

WindowBuilder 101

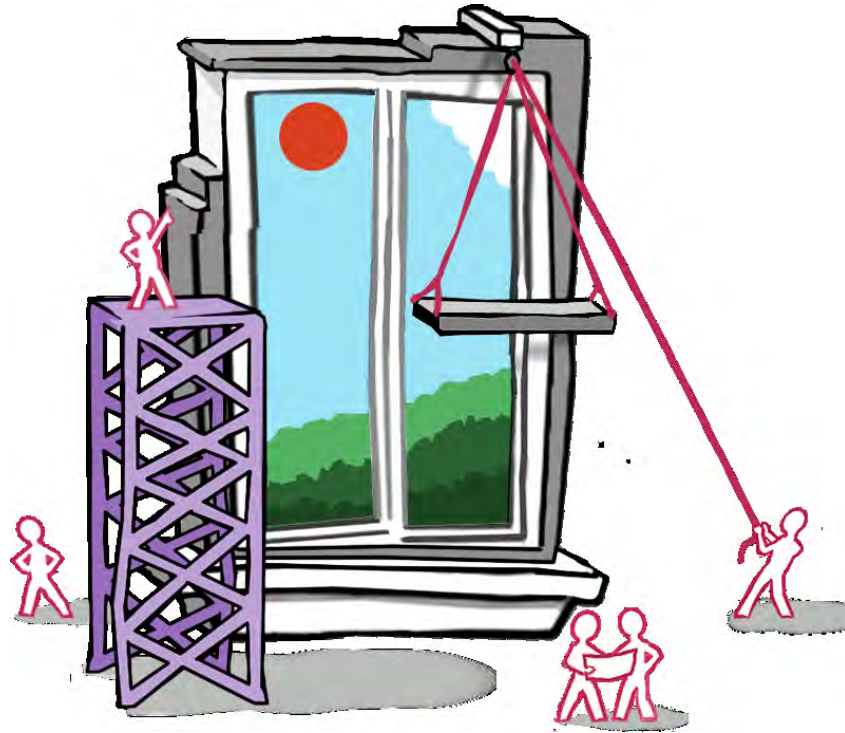
GUI Development for
Swing, SWT, RCP, XWT, GWT & Android

Intro



Agenda & Overview



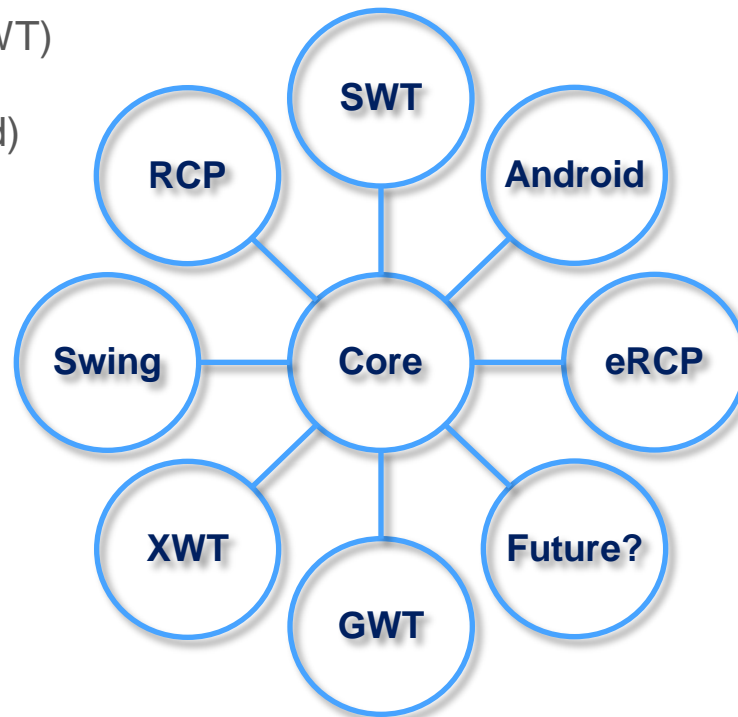
- History
- Overview
- Key Features
- User Interface
- Customization



WindowBuilder has a very long history spanning multiple technologies and companies

- | | | | | | |
|--------------|---------|---|--|-----------|--|
| Smalltalk | • 1991 | Original release for Smalltalk/V by Cooper & Peters |  | Same Team | |
| | • 1993 | VisualSmalltalk release by ObjectShare | | | |
| | • 1994 | VisualAge Smalltalk release by ObjectShare | | | |
| | 1996 | Briefly owned by ParcPlace-Digitalk | | | |
| | • 1997 | VisualAge Smalltalk release by Instantiations | | | |
| Eclipse/Java | • 2003 | New Eclipse/Java version for SWT/RCP (SWT Designer) | | | |
| | • 2004 | Swing support added (Swing Designer) | | | |
| | • 2006 | Google Web Toolkit (GWT) support added (GWT Designer) | | | |
| | • 2009 | Eclipse community award for Best Commercial Add-on |  | | |
| | • 2010 | Acquired by Google and released free to the world | | | |
| | • 2011+ | Contributed to Eclipse.org as new open-source project;
Part of Indigo & Juno release trains (Eclipse 3.7, 3.8 & 4.2) | | | |

- Available now from <http://www.eclipse.org/windowbuilder>
- Composed of WindowBuilder Engine, SWT, eRCP, XWT & Swing Designer
- WindowBuilder Engine provides a rich API for creating UI designers
 - Very modular with dozens of extension points
 - Pluggable support for different languages and parsers
 - Java-based UI frameworks (e.g., Swing, SWT/RCP, eRCP, GWT)
 - XML-based UI frameworks (e.g., XWT, GWT UiBinder, Android)
- Exemplary tool examples:
 - SWT Designer
 - Swing Designer
 - eRCP Designer
 - XWT Designer
- 3rd Party Tools
 - JBuilder Swing Designer
 - GWT Designer
 - Android Designer



“WindowBuilder delivers the kind of GUI building productivity that we used to have before we converted to Java. WindowBuilder not only dramatically improves productivity for design and maintenance, but it also enables us to significantly improve the look-and-feel of our GUIs without costing days of coding. Until discovering WindowBuilder, I had forgotten just how much fun and easy it can be building Java GUIs.”

Sally Rich,
Senior Software Engineer at RSS
Solutions Inc

“In 25 years of software development I have used a plethora of development tools. I can honestly say that WindowBuilder is head and shoulders above anything I have used for serious development. The features I particularly like include the bi-directional edit process, the native look and feel of cross platform GUIs and the manner in which component management is greatly simplified. It all adds up to allowing the developer to get on with the process of creating an application rather than worrying about the technology beneath it.”

John Bond,
Developer

Key Features

State-of-the-art GUI tool features



WindowBuilder supports many state-of-the-art features

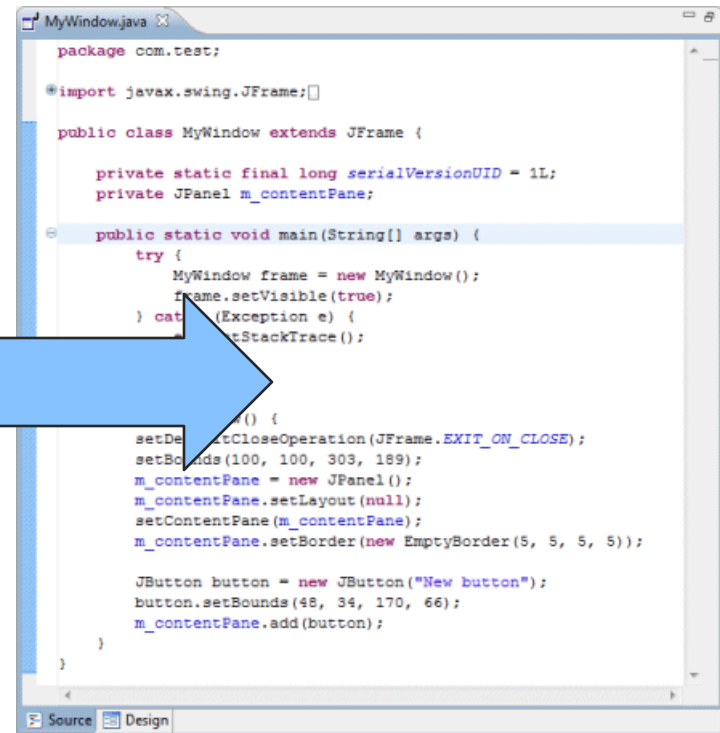
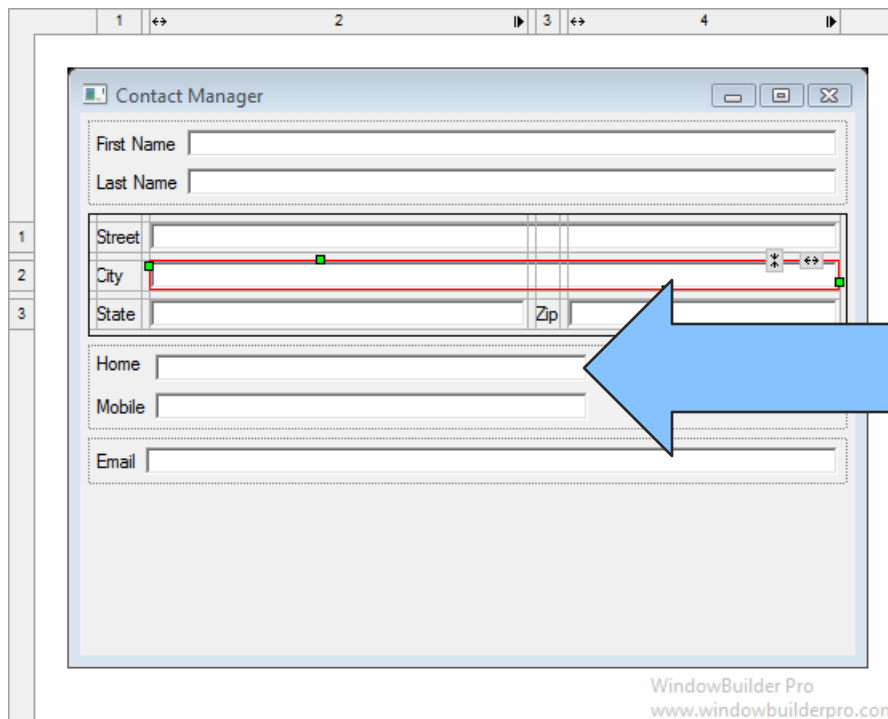
- WYSIWYG & Bi-directional Code Generation
- Powerful & Flexible Code Parser
- Single Representation
- Read & Write Any Style
- Micro Edits
- Internationalization
- Visual Inheritance
- UI Factories
- Morphing
- Widgets & Layout Managers
- Graphical Menu Editing



WYSIWYG & Bi-directional Code Generation

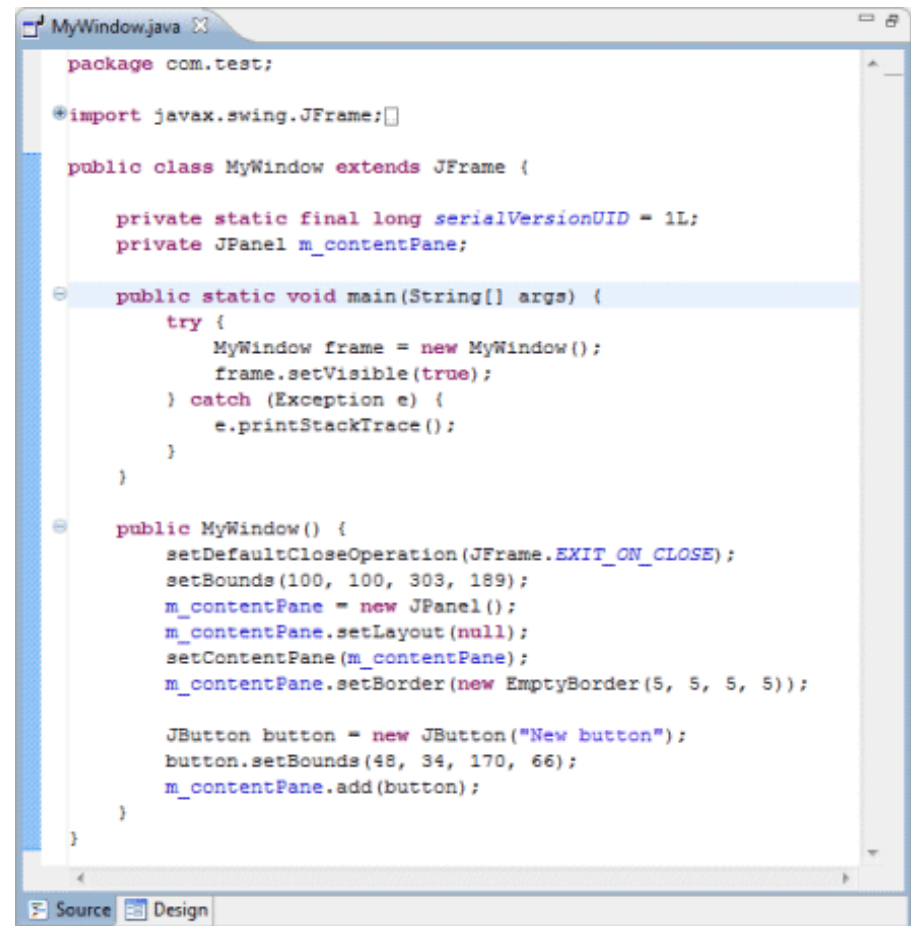


Provides a WYSIWYG visual designer coupled with bi-directional code generation



Contains a powerful & flexible code parser

- Can parse its own code
- Can parse code from any GUI tool
- Can parse code written by hand
- No protected code blocks
- Understands data flow
- Ignores and preserves non-UI code
- Refactoring friendly
- Resilient to hand-made changes



```
package com.test;

import javax.swing.JFrame;

public class MyWindow extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel m_contentPane;

    public static void main(String[] args) {
        try {
            MyWindow frame = new MyWindow();
            frame.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

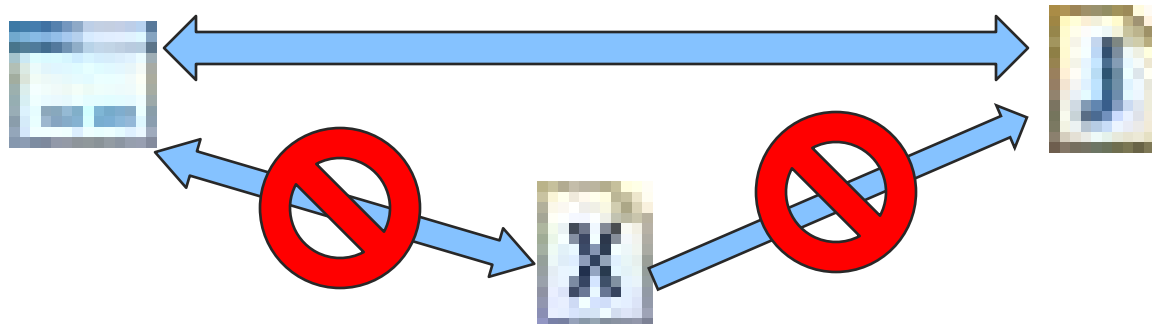
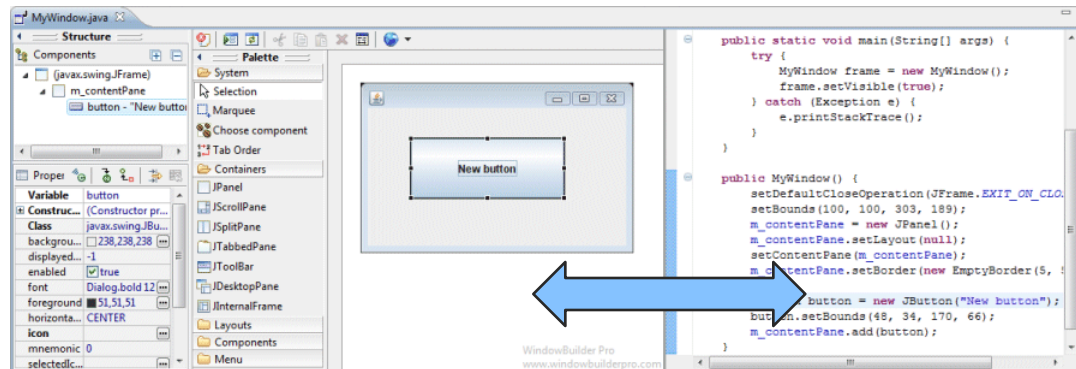
    public MyWindow() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 303, 189);
        m_contentPane = new JPanel();
        m_contentPane.setLayout(null);
        setContentPane(m_contentPane);
        m_contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        JButton button = new JButton("New button");
        button.setBounds(48, 34, 170, 66);
        m_contentPane.add(button);
    }
}
```

Single Representation

Maintains only one representation of the GUI code

- One-to-one relationship between UI and Java/XML code
- No intermediate metadata file to get lost or out of sync

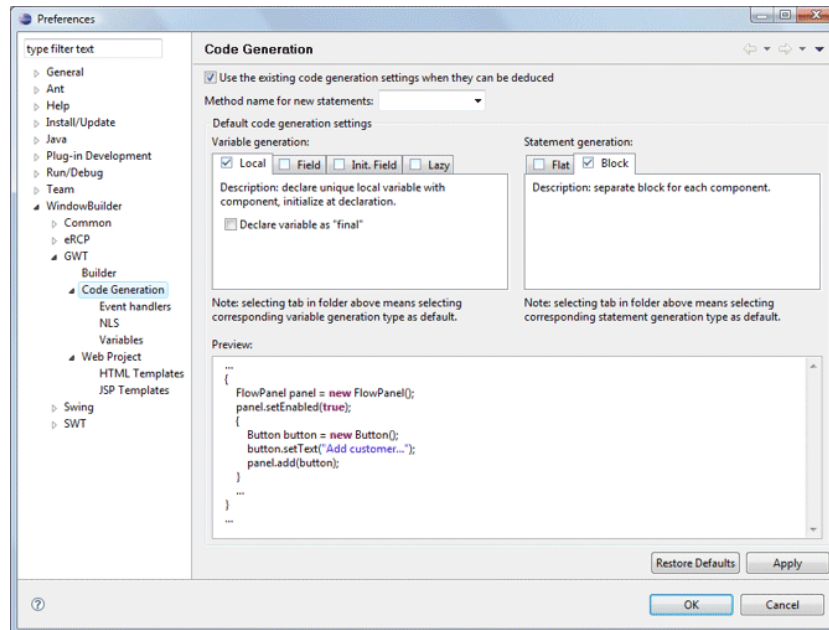


Read & Write Any Style



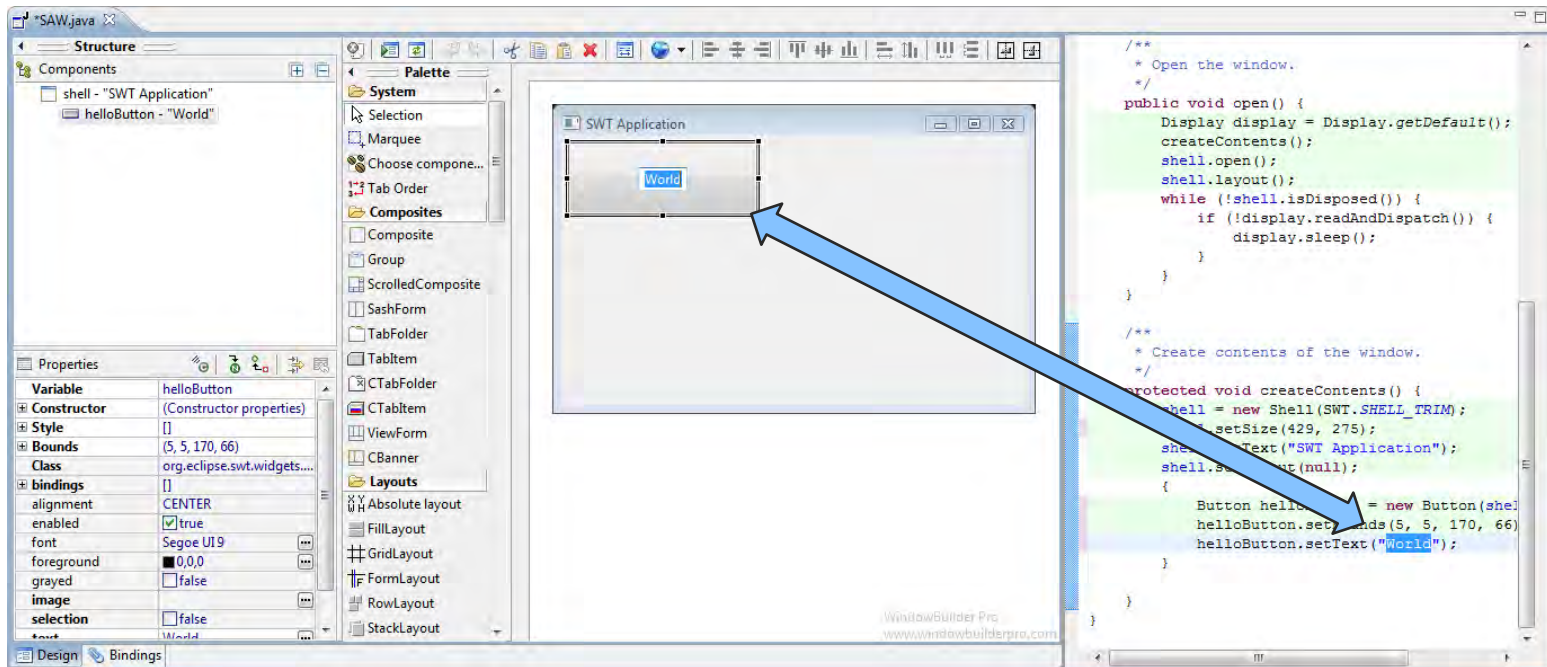
Reads and writes code in any format or style

- Local variables vs. Fields
- Initialized fields
- Lazy declaration
- Flat vs. Block

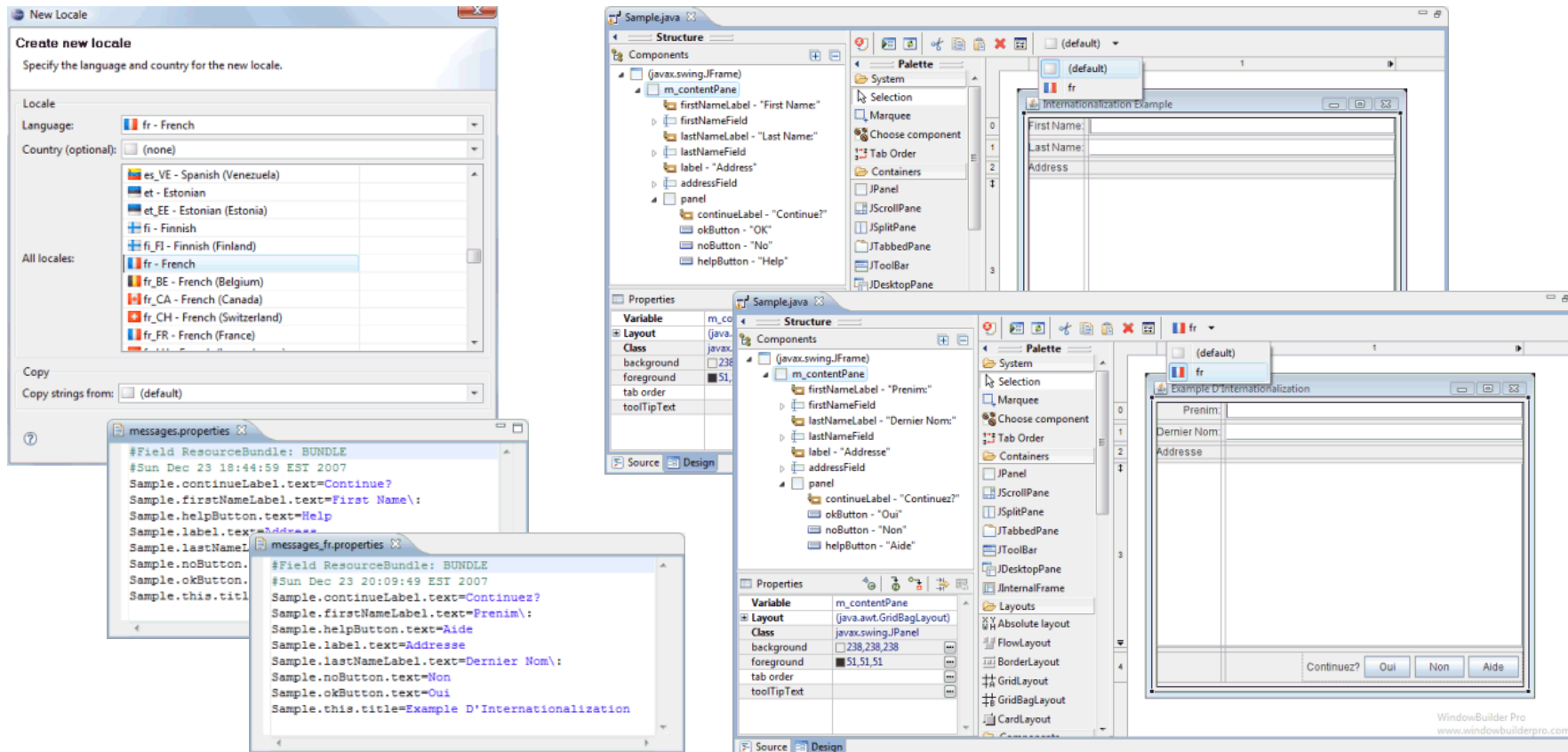


Makes micro edits to the source GUI code

- Smallest possible edit to make a code change
- Respects user formatting and refactoring

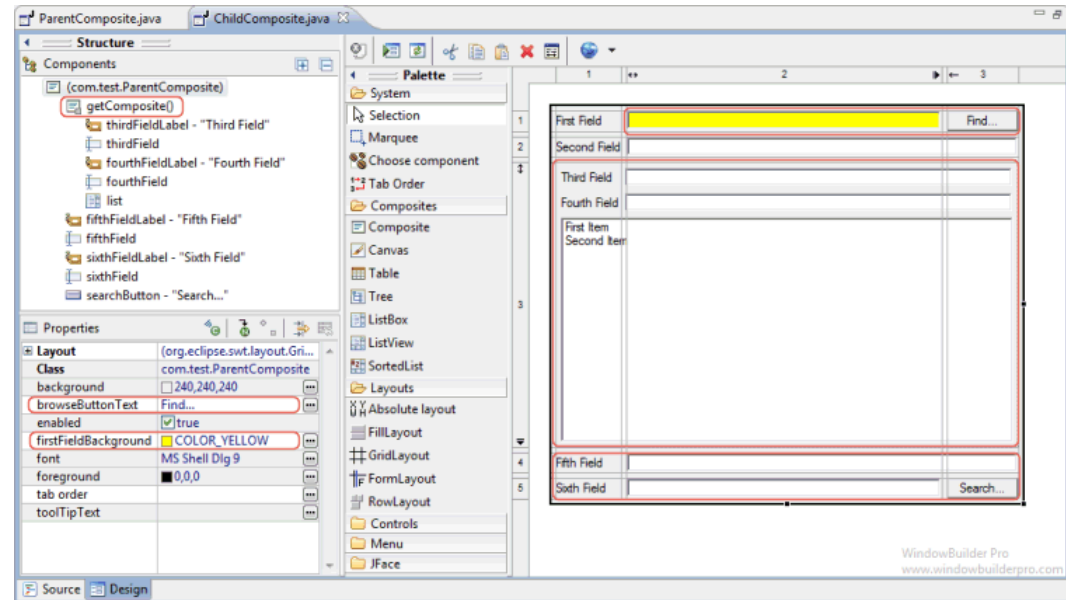
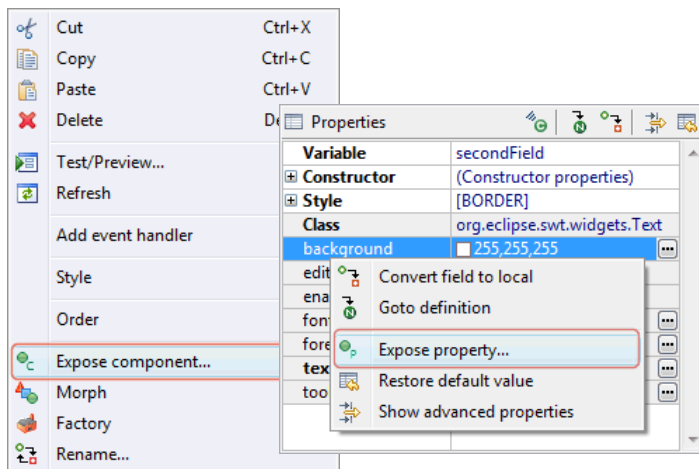


Offers easy-to-use Internationalization and Localization tools

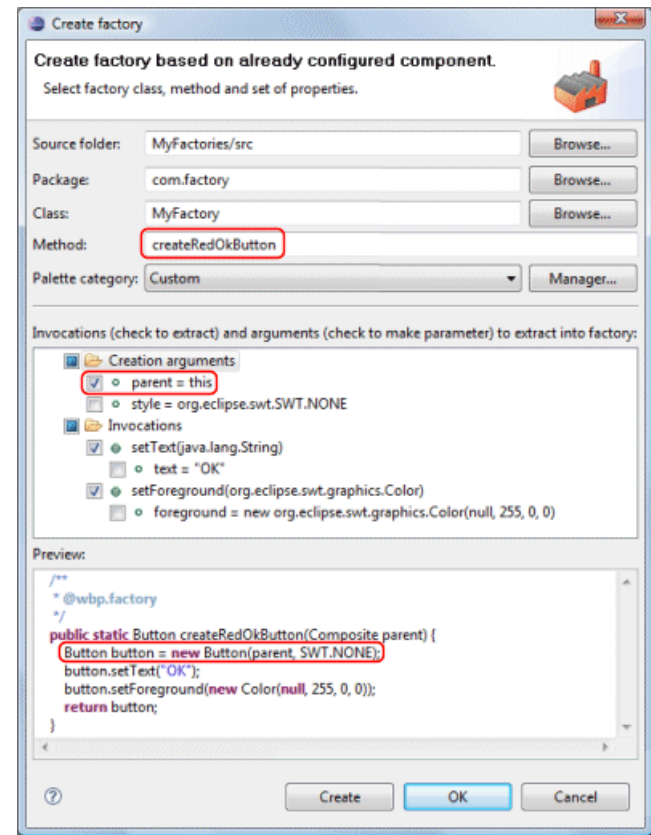
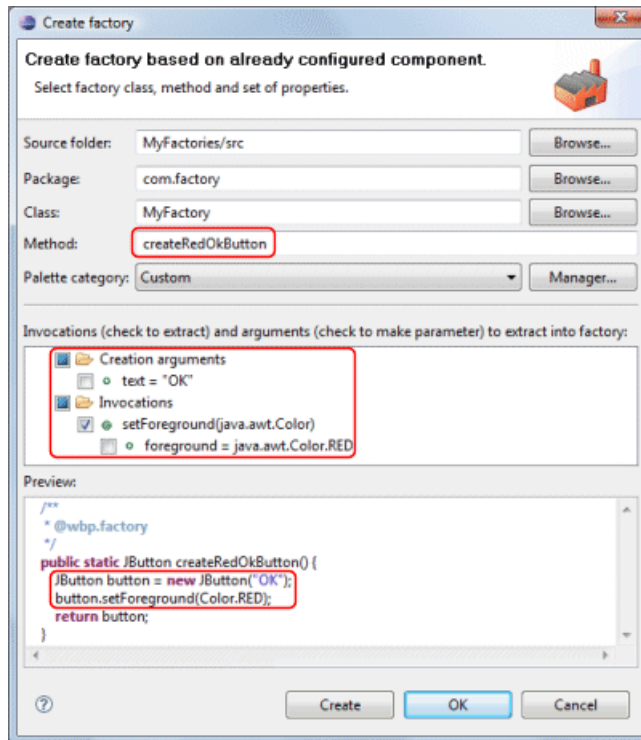


Provides visual inheritance so that code features can be easily inherited from a parent – child hierarchy

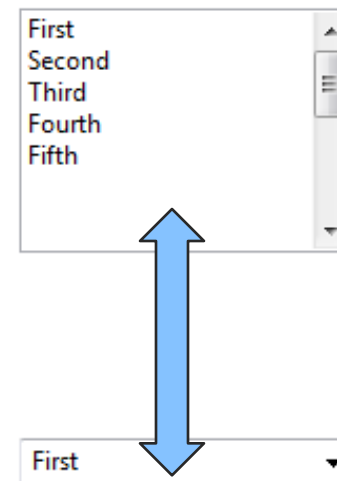
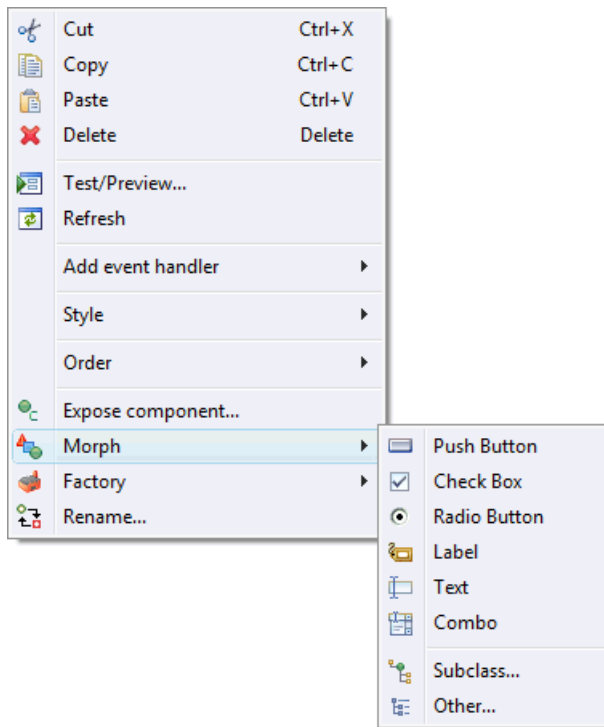
- Change properties of inherited fields
- Add components & event handlers to inherited fields
- Change public properties of added components
- Easily expose fields and properties



Contains support for UI Factories and reusable customized GUI elements



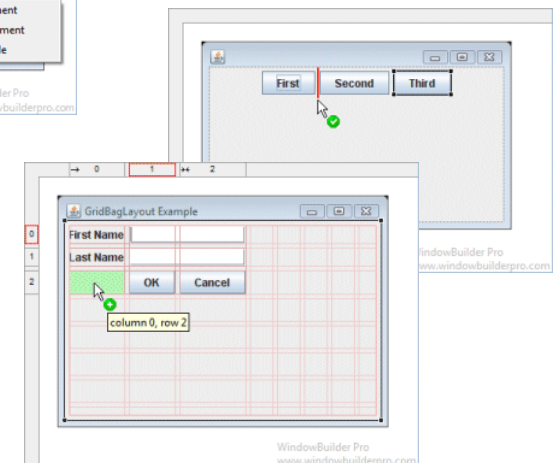
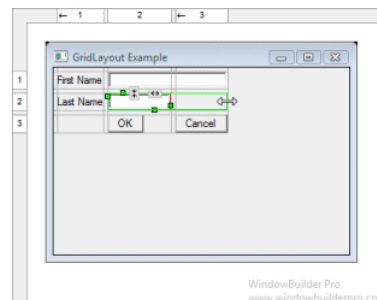
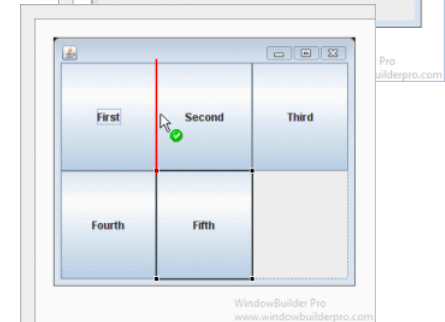
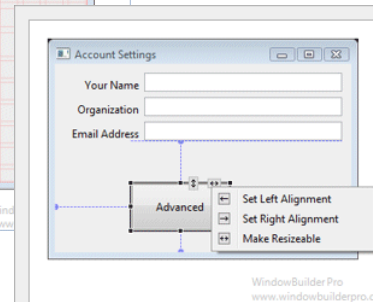
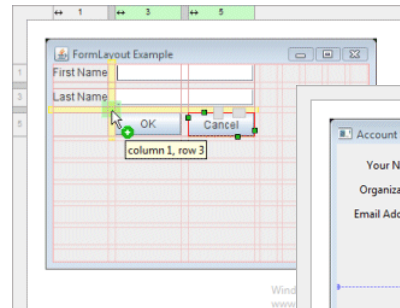
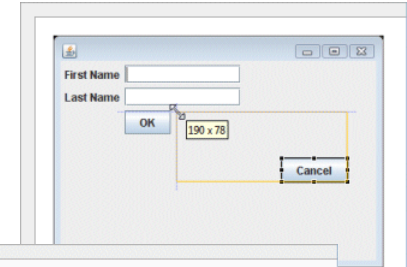
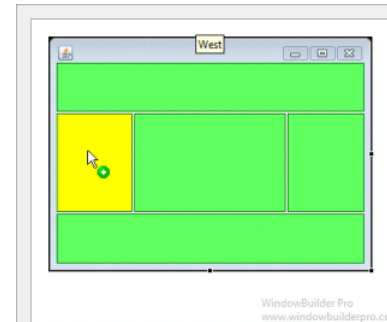
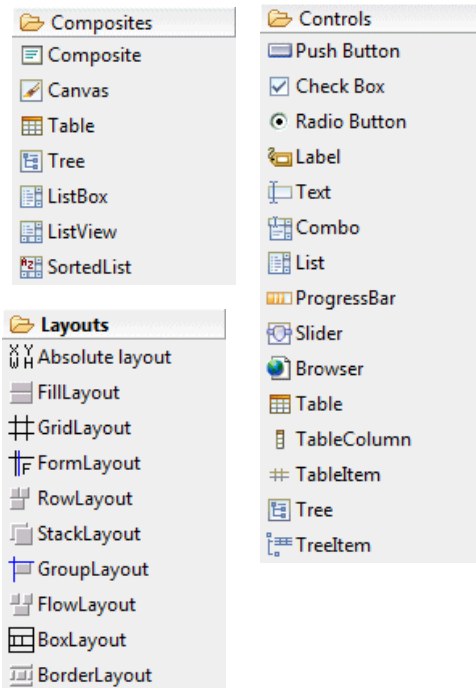
Provides a Morphing tool to easily change one widget type into another



Widgets & Layout Managers



Fully supports all standard widgets and layout managers as well as select third-party widgets and layout managers

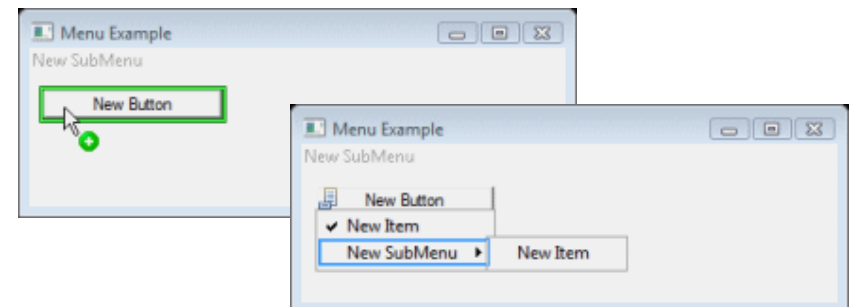
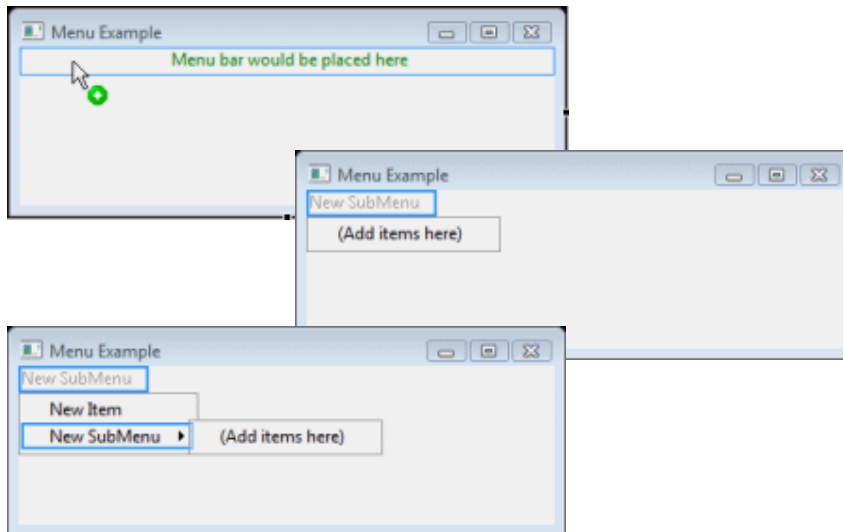
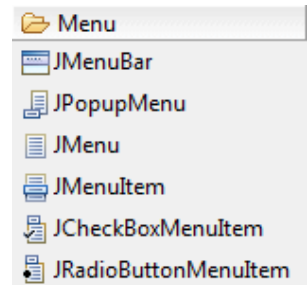
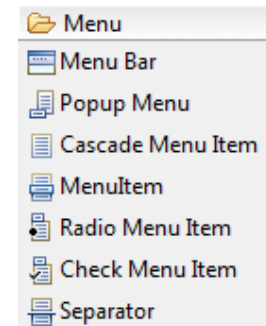


Graphical Menu Editing



Supports WYSIWYG Graphical Menu Editing

- Graphical edit menubars and popup menus
- Use drag/drop to rearrange menus
- Direct edit menu labels

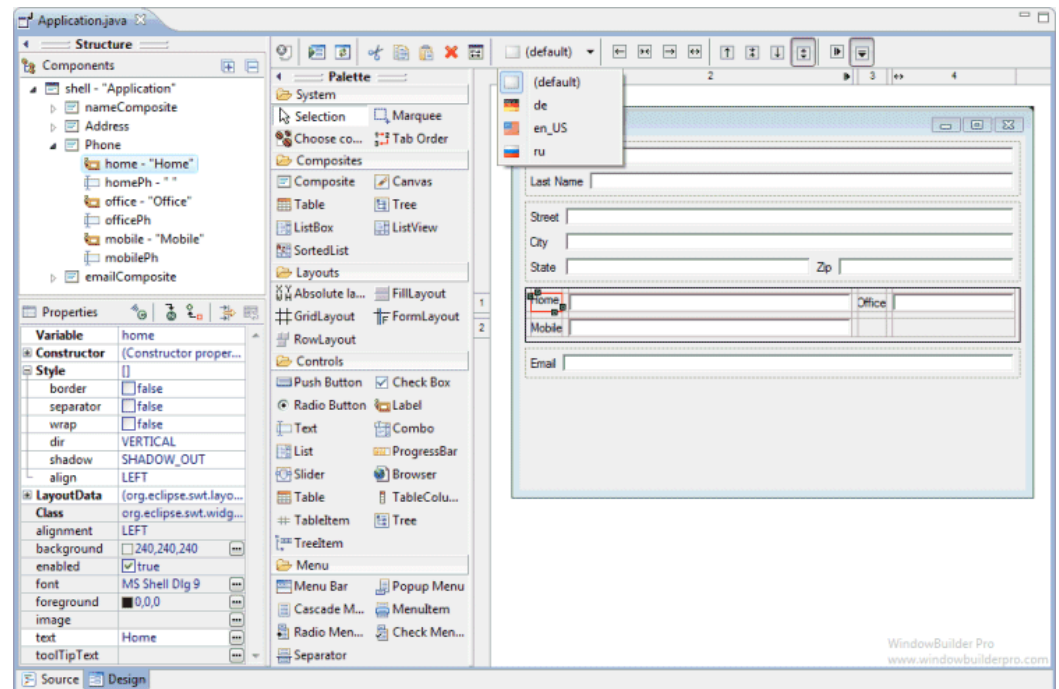


WindowBuilder User Interface



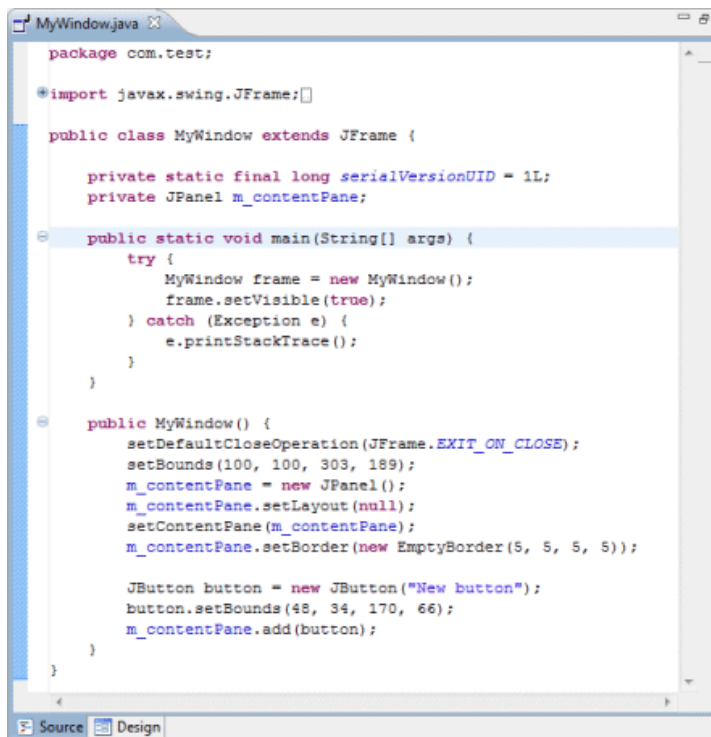
WindowBuilder is composed of the following major components

- Source View
- Design View
- Structure View
 - Component Tree
 - Property Pane
- Palette
- Wizards
- Toolbars & Context Menus
- Event Handlers
- Data Binding



Source View

- Separate tabs or top/bottom/left/right with splitbar
- Bi-direction code generation



```
package com.test;

import javax.swing.JFrame;

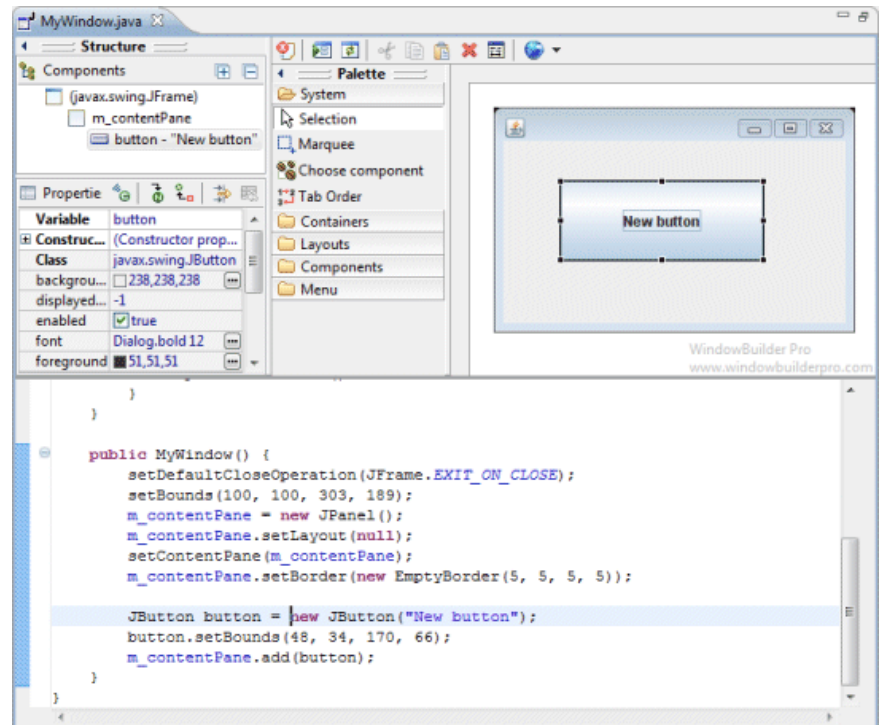
public class MyWindow extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel m_contentPane;

    public static void main(String[] args) {
        try {
            MyWindow frame = new MyWindow();
            frame.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

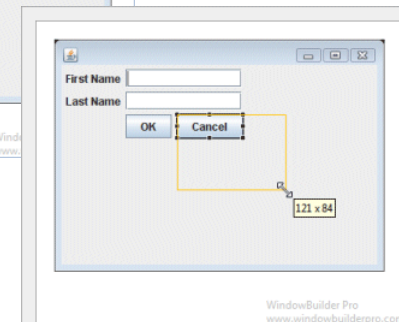
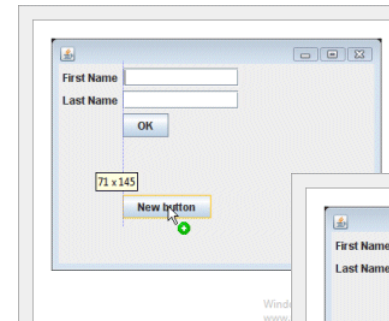
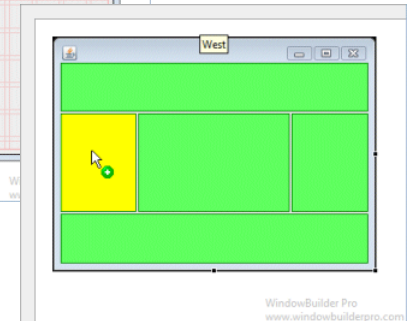
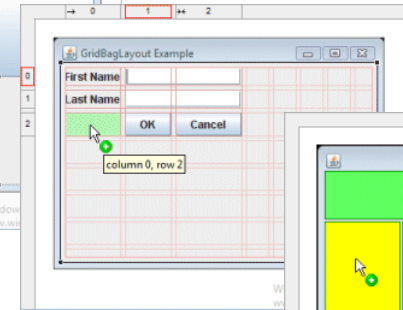
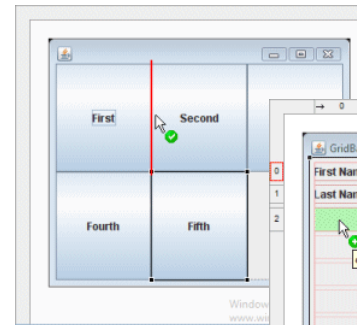
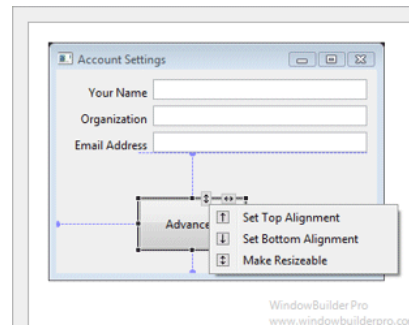
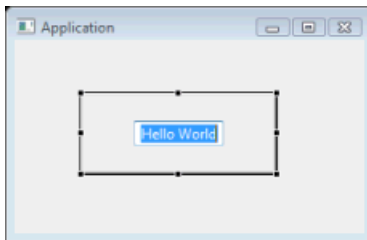
    public MyWindow() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 303, 189);
        m_contentPane = new JPanel();
        m_contentPane.setLayout(null);
        setContentPane(m_contentPane);
        m_contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

        JButton button = new JButton("New button");
        button.setBounds(48, 34, 170, 66);
        m_contentPane.add(button);
    }
}
```



Design View

- Each layout has its own UI model
- Move & resize components visually
- Direct edit component labels
- Select attachments using popup context menu

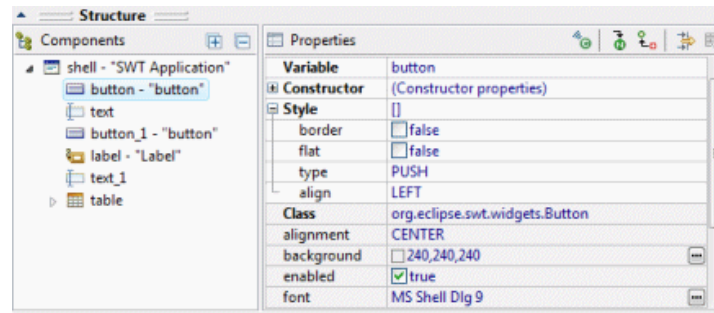
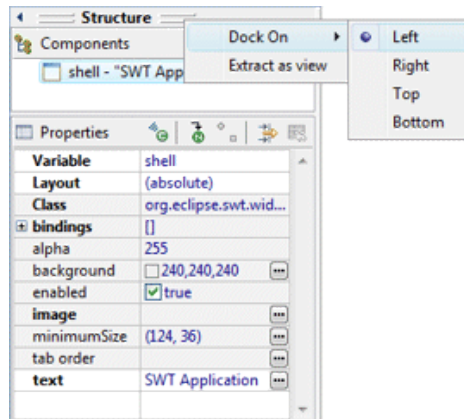
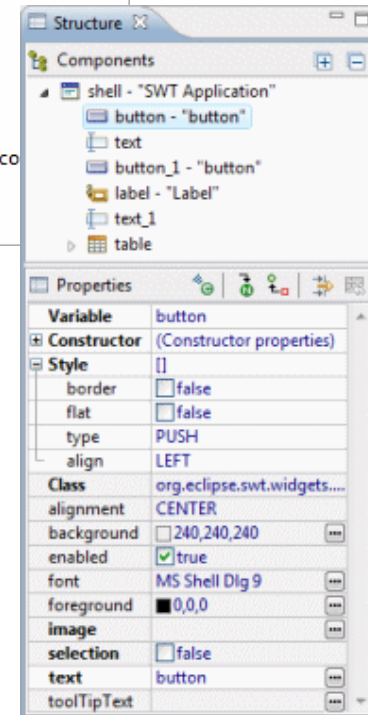
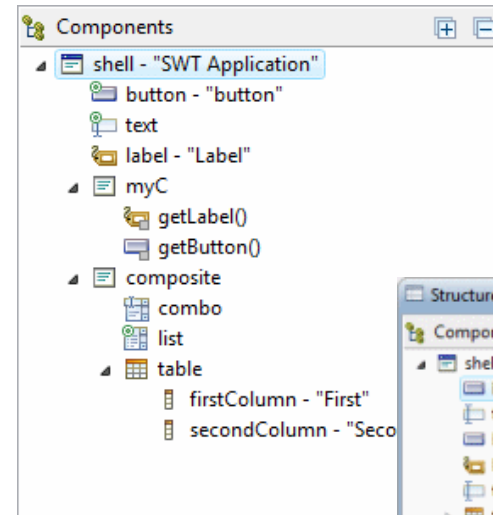


Component Tree



Component Tree





- Hierarchical view of all components
- Displays inst var name of component
- Shows text of component
- Right-click to cut/copy/paste multiple widgets
- Use drag/drop to reorder and nest
- Dock to editor or open as standalone view

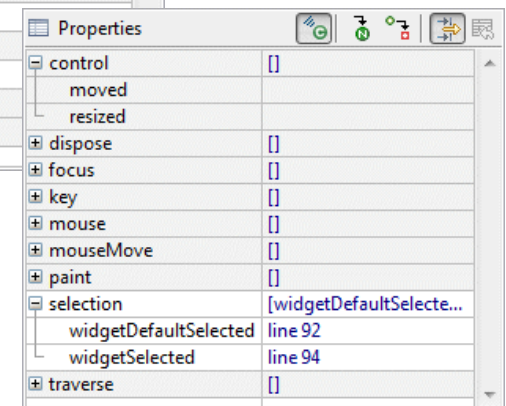
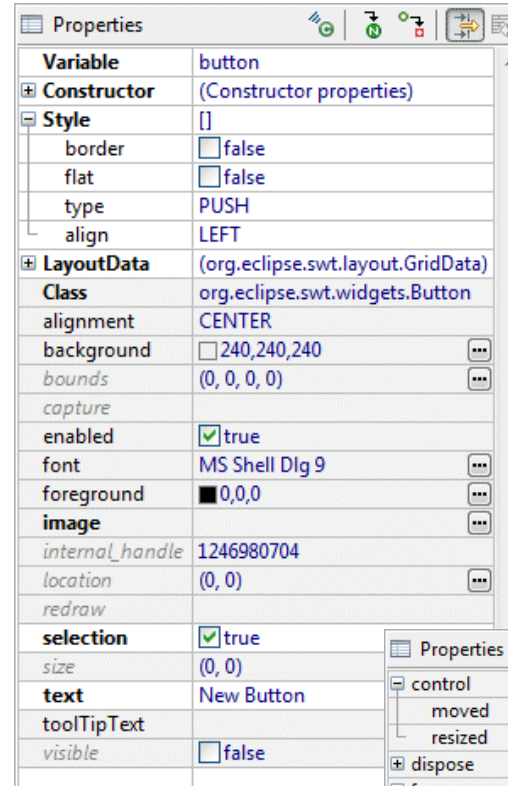
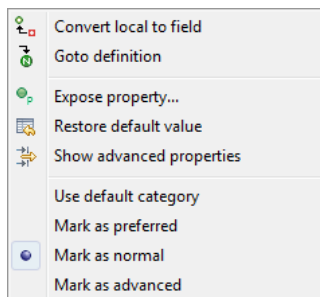


Properties View

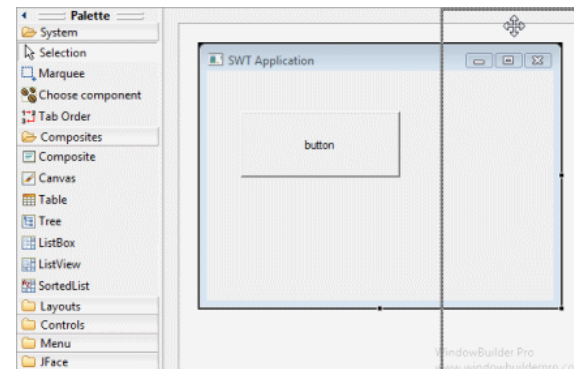
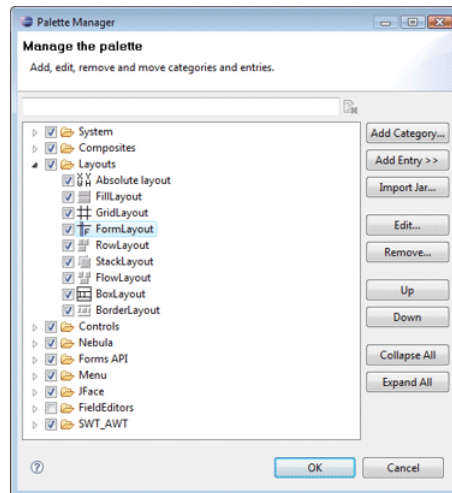
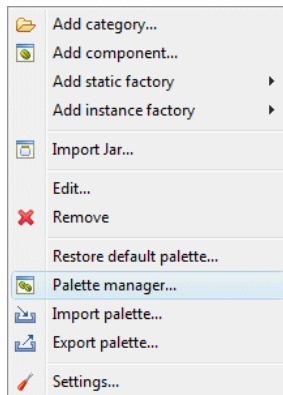
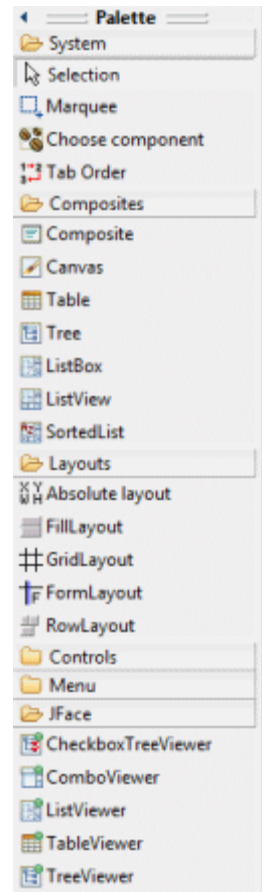
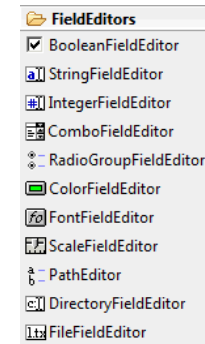


Properties View

- Supports all common properties
- Hide & Show selected properties 
- Highlight important properties
- Convert local/field 
- Open definition 
- Show events 
- Mark hidden & important properties
- Hover help for all properties



- Single select or marquee select
- **Choose component** to select arbitrary components
- Palette adapts to window type
- Special palettes for Eclipse Forms, PreferencePage Field Editors
- Palette can dock to editor or open as standalone view
- Fully customizable by user

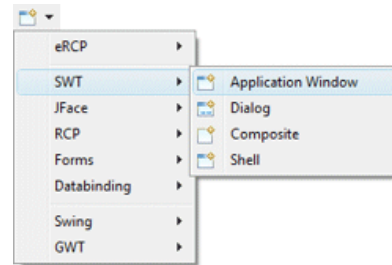


WindowBuilder contains dozens of new UI wizards

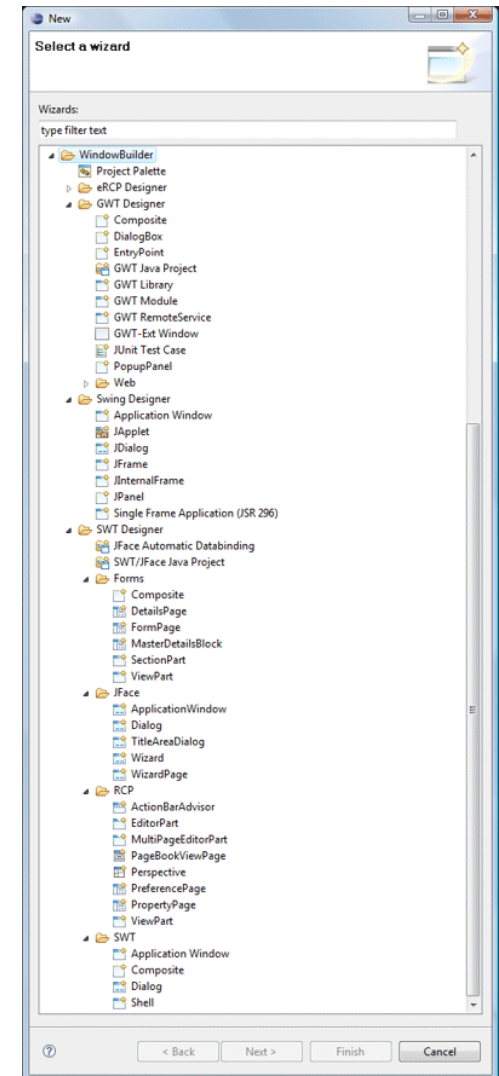
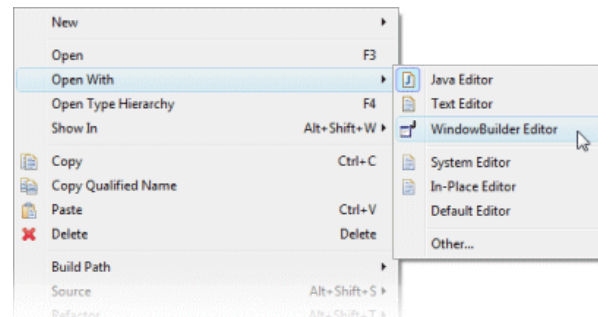
- Select **File > New > Other > WindowBuilder**

- Or use drop down tool button menu

- Dozens of templates for creating RCP, JFace, SWT, XWT, Swing and GWT components
 - RCP – Views, Editors, Perspectives, Preference & Property pages
 - JFace – App Windows, Wizards Pages, etc.
 - SWT – Composites, Shells & Dialogs
 - Swing – Frames, Panels, Dialogs, etc.
 - GWT – EntryPoint, Composites, etc.

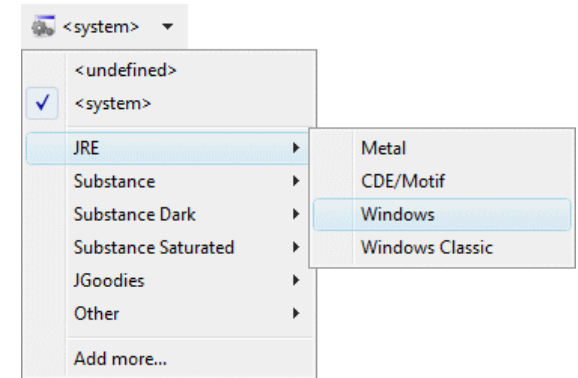
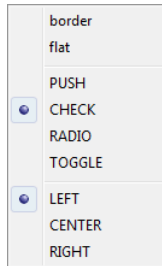
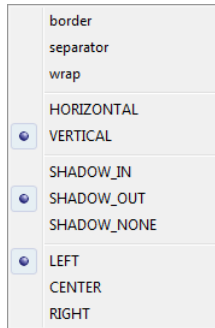
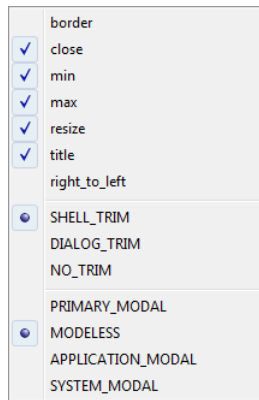


- Edit an existing class using **Open With > WindowBuilder Editor**

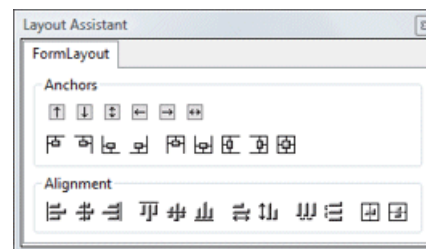
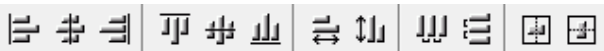
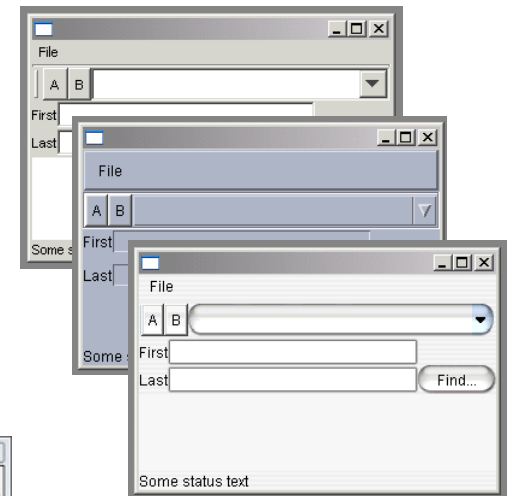
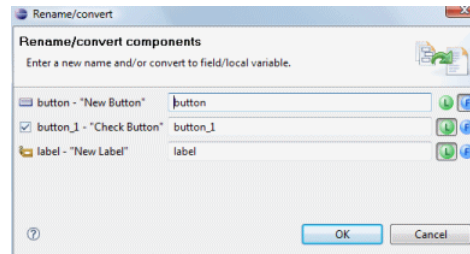


Toolbars & Context Menus

- Set Swing LookAndFeel using drop down look menu
- Change SWT styles using the context menu

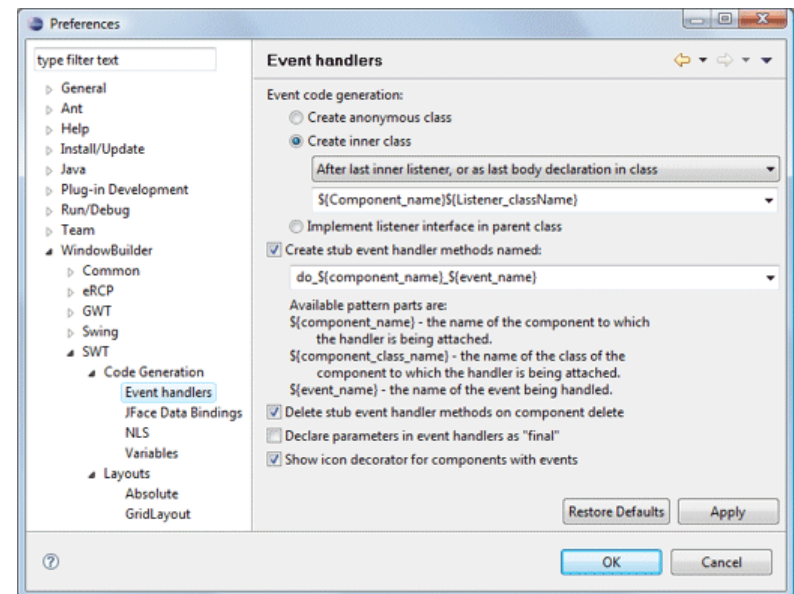
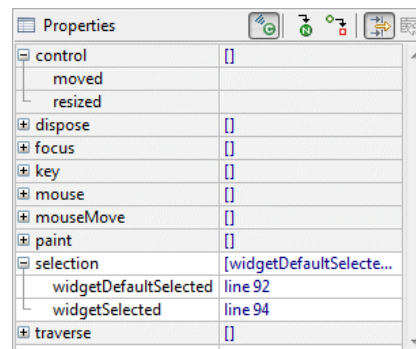
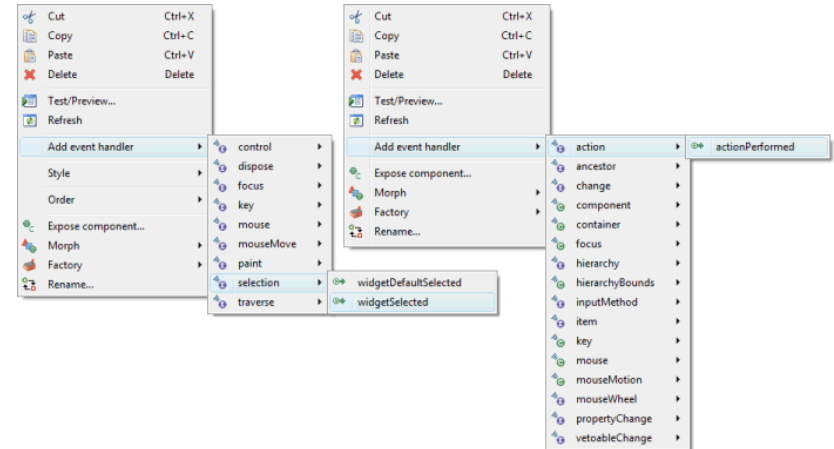


- Rename components
- Delete components
- Floating Layout Assistant
- Align & Distribute components



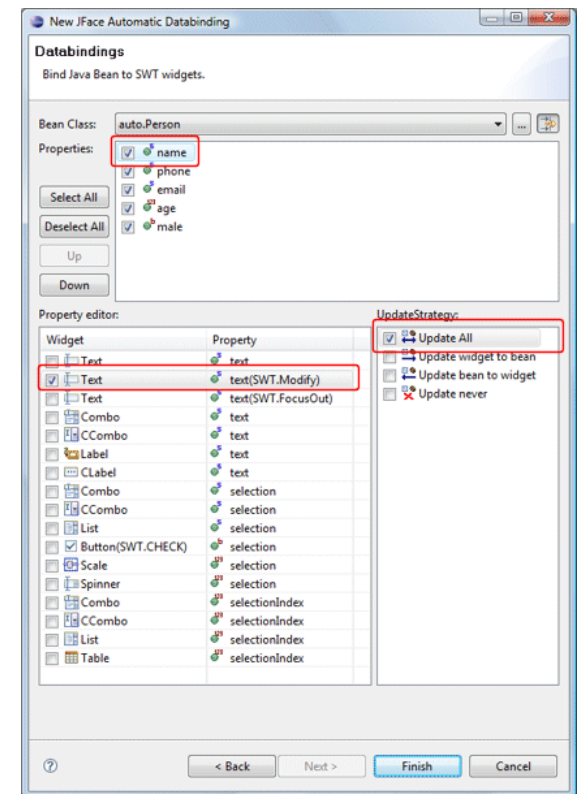
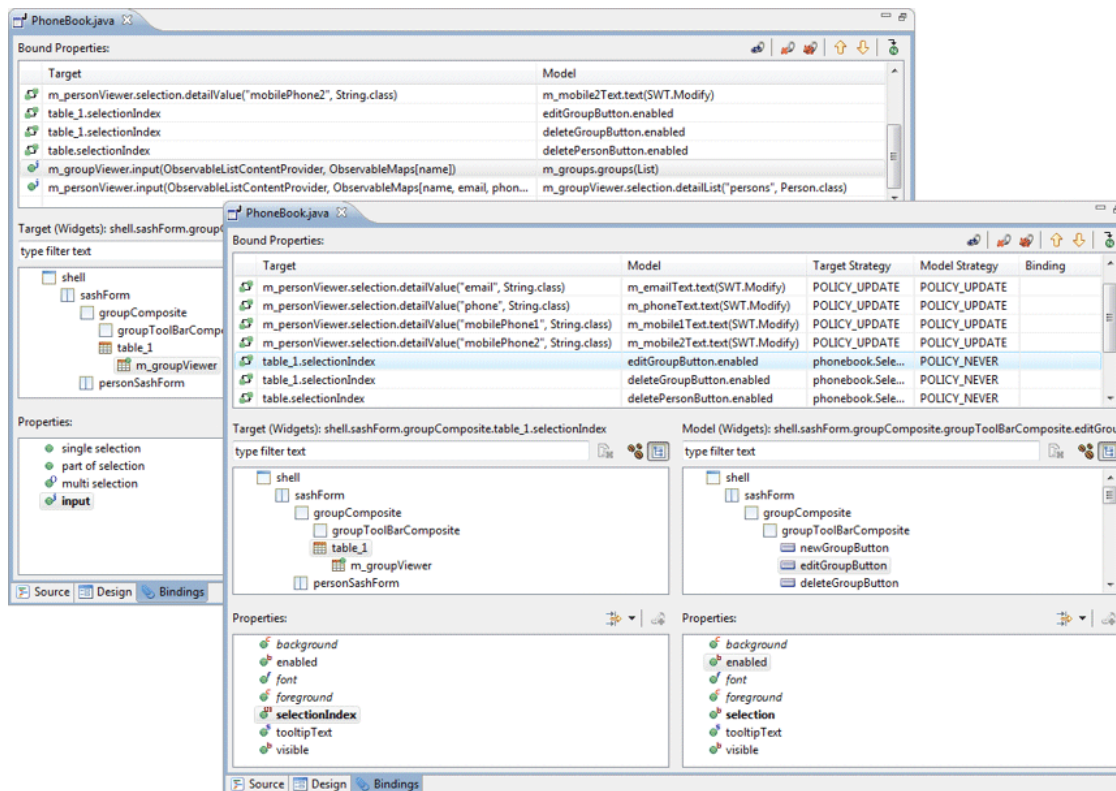
Event Handling

- Add events using Events panel of the Property pane or context menu
- Create event handler
 - as anonymous inner classes
 - as named inner classes
 - add listener interface to the class itself
- Handle event inline within the inner class or add a stub method handler called from the inner class
- Delete a handler by hitting Delete in Property pane



Data Binding

- Create and edit SWT, JFace & Swing data bindings
- Automatic Data Binding wizard



Customization

Developer API



There are three layers to adding new components

- Just add component to palette
 - Palette contribution in project
 - Palette contribution in jar
 - Palette contribution from plugin.xml
 - Palette commands in project
- Describe component using *.wbp-component.xml
 - Component constructors
 - Component properties
- Write Java code for special model/layout features
- UI toolkit auto-discovery mechanism

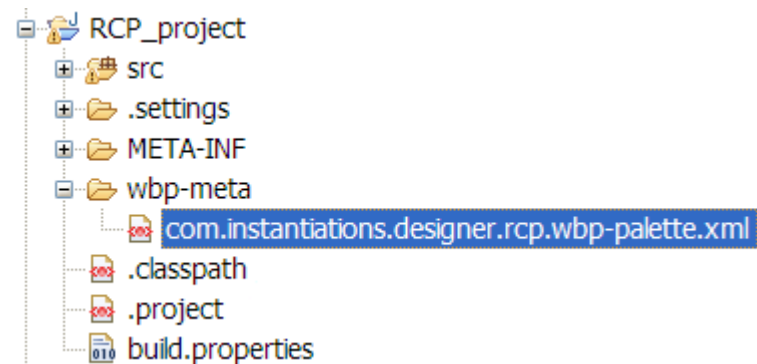
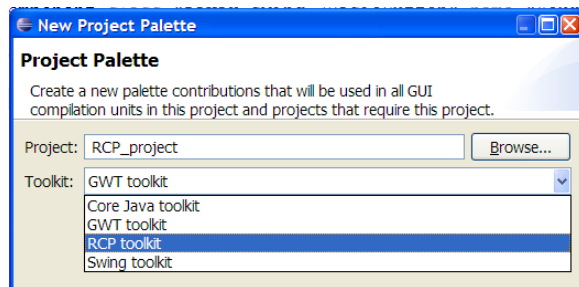


Palette Contribution in Project – New Category

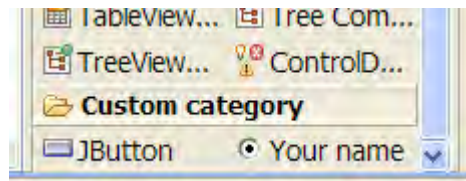
- Place a **«toolkitID».wbp-palette.xml** file into your project's **«wbp-meta»** folder with the desired **«category»** and **«component»** entries

```
<?xml version="1.0" encoding="UTF-8"?>
<palette>
  <category id="someUniquelid" name="Custom category" description="Custom category" open="true">
    <component class="javax.swing.JButton"/>
    <component class="javax.swing.JRadioButton"
      name="Your name" description="Any description here."/>
  </category>
</palette>
```

- You can use a wizard to generate the file for you



- Here's how it looks



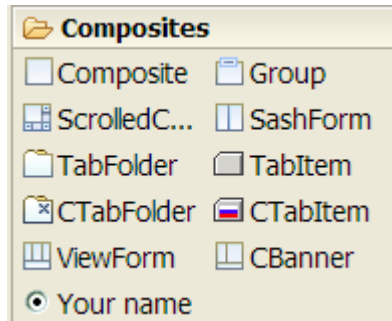
Palette Contribution in Project – Existing Category



- You can define a component in an existing category

```
<component  
  category="org.eclipse.wb.rcp.composites"  
  class="javax.swing.JRadioButton"  
  name="Your name"/>
```

- Here's how it looks



- For a custom icon, place a png/gif icon with same name in the same package or the same package within the «wbp-meta» folder

- The «**toolkitID**».wbp-palette.xml file can be placed in any project required by your project
- The «**toolkitID**».wbp-palette.xml file can be placed in the same jar file as the contributed components
- Simply adding **my-components.jar** to classpath will also automatically add them to the palette

- If you want more control over the palette, create a plugin and contribute to the palette via the **plugin.xml** file

```
<extension point="org.eclipse.wb.core.toolkits">
  <toolkit id="org.eclipse.wb.rcp">
    <classLoader-library bundle="org.eclipse.wb.rcp.nebula.lib" jar="cdatetime-0.9.0.jar"/>
      <palette>
        <category id="org.eclipse.wb.rcp.nebula"
          name="Nebula" description="Nebula custom widgets"
          next="org.eclipse.wb.rcp.FormsAPI">
          <!-- CDateTime -->
          <component class="org.eclipse.swt.nebula.widgets.cdatetime.CButton">
            <library type="org.eclipse.swt.nebula.widgets.cdatetime.CButton"
              bundle="org.eclipse.wb.rcp.nebula.lib" jar="cdatetime-0.9.0.jar"/>
            </component>
          </category>
        </palette>
      </toolkit>
    </extension point>
  </extension point>
</extension point>
```

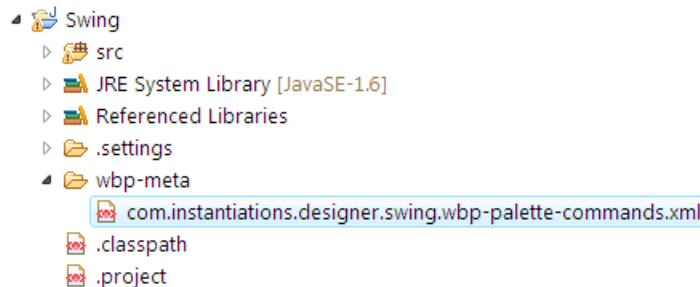
- The «**classLoader-library**» tag tells WindowBuilder that this jar should always be added into the ClassLoader (even if it is not in classpath)
- Specify the «**library**» element for a component to tell WindowBuilder that the library should be added to the classpath the first time you try to use the component
- For a full description of palette related attributes, see **toolkits.exsd** schema

Palette Commands in Project



If you wish to use different palettes in different projects or share a palette between developers, you should:

- Place an empty **«toolkitID».wbp-palette-commands.xml** file into your project's **«wbp-meta»** folder



- WindowBuilder will save all palette modification operations (move, add or delete components) to this file and read from it later
- For example, after moving the **«Layouts»** category before **«Containers»**, the file will have the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<commands>
  <moveCategory
    id="org.eclipse.wb.swing.layouts"
    nextCategory="org.eclipse.wb.swing.containers"/>
</commands>
```

- Use a ***.wbp-component.xml** file to describe the component
- Place a ***.wbp-component.xml** file with same name in the same package or the same package within the «wbp-meta» folder

```
<?xml version="1.0" encoding="UTF-8"?>
<component xmlns="http://www.eclipse.org/wb/WBPComponent">
  <model class="org.eclipse.wb.swt.model.widgets.LabelInfo"/>
  <description>Instances of this class represent a non-selectable user interface object that displays a string or image.
    When SEPARATOR is specified, displays a single vertical or horizontal line.</description>
  <!-- CREATION -->
  <creation>
    <source><![CDATA[new org.eclipse.swt.widgets.Label(%parent%, org.eclipse.swt.SWT.NONE)]]></source>
    <invocation signature="setText(java.lang.String)"><![CDATA["New Label"]]></invocation>
  </creation>
  <creation id="separatorHorizontal" name="Horizontal Separator">
    <source><![CDATA[new org.eclipse.swt.widgets.Label(%parent%, SWT.SEPARATOR | SWT.HORIZONTAL)]]></source>
    <description>Horizontal separator.</description>
  </creation>
</component>
```

- For a full description of component related attributes, see **schema/wbp-component.xsd** schema in the **org.eclipse.wb.core** plugin

- Components may have multiple constructors with values bounds to different method-based properties
- Use the «**property**» attribute to map a constructor argument to a method signature or field name

<!-- CONSTRUCTORS -->

```
<constructors>
  <constructor>
    <parameter type="java.lang.String" property="setText(java.lang.String)"/>
  </constructor>
  <constructor>
    <parameter type="java.lang.String" property="setText(java.lang.String)"/>
    <parameter type="int" property="setHorizontalAlignment(int)"/>
  </constructor>
  <constructor>
    <parameter type="javax.swing.Icon" property="setIcon(javax.swing.Icon)"/>
  </constructor>
  <constructor>
    <parameter type="javax.swing.Icon" property="setIcon(javax.swing.Icon)"/>
    <parameter type="int" property="setHorizontalAlignment(int)"/>
  </constructor>
  <constructor>
    <parameter type="java.lang.String" property="setText(java.lang.String)"/>
    <parameter type="javax.swing.Icon" property="setIcon(javax.swing.Icon)"/>
    <parameter type="int" property="setHorizontalAlignment(int)"/>
  </constructor>
</constructors>
```

- Mark properties as rarely used (advanced), sometimes used (normal), frequently used (preferred) or hidden

```
<!-- PROPERTIES ->
```

```
<properties-preferred names="text icon labelFor"/>
```

```
<properties-advanced names="border disabledIcon displayedMnemonicIndex iconTextGap"/>
```

```
<properties-hidden names="UI"/>
```

```
...Or...
```

```
<property id="setDisplayedMnemonic(char)">
```

```
  <category value="preferred"/>
```

```
</property>
```

- Special tags are supported for some properties such as the «isText» tag to mark which property to support with direct edit

```
<property-tag name="text" tag="isText" value="true"/>
```

- For property types, WindowBuilder includes a standard editor. For enumerations, special configuration options are available

```
<property id="setVerticalAlignment(int)">
```

```
  <editor id="staticField">
```

```
    <parameter name="class">javax.swing.SwingConstants</parameter>
```

```
    <parameter name="fields">TOP CENTER BOTTOM</parameter>
```

```
  </editor>
```

```
</property>
```

```
<property id="setHorizontalAlignment(int)">
```

```
  <editor id="staticField">
```

```
    <parameter name="class">javax.swing.SwingConstants</parameter>
```

```
    <parameter name="fields">LEFT CENTER RIGHT LEADING TRAILING</parameter>
```

```
  </editor>
```

```
</property>
```


Sometimes descriptions are not enough and you need special editing behavior (for example, layout edit policies for various layout managers).

- Custom code need to live in a plug-in
- Specify the «**model**» to use for a component in its description

```
<model class="org.eclipse.wb.swing.model.component.ComponentInfo"/>
```

- Class **ComponentInfo** is an indirect subclass of **JavaInfo** that provides basic support for all Java models.
- All of your models will be subclasses of **ComponentInfo/ContainerInfo** (for Swing) or **ControllInfo/CompositeInfo** (for SWT).
- WindowBuilder provides various broadcast notifications that can be used to be informed about an event or participate in them
 - Add properties to any component:
`org.eclipse.wb.core.model.broadcast.JavaEventListener.addProperties(JavaInfo, List<Property>)`
 - Delete components and associated resources:
`org.eclipse.wb.core.model.broadcast.JavaEventListener.deleteAfter(JavaInfo, JavaInfo)`

UI Toolkit Auto-Discovery Mechanism

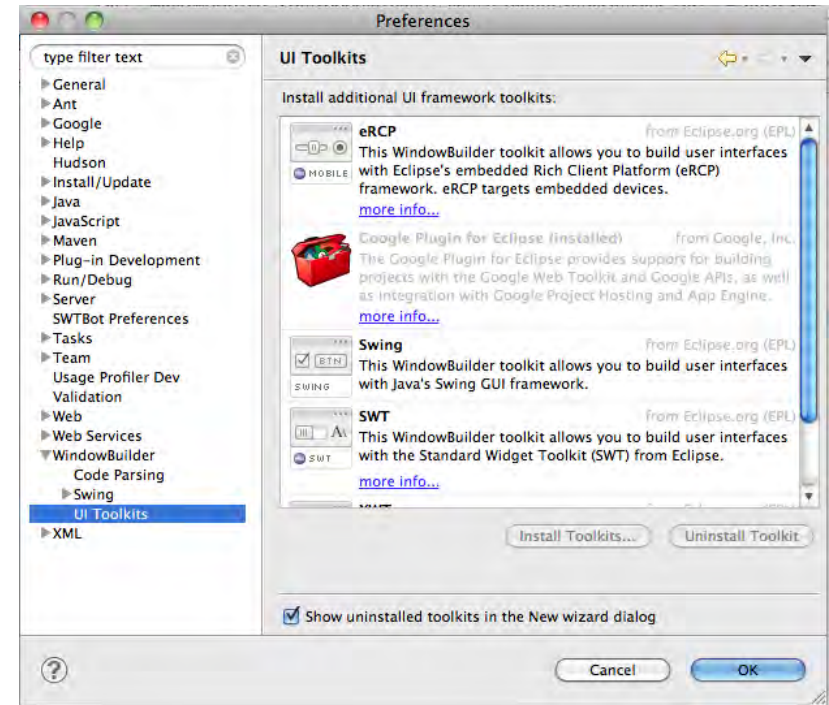


UI toolkit auto-discovery available in WindowBuilder Engine (WBE)

- Concept pioneered by Mylyn
- "Vendor-neutral" but pre-populated with eRCP, SWT, Swing and XWT
- Use Eclipse p2 mechanism to download and install code
- One-click install of any UI toolkit within any Eclipse client containing the WBE

Available at multiple user access points

- WBE preferences - show an explicit list of all known UI libraries supported by WBE and allow user to load any of them
- New Wizard - show a tree of all known UI types and offer to auto-load if user selects one not currently installed
- Open WBE editor on existing UI file - if user tries to edit a UI file for which no editing support is currently loaded, offer to auto-load it



Thank You!

Q&A

