

# CHAPTER 15

## USING THE UI BUILDER

The UI Builder enables you to visually design and build a graphical user interface (GUI) for a Java™ application. This chapter includes the following topics:

- “Overview of the UI Builder” on page 284
- “Activating the UI Builder” on page 285
- “Setting UI Builder options” on page 286
- “Creating visually editable UI classes” on page 287
- “Using the UI Designer” on page 290
- “Designing menus” on page 309
- “UI Builder audits and metrics” on page 323
- “Generating user interface documentation” on page 324
- “Customizing the Toolbox component palette” on page 324
- “Java features supported in designable classes” on page 328

### Overview of the UI Builder

---

The UI Builder is connected to source code, meaning that the relevant Java source code automatically generates and updates in real time as you add visual, nonvisual, and menu components to your user interface. You can optionally develop user interfaces by writing code and view the results using the UI Builder.

The UI Builder is an *activatable feature* in Together, meaning that you can turn it off when not needed to conserve system resources.

**WARNING!** The UI Builder depends on the RWI2 Inspector. This module is activated by default, when the UI Builder is activated. Never deactivate the RWI2 Inspector - this will make the UI Builder work incorrectly.

When activated, the UI Builder commands appear on the main View menu, a button is added to the Designer pane toolbar, and two additional views are available in the Designer pane:

- **UI Design.** This is a WYSIWYG<sup>1</sup> design view that enables you to see frames and other containers, and the components these contain.
- **Menu Design.** This is a specialized view enabling WYSIWYG design and implementation of menu bars and right-click menus.

When you work in one of the above Designer pane views, the **UI Builder** tab is added to the Explorer. This tab displays the components of your user interface, enables you to navigate the components of your user interface, and switch between UI Designer and Menu Designer views in the Designer pane. This tab is referred to as the *Component Explorer* throughout this chapter.

Any component selected in the Designer pane, is highlighted in the treeview of the Components Explorer, and vice versa.

## Component Explorer

---

The Component Explorer displays a treeview of the three possible categories of UI components. These are:

- **UI Components**  
The visual UI elements, or the UI Components per se (labels, buttons, combo boxes and other controls) are listed in this node.
- **Menu Components**  
This node contains the various menu elements (menu bar, popup menu, menu items etc.)
- **Non-Visual Components**  
The various UI elements that have no visual representation (for example, be responsible for connection or queries to a database)

**TIP:** Visual Components can be added directly to the allowed locations of the container.

There is a sample Together project that you can use to explore the various kinds of things you can construct with the UI Builder. See: `$TGH$/samples/java/UIBuilder/UIBuilderSamples.tpr`.

## Activating the UI Builder

---

The UI Builder is an *activatable feature*. It is activated by default. This section explains the steps for activation and deactivation, and shows you what to look for in the user interface to know when the UI Builder is activated.

*To activate the UI Builder feature:*

1. On the main menu, choose **Tools | Activate/Deactivate Features**.

1. (“What-You-See-Is-What-You-Get”)

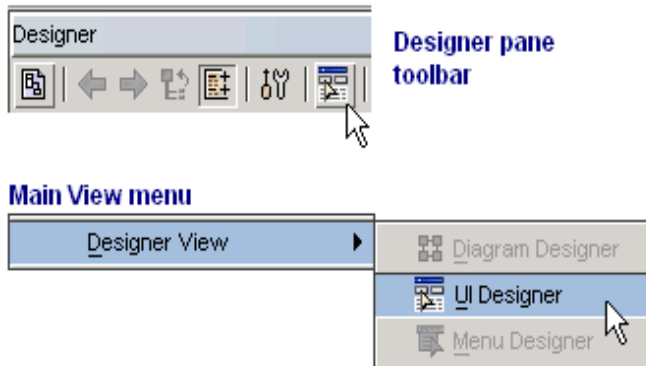
2. On the **Together Features** tab, check *UI Builder*.

**NOTE:** In some pre-release builds, the UI Builder may appear on the **Early Access** tab of the Activate/Deactivate Features dialog.

3. Click **OK** to activate the UI Builder module.

The UI Builder button is now enabled on the Designer pane toolbar and the UI Builder command is added to the main View menu, as shown in Figure 76.

**Figure 76** When activated, icons show on the Designer pane toolbar and main menu



## Deactivating the UI Builder

When you no longer need to work with the UI Builder, it is advisable to deactivate it. Deactivating decreases the use of system resources and helps optimize performance.

*To deactivate the UI Builder feature:*

1. On the main menu, choose **Tools | Activate/Deactivate Features**.
2. On the **Together Features** tab, clear the *UI Builder* box.
3. Click **OK** to confirm deactivation of the UI Builder module.

## Setting UI Builder options

There are a number of configuration options for the UI Builder feature. The default settings enable you to get started creating new graphical user interfaces. When you have existing user interface code, it may be necessary to change some of the settings.

*To access the UI Builder options:*

1. Choose **Tools | Options | Default** on the main menu to open the Options dialog.
2. Choose the *UI Builder* node in the Options explorer.

## Choosing a profile

---

Choose one of several pre-defined profiles in the *IDE profile* option, including one user-defined profile. (All the profiles are individually customizable.) The profile settings define what the UI Builder looks for in a UI class, and enable the UI Builder to handle source code created with other IDE products.

If you choose a profile for a competing IDE product, the UI Builder recognizes UI classes in source code that were created with the competing product. Such classes are treated as *visually editable* and are loaded into the UI Designer view in the Designer pane. Note that any new UI classes you create in Together are *not* created according to the IDE profile setting.

## Customizing a profile

The default values of the various profiles should suffice in most cases. You might want to customize an IDE profile if:

- A new version of a competing product changes the way it handles UI initialization, and so on in the source code it creates or generates, and you want to use Together to work on code created with that version.
- Your organization coding standards call for something different from the default values of the *Together* or *User-defined* profiles.

## Profile options

The following options are available for each profile:

- **InitGUI operation.** Defines the operation names that UI Builder requires to be present in any class. This allows UI Builder to treat it as a visually editable class that can be loaded into UI Designer view in the Designer pane. Specify multiple names separated by semi-colons. When multiple names are specified, a class is treated as visually editable if it contains any of the specified operations.
- **Visibility.** Defines the visibility scope of new attributes added to UI classes as you work on them.
- **Event handler style.** Defines the way UI Builder generates code for event handlers in new UI components you create. Choose between standard or anonymous inner class.
- **Object initialization point.** Defines the way UI Builder generates initialization code for new objects. Choose to have the object initialized at its declaration point or at the beginning of the class initialization method.

Of the above options, the first one is most likely to be significant for you initially. For example, if you are accustomed to JBuilder™, you might write a new UI class with a *jbinit()* operation. But if you have the *Together* profile selected, and you try to load the class into UI Designer view, the UI Builder rejects the attempt because the profile tells it to expect an *InitGUI()* operation.

## Creating visually editable UI classes

---

In order to develop a user interface using the UI Builder, complete the following preliminary steps:

1. Open a project. For instructions, see “Opening a project” on page 43 and “Creating new projects” on page 96.
2. Focus on a source code file in the Editor that represents a visually editable UI class. Normally you would do this by selecting the relevant class in a class diagram in the Designer pane. The next section provides more information on creating visually editable UI classes.

## What is a visually editable class?

---

Any file submitted to the UI Builder is scrutinized to determine whether it is suitable for visual editing. Such file should have \*.java extension, and should contain a designable (or visually editable) class.

A designable class must meet certain requirements on the source code level:

- The class should contain the initialization operation defined in the currently selected IDE profile (**Tools | Options | Default (or Project) - UI Builder: IDE Profile**).
- Its superclass should have either default (parameterless) constructor, or Creator Class, specified in the BeanInfo.
- The class scope must be declared *public*.

## Creating UI classes

---

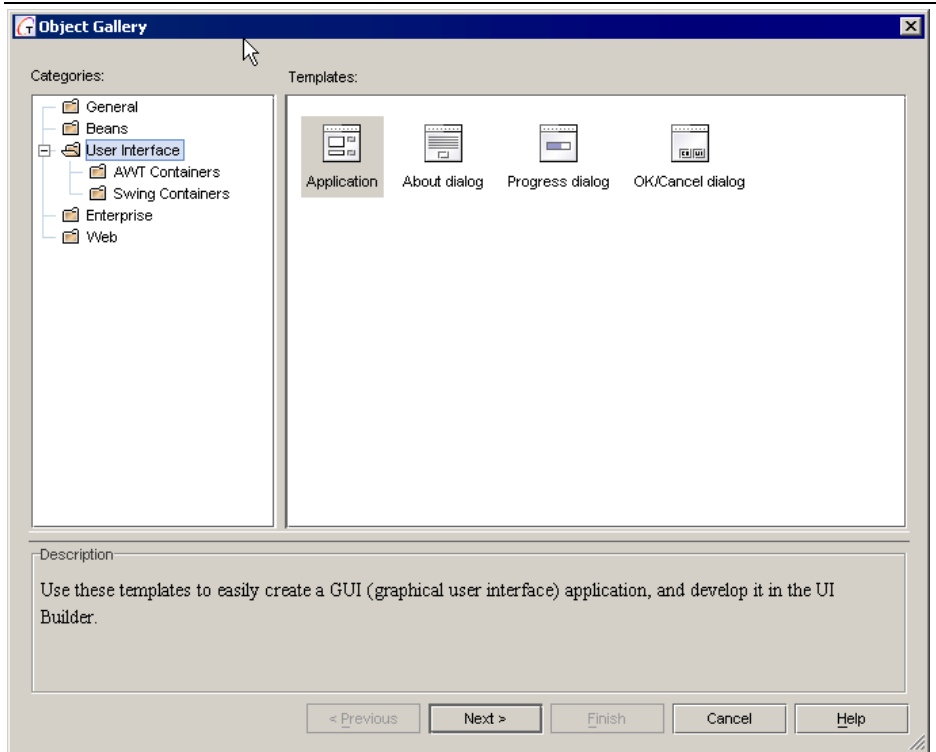
The *User Interface* category of the Object Gallery (**File | New: User Interface**) provides a quick way to create a number of designable UI classes. There you will find both AWT and Swing containers. It also provides a number of templates for more robust things such as applications and dialogs. (See Figure 77.)

*To create a designable class:*

1. On the main menu, choose **File | New: User Interface**.
2. Click on the desired category.
3. In the *Templates* area, choose the template, and click **Next**.
4. In the dialog window, fill in the necessary information, following the instructions provided by online help. Click **Finish** to complete.

**NOTE:** The User Interface templates are just patterns. They are not dynamic, and therefore do not look to the current IDE profile for the name of the initialization operation. The default initialization operation is *InitGUI()*.

**Figure 77** Object Gallery categories for UI components



There is an alternative way to create designable classes using the Choose Pattern dialog:

1. On the diagram right-click menu, choose **New | Class by Pattern**.
2. In the Choose Pattern dialog, expand the *UI Components* node.
3. Choose the desired pattern, and click **Finish**.

**TIP:** It is also possible to turn a non-UI class into a designable class by applying some of the UI patterns. To do that, choose **Choose Pattern** on the right-click menu of the class, and apply the desired pattern.

# Using the UI Designer

---

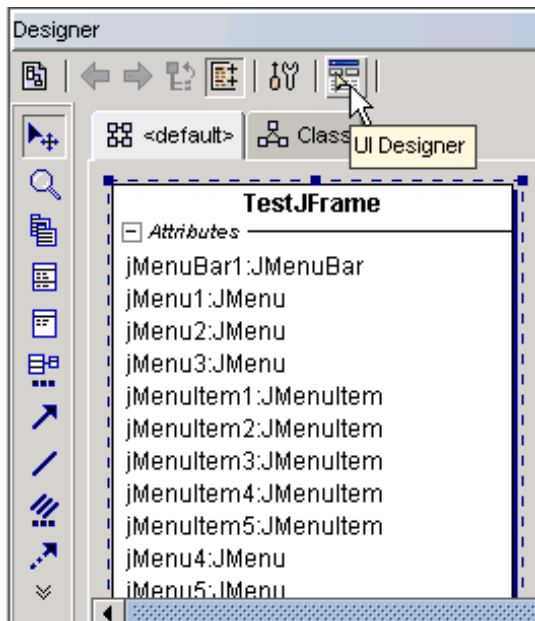
Working on a user interface visually requires the Designer pane, which may or may not be visible depending on which Workspace you currently use. When you are ready to use the UI Builder, switch to a Workspace that shows the Designer pane or create a Workspace in which it is visible. (You can find the Workspaces feature on the main view menu and main toolbar. For more information, see “Setting up workspaces” on page 72.)

When you create a visually editable class using the Object Gallery, it appears in the default class diagram of the package containing the source code. Open that diagram in the Designer pane (Diagram Designer view) and visually add attributes and operations to the class in the same way you would for any other class. You can also write code using the Editor.

The Designer pane has three possible views, as shown in Figure 78: Diagram Designer, UI Designer, and Menu Designer.

**Figure 78** Diagram designer view

---



## UI Designer view

---

When it comes to adding visual and nonvisual UI components within a container, you may decide you prefer to work on the class visually in the Designer pane in UI Designer view.

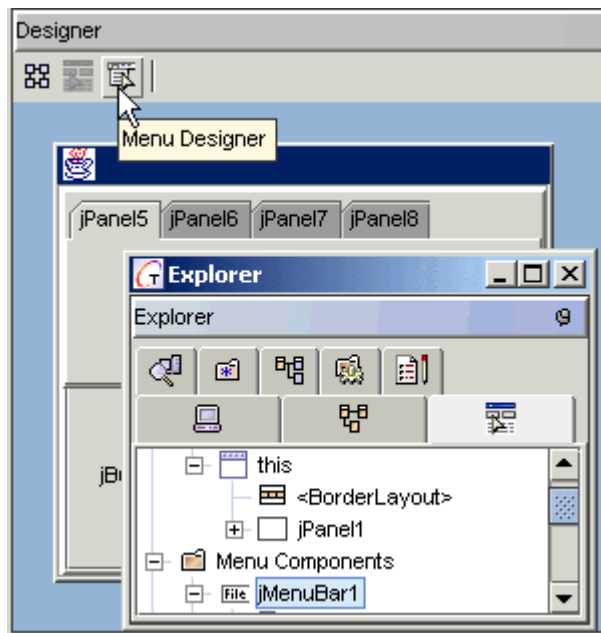
To work on a visually editable class in UI Designer view:

1. Open the class diagram containing the visually editable class if that diagram is not already open, *or* open the visually editable class in the Editor.
2. If working from the class diagram, select the editable class to load its source into the Editor.
3. If working with the class diagram, click the UI Designer button on the Designer pane toolbar. If using the Editor, choose **UI Designer** on the Editor right-click menu.

**NOTE:** Alternatively, choose **View | Designer View | UI Designer** on the main menu.

The UI Builder looks to the Editor (*not* the class diagram or Explorer selection) for the class to work on. It accepts the class for visual editing only if it is declared *public* and contains the required initializing operation (see “What is a visually editable class?” on page 288).

**Figure 79** UI Designer view



## Menu Designer view

---

When a container class you are working on contains a menu component, develop the menu visually in Menu Designer view.

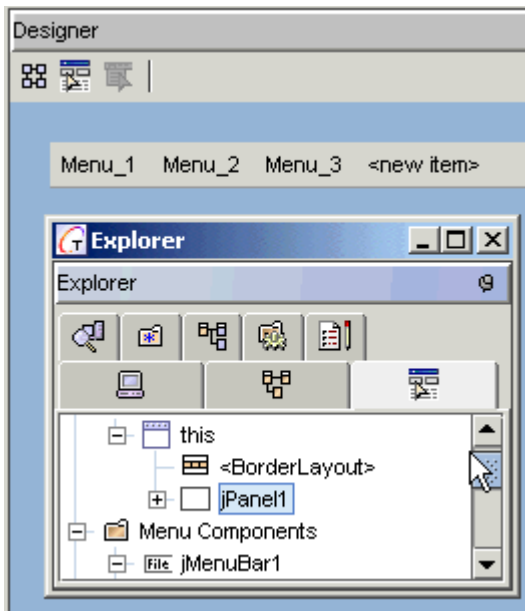
*To work on a menu component visually in Menu Designer view:*

1. Select the menu component in the UI Component Explorer (Explorer pane). The Menu Designer button is enabled on the Designer pane toolbar and the Menu Designer command is enabled on the main menu (**View | Designer View | Menu Designer**).
2. Click the Menu Designer button on the Designer pane toolbar, or choose **Activate Designer** from the right-click menu for the menu component in the UI Component Explorer.

**TIP:** Double-click a menu component in the UI Component Explorer to switch to Menu Designer view. To switch back to UI Designer view, double-click a visual UI component instead of a menu component.

**Figure 80** Menu Designer view

---



## Adding components from the Toolbox

---

The Toolbox is available in the UI Designer and Menu Designer views. It contains the available set of components for building a user interface. If the Toolbox is hidden, display it using **View | Main Panes | Toolbox**.

For a list of available UI components provided in the Toolbox, search Online Help for *Toolbox*.

The procedure of adding visual, non-visual and menu components are slightly different.

*To add visual components from the Toolbox:*

Provided that a visually editable class is already created, as described earlier in this chapter, you can add a component to it.

1. Open the page of the Toolbox that contains the desired components.
2. Click the component icon in this page of the Toolbox.
3. Move the pointer over the target container on the Designer pane. The pointer changes to the drop shape, and the container changes to show the current layout (the default layout is BorderLayout).
4. Click inside the container frame to place the component.

**Result:** the component is allocated in the container, and an entry is added to the UI Components node of the Components Explorer.

**TIP:** You can drop a visual component on the background of the UI Designer view. In this case, the component is not visible. You can later place it to a container using Cut-Paste in the Components Explorer (or just drag and drop it to the desired container).

Menu components are added in a different way.

*To add menu components from the Toolbox:*

Provided that a visually editable class is already created, as described earlier in this chapter, you can add a menu component to it.

1. Open the menu page of the Toolbox that contains the desired menu components.
2. Click the component icon in this page of the Toolbox.
3. Move the pointer over the background on the Designer pane. The pointer changes to the drop shape.
4. Click on the background to place the component.

**Result:** the menu component is created, and the Menu Designer icon becomes enabled on the UI Builder toolbar. Now you can design the menu, and, when ready, place it in the container. This procedure is described in details in the section “Setting a menu bar into the container frame” on page 311.

**NOTE:** You can customize the contents of the Toolbox. For more information, see “Customizing the Toolbox component palette” on page 324.

## Editing component properties

---

Every UI component, represented by a Bean, has a set of properties that control how a component behaves at runtime. Properties can be edited at design time and manipulated with code at runtime.

There are two ways to customize a UI component:

- By using *property editors*. Each Bean property has its own property editor, associated with it.
- By using *customizers*. Customizers are used where property editors are not practical or applicable. Unlike a property editor, which is associated with a property, a customizer is associated with a Bean.

The two ways of editing properties are described in the following sections: , “Editing properties using Customizers” on page 294, and , “Editing properties using Property Editors” on page 296.

## Exposure levels of properties

The Inspector displays several levels of properties for UI components:

- **General.** The Inspector displays properties not flagged as *Hidden* or *Expert* in the BeanInfo class of the selected component.
- **Expert.** The Inspector displays properties flagged as *Expert* in the BeanInfo class of the selected component, plus all the properties shown in the *General* level.
- **Hidden:** The Inspector displays only properties flagged *Hidden* in the BeanInfo class of the selected component.
- **Read-only:** The Inspector displays only properties flagged *Read-only* in the BeanInfo class of the selected component.

*To change the exposure level in the Inspector:*

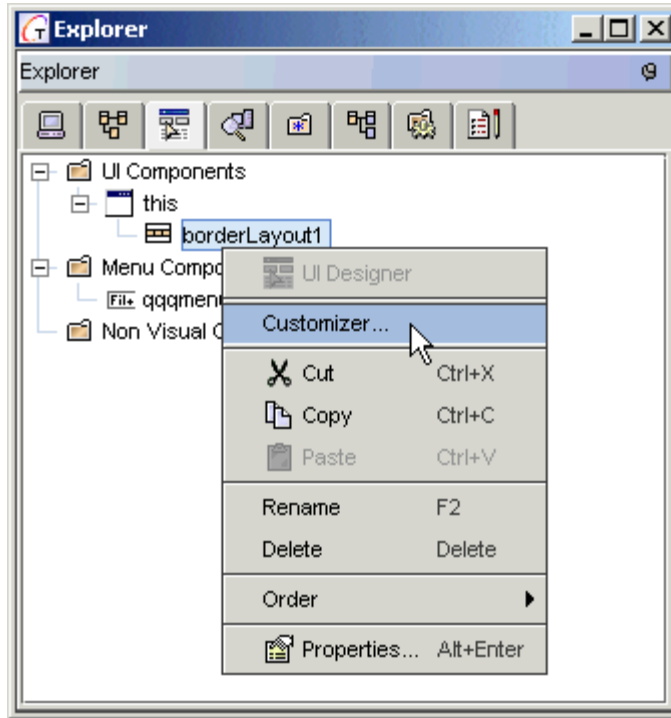
1. Right-click any property in the *Name* column of the Inspector.
2. Choose **Property Filter** on the right-click menu, followed by the desired exposure level on the submenu.

## Editing properties using Customizers

---

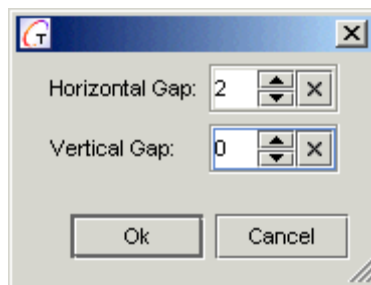
Sometimes properties are not sufficient to represent configurable attributes of UI components. If a UI component has a Customizer class, associated with it via its BeanInfo class, the Customizer command is enabled on the right-click menu of this component in the Components Explorer, as shown on the Figure 81, “Customizer command in the Components Explorer” on page 295.

**Figure 81** Customizer command in the Components Explorer



The appearance of the customizer dialogs depends on the specific implementation. A sample Customizer dialog is shown on the Figure 82, “Customizer dialog for a BorderLayout container” on page 295.

**Figure 82** Customizer dialog for a BorderLayout container



As you enter the required values, the Customizer directly repaints the relevant properties of the displayed object. The further behavior of the customizer depends on the button pressed to complete operation.

- **OK:** if you press this button, the changes that you have done to the object, are committed to the source code.
- **Cancel:** changes are not committed to the code, but properties are changed with the nearest update. Default values are not restored with Cancel button.

## Editing properties using Property Editors

---

Properties of UI components are displayed in the Inspector when a component is selected in UI Designer or Menu Designer view in the Designer pane. Use the Inspector to edit default property values during development.

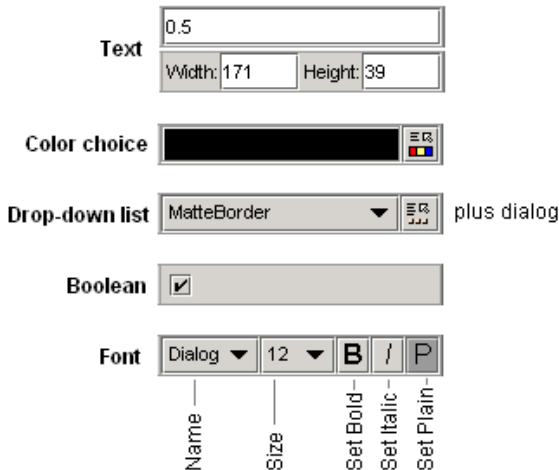
*To view the Inspector:*

1. Select a visual component in the Designer pane (UI Designer view), or a visual, nonvisual, or menu component in the UI Component Explorer.
2. Press `Alt+Enter` or choose **Properties** on the right-click menu.

There are several types of property editors ranging from simple text fields to drop-down lists, color choosers, and entire dialogs (Layout Manager, for example). Figure 83 illustrates the different types.

**Figure 83** UI Builder property editors

---



As you edit properties, relevant changes are written to the source code in real time.

Some property editors have buttons leading to dialogs, such as Bean Chooser or File Selection. Online Help is provided for these dialogs.

## Changing the layout manager for UI components

---

Java UI containers use a special *layout manager* object to determine how components are placed and sized at runtime. This means your layout will be consistent on various operating platforms.

When you add a component to a container (or Applet), the container uses the layout manager to determine where to put each component. Java defines the `java.awt.LayoutManager` interface that is implemented by the five main layout manager classes: *BorderLayout*, *CardLayout*, *FlowLayout*, *GridLayout*, and *GridBagLayout*. (There are several other layout managers, but these are the most often used.)

### The *layout* property

In Together, the layout manager used by a container component (frames and panels, for example) is shown visually as a property of the component. The *layout* property is shown in the Inspector whenever you select a container in the UI Designer or the UI Component Explorer.

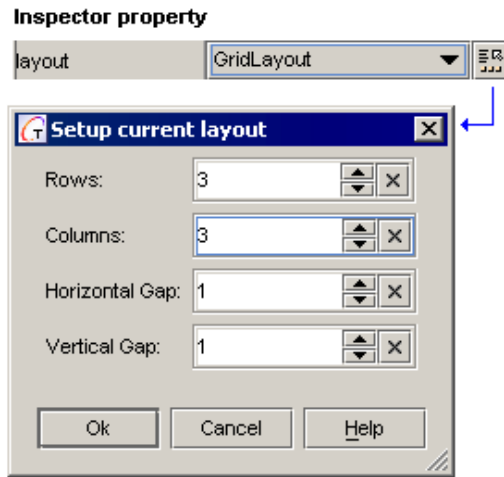
*To view and change the layout property:*

1. Select the component in the UI Components Explorer, or in the UI Designer view in the Designer pane.
2. If the Inspector is not visible, use the View menu or press `Ctrl+Alt+I` to show it.
3. Scroll the Inspector properties list, and locate the *layout* property.
4. Choose the desired layout manager from the drop-down list.

### Using layout setup

The *layout* property includes a special editor dialog you can use to visually set the parameters for each layout. For example, if using *GridLayout*, set the number of rows and columns. Also set the size of the horizontal and vertical gaps between them.

**Figure 84** The layout setup dialog



**Layout-specific setup dialog**

The visual setup shown in Figure 84 results in the following statement in the source code (when used for a JPanel component):

```
jPanel1.setLayout(new java.awt.GridLayout(3, 3, 1, 1));
```

## Supported layout managers

The following layout managers are supported as a property of UI containers and are displayed as choices in the *layout* property of the Inspector:

- **BorderLayout** (`java.awt.BorderLayout`). Organizes the container into 5 regions: *North*, *South*, *East*, *West*, and *Center*. It is not necessary to use all regions. Default layout manager for frames.
- **GridLayout** (`java.awt.GridLayout`). Divides a container into a specified number of rows and columns to form a grid of cells. Each cell is limited to a single component. Most useful when all components in the container are nearly equal in size. Good for laying out panels.
- **GridBagLayout** (`java.awt.GridLayout`). The most robust of the layout managers. It enables you to use components of different sizes and precisely lay them out in the container.
- **CardLayout** (`java.awt.CardLayout`). The concept is similar to breaking the container into a deck of cards, each card having its own layout manager. Flipping through the cards displays a different set of components. Conceptually analogous to HyperCard on the Macintosh, and Toolbook on Windows.
- **FlowLayout** (`java.awt.FlowLayout`). Arranges components from left to right until no space remains, then begins a new row. Mainly useful for button layouts. Default manager for panels and Applets.
- **OverlayLayout** (`javax.swing.OverlayLayout`). Allows components to be physically placed one on top of another.

- **BoxLayout** (`javax.swing.BoxLayout`). Aligns components from either left to right or top to bottom in a container. The class is not generally used directly, as the `Box` class provides methods for screen design.
- **Null**. A special design-time layout that disables any automatic control of the component layout, allowing for easier visual prototyping at design time. Not generally suitable for runtime.



You are not limited to these layout managers. Use your own custom layout managers or third-party layout managers, but you will need to use code to implement them rather than using a visual property in the Inspector.

## Containers with different layouts

Once you have set the *layout* property for a container, place components into it (for instructions, see “Adding components from the Toolbox” on page 293). To visually resize containers such as panels, select the component and drag the corners or sides.

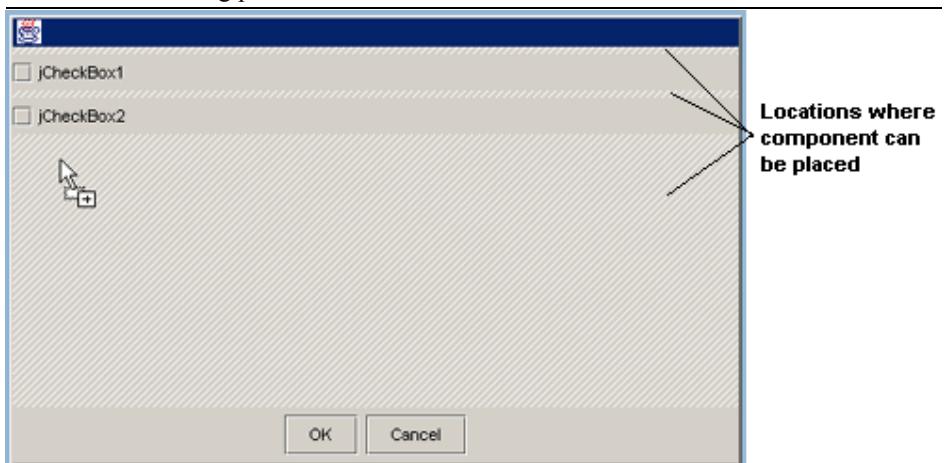
When you select a component in the Toolbox and then hover the pointer over the container, the visual image of the container areas that can accept the component change to a cross-hatch pattern.

**TIP:** The mouse pointer changes its shape depending on the container area:

- Where the container allows placement of a component, the pointer changes to a “drop” shape. 
- On areas that cannot accept a component, the pointer changes to a “prohibit” shape. 

The main point to remember when working with different layouts in a container is that Together only allows you to place a component in a location that the current layout manager supports. Those locations show the white cross-hatching pattern, as shown in Figure 85.

**Figure 85** Valid placement locations for any layout manager always display the cross-hatching pattern



Other manipulations with the components also depend on the properties of the container layout manager. When a component is selected by a mouse-click, it is marked with bullets at the component's borders. Where the layout manager enables resizing or relocation of the component, the bullets are black. If any visual manipulations with the components are prohibited, the bullets are grey. In case of multiple selection, the first component selected is marked with the white bullets and plays the role of an anchor, that serves as a reference point for the various adjustment operations.

The following sections (“NullLayout”, “BorderLayout”, “CardLayout”, and “GridBagLayout”) provide specific information on the appropriate layout managers.

## NullLayout

NullLayout is the most basic layout that enables rather free visual manipulations with components. You can refine the results of your exercises using the special adjustment commands of the right-click menu. Thus, it is possible to align the components against the selected anchor, spread them within a container, and resize them as required.

Components right-click menu contains the following command nodes:

- **Layout**, that allows to align the components (**Align**), allocate them evenly against an anchor (**Space Evenly**), and spread them within a container (**Distribute in Container**).
- **Match Size**, that allows to produce components with uniform dimensions.
- **Fill**, that allows to produce container-size components.

**IMPORTANT:** All adjustment operations are performed on a selection of components. The element that is selected first, becomes an anchor. This is denoted by the white square bullets, unlike the black bullets for the ordinary selection.

It is vital to right-click on the anchor, to preserve selection. If you right-click on another selected component, it will become an anchor, and the result will be different from what you expect to obtain.

*To align components against an anchor:*

1. Select the components to be aligned.
2. Right click on the selected anchor to open the right-click menu.
3. On the right-click menu, expand the **Align** command node, and choose one of the available options (Left, Right, Center, Top, Bottom, Middle).

*Result:* the selection is aligned against the selected anchor. For example, if you choose Align Left, all the components will be aligned against the left edge of the anchor.

*To place the components at the equal distances against the anchor:*

1. Select the components to be spaced, keeping in mind the anchor.
2. On the right-click menu of the anchor, choose **Layout | Space Evenly** command.

**TIP:** The nested commands are only enabled if three or more components are selected.

3. Choose the required direction (Horizontal or Vertical).

*To spread the components in the container:*

1. Select the components to be distributed, keeping in mind the anchor.
2. On the right-click menu of the anchor, choose **Layout | Distribute in Container** command.
3. Choose the required direction (Horizontal or Vertical).

*To resize components against the anchor:*

1. Select the components to be resized, keeping in mind the anchor.

**TIP:** At least two components should be selected.

2. On the right-click menu of the anchor, choose **Match Size** command.
3. Choose the required dimension (Horizontal, Vertical, or both).

*Result:* the components are resized so that the selected dimension matches that of the anchor.

*To resize components to the container size:*

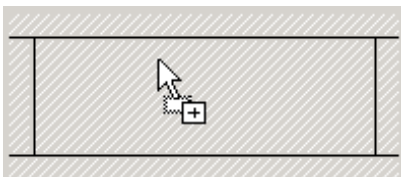
1. Select the components to be resized.
2. On the right-click menu of the anchor, choose **Fill** command.
3. Choose the required dimension (Horizontal, Vertical, or both).

**TIP:** `NullLayout` enables visual resizing of components. As the mouse pointer hovers over one of the black selection bullets, it changes its form to a double arrow. Click on the bullet and drag the component border in the required direction.

## BorderLayout

`BorderLayout` allows only five components to be added to a container in the North, South, East, West, and Center regions. The mouse pointer changes to a “drop” shape whenever you hover over an area of the container that can accept the component you are placing. As soon as a component is added to a region, the number of allowed regions is reduced. Figure 86 shows the `BorderLayout` container with its regions allowed for allocating components.

**Figure 86** Empty `BorderLayout` container: North, South, East, West, Center regions



### Example

Figure 87 shows an example of a `JFrame` using `BorderLayout`. You can clearly see the North, South, East, West, and Center regions of the container.

**Figure 87** JFrame using BorderLayout shows where components can be placed

---

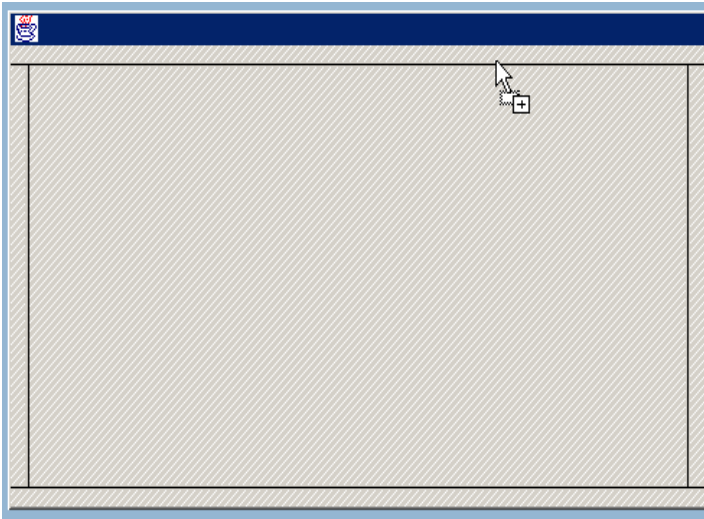
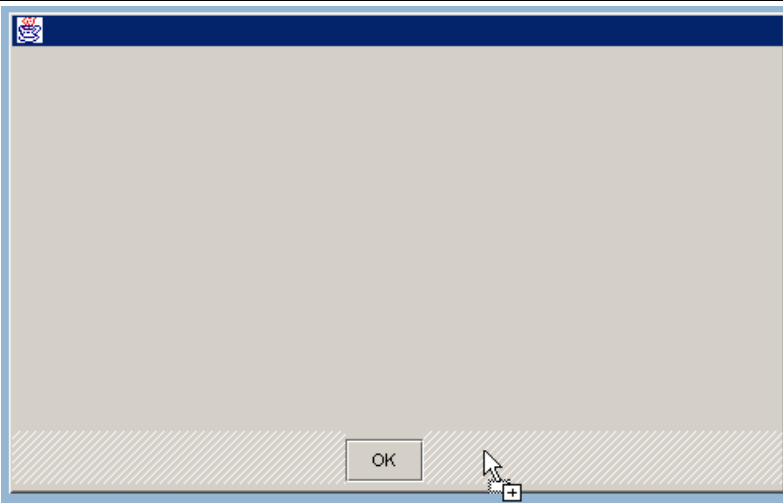


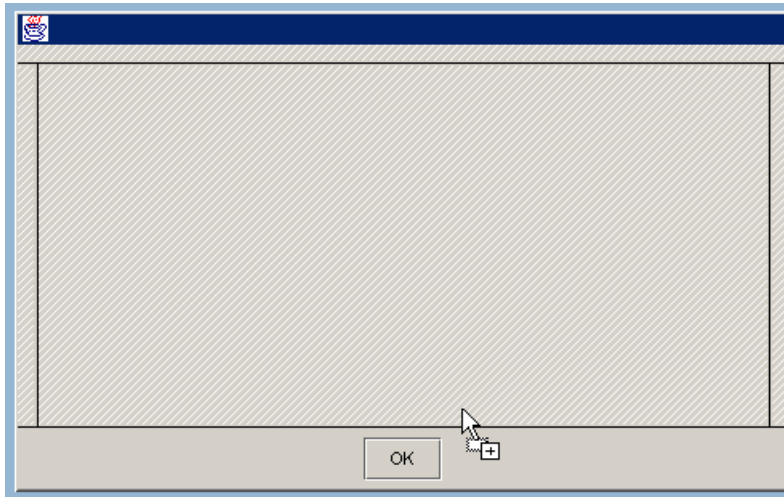
Figure 88 shows the same JFrame as Figure 87, with a button being placed into the JPanel in the South region of the frame. As you can see, only the pane displays the cross-hatching. If the pointer were moved over the JFrame, cross-hatching would show for the other regions of the frame, but the JPanel would not, as in Figure 89.

**Figure 88** Placing a second JButton into a JPanel using FlowLayout

---



**Figure 89** Pointer hovering outside the JPanel



## CardLayout

When adding a new component to a CardLayout container, keep in mind that you need to place the mouse pointer between the tabs of the already existing components. Figure 90 below demonstrates adding a new component between the existing cards.

**Figure 90** Placing components in a CardLayout container



You can mix components of different types (Swing and AWT) in a single container. When mixing components in a CardLayout container, the AWT components are always displayed on top. You can navigate through the components using the treeview in the UI Builder (Explorer).

## GridBagLayout

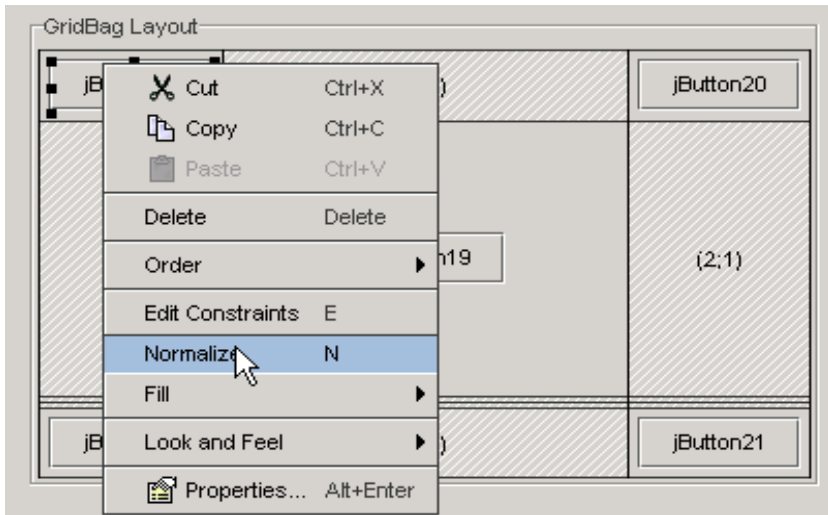
When arranging components in a GridBagLayout container, you may place them in any allowed location, which automatically creates a new cell for each component. However, if a component is deleted, its cell remains in place, and sooner or later you might end up with a number of empty cells. To remove unnecessary empty cells, use the normalization feature. Normalization simplifies constraints of all the components within a container, without actually changing its layout.

*To normalize a GridBagLayout container:*

1. Right-click a component in a GridBagLayout container.
2. Choose **Normalize** on the right-click menu.

This removes all the unnecessary empty rows and columns, and the spaces between occupied rows and columns, as shown in Figure 91.

**Figure 91** Normalizing the GridBagLayout container



When placing components into a GridBagLayout container, you have to customize their constraints.

This is done using the Edit Constraints dialog, which is opened from the right-click menu of a component, as shown on Figure 91 above. Figure 92 displays the Edit Constraints dialog window. The controls of this dialog correspond to the variables of the GridBagLayout class.

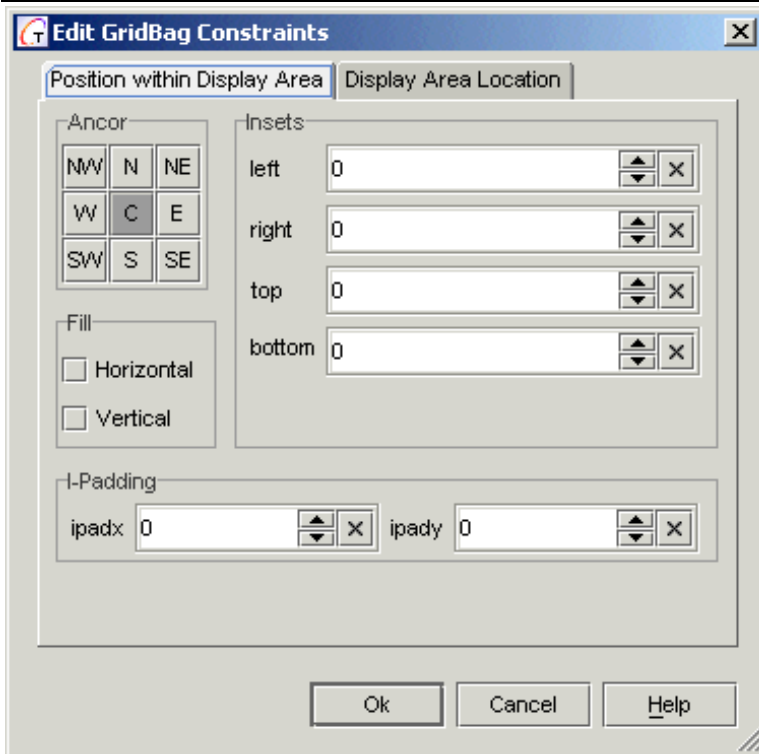
Use the spinners to increase or reduce the values, and the **x** character to reset the values to the default.

The Position section within the **Display Area** tab allows you to specify the anchor, insets, fill and padding of a component in its display area.

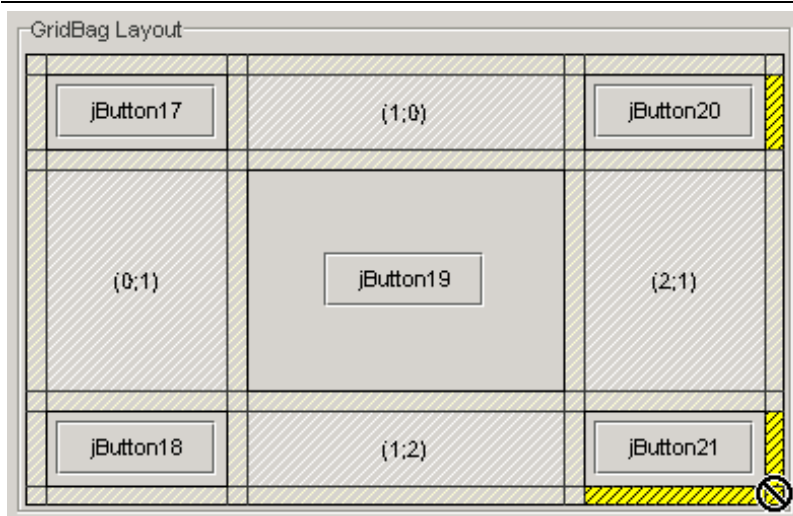
The **Display Area Location** tab enables you to customize the properties of the display area within a container. Using this tab, you can customize the resizing behavior of the component using the *weight* fields, and the gaps between components and the container border. These gaps are regulated by the *width* and *height* fields. If the value *REMAINDER* is selected in these fields, the appropriate areas around the component become prohibited for the allocation of new components. Such areas are marked with bright yellow hatch marks, as shown in Figure 93.

For more details, refer to the relevant topic of the online help system, and to <http://java.sun.com/j2se/1.4/docs/api/java/awt/GridBagLayout.html>.

**Figure 92** Edit Constraints dialog



**Figure 93** Remainder areas in a GridBagLayout container



## Where to learn more about layout managers

There is a great deal of information published on the Web about basic as well as advanced techniques for using layout managers in Java. One useful tutorial is available (at the time of publication) on the Sun website at:

<http://java.sun.com/docs/books/tutorial/uising/layout/using.html>

## Other tips and tricks for working on UI components

---

This section provides some information that you may find useful as you learn your way around the UI Builder.

### Selecting multiple components

To select multiple components in the UI Designer, click to select the first component, press and hold down **Ctrl**, and then click the other components you want to select.

In the UI Component Explorer, use `Shift + click` to select multiple contiguous components, or `Ctrl + click` to select multiple non-contiguous components.

### Changing the look and feel

To change the look and feel of frames, choose the **Look and Feel** command on the right-click menu of the container in the UI Designer. The look and feel options are:

- Metal (the default)
- CDE/Motif
- Windows

**IMPORTANT:** You can change the look and feel only to the one supported on the given platform. For example, on the Unix platform, the Windows L&F is not supported, and the appropriate menu command is not available.

## Creating event handlers for UI components

---

Each component has a set of event handlers. Code defines what behavior the component should have, when and if an event occurs.

*To create an event handler:*

1. Select the component in the UI Designer view or the UI Component Explorer.
2. Open the *Events* page of the Inspector.
3. Click in the edit field next to the event for which you want to write event handler code. A default identifier for the event handler block appears in the field.
4. Press **Enter** to create the event handler block in the source code. The Editor scrolls to the newly inserted event handler line.

## Changing alignment, spacing, and distribution

You can change the alignment, spacing, and distribution of components in a frame container with Null layout, using the **Layout** command on the right-click menu of the frame.

**TIP:** To open the right-click menu for a frame when it contains other components such as panels, right-click on the title bar of the frame in the UI Designer.

The Layout menu provides the following commands:

- **Align.** Opens a submenu enabling you to specify horizontal alignment values of Left, Center, and Right, and vertical alignment values of Top, Middle, and Bottom.
- **Space Evenly.** Spaces components evenly either horizontally or vertically.
- **Distribute in Container.** Evenly distributes components over the entire container either horizontally or vertically.

## Changing the order of components

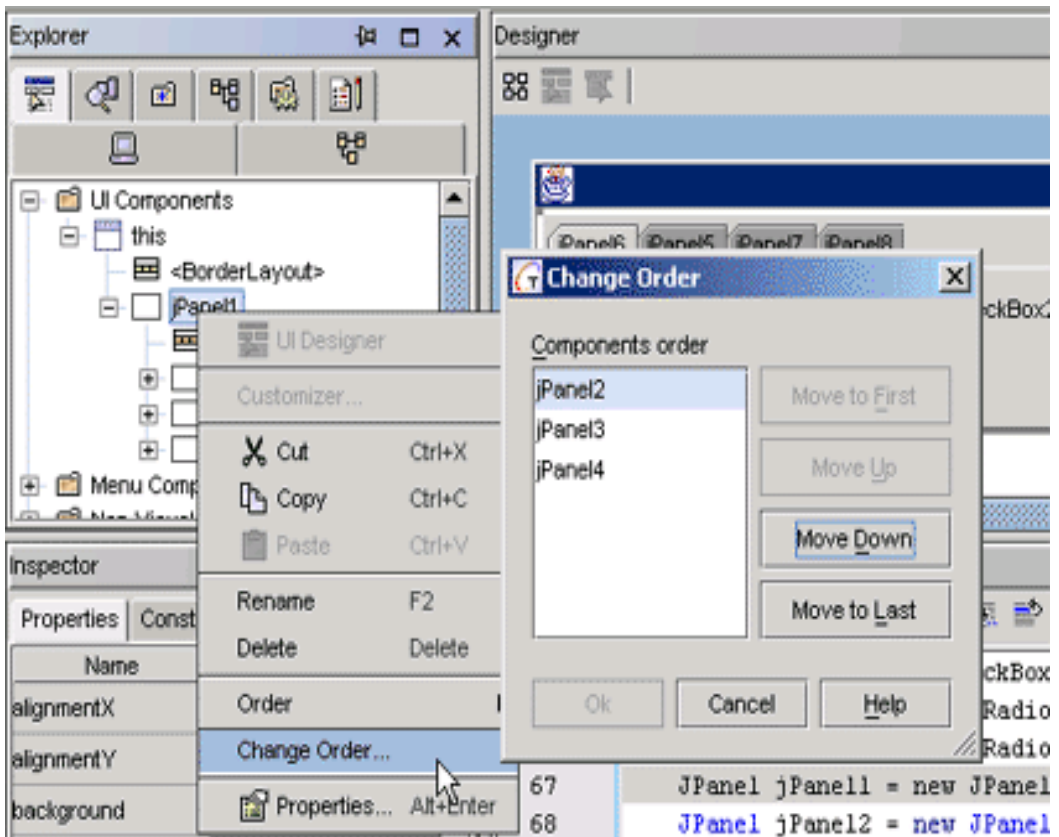
To change the appearance order of components or containers, choose the **Order** command on the right-click menu of a selected component or container. You can make the selected component first or last in its container, or move it backward and forward relative to the adjacent components.

To rearrange all the components within the selected container in one session, use the **Change Order** command on the right-click menu of a container in the **UI Builder** tab of the Explorer (see Figure 94).

*To change the order of components in a container:*

1. Select the container whose components should be rearranged.
2. Right-click on this component in the **UI Builder** tab of the Explorer, and choose the **Change Order** command.
3. In the Change Order dialog, choose one of the displayed components.
4. Using the direction buttons, move the component to the desired location.
5. Select the other components and repeat step 4 as required until you end up with the proper arrangement of components.
6. Click **OK** to confirm the changes and close the dialog.

**Figure 94** Reordering components in a container using the Change Order dialog



# Designing menus

---

Use the UI Builder feature to create both Swing menu components (*javax.swing.JMenuBar* and *javax.swing.JPopupMenu*) and AWT menu components (*java.awt.MenuBar*, *java.awt.PopupMenu*) in applications. When the UI Builder feature is activated, the Designer pane has a special Menu Designer view for visually developing menu components.

The “Designing menus” section covers the following topics:

- Creating a new main menu (see “Creating a new main menu” on page 309).
- Creating a new popup menu (see “Creating a new popup menu component” on page 314).
- Defining menus visually (see “Defining menus visually” on page 315), which includes:
  - Defining menu items including checkbox and radio button items (see “Defining new menu items” on page 316).
  - Creating nested menus (see “Creating nested menus” on page 319).
  - Enabling, disabling, hiding, and showing menu items (see “Disabling and hiding a menu using the right-click menu” on page 320).
  - Defining menu item properties such as icons, mnemonics, and keyboard shortcuts (see “Adding icons to menus” on page 321), and more.
- Alternative way of defining menu commands (see “Defining menu items using the UI Component Explorer” on page 322)

**IMPORTANT:** The whole procedure of designing menus involves the following major independent steps:

- creating a container
- creating a menu
- setting menu component in the container frame.

We’ll consider this procedure step by step for a main menu.

## Creating a new main menu

---

*To design a main menu:*

1. Using the **Object Gallery**, create a container.
2. Create the menu bar component (see “Creating a new menu bar component” on page 310).
3. Add commands to the menu (see “Adding commands to the menu” on page 310).
4. Add the created menu to the container (see “Setting a menu bar into the container frame” on page 311), and set the required properties.

Find the detailed descriptions of specific steps in the following sections.

## Creating a new menu bar component

Create main menu (menu bar) components in much the same way you create other components. There are minor but important differences which are covered in detail in this section.

*To create a new menu bar component:*

1. On the Diagram toolbar, click the **UI Designer** button to switch to the UI Designer view (see “UI Designer view” on page 290).
2. In the UI Designer view in the Designer pane, open the container frame that will contain the menu.
3. In the Toolbox, open the *AWT Menu* or *Swing Menu* page, depending on the type of components you are using.
4. Click **MenuBar** (AWT) or **JMenuBar** (Swing) button on the Toolbox.
5. Move the pointer over the Designer pane and click anywhere on the background of the Designer pane. (You cannot click inside the frame component.)
6. In the UI Component Explorer, expand the *Menu Components* node and choose the new menu component (*MenuBar1* or *JMenuBar1*).
7. Optionally change the name of the menu component in the Explorer (for more information, see “Renaming menu component identifiers visually” on page 322).

## Adding commands to the menu

Now you can populate the menu component with commands. This is how it’s done.

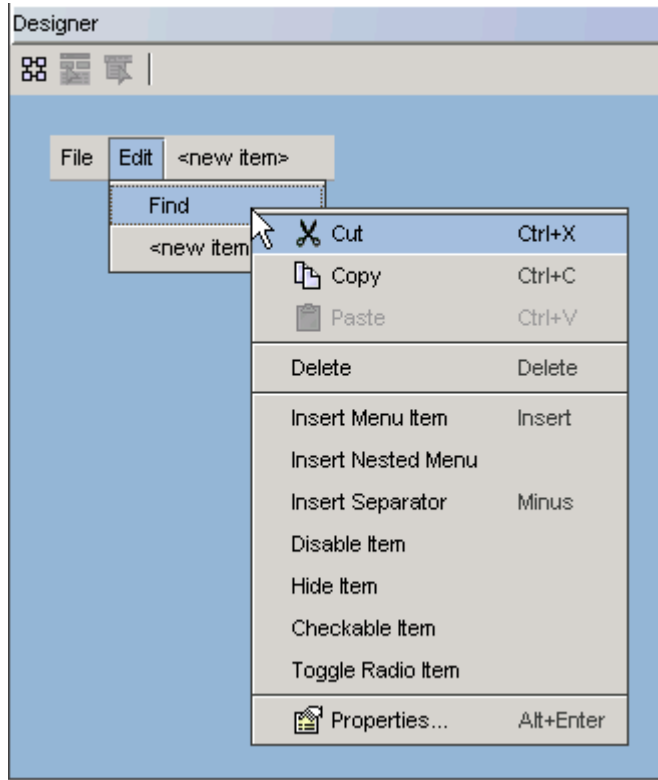
*To add commands to the menu:*

1. On the Diagram toolbar, click the **Menu Designer** button to switch to the Menu Designer view. You will see the placeholder for the new menu item.
2. On the right-click menu of the new item, choose **Insert Menu** command. This will create a new command on the menu, add a `<new item>` entry on the menu bar, which allows to further extend the menu bar, and another `<new item>` entry under the command, to further extend the command sub-menu. (see Figure 88, “Adding commands to the menu bar” on page 311)
3. Repeat the step 2 as required.

**NOTE:** Alternatively, you can just double click on a `<new item>`, and type in the command name This action will both visually rename the command, and automatically add a new entry to the menu.

Using the right-click menu of the menu component, you can create nested menus, insert or remove separators between the groups of commands, disable / enable or show / hide the selected items. It is also possible to create checkable items and radio buttons. These features are described further in the sections under “Defining menus visually” on page 315.

**Figure 95** Adding commands to the menu bar



Finally, the menu is ready. Switch to the UI Designer view - and you see no visible results! The container is still empty... But don't be frustrated - with the next step you will put the menu to its container.

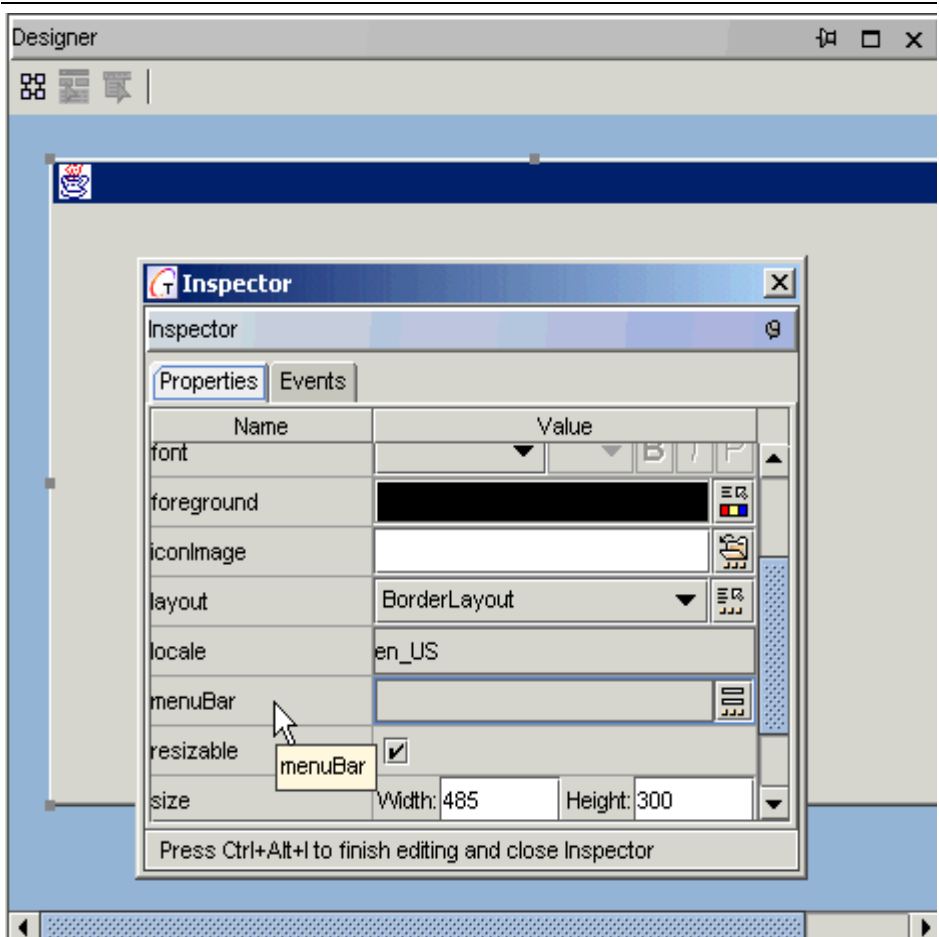
## Setting a menu bar into the container frame

A frame component does not display a menu bar until an appropriate *set* statement is placed into the source code (see “Understanding the code” on page 313). There is no way to visually place the menu to a container. To do so, you have to use the Inspector, which enables you to set a menu bar component into a container frame.

*To set a menu to a container:*

1. Switch back to the UI Designer view
2. On the right-click menu of the container, choose **Properties** command, to open the Inspector.
3. In the Properties tab of the inspector, choose the *menuBar* field. The *menuBar* property specifies which menu bar component should be set in the frame.

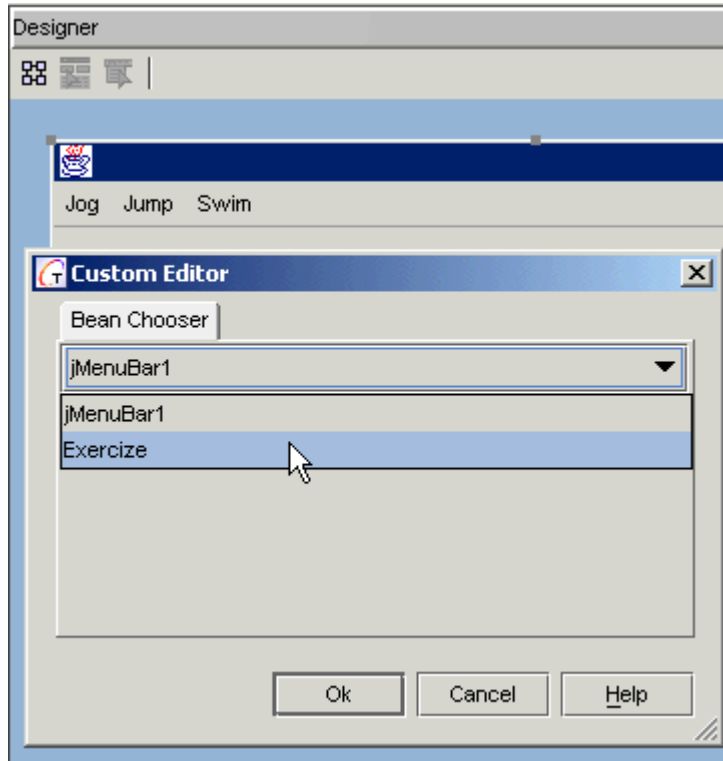
**Figure 96** *menuBar* property of the container



4. In the *menuBar* field, click on the Editor button, to open the editor for this property.
5. In the Custom Editor dialog, click on the drop down list, and choose the menu that will be actually added to the container.

**NOTE:** The drop-down list of the Custom Editor dialog contains all menu bar components that have been declared in the current source code file. When you select a menu bar component, an appropriate `set` statement generates or updates in the code and the selected menu bar appears in the visual representation of the frame in the UI Designer view, as shown on the Figure 97, “Using the property editor to set a menu to a container frame,” on page 313. See also the section “Understanding the code” on page 313 to learn how the visual operations are reflected in the source code.

**Figure 97** Using the property editor to set a menu to a container frame



**TIP:** View an example of how a menu bar is set in the main JFrame class created by the *Application* pattern in the Object Gallery (**File | New: User Interface - Application**).

## Understanding the code

When you first place a menu component visually, its declaration is written in source. For example, if you place a JMenuBar component, the following line is written:

```
private JMenuBar JMenuBar1 = new JMenuBar();
```

When you add menu items to the menu bar, their declarations are also written to the source file. You might end up with the following code for a small menu:

```
private JMenuBar JMenuBar1 = new JMenuBar();
private JMenu JMenu1 = new JMenu();
private JMenu JMenu2 = new JMenu();
```

The code will also contain some *add* and *set* statements such as:

```
JMenuBar1.Add(JMenu1);
JMenuBar1.Add(JMenu2);

JMenu1.SetText("File");
JMenu1.SetMnemonic("F");
```

```
JMenu2.SetText("Edit");
JMenu2.SetMnemonic("E");
```

At this point the *JMenuBar1* component appears as an attribute of the container class when the Designer pane is in Diagram Designer view. But the menu does *not* appear on the WYSIWYG image of the frame in UI Designer view. In order for the menu to appear there, add an appropriate *setJMenuBar* statement to the code. In the foregoing example, the following statement is required after the *add* statements:

```
JMenuBar1.Add(JMenu1);
JMenuBar1.Add(JMenu2);
setJMenuBar(JMenuBar1);

JMenu1.SetText("File");
JMenu1.SetMnemonic("F");
JMenu2.SetText("Edit");
JMenu2.SetMnemonic("E");
```

Write the code yourself or use the *menuBar* property of the frame component, which generates the necessary code.

## Creating a new popup menu component

---

The steps for creating a new popup menu are the same as for creating a new main menu, except that you choose a popup menu component in the Toolbox instead of a menu bar component.

### Displaying a popup menu at runtime

The code for calling a popup menu from a component is somewhat more complex than that required to set a menubar component. Thus, there is no Inspector property for components that “attach” a popup menu.

#### Example:

The following code fragment demonstrates one way to invoke popup menus at runtime.

```
public void commentFieldMouseReleased(MouseEvent e) {
    if(e.isPopupTrigger()){
        jPopupMenu1.show(commentField,e.getX(),e.getY()) ;
    }
}
```

(where *commentField* is the name of the component that the popup menu was called on, and *jPopupMenu1* is the name of a popup menu component for *commentField*).

Also add a mouse listener using the Inspector (for instructions, see “Creating event handlers for UI components” on page 306). For example:

```
commentField.addMouseListener(
    new MouseAdapter() {
        public void mouseReleased(MouseEvent e) {
commentFieldMouseReleased(e); }
    });
```

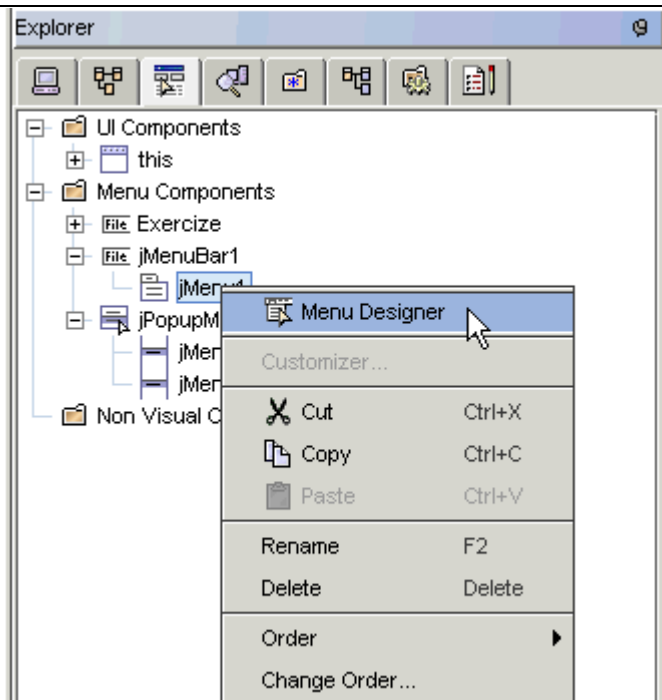
**NOTE:** The example above refers to the PC only, because on the different operating systems the popup menus are invoked in different ways. For example, in MacOS popup menus are invoked on CTRL+mouse press. Thus, the better way to display a popup menu is to catch both `mouseReleased()` and `mousePressed()` events and test them by means of `isPopupTrigger()` method. The subtle issue here is that both events should invoke the same method:

```
jButton.addMouseListener(  
    new MouseAdapter() {  
        public void mousePressed(MouseEvent e) {  
jButtonshowPopup(e); }  
        public void mouseReleased(MouseEvent e) {  
jButtonshowPopup(e); }  
    });  
}
```

## Defining menus visually

When opening a UI container class in the Designer pane, a visual representation of the container appears in the Designer, and the contained components are listed in the UI Component Explorer, as shown in Figure 98. The *Menu Components* node of the Explorer lists all the menu and menu item components.

**Figure 98** The UI Component Explorer and its right-click menu



To activate the Menu Designer view:

1. In the Explorer, choose any menu, popup menu, or menu item component
2. Choose **Menu Designer** command on the right-click menu, or click the Menu Designer icon on the Designer pane toolbar.

## Defining new menu items

When you create a new menu component, the first item is automatically created and labeled *<new item>*. To define the new menu item:

1. Double-click *<new item>* to activate in-place editing of the item name.
2. Type the text of the menu item, optionally using the syntax described in the next section.
3. Press **Enter** to accept the entry.

For second-level main menu items and popup menu items, a *<new item>* item is added to the end of the menu. Repeat the above steps to define new menu items.

For top-level items in a main menu component, *<new item>* items are created to the right of each item.

To create checkbox or radio-button menu items, see the respective sections in this chapter: “Defining a checkbox menu item” on page 317 and “Defining a radio-button menu item” on page 318.

## Menu definition syntax and syntax helpers

You can define a number of properties of new menu items in the in-place edit field for the items, using a special syntax. This is very productive when creating a menu structure from a specification. The properties you can define are:

- Menu text
- Mnemonic (accelerator)
- Keyboard shortcut
- Enabled
- Visible
- Checked (for checkbox and radio button items only)

The syntax looks like this:

```
<menu &text>;<shortcut>;<enabled>;<visible>[ ;<checked>]
```

### Examples:

```
&New file;Ctrl+F;0;1
```

In the above example, the menu text is *New file*, the mnemonic accelerator is *N*, the shortcut is *Ctrl+N*, the item is *disabled* (0, or Boolean *false*), and the item is *visible* (1, or Boolean *true*).

```
&Toolbox;Ctrl+Alt+T;1;1;0
```

In the above, the menu text is *Toolbox*, the mnemonic accelerator is *T*, the shortcut is *Ctrl+Alt+T*, the item is enabled (1), visible (1), and unchecked (0).

**NOTE:** Menu items defined as *not visible* are still displayed in the Menu Designer view, but their background color is white instead of gray.

## Using the menu definition syntax helpers

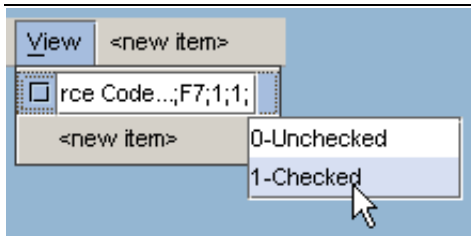
You do not have to memorize the in-place menu definition syntax. Together provides a set of syntax helpers that guide you through your in-place editing options.

*To use the syntax helpers:*

1. Type in the menu text followed by a semi-colon. A context-sensitive pop-up helper menu appears after a short pause.
2. Choose the desired option from the menu, navigating with the mouse or arrow keys.
3. To set further properties, type another semi-colon and select the desired option from the next helper.
4. Repeat step 3 until no further pop-up appears.

**TIP:** Press **Enter** in the helpers to select the default.

**Figure 99** Design syntax helper for checkbox item



## Defining a checkbox menu item

Optionally define menu items below the menu bar level and popup menu items as checkbox items in both AWT and Swing. Checkbox items provide on/off true/false toggles.

*To define a checkbox menu item:*

1. In Menu Designer view, select the *<new item>* you want to define as a checkbox menu item.
2. Right-click and choose **Checkable Item**.
3. Double-click to activate in-place editing and proceed as you would for any menu item.

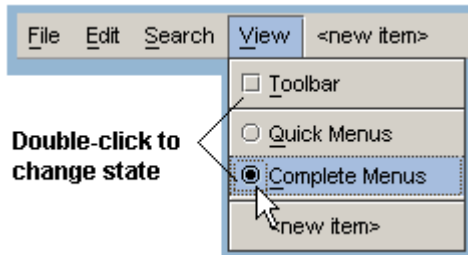
**TIP:** To revert to a regular menu item, right-click on the item in the designer and choose **Non-checkable item**.

## Changing the default state of a checkbox

To set the checked state of a checkbox type menu item, use the in-place editing field. Double-click to activate editing, then enter the desired Boolean value (0 or 1) in the syntactical position for the *checked* state. For more information, see “Menu definition syntax and syntax helpers” on page 316.

Once a checkbox menu item has been created, change the checked state of the checkbox by double-clicking on the checkbox itself, as shown in Figure 100.

**Figure 100** Changing menu item state at design time



## Defining a radio-button menu item

The menu items of a menu bar, and popup menu items can optionally be defined as radio-button items (Swing only). Radio-button items provide mutually exclusive choices.

*To define a radio button menu item:*

1. In the Menu Designer view, select the *<new item>* that you want to define as a radio-button menu item.
2. Right-click and choose **Toggle Radio Item**.
3. Double-click to activate in-place editing and proceed as you would for any menu item.

**TIP:** To revert to a regular menu item, right-click and choose **Non-checkable item**.

## Changing the default state of a radio-button

Use the in-place editing field to set the selection state of a radio-button type menu item. Double-click to activate editing, then enter the desired Boolean value (0 or 1) in the syntactical position for the *checked* state. For more information, see “Menu definition syntax and syntax helpers” on page 316.

Once a radio-button menu item has been created, you can quickly change the selection state of the radio button by double-clicking on the button itself, as shown in Figure 100.

## Creating nested menus

You can create any number of nested menu levels (sometimes called “submenus”) beneath any menu item or popup menu item. (Note that good user interface design practice generally limits the number of levels to about three.)

*To create a nested menu:*

1. Select the existing menu item you want as the parent of the nested menu.
2. Right-click on the parent menu item and choose **Insert Nested Menu**. This creates the new level and inserts a *<new item>* menu item.
3. Define the menu items for the nested menu in the same manner as for any other menu item.

## Inserting separators

You can insert separators between menu items to create groups of related commands. There are three ways to do this:

- Using in-place editing.
- Using the menu item right-click menu.
- Using the Toolbox.

### Creating separators using in-place editing

This is the quickest way to create a separator when creating a new menu.

1. Create or select a *<new item>* item at the place where the separator should be.
2. Activate in-place editing and type a dash (-). Press **Enter**.

### Creating separators using the right-click menu of the menu item

1. Select the menu item you want to appear immediately *below* the new separator.
2. Right click and choose **Insert Separator**.

A separator is inserted *above* the selected menu item.

### Creating separators using the Toolbox

Select a separator component in the AWT Menu or Swing Menu page of the toolbox and drop it into the menu at the desired location. See “Defining menu items using the UI Component Explorer” on page 322 for more information about using the Toolbox.

## Moving menu items

Once you have a menu structure developed, move menu items to different locations using drag-and-drop in the Menu Designer view (Designer pane). As you drag an item, a red bar appears at valid drop target locations.

## Defining menu item properties

---

Every menu item has a set of properties that you can modify at design time. As we saw in earlier sections, some properties can be set as you visually create new items, and

some can be modified using the menu item right-click menu. The full set of properties is listed in the Inspector when a menu item is selected in Menu Designer view.

*To view the Inspector properties of a menu item:*

1. Select the item in the Menu Designer view (Designer pane).
2. Press **Alt+Enter** or click the Properties button on the main toolbar to open the Inspector.

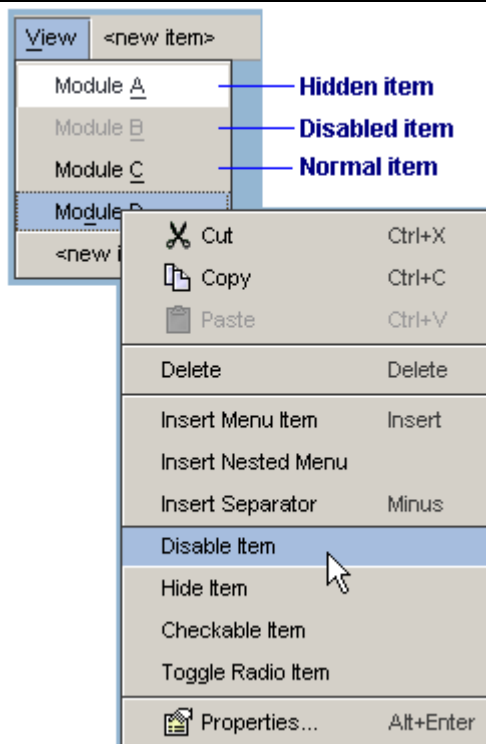
**TIP:** You might find it convenient to undock and resize the Inspector.

## Disabling and hiding a menu using the right-click menu

As mentioned in “Menu definition syntax and syntax helpers” on page 316, you can use a special syntax in the in-place editing field to enable/disable or show/hide menu items at design time.

You can modify the settings in the same way, but you might find it more convenient to change the default state of menu items at design time using the right-click menu of any menu item.

**Figure 101** Setting design time properties for menu items using the right-click menu



*To make a menu item hidden by default:*

1. Select the item in the WYSIWYG menu hierarchy and right-click.

## 2. Choose **Hide Item** on the right-click menu.

Menu items defined as hidden are still visible at design time. The background color of these items turns to white, as shown in Figure 101.

*To make a menu item disabled by default:*

1. Select the item in the WYSIWYG menu hierarchy and right-click.
2. Choose **Disable Item** on the right-click menu.

The foreground color of items defined as disabled turns to dark gray, as shown in Figure 101 above.

## Adding icons to menus

Menu items can display icons next to them. You can show static and rollover images using existing image files in BMP, GIF, or JPEG format. Images are defined as properties of menu items in the Inspector.

**TIP:** One fairly widespread convention for the use of icons in menus is that menu items having a parallel function on a toolbar should display the same icon as the toolbar button. Other menu items do not show icons.

### Icon properties

Optionally define in the Inspector the following icon properties:

- ***disabledIcon***. The image to display when the item is disabled but not selected.
- ***disabledSelectedIcon***. The image to display when the item is disabled and selected.
- ***icon***. The image to display when the item is enabled but not selected.
- ***rolloverIcon***. The image to display when the mouse pointer hovers over the item when the item is not in the selected state. Note that the *rolloverEnabled* property must be checked (*true*) in order for this to work at runtime.
- ***rolloverSelectedIcon***. The image to display when the mouse pointer hovers over the item and the item is in the selected state. Note that the *rolloverEnabled* and *selected* properties must be checked (*true*) in order for this to work at runtime.
- ***selectedIcon***. The image to display when the item is enabled and selected. Note that the *selected* property must be checked (*true*) in order for this to work at runtime.

*To display one or more images for the menu item:*

1. Select the menu item in the Menu Designer view (Designer pane).
2. Display the properties of the item in the Inspector (to open the Inspector, press `Alt+Enter` or click the Properties button on the main toolbar).
3. In the Inspector, select one of the iconic properties described earlier in this section. Use the browse button in this field to select an image file.
4. Repeat this process for any or all of the other iconic properties as required to get the effect you desire.

**TIP:** According to Sun user interface guidelines, the standard size for menu icons is 16x16 pixels.

## Defining menu colors and font

It is possible to modify the default foreground and background colors and the default font properties of menu items.

- Use the *foreground* property in the Inspector to change default foreground (text) color.
- Use the *background* property in the Inspector to change default background color.
- Use the *font* property in the Inspector to change default font properties of the selected menu item.

## Renaming menu component identifiers visually

As you create new menu items either visually or non-visually, new menu components are declared in source code with default identifiers. If desired, opt to change the default component names (identifiers) to names that are more meaningful and easier to maintain in the future.

*To rename a component identifier:*

1. Select the menu component in the UI Component Explorer.
2. Press **F2**, or right-click and choose **Rename**. In-place renaming is activated.
3. In the menu component node, type in a legal Java identifier for the component and press **Enter**.

**NOTE:** Do not confuse this process with in-place editing of the menu text and properties as described in, “Menu definition syntax and syntax helpers” on page 316.

Renaming components this way automatically changes every occurrence of the component name in the source code file.

## Renaming component identifiers in source code

You can optionally rename menu components manually in the source code. However, you will have to change *every instance* of the name in the file. To accomplish this task, use the **Search | Replace** command on the main menu when the Editor pane has focus.

## Defining menu items using the UI Component Explorer

---

There is an alternate technique to creating new menu items once you have defined a menu bar or popup menu component. Use the Toolbox to select different types of menu items and place them into the menu hierarchy displayed in the UI Component Explorer. This technique may be easier if you have to add a new item to a large existing main menu system.

*To add a menu item in the Explorer using the Toolbox:*

1. Expand the *Menu Components* node of the **UI Builder** tab of the Explorer (the UI Component Explorer), and locate the menu item that you want as the parent of the new item.
2. Select the Toolbox page that contains the menu component you want to add (such as Swing Menu).

3. In the Toolbox, click the menu component you want to add to the hierarchy.
4. Move the mouse over the parent menu component in the Explorer. (The pointer changes to the “drop” shape when you hover over a valid target.)
5. Click the node of the parent item to create the new menu item with a default identifier.

## UI Builder audits and metrics

---

When the UI Builder feature is activated, special audits and metrics are provided to help you check the quality of your user interface classes. In general, these work like all other audits and metrics. See Chapter 25, “Audits and Metrics” for information on how to use audits and metrics.

### UI audits

---

When the UI Builder feature is activated, a special *UI Builder Specific* category is added to the Java Audits dialog. It contains a special set of audits for UI classes that are used to check for problems including conflicting mnemonics in the same menu, use of nonstandard mnemonics in menus, duplicate context menu items, and so on. Detailed descriptions of each individual audit are provided in the Java Audits dialog.

*To use UI Builder audits:*

1. Make sure your project includes Java GUI classes as defined in this chapter under “Creating UI classes” on page 288.
2. Make sure the UI Builder feature is activated as described in this chapter under “Activating the UI Builder” on page 285.
3. On the main menu, choose **Tools | Audits** to open the Java Audits dialog.
4. Locate the *UI Builder Specific* category in the list of audit titles, and expand the category node to see all available audits.
5. Check the audits you want to run against your project and click **Start**.

### UI metrics

---

When the UI Builder feature is activated, a special *UI* category is added to the Java Metrics dialog. The first release of the UI Builder feature provides a single metric, *Number of Controls in Form*, that returns a number count of controls in a form type component (such as a JFrame derived class). Future releases will provide additional metrics.

*To use UI Builder metrics:*

1. Make sure your project includes Java UI classes as defined in this chapter under “Creating UI classes” on page 288.
2. Make sure the UI Builder feature is activated as described in this chapter under “Activating the UI Builder” on page 285.
3. On the main menu, choose **Tools | Metrics** to open the Java Metrics dialog.

4. Locate the *UI* category in the list of metric titles, and expand the category node to see all available metrics.
5. Check the metrics you want to run against your project and click **Start**.

## Generating user interface documentation

---

The Together documentation generation feature provides a special documentation template that you can use to generate documentation for the user interface of a project. Generate documentation in HTML, RTF, or text format using the standard template, modify the standard template, or design your own template.

For more detailed information on documentation generation features, see Chapter 17, “Generating Documentation for Together Projects”.

*To generate documentation for your user interface:*

1. Open the project containing the UI code you want to document.
2. On the main menu, choose **Project | Documentation | Generate Using Template**. The Generate Documentation Using Template dialog opens.
3. Choose the scope of the information to include in the generated documentation.

**TIP:** To document only user interface classes, open the class diagram that contains the UI classes and choose *Current Diagram*. If UI classes are in a specific package, select that package in the **Model** tab of the Explorer and choose *Current Package* (or *Current package with subpackages* if there are classes further down in the package hierarchy).

4. Use the drop-down list of templates and choose `$TG_HOME$\modules\com \togethersoft\modules\gendoc\templates\UIBuilderReport.tpl` (where `$TG_HOME$` is the root of your Together installation).
5. Choose the output format and output location, then click **OK**.

## Customizing the Toolbox component palette

---

User interface components are simply JavaBeans™. The toolbox comes with a default set of standard AWT and SWING bean components (for a full listing, search Online Help for *Toolbox*). The bean components are divided into named categories (such as *Swing General*, and *AWT Menu*). Each category has a scrolling page in the Toolbox that contains the various bean components assigned to that category. You can:

- Fully customize the Toolbox categories and the components each one contains.
- Reorder existing categories and add new categories.
- Add your own custom user interface components to the new categories you create, or to any existing category.

## Creating a component archive

---

Before you try customizing the Toolbox, create one or more JAR files containing the bean components to add to the Toolbox. These can be your own components or components of a third-party library.

To create new categories, a good rule of thumb is to create one appropriately named JAR file for each category page you want to add to the Toolbox. This helps to organize your components, but it is not required. Add specific beans from any JAR file to any Toolbox category. The JAR files containing custom bean components can reside on any drive accessible to your system.

**TIP:** Consider adding a *Custom* directory to `$TG_HOME$/lib` and placing your custom component archives there.

Once you have your component archives ready, you can begin customizing the Toolbox.

## Invoking the Toolbox customization dialog

---

The Palette Customization dialog is available from the right-click menu of any category header in the Toolbox.

*To access the Palette Customization dialog:*

1. Load a visually editable class in the Editor as described in “What is a visually editable class?” on page 288 and “Creating UI classes” on page 288.
2. Activate the UI Designer view by clicking the UI Designer button on the Designer pane toolbar (for more information, see “UI Designer view” on page 290).
3. To display the Toolbox, choose **View | Main Panes | Toolbox** on the main menu. Alternatively, click the Toolbox button on the main toolbar.
4. Right-click any of the category headers and choose **Customize Palette** in the Toolbox.

## Customizing toolbox categories

---

Reorder existing component categories, add new categories, or remove existing categories in the Palette Customization dialog.

### Reordering existing component categories

*To reorder existing categories:*

1. In the *Categories* list, select the category name you want to move. `Shift+click` to select multiple categories to be moved at once.
2. Using the toolbar immediately above the *Categories* list, click the up arrow or the down arrow to change the position of the selected category in the list. Continue clicking until the selected items are in the desired position.

## Creating new component categories

To create a new component category:

1. Click the **New Category** button on the toolbar immediately above the *Categories* list.
2. Edit the default name in the *Category Name* field. Press **Enter** to accept the edit.
3. Optionally change the appearance order of the new category as described in the previous section.

## Removing component categories

You can remove any existing component category. Removing a category automatically removes all the bean components it contained. The JAR file containing the components is *not* deleted by the remove operation.

To remove a component category:

1. In the *Categories* list, select the category name you want to remove. **Shift+click** to select multiple categories to be removed at once.
2. Click the **Remove** button on the toolbar immediately above the *Categories* list.

## Restoring the default component categories

You are not constrained from removing any component category, including the default categories that ship with Together. Should you remove any of the default categories and later want them restored, you can use the **Restore Defaults** button in the Palette Customization dialog.

**IMPORTANT:** If you click the **Restore Defaults** button, you will lose any customizations you may have made to any of the default categories. However, the user-defined categories will stay intact.

## Adding custom bean components

---

Add custom bean components to any existing component category by selecting a category and then selecting a JAR file that contains the components you want in the selected category. Once added to the category, you can change the appearance order of the individual bean components, remove individual bean components from the category, and customize the component names and icons of each bean component.

## Adding components to a category

To add bean components to a component category:

1. Select the desired category in the *Categories* list box.
2. Click the button labeled **Add Beans to Category**.
3. Click the browse button next to the *Path to JAR* field. In the file chooser dialog, select the JAR file containing the components.

4. If any components in the bean component JAR file require classes in other JAR files in order to work correctly, use the Advanced mode of the dialog to add these archives.
5. Click **OK** to close the Add Beans dialog and add the selected JAR file bean components to the category.

**NOTE:** All components contained in the archive are added to the category. If there are some you do not want to use, you can remove them as described in the following section.

## Removing components

You can remove any component from the category. Removed components do not appear in the Toolbox. Components are not removed from the underlying JAR file.

*To remove a component from a category:*

1. In the *Categories* list, select the category from which you want to remove components.
2. In the *Components* list, select the component name you want to remove.  
Shift+click to select multiple contiguous components to be removed at once.  
Ctrl+click to select multiple non-contiguous components to be removed at once.  
(Or use the multi-selection operations for your operating system, if different.)
3. Click the **Remove** button on the toolbar immediately above the *Categories* list.

## Changing component display names

You can change the display name of any component. The naming in the underlying JAR file is not affected.

*To change a component display name:*

1. In the *Components* list, select the component you want to rename. Its name is shown in the *Component Name* field.
2. In the *Component Name* field, edit the display name and press **Enter** to accept the change.

## Changing component icons

Each component has two associated icons, termed *large* and *small*. The large icon is displayed in the Toolbox. The small icon is displayed in the UI Component Explorer when the component is used in a user interface you are building in your project. Normally the icons are provided by the developer of the component.

If the component developer does not provide default icons, or if you want to use different icons, you can specify the icon image file you want displayed for each component.

*To use custom icon images:*

1. In the *Components* list, select the component for which you want to use custom icon images.
2. In the *Choose Icon* group, choose *Select icons*.

- Click the browse buttons next to the *Large icon* and *Small icon* fields in turn, and select the GIF or JPEG image you want to use for each of the two icon sizes.

## Java features supported in designable classes

Normally, you can design a correct user interface using a basic set of operations. The UI Builder supports Java grammar and lexical structure to the extent they are supported by the Together SCI, and provides partial support for a rather extended subset of features defined by the Java Language Specification, second edition ([http://java.sun.com/docs/books/jls/second\\_edition/html/jTOC.doc.html](http://java.sun.com/docs/books/jls/second_edition/html/jTOC.doc.html)).

The following tables summarize the supported Java Language Specification (JLS) features:

**Table 37** Supported JLS features

JLS feature	Not supported	Partly supported	Supported	Note
Java Syntax			X	
Keywords		X		Table 38 on page 329
Literals			X	
Separators			X	
Operators		X		Table 39 on page 329
Integer operations		X		Table 40 on page 330
Types, values and variables		X		
Conversions and promotions		X		Identity Conversion, Widening Primitive Conversion, Narrowing Primitive Conversion, Widening Reference Conversion, Narrowing Reference Conversion, String Conversion
Names		X		Array members are not supported
Classes		X		Classes with <code>abstract</code> , <code>final</code> , <code>strictfp</code> modifiers are not supported
Arrays	X			
Exceptions	X			
Execution		X		

**Table 37** Supported JLS features

JLS feature	Not supported	Partly supported	Supported	Note
Blocks and Statements		X		
Threads and Locks	X			

**Table 38** Supported Keywords

Supported Keyword	Note
boolean	May be used as a type of a variable
byte	May be used as a type of a variable
char	May be used as a type of a variable
class	May be used with some limitations
double	May be used as a type of a variable
float	May be used as a type of a variable
int	May be used as a type of a variable
long	May be used as a type of a variable
new	
private	May be used as a modifier
protected	May be used as a modifier
public	May be used as a modifier
return	
short	May be used as a type of a variable
static	Variables defined in a class body with the <code>static</code> modifier (class variables) are processed prior to other variables (instance variables)
super	
this	

**Table 39** Supported Operators

Operator	Note
=	
!	For boolean type only
? :	The type of conditional expression is not determined
==	
!=	
&&,	For boolean type only
++, --, +, -, *, /, &,  , ^, %	

**Table 39** Supported Operators

Operator	Note
+=, -=, *=, /=, %=	Not supported for arrays and array elements

**NOTE:** As of this writing, the left shift <<, signed right shift >>, unsigned right shift >>> and bitwise complement operators ~ are not supported.

**Table 40** Supported integer operations

Supported integer operation	Note
Numerical comparison ==, !=	Returns boolean value
Unary +, -	Returns int or long
Multiplicative *, /, %	Returns int or long
Increment prefix and postfix (++)	Returns int or long
Decrement prefix and postfix (--)	Returns int or long
Integer bitwise operations &,  , ^	
Conditional operator ?:	
Cast operator	Converts from an integral value to a value of any specified numeric type
String concatenation +	