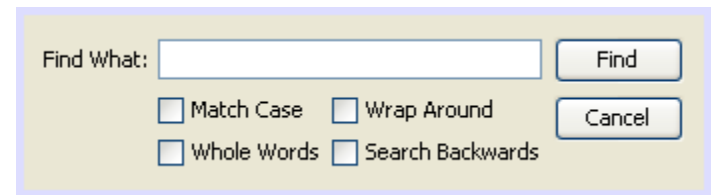
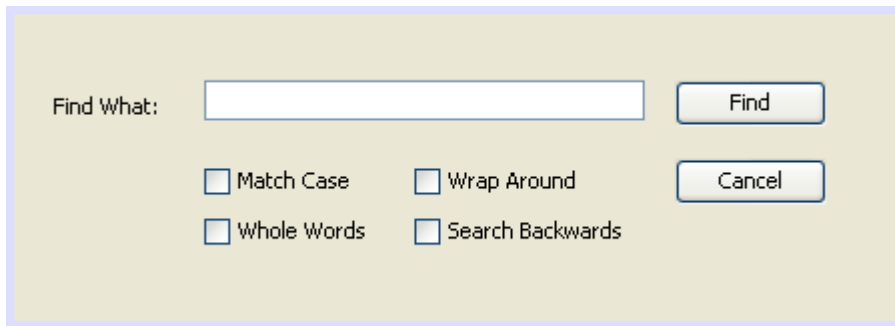
- Follow the guidelines Matisse offers
 - It is easy to make the components aligned right away
- Pay attention to the suggested alignment
 - There can be more guidelines offered on close positions but with different alignment



Good Design Practices

Avoid the “I have lots of space” syndrome

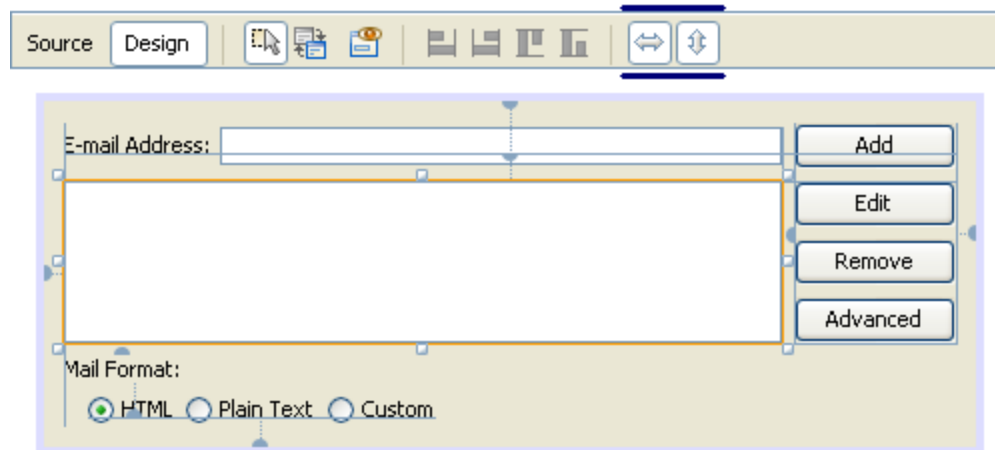
- Snap components to preferred positions
- Draw components to container borders
- Reduce superfluous space
- What you see is **really** what you get



Good Design Practices

Default size and resizability

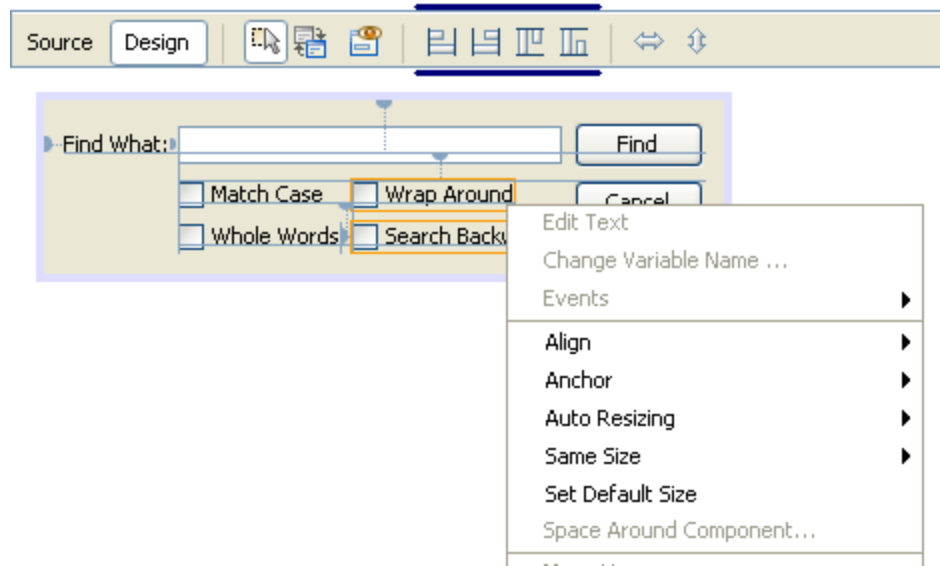
- Keep components in their default size, or make them resizable
- Design every container as resizable



Good Design Practices

Aligning actions

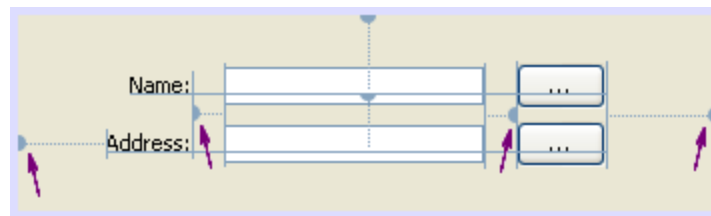
- You don't need to do everything via mouse
 - There are aligning actions in toolbar
 - ...and more actions in context menu



Good Design Practices

Layout design tips

- Matisse does not “remember” the absolute positions but relations between components
 - Pay attention to anchors, lines of alignment, and resizing
- Components may move to preserve the established relations



DEMO

Designing Layout



Good Design Practices

More design tips

- Use “Space Around Component” dialog to fine-tune gaps between components
 - Gaps can also be resizable
 - You can provide custom `LayoutStyle` implementation for runtime
- Use “Set Same Size” action to impose same width on related components (e.g., buttons)
- When dragging
 - Use Shift to add multiple components
 - Use Control to hold to a guideline
 - Use Alt to suppress snapping on guidelines

Matisse Behind the Scene

New layout features in JDK 6 software

- **GroupLayout**—brand new layout manager
- **LayoutStyle**—responsible for spacing
- Swing Layout Extensions library used before
- Practical tips for manual coding:
 - Can provide custom **LayoutStyle** implementation
 - Can replace part of the layout dynamically
 - Controlling visibility—can show/hide components without affecting the layout

GroupLayout

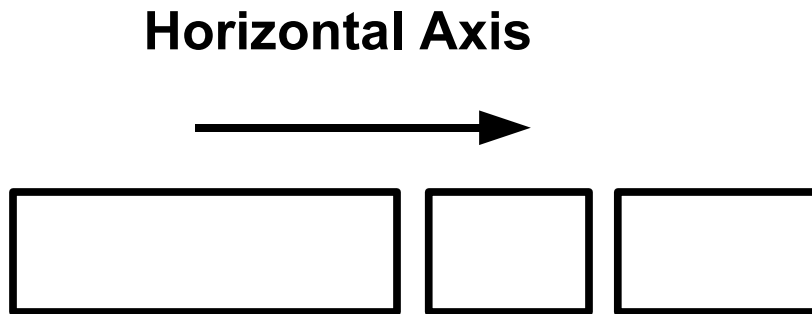
- Not intended for hand coding
- Can automatically add preferred gaps
 - Uses `LayoutStyle` to determine preferred gaps
- Ability to align components along their baseline
 - Finally!
- Each axis treated independently
 - Must configure horizontal and vertical axis separately

GroupLayout

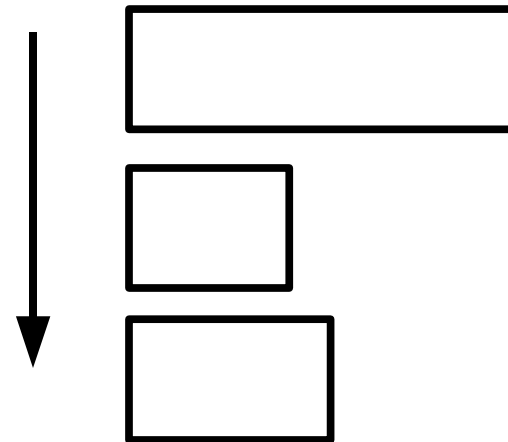
Groups

- Group
 - Contains components and other groups
- Horizontal Group sets x and width
- Vertical Group sets y and height
- Two types of Groups
 - Sequential Group
 - Aligns contents one after another
 - Parallel Group
 - Aligns contents on top of each other
 - Typically used in conjunction with sequential group along opposite axis

Sequential Group



Vertical Axis



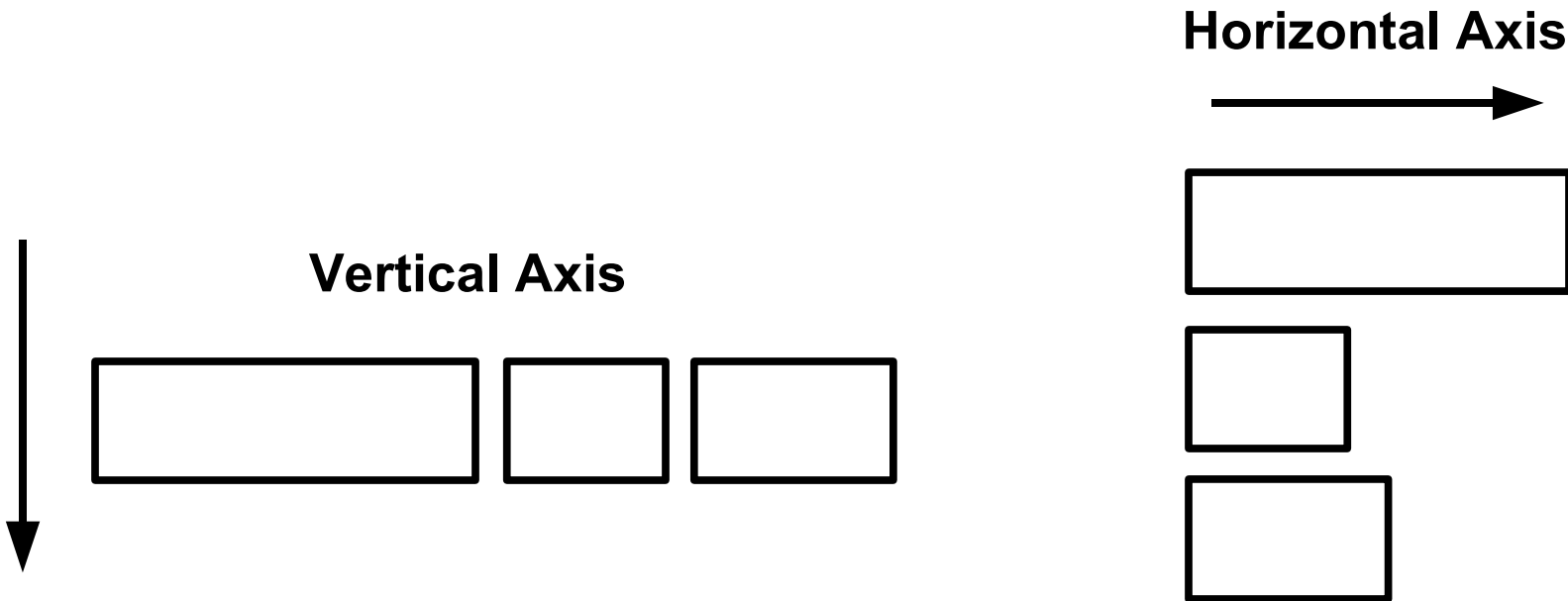
```
SequentialGroup hg = layout.createSequentialGroup();
hg.addComponent(c1).addComponent(c2).addComponent(c3);
layout.setHorizontalGroup(hg);
```

Parallel Group Along One Axis



```
ParallelGroup hg = layout.createParallelGroup();
hg.addComponent(c1).addComponent(c2).addComponent(c3);
layout.setHorizontalGroup(hg);
```

Parallel Group Used with a Sequential Group



```

ParallelGroup hg = layout.createParallelGroup();
hg.addComponent(c1).addComponent(c2).addComponent(c3);
layout.setHorizontalGroup(hg);
SequentialGroup vg = layout.createSequentialGroup();
hg.addComponent(c1).addComponent(c2).addComponent(c3);
layout.setVerticalGroup(vg);
  
```

Agenda

NetBeans GUI Builder Introduction

Cross Platform UI

How to Design Layout

Internationalization

Using Custom Components

Managing Generated Code

Internationalization

How to make the GUI localizable

- Allow translation without rebuilding the GUI
- No hard coded strings
 - Stored in `.properties` file
 - Accessed via `ResourceBundle`
- All visible text should be internationalized
 - Text on labels, buttons, tabs, titled borders, etc.
 - Window titles
 - Tool tips
 - Mnemonics
 - Accessibility descriptions

DEMO

Internationalization



Agenda

NetBeans GUI Builder Introduction

Cross Platform UI

How to Design Layout

Internationalization

Using Custom Components

Managing Generated Code

Custom Components

Using custom components in GUI builder

- Only requirement: must be JavaBeans architecture-compliant
- Can be installed to palette via Palette Manager
 - From an external JAR file (library)
 - From a NetBeans IDE project
- If in a project, the component can be copied or dragged from the project explorer directly

Custom Components

Developing your own component

- JavaBeans architecture requirements:
 - Public non-abstract class
 - Public no-arg constructor
- Good practice is to only expose properties that makes sense (via **BeanInfo**)
- Return superclass' **BeanInfo** as additional **BeanInfo**
- Use Java-Bean tag in the JAR file's manifest
- Define icon for the component (**BeanInfo**)

Example: Java-Bean Tag in manifest.mf

```
Manifest-Version: 1.0
```

```
X-COMMENT: Main-Class will be added automatically by build
```

```
Name: com/me/beans/MyComponent.class
```

```
Java-Bean: True
```

- Go to Files view
- Open `manifest.mf` file under project root
- Enter the last two lines as marked above, separated by an empty line

Example: Component's Icon

```
package com.me.beans;

import java.beans.SimpleBeanInfo;
import java.awt.Image;

/**
 * Simple BeanInfo implementation for MyComponent.
 * Only provides an icon (the rest is omitted).
 */
public class MyComponentBeanInfo extends SimpleBeanInfo {
    public Image getIcon(int iconKind) {
        return loadImage("/com/me/beans/cool_icon.gif");
    }
}
```

Custom Components

Troubleshooting, common errors

- Class loading error
 - Check the required JAR file (external component)
 - Note: a **library** needs to be defined for multiple JARs
 - Check the class is compiled (component from a project)
- Instantiation error—check the sources:
 - Whether the component is a bean
 - What it does in constructor
 - Look at the exception stack trace
- See NetBeans IDE FAQs for complete guide

DEMO

Using Custom Components



Agenda

NetBeans GUI Builder Introduction

Cross Platform UI

How to Design Layout

Internationalization

Using Custom Components

Managing Generated Code

Source Code

Generated code vs. user code

- Standard Java technology code is the only output
- Generated code is protected from changes
 - `initComponents ()` method
 - Field variables for components
 - Event handlers (headers)
- Can write any code outside the protected area
 - To do more initialization/customization
 - To implement additional logic (e.g., event handlers)
 - To change components dynamically

Source Code

Customizing generated code

- See “Code” tab in the property sheet
- Generated code can be configured freely
 - Local variables vs member fields, modifiers, way of dispatching events...
- Custom code can be inserted almost anywhere
 - Custom code for property values
 - Custom code for creating components
 - Pre-init and post-init code for components/properties
- Synthetic properties

Source Code

Examples of using custom code

- Setting up a complex property
- Special way of initialization (not via properties)
 - For example binding to a dynamic content
- Special way of constructing a component
 - For example `JFormattedTextField`
- Iterating over many components, for example:
 - Creating a collection of components
 - Setting some property to all components

DEMO

Customizing Generated Code



Summary

- NetBeans GUI Builder is designed to help you create professional UIs
- The GUI Builder honors platform independence and internationalization
- Standard Java technology code is produced that can be used anywhere
- You can use third-party components and develop own custom components

For More Information

- <http://form.netbeans.org/JavaOne>
- <http://www.netbeans.org/kb/50/quickstart-gui.html>
- http://www.netbeans.org/kb/faqs/#GUI_Editor_Matisse
- <http://swing-layout.dev.java.net>
- **Related Sessions**
 - TS-1594 Best Practices: Data Binding
 - TS-3399 A Simple Framework for Desktop Applications
- **Blogs**
 - <http://weblogs.java.net/blog/zixle>
 - <http://weblogs.java.net/blog/tpavek>

Q&A





the
POWER
of
JAVA™



netBeans.ORG



Creating Professional Swing UIs Using the NetBeans GUI Builder

Tomas Pavek, Jan Stola, Scott Violet

Sun Microsystems

<http://www.netbeans.org>
<http://swinglabs.dev.java.net>

TS-4916