12 - An Eclipse GUI Builder

12 - An Eclipse GUI Builder [1, 2]

by Raphael Enns and Shantha Ramachandran

Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada

Last revised: May 21, 2004

Overview:

In this tutorial, we describe how to use a GUI Builder plug-in for Eclipse, as well as look at its features. The GUI Builder that we will look at in this tutorial is the Designer created by Instantiations, Inc., and is available at http://www.swt-designer.com. We will be using version 2.0.0 of the Designer in this tutorial.

A GUI Builder:

In the previous tutorials, we have seen that SWT and Eclipse are fairly simple and straightforward to use. One of the most common questions about SWT, however, is "Where is the GUI Builder?" The answer is that there is no GUI Builder that comes with Eclipse. However, because Eclipse is open source and designed for plug-ins, a GUI Builder plug-in is able to be made and can be integrated right into Eclipse.

The Designer is a plug-in that allows you design SWT and Swing GUIs in Eclipse. The Designer is available at http://www.swt-designer.com. A restricted version of the plug-in is available for free download at the above website. There is also a professional version, which includes extra features, that is available for $199 USD. Look at http://www.instantiations.com/swt-designer/comparison.html to see a comparison between the free and the professional versions.

As of version 2.0 of the Designer, Instantiations includes both SWT and Swing support in the Designer. The SWT part is the same as in previous versions and is called the SWT Designer. The Swing part is called the Swing Designer and is similar to the SWT Designer. There is also a WindowBuilder product in the Designer which is simply both the SWT Designer and the Swing Designer.

Since most of the features are the same or similar between the SWT Designer and the Swing Designer, this tutorial will mainly look at features of the SWT Designer. To demonstrate the similarities and the differences between the SWT Designer and the Swing Designer, one of the examples at the end of this tutorial will be done in Swing.

[1] This work was funded by an IBM Eclipse Innovation Grant.

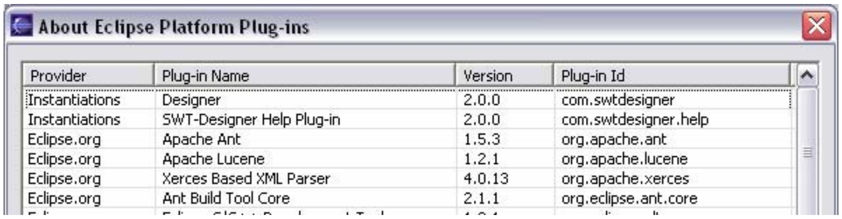
Acquiring the Designer Plug-in:

You can download the Designer from the website, http://www.swt-designer.com. It can be found in the Download section. Make sure you get the correct file for your version of Eclipse. In order to download the Designer, you must have a username and password. You can get a username and password by registering at the Designer's website.

Installation:

Once you have downloaded the Designer, installing it is quite simple. First make sure Eclipse is closed. Move the zip file into your Eclipse directory and then unzip the file. It will add three folders to your plugin directory that start with com.swtdesigner.

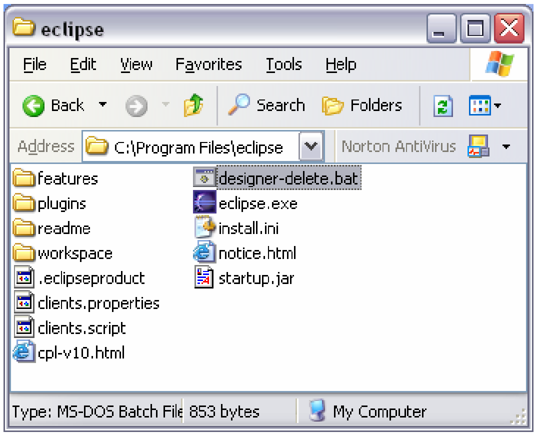
To make sure that the plug-in was successfully installed, start Eclipse. Go to the menu item Help > About Eclipse Platform. Click on the Plug-in Details button, and a list will appear. The Designer and the SWT-Designer Help Plug-in items should appear in the plug-in list.



About Eclipse Platform Plug-ins

Provider	Plug-in Name	Version	Plug-in Id
Instantiations	Designer	2.0.0	com.swtdesigner
Instantiations	SWT-Designer Help Plug-in	2.0.0	com.swtdesigner.help
Eclipse.org	Apache Ant	1.5.3	org.apache.ant
Eclipse.org	Apache Lucene	1.2.1	org.apache.lucene
Eclipse.org	Xerces Based XML Parser	4.0.13	org.apache.xerces
Eclipse.org	Ant Build Tool Core	2.1.1	org.eclipse.ant.core

In the case that you would wish to remove the Designer from your computer, the Designer creates a batch file called **designer-delete.bat** in your Eclipse program directory. Run this file to remove the Designer.



Address C:\Program Files\eclipse

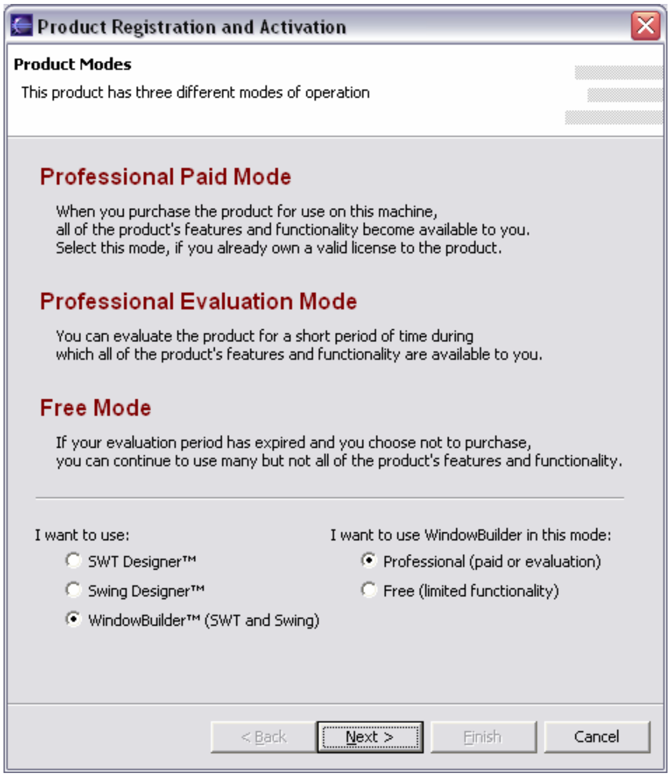
features
plugins
readme
workspace
.eclipseproduct
clients.properties
clients.script
cpl-v10.html
designer-delete.bat
eclipse.exe
install.ini
notice.html
startup.jar

Type: MS-DOS Batch File 853 bytes My Computer

Activating the Designer:

Before you begin using the Designer you have to activate it. To start the activation process, select Window > Preferences inside Eclipse. Select the "+" beside "Designer" on the left side of the Preferences window, click on License, and then click on the button that says "Registration and Activation".

Both SWT and Swing support comes with the Designer product, but depending on your serial number, only one of the three products is activated. For example, if you have a valid serial number for the SWT Designer, you can use the SWT functionality of the Designer, but you will not be able to use the Swing functionality except in evaluation or free mode. If you have a serial number for WindowBuilder, you will have full use of both the SWT and Swing functionality of the Designer.



Product Registration and Activation

Product Modes

This product has three different modes of operation

Professional Paid Mode

When you purchase the product for use on this machine,
all of the product's features and functionality become available to you.
Select this mode, if you already own a valid license to the product.

Professional Evaluation Mode

You can evaluate the product for a short period of time during
which all of the product's features and functionality are available to you.

Free Mode

If your evaluation period has expired and you choose not to purchase,
you can continue to use many but not all of the product's features and functionality.

I want to use:
- ○ SWT Designer™
- ○ Swing Designer™
- ● WindowBuilder™ (SWT and Swing)

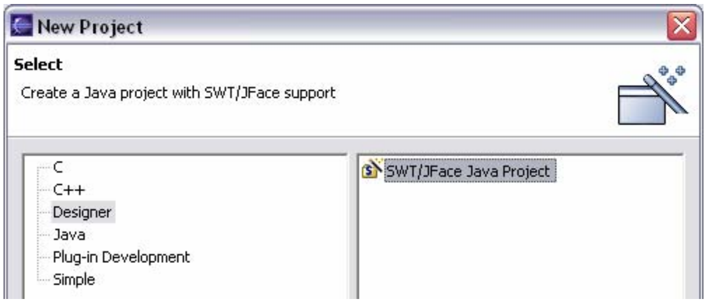
I want to use WindowBuilder in this mode:
- ● Professional (paid or evaluation)
- ○ Free (limited functionality)

Select the type of Designer that you want to activate, select the desired mode, and then click Next. Follow the directions in the following pages to activate the Designer. Once it has been activated you are ready to start designing GUIs.

Creating a new SWT/JFace Java Project:

An SWT/JFace Java Project and a regular Java Project are nearly equivalent. They both create a Java project and you can use the Designer with both types. The only difference between an SWT/JFace Java project and a regular Java project is that the SWT/JFace Java project will automatically add the SWT and JFace jar files to the build path.

To create an SWT/JFace Java Project, select File > New > Project, and then in the resulting window select SWT > SWT/JFace Java Project.



Select

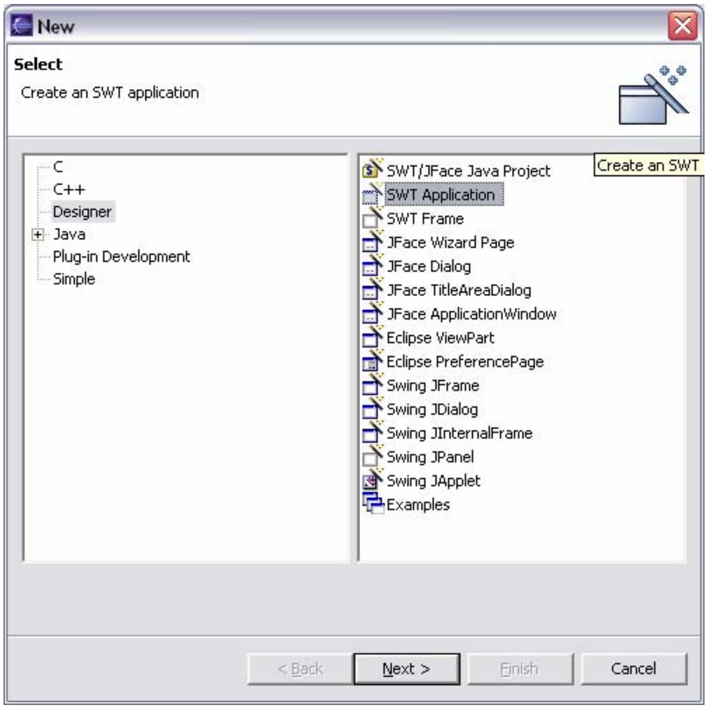
Create a Java project with SWT/JFace support

Click Next and then create the project the way you would normally create a Java Project. You do not have to add the SWT and JFace jar files to the build path because they are added automatically.

Note: To be able to use SWT, the SWT runtime DLL must be accessible by the SWT jar file. For information on how to do this, see the *Installing Eclipse* tutorial, which can be found at http://www.cs.umanitoba.ca/~eclipse.

Creating a new SWT Java File:

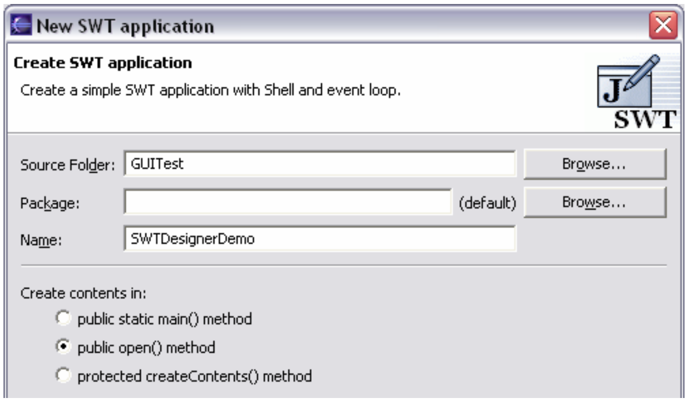
The easiest way to set up a file to use the Designer is to use the Designer Wizard. Select File > New > Other from the menu and select Designer > SWT Application.



Select

Create an SWT application

Click Next. In the window that opens, choose the project the file will go in and give the file a name. You can also select which method the SWT code will initially appear in by selecting the appropriate radio button as shown below. Click Finish to create the new file.



New SWT application

Create SWT application

Create a simple SWT application with Shell and event loop.

Source Folder: GUITest

Package: (default)

Name: SWTDesignerDemo

Create contents in:

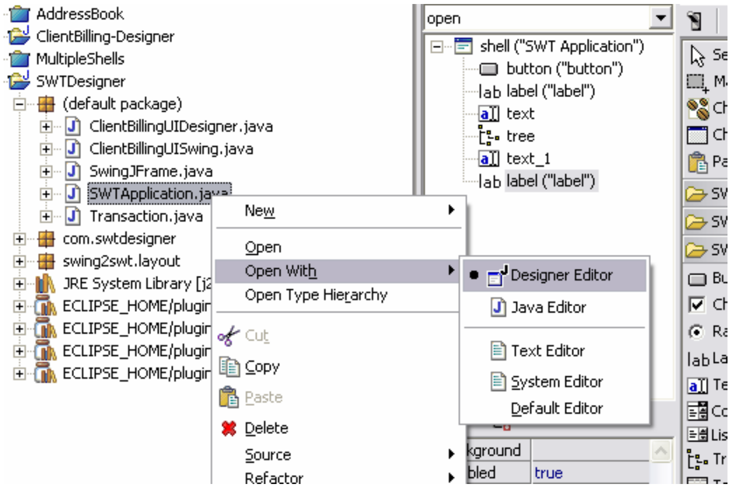
○ public static main() method

● public open() method

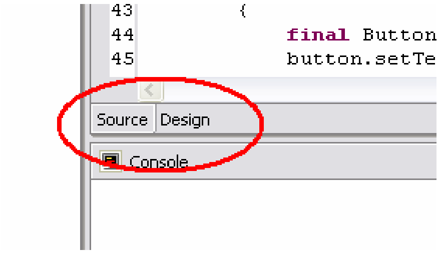
○ protected createContents() method

This will add a file to your project that already has an SWT event loop and a shell created for you. The file should automatically open in the Designer Editor.

To open a file that has not been created by the Designer, simply right-click on the file and select Open With > Designer Editor.

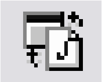


This will open the file in the Designer Editor in the source view. There are two tabs at the bottom of the editor window labeled Source and Design as shown below. If you click on the Design tab, the file will be displayed in the design view and you will be able to visually design your GUI.



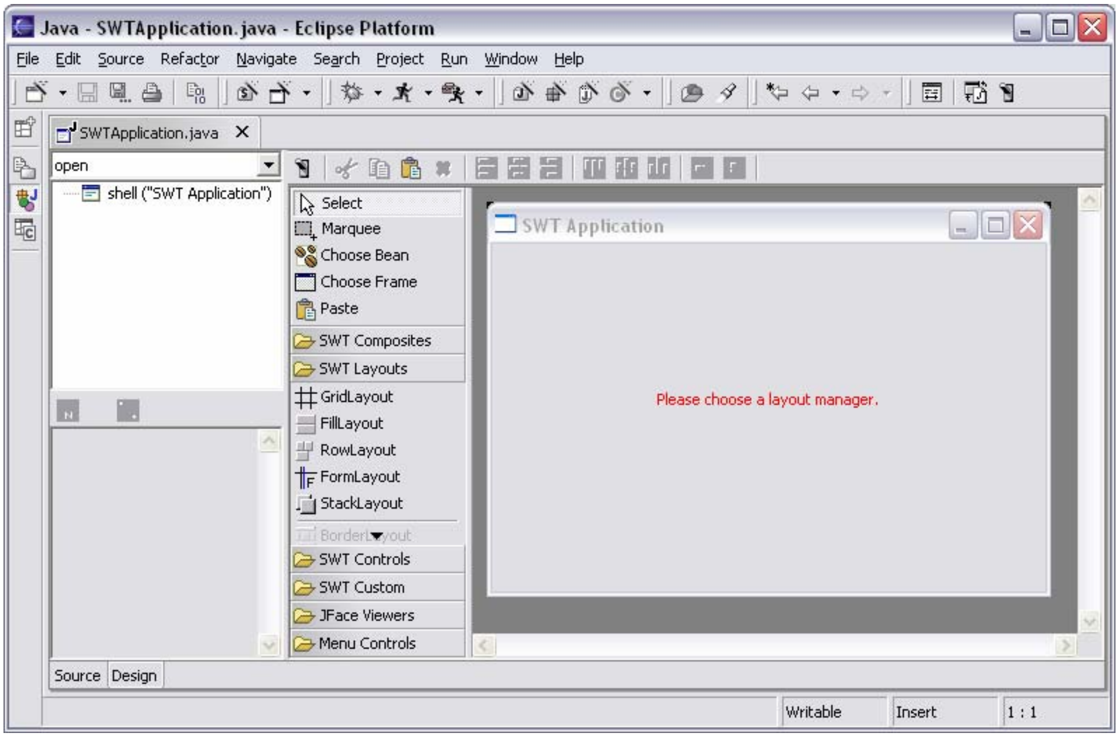
```
        {
            final Button
            button.setTe
```

Two other quick ways to switch between the source and design view are to click on the "Switch between source and designer" button in the toolbar that is shown below, or simply press F12.



Note: If a file was last opened in the Designer Editor, it will continue to open in that editor until a different editor is selected to open the file with.

Below is an image of the Designer in the design view for a new file that was created using the SWT Application wizard. Initially the class only has a shell with no controls and no layout.



Please choose a layout manager.

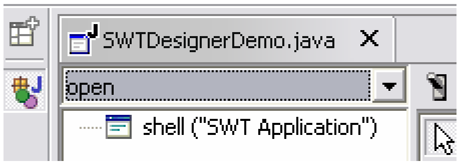
Note: To view the Designer Editor using the full screen, simply double click on the tab of the current file at the top of the editor window.

Note: The above image is of the professional version of the SWT Designer. The free version looks identical with a few less controls in the Control Palette and a few less items in the editor's toolbar.

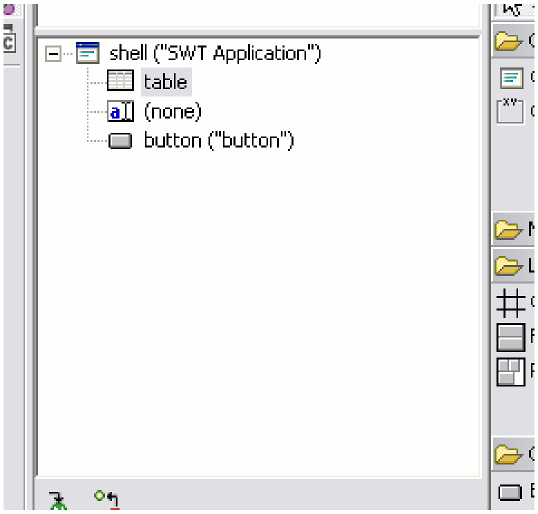
Parts of the Designer:

There are five different parts in the Designer design view: the Method List, the Control Tree, the Control Palette, the Content Pane, and the Inspector.

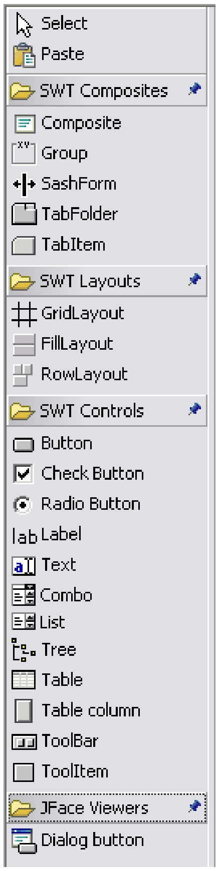
The **Method List** is a dropdown list of all the methods where controls are created. In this example, the controls are created in the method "open". When a method is selected, its controls appear in the Content Pane.

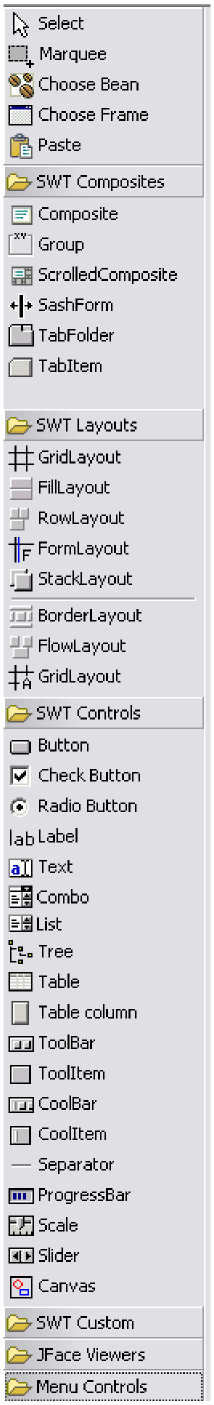


The **Control Tree** shows the hierarchy of controls in the selected method. You can select a control by clicking on it.

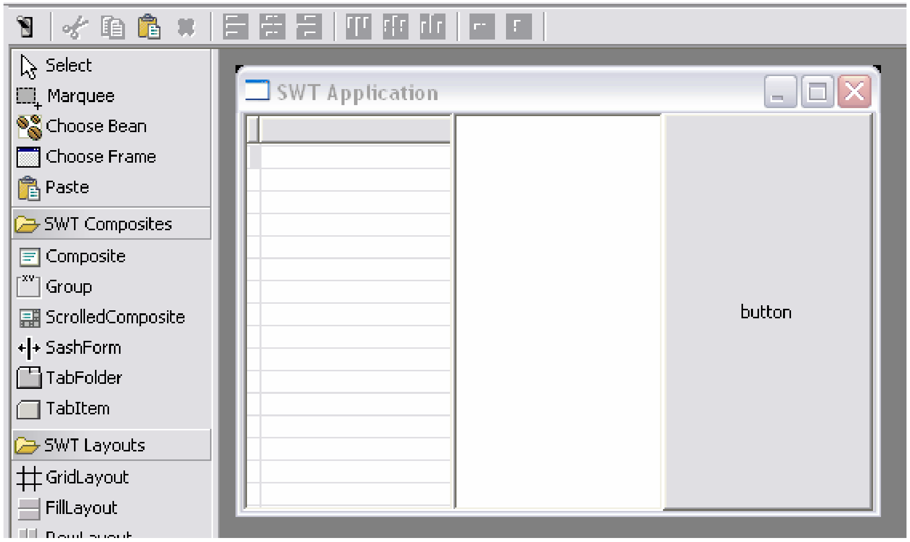


The **Control Palette** is the palette from which to choose your controls. It contains all the controls that can be added to your GUI with the Designer. Below is shown the Control Palette for the free version on the left and the professional version on the right.

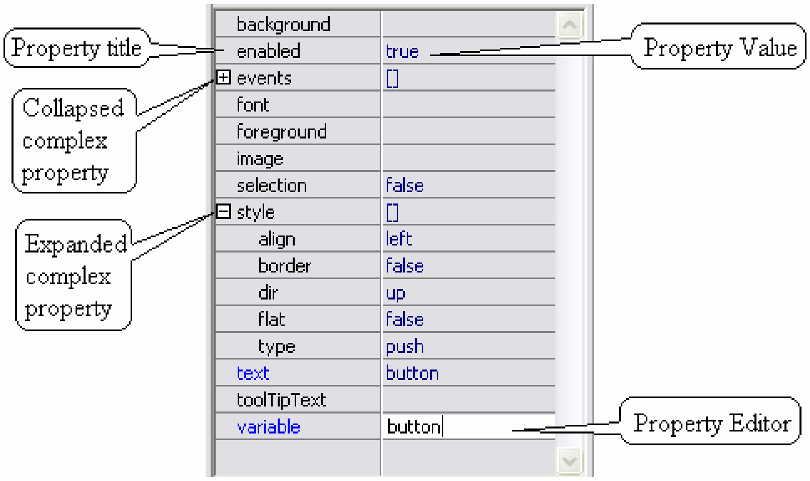




The **Content Pane** shows the GUI design as it will look when your application is run. It will also display some extra features to help in editing the design. There is a toolbar at the top of the Content Pane which has buttons for test, cut, copy, paste, delete, as well as several buttons to help position controls. The positioning buttons will only work with controls on a NullLayout. Some of the positioning buttons are only available in the professional version.



The **Inspector** shows all of the properties and events of a selected control and allows you to modify them without going into the code.



background	
enabled	true
⊞ events	[]
font	
foreground	
image	
selection	false
⊟ style	[]
align	left
border	false
dir	up
flat	false
type	push
text	button
toolTipText	
variable	button

Property title

Collapsed complex property

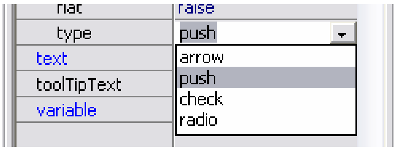
Expanded complex property

Property Value

Property Editor

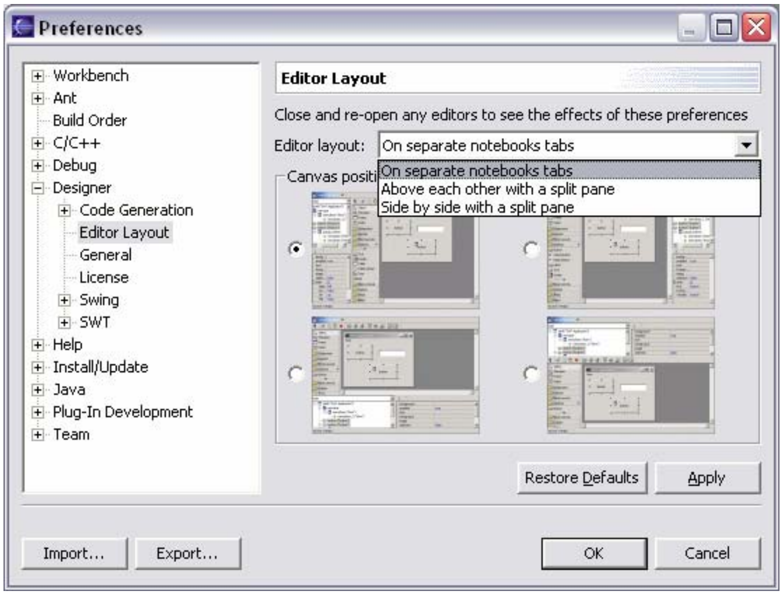
Select a property by either clicking on the property title or the property value. A property editor is then displayed which allows you to change its value. While editing a property value, you can press Enter to apply the value or press Esc to return to the previous value. If a property's value is not the default value, the property title changes color. Expand or collapse complex properties such as style by clicking on the '+' or '-' beside the property title.

Some properties are selected through a selection property editor. Simply select the value from the list that appears after clicking on the down arrow as shown below.



type: push

The layout of the Designer's editor can be changed in the Designer's preferences. Select Preferences from the Window menu to open the Preferences window. Expand the Designer item and select Editor Layout. The Editor Layout property page is shown below.



Editor Layout

Close and re-open any editors to see the effects of these preferences

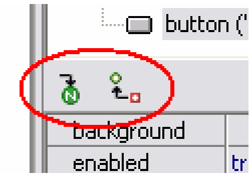
Editor layout: On separate notebooks tabs

You can choose one of three options in the Editor layout dropdown list. The "On separate notebooks tabs" option is the default option which places the design view and the source view on separate pages. The "Above each other with a split pane" option will remove the Source and Design tabs and place the design view above the source view in the same editor. The "Side by side with a split pane" option is similar except it places the design view on the left and the source view on the right. This allows you to view the source while designing in the design view without having to switch back and forth between the design and source views. Since the Designer updates the GUI bidirectionally, making changes in either the design or source view will instantly update the other view.

There are four pictures in the Canvas position group, each with a radio button next to it. These options place the Control Tree and the Inspector in different positions in the editor. You can see where each part is by looking at the pictures. The default option is the top-left option where the Control Tree and the Inspector are at the far left of the editor.

Open Definition and Convert Local to Field Buttons

There are two small buttons between the Inspector and the Control Tree. These are "Open Definition" and "Convert Local to Field" (or "Convert Field to Local" if it has already been clicked once).



When a control is selected and the "Open definition" button is pressed, the Designer will switch to the source view and place the cursor on the line where the selected control is instantiated.

By default, the Designer generates code using the block style: a control is declared and instantiated inside its own block of code. When a control is declared inside its block of code, it is visible only to its children and itself.

When a control is selected and the "Convert local to field" button is pressed, the Designer will move the declaration of the control from inside its block of code to being a field in the GUI class. When a control is declared as a field, it is visible to everything in that class. If there is a conflict of names when converting a control to a field, the Designer will give the control a unique name.

After the "Convert local to field" button is pressed once it becomes the "Convert local to field" button. When the button is pressed now, the Designer will move the declaration of the control back inside its block of code.

In the examples as the end of this tutorial we will see in more detail how these buttons work.

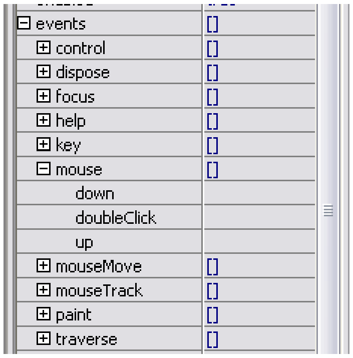
Note: An alternative to the block style of coding is the flat style. Controls are not declared in their own blocks, but are rather declared in the same scope. This is often how a GUI is coded when done by hand. The Designer supports generating code in this style. We will look at how to make the Designer generate code in this style in the Code Generation section of this tutorial.

Handling Events with the Designer:

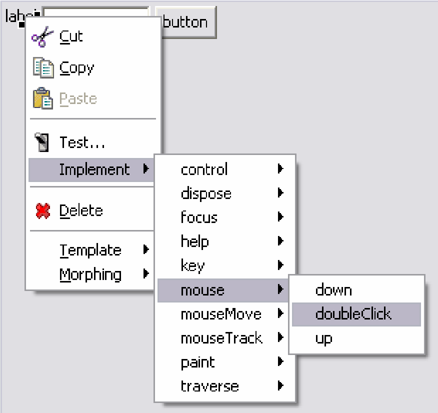
Events occur when the user interacts with the UI. The appropriate event-handling code is then executed. In order to know when events occur, event listeners must first be added to your controls. The Designer makes it very easy to add and remove event listeners to your controls.

Adding an Event Listener with the SWT Designer:

There are two ways to add an event listener with the SWT Designer. The first way is through the Inspector. First select a control in either the Control Tree or the Content Pane. Expand the "events" complex property in the Inspector. Then expand an event and double-click or press Enter on the method you wish to implement.



The second way to add an event listener is to simply right-click on a control (either in the Content Pane or in the Control Tree), select Implement > [name of the event] > [name of the method to implement].

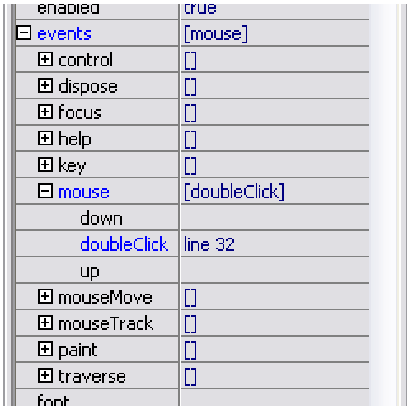


A quick way to add event listeners to buttons (including check and radio buttons) and menu items (professional version only) is to simply double-click on the button or menu item. When double-clicked, a selection event listener will be created.

Any way you add an event, the SWT Designer will automatically create an event listener for the selected event method. The SWT Designer will then switch to the source view and go directly to the new event handler method. The code generated looks like the following:

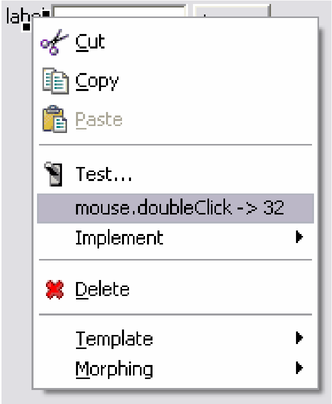
```
{
        final Label label = new Label(shell, SWT.NONE);
        label.addMouseListener(new MouseAdapter() {
                public void mouseDoubleClick(MouseEvent e) {
                }
        });
        label.setText("label");
}
```

There are two ways to quickly return to the event-handling code from the design view. In the Inspector, expand "events" and then the event. Then double-click or press Enter on the event method to return to the code. The events' property value in the Inspector is a list of all the events implemented, and each individual event's property value is the list of methods implemented. If a method has been implemented, its property value is the line number in the source code where its event-handling code begins.



enabled	true
⊟ events	[mouse]
⊞ control	[]
⊞ dispose	[]
⊞ focus	[]
⊞ help	[]
⊞ key	[]
⊟ mouse	[doubleClick]
down	
doubleClick	line 32
up	
⊞ mouseMove	[]
⊞ mouseTrack	[]
⊞ paint	[]
⊞ traverse	[]
font	

The other way to quickly return to the event-handling code is to right-click on a control (either in the Content Pane or in the Control Tree), and then select the menu item for the correct method that was implemented. Shown below is "mouse.doubleClick > 32" being selected. In this example, "mouse" is the event, "doubleClick" is the event method, and "32" is the line number in the source code on which the method begins.



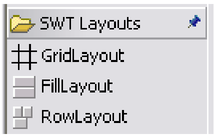
Note: The professional version of the Designer is able to generate an event handler stub method when a new event is created. We will look at how to make the Designer generate this method in the Code Generation section of this tutorial.

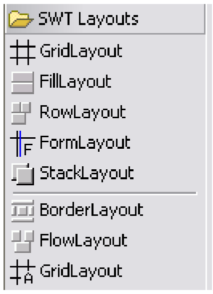
Deleting an Event Listener with the SWT Designer:

There is only one way to delete an existing event handler in the SWT Designer. Select a control in either the Content pane or in the Control Tree. In the Inspector expand "events" and then expand the event. Click on the event method you wish to delete and then press Delete. If there are no other methods that have been implemented for that event, the SWT Designer will also delete the event listener for that event.

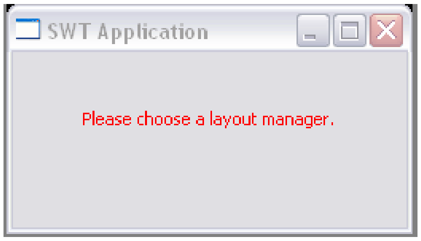
Using Layout Managers in the SWT Designer:

The first step to creating an application is to choose a layout. You can use either the default NullLayout or you can choose one of the layouts listed under the Layouts tab in the Control Palette. FillLayout, RowLayout, and GridLayout are all supported in the free version of the SWT Designer. In addition to the layouts in the free version, the professional version also supports FormLayout, StackLayout, BorderLayout, FlowLayout, and AWT GridLayout. BorderLayout, FlowLayout, and AWT GridLayout are ports of AWT layouts. Below are shown the Layout tabs in the free version (left) and the professional version (right).



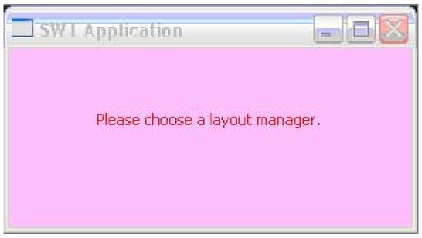


By default, the SWT Designer does not support designing a GUI using the NullLayout. If no layout is selected for a container, the container in the Content Pane will have red text in the center saying "Please choose a layout manager." We will find out how to get rid of this message and how to enable support for the NullLayout in the SWT Designer in the NullLayout section of this tutorial.



Please choose a layout manager.

After you select one of the layouts from the Control Palette, place your mouse over a container in the Content Pane. The area where the layout can be applied will appear highlighted, as shown below. Click the mouse and the layout will be applied. You can check and make sure it was applied by looking at the Inspector. The value of the layout property should now say the name of the layout selected (it will be blank if the NullLayout is used).



Please choose a layout manager.

To remove a layout from a container, select the container in either the Content Pane or in the Control Tree, click on the layout property in the Inspector, and then press Delete. Another way to remove a layout is to right-click on the container holding the layout in either the Content Pane or Control Tree and select Delete Layout.

When a layout is removed from a container, the SWT Designer converts the layout to an equivalent NullLayout by using the setBounds() method call for each control in the container. This means that after you delete a layout, the controls will be in the same position and the same size as before, even though the layout is now the NullLayout.

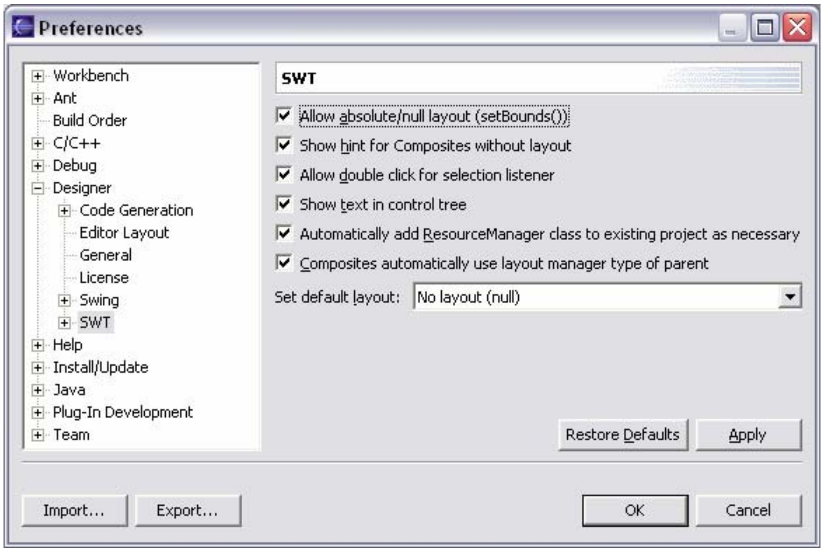
To switch from one layout to another on a container that already has one or more controls on it, you must first delete the current layout and then apply the new layout. If the container has no controls on it, you are able to change layouts without deleting the current layout first, unless the current layout is a FormLayout or a BorderLayout.

Adding controls to a container is similar to placing a layout. Simply select the desired control in the Control Palette, move your mouse to the desired area in the Content Pane, and click to place the control. A control can also be added by placing the control on the Control Tree. To delete a control, simply select it and press Delete. More specific examples of how to place controls for each layout are shown in the individual layout sections.

For more information on SWT layout managers, see the tutorial *Eclipse Layouts*.

NullLayout:

In order to begin designing applications using the NullLayout, you must first enable it in the SWT Designer. To do this, go to the menu item Window > Preferences. On the left side of the Preferences window, expand Designer and click on SWT. On the right side, check “Allow absolute/null layout (setBounds())”. Then click OK.



SWT

☑ Allow absolute/null layout (setBounds())

☑ Show hint for Composites without layout

☑ Allow double click for selection listener

☑ Show text in control tree

☑ Automatically add ResourceManager class to existing project as necessary

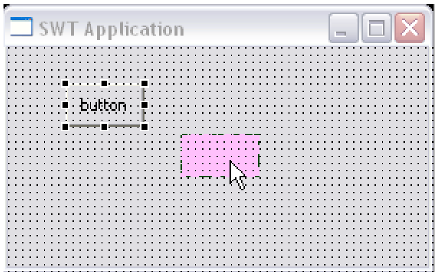
☑ Composites automatically use layout manager type of parent

Set default layout: No layout (null)

The NullLayout is very easy to use. You simply have to place and size a control exactly as you want it. One downfall with using the NullLayout is that you may lose some portability. Though your application may display correctly on your platform and screen resolution, on another platform, or on a different screen resolution, it may not display correctly.

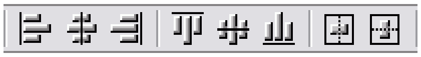
The NullLayout has no layout properties to set. To place a control on a shell with a NullLayout, select a control from the Control Palette, click on the shell to set the top-left corner of the control, and then drag to set the size of the control. The new control will then be placed on the area of the shell that you selected. To move a control, simply click on the control and drag it to where you want it. To resize a control, select the control either in the Content Pane or in the Control Tree. Then click and drag the handles that are around the edges of the control. To delete a control, simply select the control and press Delete.

When you create, move, or resize a control that is on a NullLayout, a grid is shown. The edges of the control you are creating, moving, or resizing will always snap to the grid points. The grid helps you to place the controls evenly. A button being moved on a NullLayout is shown below.



You are able to change the distance between the points on the grid. To do this, select Preferences from the Window menu. Expand the Designer item and select the General item. Setting the Grid step value will set the distance between the points. You can enter a value between 1 and 100. This value is the number of pixels between points. The default Grid step value is 5.

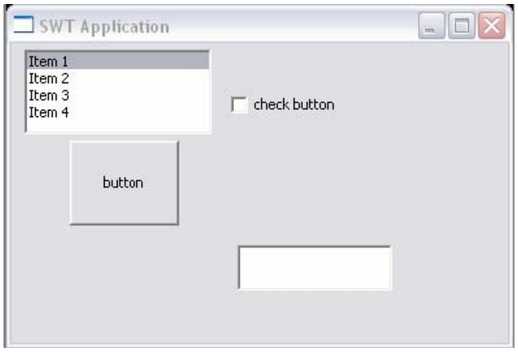
There are several buttons on the toolbar above the Content Pane that can be used to align controls on a NullLayout. Most of the buttons are only available in the professional version of the Designer. Only the two buttons on the far right are also available in the free version.



The buttons on the far right are "Center horizontally in window" and "Center vertically in window". To use them, select a control on a NullLayout and press one of the buttons. The selected control is now centered in the container.

To enable the other buttons, multiple controls must be selected. When multiple controls are selected and one of the alignment buttons is pressed, the controls will line up their edges or centers, depending on which button is pressed, to one of the selected controls. Generally the selected controls will always be aligned to the selected control highest in the z-order. The only way to align controls to a specific control regardless of its location in the z-order is to select the control you wish to align to, hold down Ctrl or Shift, and then in the Content Pane select the controls you wish to align. If you then press one of the alignment buttons, the other controls will align themselves to the first control selected. Once an alignment button has been pressed, any subsequent presses to any alignment button will return to aligning the selected controls to the control highest in the z-order.

Below is shown a shell with a NullLayout which has several controls placed in it:



Note: You cannot change the size and position of a control in the Inspector using NullLayout. To do this you must go into the source code and modify the setBounds() method call or move and resize the control on the Content Pane.

Note: If there is any container which uses a layout other than a NullLayout, and that container's parent uses the NullLayout, it will display controls as if it had a NullLayout. That means that every control still needs a setBounds() call even though the container it is on does not use the NullLayout. This makes using a different layout manager on the container to be useless. This is why you should either use the NullLayout for everything in an application, or do not use it at all.

FillLayout:

FillLayout is the easiest layout manager to use, both with and without the SWT Designer. It places your controls in either a single row or single column. FillLayout has only one style property to set – dir, which can be horizontal or vertical. There are two ways to set this style. The first is through the Inspector. Select the container in the Control Tree and then expand the layout property and the style property. The property value of the dir property can be set to either horizontal or vertical.

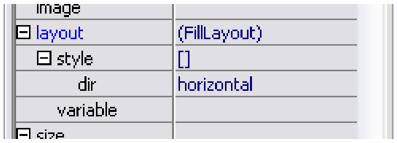
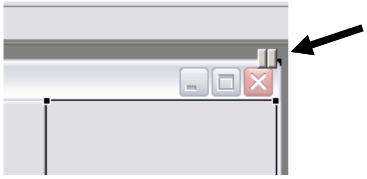


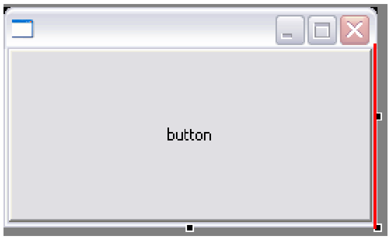
image	
⊟ layout	(FillLayout)
⊟ style	[]
dir	horizontal
variable	
⊟ size	

The second way to set this is within the Content Pane. When a control in a container with FillLayout is selected, a small switch appears in the top right corner of the container indicating which direction the layout is in. Clicking on this switch changes the direction. In the example below, the switch is indicating a horizontal FillLayout, as there are two "controls" side-by-side, horizontally.



For a vertical FillLayout, two "controls" are pictured one on top of the other, vertically.

To add the first control to a container using FillLayout, select a control from the Control Palette and click anywhere in the container. To add more controls, select a control from the Control Palette, and then place the cursor to the left or right of a control already placed (if the dir property is horizontal), or to the top or bottom of a control (if the dir property is vertical). A small red line will appear where the control is to be placed, as shown below.



If no red line appears when trying to place a control, make sure that the container that the control will be placed in is selected in the Control Tree. To move a control, simply click and drag the control to the desired position.

RowLayout:

Using RowLayout is similar to FillLayout in that you can set a dir property to horizontal or vertical. However, RowLayout does have a few more properties that can be set. These can all be set using the Inspector.

If the justify property is set to true, the controls will be evenly spaced out across the screen either horizontally or vertically depending on the dir property. The margin properties set how many pixels away from the edges of the container to place to controls. When pack is true, all the controls are their natural size. When pack is false, all the controls are the same size. The spacing property sets how many pixels are to be between each control. When wrap is true, the controls are wrapped onto the next line if they cannot fit all on one line.

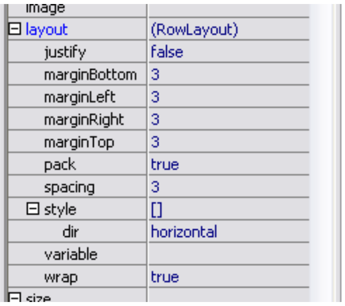
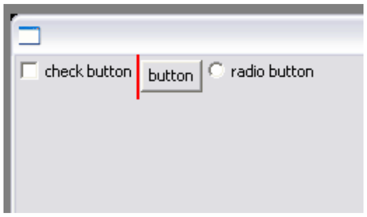
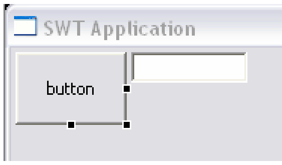


image	
⊟ layout	(RowLayout)
justify	false
marginBottom	3
marginLeft	3
marginRight	3
marginTop	3
pack	true
spacing	3
⊟ style	[]
dir	horizontal
variable	
wrap	true
⊟ size	

When placing a control onto the container, a red line will appear where the control will be placed. By moving your mouse around, you can specify exactly where in the layout you wish to place another control.

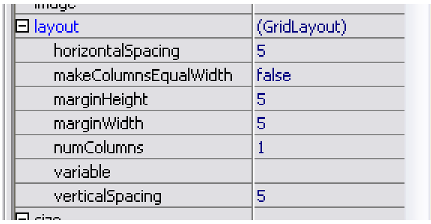


When a control placed on a container using RowLayout is selected, it will have three small square handles. The handles are on the bottom edge, right edge, and bottom-right corner, as shown below. If you click and drag these handles, you can resize individual controls.



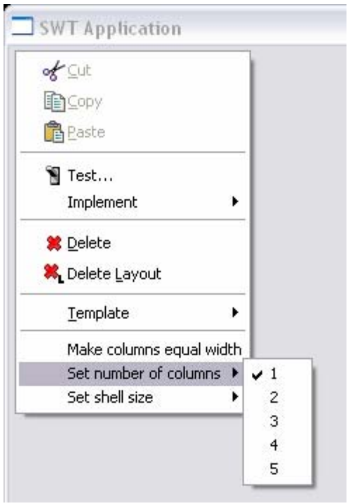
GridLayout:

GridLayout is significantly different from FillLayout and RowLayout because of the amount of layout data that can be manipulated. There are a number of properties that appear in the Inspector that can help you configure the GridLayout. The most important one is numColumns, which specifies how many columns are in your grid and therefore determines the size of the grid. The horizontalSpacing and verticalSpacing properties specify how many pixels are between each control. When the makeColumnsEqualWidth property is set to true, it sets each column in the grid to be equal in width.

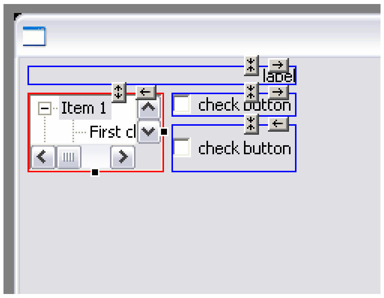


⊟ layout	(GridLayout)
horizontalSpacing	5
makeColumnsEqualWidth	false
marginHeight	5
marginWidth	5
numColumns	1
variable	
verticalSpacing	5

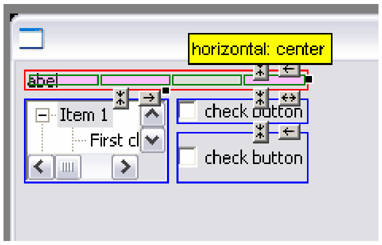
You can also set the makeColumnsEqualWidth and numColumns properties by right-clicking on the container in either the Content Pane or the Control Tree.



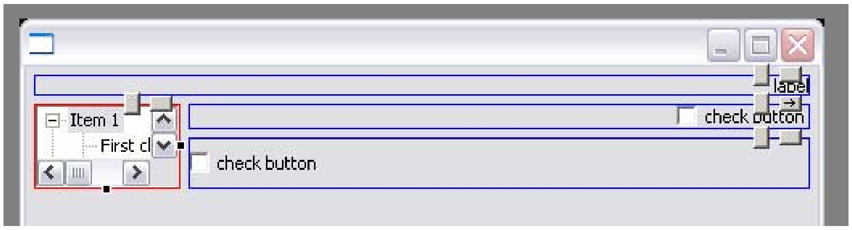
In the following example, there are two columns. You can see how each control in the grid has vertical and horizontal alignment buttons. By clicking on these buttons you can quickly and easily change the alignment of the controls, to beginning, ending, center, or fill.



You can also change the alignment simply by clicking on a control and dragging it through four different areas. A tool tip will appear indicating which type of alignment you are currently in. Simply drag the control to where you want it to be. To change the vertical alignment, press Ctrl while dragging the control through the different areas.



When you press the Ctrl key on your keyboard, the arrows on the alignment buttons disappear and they change into grab buttons. By holding down the Ctrl key and clicking on a grab button, the control will switch between grabbing excess space, and not grabbing excess space. Horizontal and vertical span can be set to span one or more columns or rows by dragging the handles of the controls.



You can also make a control grab horizontally or vertically by right-clicking on the control and selecting “Grab horizontal” or “Grab vertical” from the popup menu.

To facilitate these processes even more, there is a series of hot keys that can be used to set alignment and grab properties. To change a property, simply select a control and press the hot key on your keyboard. Here are the implemented hot keys:

l – set horizontal alignment to left
r – set horizontal alignment to right
c – set horizontal alignment to center
f – set horizontal alignment to fill
t – set vertical alignment to top
b – set vertical alignment to bottom
C – set vertical alignment to center
F – set vertical alignment to fill
h – set horizontal grab on/off
v – set vertical grab on/off
o – set horizontal and vertical alignments to fill and switch grab on/off

All of these properties can also be modified in the Inspector under the layoutData property of a control as shown below.

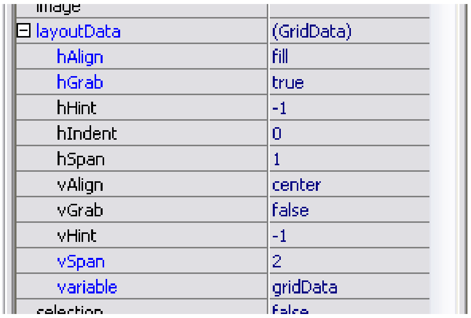


image	
⊟ layoutData	(GridData)
hAlign	fill
hGrab	true
hHint	-1
hIndent	0
hSpan	1
vAlign	center
vGrab	false
vHint	-1
vSpan	2
variable	gridData
selection	false

As with FillLayout and RowLayout, when you are placing an item on the shell, a red line will appear to indicate where on the grid your new control will be placed. Once a control is placed, the layoutData properties will appear in the Inspector for the control.

FormLayout (Professional Version Only):

Though the FormLayout is one of the most complicated layouts, it is also one of the most useful and powerful. By using the FormLayout, you are able to configure each side of a control to "snap" to window margins, percentage points, and even to the sides of other controls. Controls can be freely placed on a container with FormLayout. They can be freely moved and resized as well.

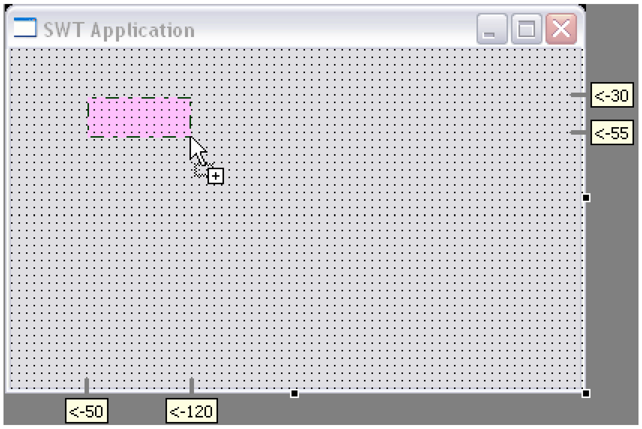
The SWT Designer fully supports the FormLayout and you can easily configure the layout graphically. The SWT Designer has dynamic snap points at the window margins, at predefined percentage points, and near other controls. By dragging a control near these snap points, the control will be set to the snap point and will dynamically move and resize to stay consistent to the snap points.

The free version of the SWT Designer does have some limited FormLayout support to allow you to see how your application will look. You are not able to add FormLayout to a container in the SWT Designer, but you are able to properly view a GUI using FormLayout. You are also able to modify the layout properties in the Inspector of controls placed on a FormLayout. The snap points in the Content Pane are only available in the professional version.

Placing the FormLayout on a container is the same as other layouts. Simply select FormLayout from the Control Palette and place it on the container. The layout itself does not have any properties. Each control on a container using FormLayout has configuration data in the form of FormData, which can be accessed in the layoutData property in the Inspector.

To place a control, simply select a control from the Control Palette and click anywhere on the container. If you just release the mouse button right where you clicked it, the control will be created at its natural size with its top-left corner exactly where you clicked. If you drag the control after clicking, you can specify the size of the control.

While placing the control, you may see some tool tips on the right and bottom of the container with a number in them. These show the number of pixels from the top or left edge of the container.



Like the NullLayout, the FormLayout displays a grid when creating, moving, or sizing controls.

If you select a control place on a FormLayout and expand the layoutData property in the Inspector, you should see four complex properties called left, right, top, and bottom. These are FormLayout settings for each of the four sides of the selected control. All of the sides have the same properties.

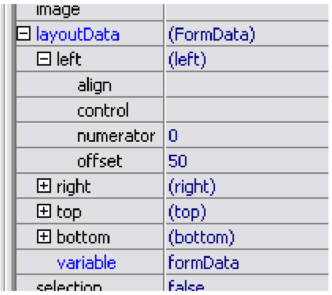


image	
⊟ layoutData	(FormData)
⊟ left	(left)
align	
control	
numerator	0
offset	50
⊞ right	(right)
⊞ top	(top)
⊞ bottom	(bottom)
variable	formData
selection	false

If you select the control property, a button with three dots will appear in the property editor. If you click this button, a window pops up with a list of all the controls that the selected control can snap to. The align property allows you to set which side of the control listed in the control property the selected control is snapped to. The align property can also be set to center or default. Setting the align property to center places the snap point in the center of the control listed in the control property. Setting the align property to default sets the snap point to the side of control listed in the control property opposite the side of the selected control. For example, if you were changing the left side of a control and selected a default alignment, the left side of the control would snap to the right side of the control listed in the control property.

If a side of the selected control is snapped to another control, the offset property specifies the number of pixels the two sides of the controls are from each other from the perspective of the control listed under the control property. For example, let's say that the selected control is to the right of the control that it is snapped to and the left side of the selected control is snapped to the right side of the other control. If the two controls have a 50 pixel offset between them, the offset property of the left side of the selected control would be 50.

Note: Not all controls are necessarily listed in the window of controls to snap to. The only controls that are listed are the siblings of the control (controls created in the same container) that are higher in the z-order (higher up in the Control Tree), any sibling of a parent control (the parent can be any number of levels deep), and any parent control (to any number of levels).

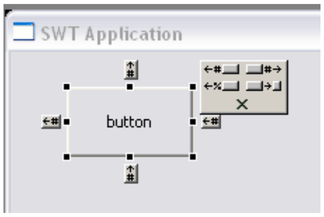
The numerator property is a percentage. Its value can be an integer from 0 to 100, inclusive. By setting this value, the selected side of a control will snap to this percentage point. The value 0 corresponds to the left edge (when modifying the left or right sides of a control) or the top edge (when modifying the top or bottom sides of a control) of the container holding the control. The value 100 corresponds to the right edge (when modifying the left or right sides of a control) or the bottom edge (when modifying the top or bottom sides of a control) of the container holding the control.

If a side of the selected control is snapped to a percentage point, the offset property specifies the number of pixels the side of the control is from the percentage point from the perspective of the percentage point. For example, if the left side of a control has a numerator value of 50 and an offset value of 10, the control is 10 pixels to the right of the center of the container.

Note: A side of a control can be snapped to either another control or to a percentage point. If you set the control property for a side of a control, the numerator property is cleared. Likewise, if the numerator property is set, the control and align properties are cleared.

Although you can fully modify the snap points of all four sides of a control in the Inspector, it is often a lot easier to do this in the Content Pane. The SWT Designer allows you easily set up snap points graphically.

In the Content Pane, you will notice a small button on each side of your control. When clicked, these buttons will bring up a popup menu with some more buttons. These buttons change what the side of the control is snapped to.



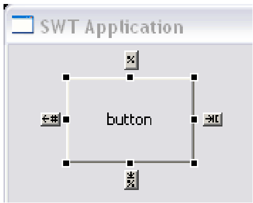
The two buttons in the popup menu that have an arrow and a number sign ←# #→ specify that the side of the control is always to be the same number of pixels from the edge of the container that the arrow is pointing to.

The button with the percent sign ←% specifies that the side of the control is always to be a specific percentage away from the left or top edge of the container.

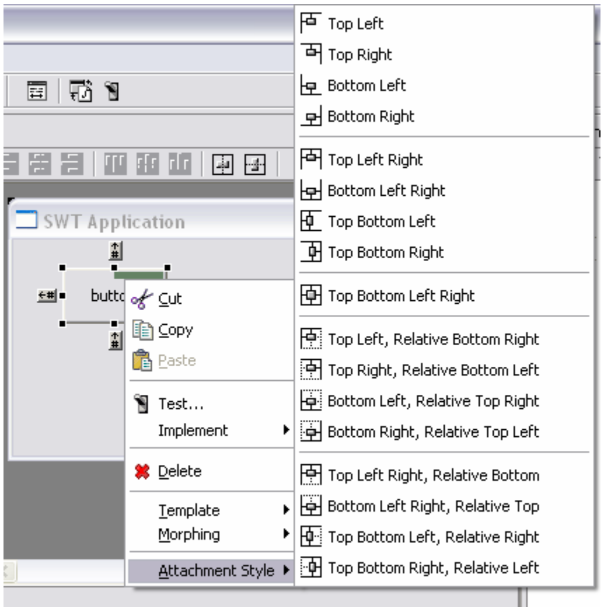
The button that shows one rectangle pointing to another rectangle ▭←▭ specifies that the side of the control is always to be the same number of pixels from the edge of another control. When this button is clicked, the window with the list of all the controls that the selected control can snap on to appears.

The ✕ button removes any snap points for that side.

Below is shown a button with each side having a different type of snap point. The left side snaps to a fixed offset from the left window edge. The top side snaps to a percentage point from the top edge of the window. The right side snaps to another control. The bottom side snaps to a fixed offset from a percentage point.

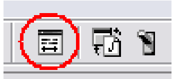


To quickly set snap points using some often used settings, simply right-click on a control, select Attachment style, and then select the style that you want to use.

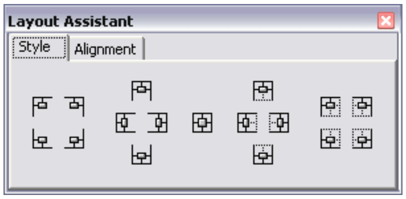

SWT Application
butto
Cut
Copy
Paste
Test...
Implement
Delete
Template
Morphing
Attachment Style
Top Left
Top Right
Bottom Left
Bottom Right
Top Left Right
Bottom Left Right
Top Bottom Left
Top Bottom Right
Top Bottom Left Right
Top Left, Relative Bottom Right
Top Right, Relative Bottom Left
Bottom Left, Relative Top Right
Bottom Right, Relative Top Left
Top Left Right, Relative Bottom
Bottom Left Right, Relative Top
Top Bottom Left, Relative Right
Top Bottom Right, Relative Left

There are five sections in the menu. The first section attaches all four sides using fixed offsets from two of the container's sides. The second section attaches all four sides using fixed offsets from three of the container's sides. The third section attaches each side to the corresponding side of the container. The fourth section attaches two of the sides to fixed offsets from two of the container's sides and attaches the other two sides to percentage offsets from the container's sides. The fifth section attaches three of the sides to fixed offsets from three of the container's sides and attaches the fourth side to a percentage offset from the container's side.

You can get all the attachment styles that appear in the menu above in a small Layout Assistant window. The button to open this window is in Eclipse's main toolbar.

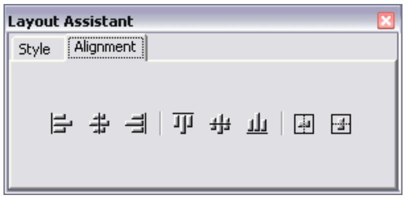


Once you have clicked the button, the following window appears. If no controls are selected or if you are not using FormLayout, the Layout Assistant window will say "Parent does not provide any layout assistance."

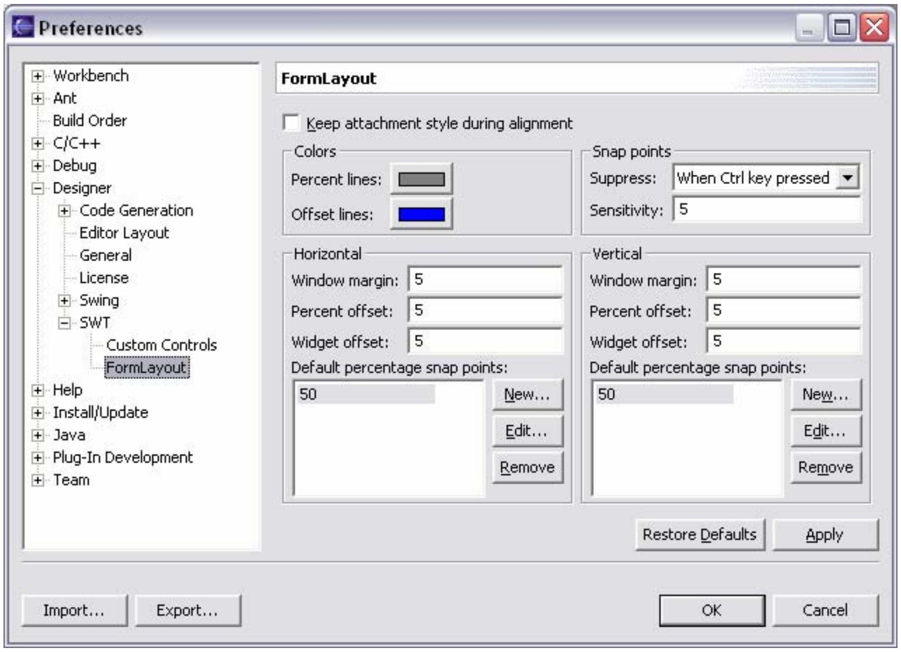


The FormLayout also supports the alignment buttons on the toolbar above the Content Pane that we looked at in the NullLayout section of this tutorial. These buttons function similarly when using the FormLayout as when using the NullLayout.

Another way to access the alignment buttons while using FormLayout is by clicking the Alignment tab in the Layout Assistant window.



You can also set the snap points to predefined offsets and percentage points. To see what these values are and how to change them, we will go to the SWT Designer's FormLayout properties page. To get there, select Window > Preferences, then select Designer > SWT > FormLayout from the pane on the left. You should see the following page:



FormLayout

☐ Keep attachment style during alignment

Colors
Percent lines: ▨
Offset lines: 🟦

Snap points
Suppress: When Ctrl key pressed
Sensitivity: 5

Horizontal
Window margin: 5
Percent offset: 5
Widget offset: 5
Default percentage snap points:
50

Vertical
Window margin: 5
Percent offset: 5
Widget offset: 5
Default percentage snap points:
50

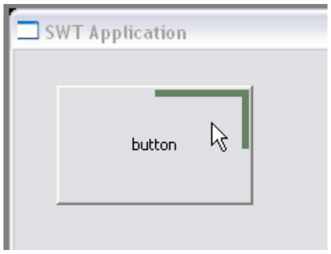
When “Keep attachment style during alignment” is checked, the attachment styles for a control do not change when the alignment buttons are used. If it is unchecked, the attachment styles of the sides of the controls you are aligning will be changed to the alignment styles of the control you are aligning to.

The Suppress dropdown box specifies when snap points will not be displayed. The default value is “When Ctrl key pressed”. This means that when you hold down the Ctrl key, snap points will not be displayed on the Content Pane. The snap point sensitivity is the maximum number of pixels that the control can be away from the snap point for the control to be “grabbed” by the snap point. If this value is set to -1, no snap points will be displayed.

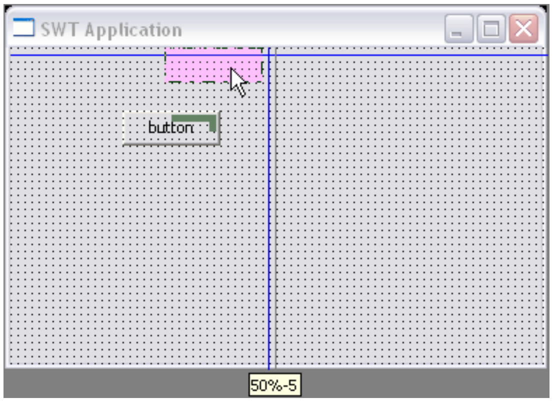
Snap point settings can be set independently for horizontal and vertical operations. There are snap points at given offsets from the window margin, from each default percentage point, and from each widget. If a value of -1 is given for any of these, that particular snap point attachment will be disabled. Default percentage snap points can be set as well. Doing this creates snap points anywhere in a container. The values can be integers from 1 to 99 inclusive (50 would be the center of the container). To show how the default percentage snap points work, we will create a horizontal and a vertical percentage point, each with a value of 50.

If you create a control and move it around, you will see lines appearing at certain places. These are the snap points that are defined in the FormLayout property page.

Every control is broken up into four quadrants. If you move your mouse over a quadrant, there will be a line on the outside edges of that quadrant as shown below. When you move the control by dragging it, the two sides that have the line will be the only sides able to snap to a snap point.



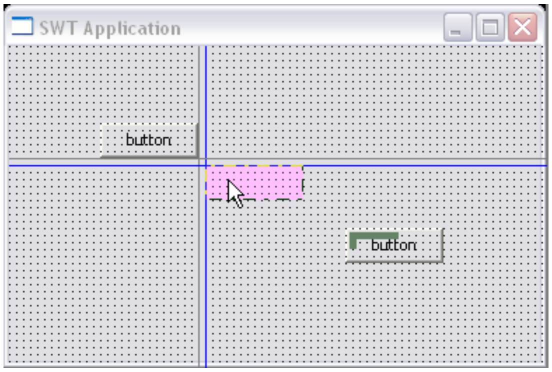
Snap points are shown as gray lines for percent lines and blue lines for offset lines. The colors for these lines can be changed in the property page. To snap to a preset snap point, move the mouse into the proper quadrant of the control, and drag the control near the snap points. Lines will appear and the control will be grabbed by the snap points when you drop it.



Above there are two blue lines. The horizontal blue line is the window margin offset, which is set to 5 pixels. The vertical blue line is the 5 pixel offset from the 50 percentage snap point that we added to the properties page. The gray line is the actual 50 percentage snap point. There is also a tool tip on the bottom displaying what the horizontal snap point is. If the button above is dropped where it is shown, the right side of the button will always be 5 pixels left of the center of the window, and the top of the button will always be 5 pixels below the top of the window.

When you drag the handles of a control to resize it, the snap points will also appear and you can snap to them.

Dragging a control near another control that is higher in the z-order (higher up in the Control Tree) will also display snap points. As shown below, the two blue lines are the 5 pixel offsets from the sides of the other control, while the gray lines are the snap points formed by the sides of the other control. Resizing a control also displays snap points to other controls. If a control is moved or resized and it has other controls that are snapped to it, the other controls will also move or resize to stay consistent with the snap points.



There are also hot keys to quickly set the majority of snap points. They are more complicated than the hot keys for the GridLayout, but it can be worthwhile to understand them. To use the hot keys, you need to press a sequence of keys. The case of the letters does not matter. Below is a list of the key presses. The information was taken from the Designer's documentation.

- L, R, T, B – select the side you with to work with (Left, Right, Top, Bottom)
  - M – set the attachment to the margin
  - A – work with Absolute pixels
    - L, R, T, B – set offset from the Left, Right, Top, or Bottom of parent
      - several digits representing the pixel offset followed by Enter
    - several digits representing the pixel offset followed by Enter (if no side of the parent is given, Left or Top is used)
  - P – work with Percentage points
    - A – uses a zero offset
      - one digit followed by Enter or two digits representing the percent
    - S – uses the negative of the offset for the attachment
      - one digit followed by Enter or two digits representing the percent
    - one digit followed by Enter or two digits representing the percent (this uses the offset for the attachment)

The above may be a little bit unclear, so below is a table of the keys. Start in the left column of the table and work your way right. If a step has several letters separated by commas, you can choose one of the options. "nothing" refers to not pressing any keys on that step. Look above to get the meaning for a sequence of keys.

L, R, T, B	M			
	A	L, R, T, B, nothing	Digits	Enter
	P	A, S, nothing	Digit	Digit
				Enter

As you start using the FormLayout, you will find that it is very powerful. Using the SWT Designer, it is also very easy to use. The SWT Designer allows you to modify FormData settings in several ways, allowing you to use the way that works the best for a given situation.

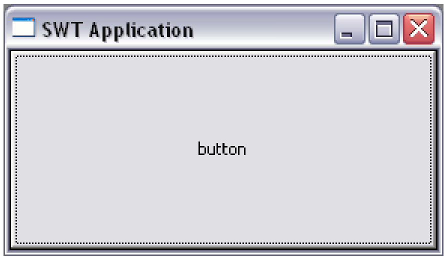
For more information on FillLayout, RowLayout, GridLayout, and FormLayout, see the tutorial *Eclipse Layouts*.

StackLayout (Professional Version Only):

StackLayout is not one of the standard layouts in SWT although it does come with SWT. StackLayout resizes all controls to fill the container. Therefore only one control is visible at a given time. This layout may be used when you want two or more controls to be in the same location, with only one of them being displayed at a time. A container using a StackLayout can be thought of as a TabFolder, except instead of switching pages by clicking on tabs, you have to specify your own actions to switch controls.

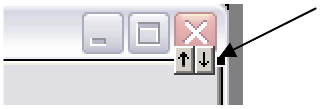
The free version of the SWT Designer can display containers using StackLayout and can modify layout properties, but it cannot place a StackLayout on a container.

Any number of controls can be placed on a container using StackLayout. Only the top control is displayed. When the application is run, the control that is on top in the design view is going to be the top control.



There are two layout properties in the Inspector that you can modify. They are the marginHeight and the marginWidth. These set the number of pixels the controls are from the edges of the container.

There are two small buttons that appear in the Content Pane that allow you to change which control is displayed. To access them, click on a control that is on a StackLayout. The buttons are shown below.



Clicking on the arrow pointing up will switch to the control higher up in the z-order. Clicking on the arrow pointing down will likewise switch to the control lower down in the z-order. Once the top or bottom control in the z-order is reached, it will wrap around.

Another way to select a control is to simply select a control in the Control Tree. That control will be displayed in the Content Pane and will be the control that is displayed when the application is run.

If you wish to switch controls when an event occurs, add these lines to the event handler:

```
stackLayout.topControl = control;
container.layout();
```

where stackLayout is the variable name for the StackLayout, control is the name of the control that you want to be placed on top, and container is the container on which the StackLayout is placed on. The top line sets the appropriate control to the top and the bottom line redraws the container. If the container is not redrawn, the top control will not switch until you perform an operation that forces a redraw such as resizing the window.

If you wish to switch controls when an event is triggered by a control outside of the container using StackLayout, you will have to convert several items to fields. The items to be converted to fields are the container using the StackLayout, the controls in the container, and the StackLayout. Converting the container and the controls in it to a field is easy. Simply select each one in turn and click on the “Convert local to field” button. Converting the StackLayout to a field is a little bit harder because there is no StackLayout item to select to use the “Convert local to field” button with. You will have to edit the source code.

Remove “`final StackLayout`” from the line that instantiates stackLayout and add the following line to the rest of the fields at the top of the class:

```
private StackLayout stackLayout;
```

Now you can access stackLayout from anywhere in the class. If you only wish to access it from the same method, an alternate way is to move the whole instantiation line of code outside of any blocks and to the top of the method. Now you can switch the top control exactly like we did before.

BorderLayout (Professional Version Only):

BorderLayout is an AWT layout that has been ported over to SWT. BorderLayout allows you to place controls on five specific spots on the container. These spots are the center and the four edges.

When a BorderLayout is placed on a container, there are a couple of properties listed under layout in the Inspector. The hgap and vgap properties are the number of horizontal and vertical pixels that are between controls.

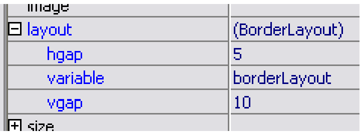
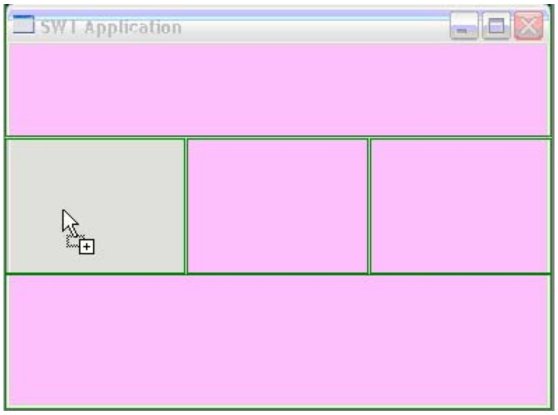
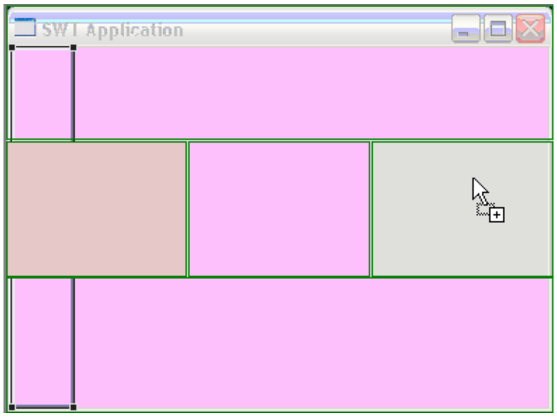


image	
⊟ layout	(BorderLayout)
hgap	5
variable	borderLayout
vgap	10
⊞ size	

When you try to place a control on a BorderLayout in the Content Pane, a grid will appear showing the different areas that you can place the control.



Once a control has been placed, the spot where it has been placed is a different color. You are not able to place another control in that spot.



When a control that is placed on a BorderLayout is selected, it will have a layoutData property. You can set this property to north, south, west, east, and center. North refers to the top of the container, south is the bottom, west is the left, east is the right, and center is the center. Only one control can be displayed in any of the five spots at one time.

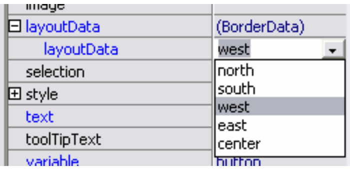
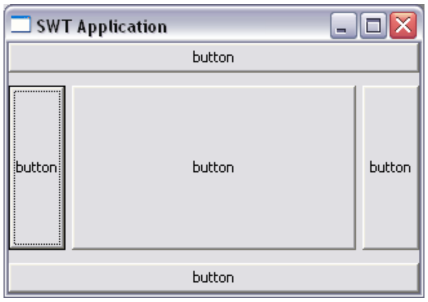


image	
⊟ layoutData	(BorderData)
layoutData	west
selection	
⊞ style	
text	
toolTipText	
variable	button

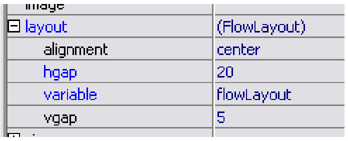
A shell using BorderLayout with the hgap set to 5, the vgap set to 10, and with a button in each spot looks like the following:



FlowLayout (Professional Version Only):

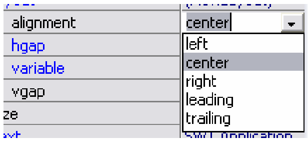
FlowLayout is also an AWT layout that has been ported over to SWT. It is similar to RowLayout in that it places controls in rows starting from the top. Once a row is full, it places controls on the next row. Unlike RowLayout, FlowLayout is able to use different alignments. The controls on a FlowLayout are always their natural size and cannot be resized.

You can change the alignment, horizontal gap (hgap) and vertical gap (vgap) properties of the layout in the Inspector.

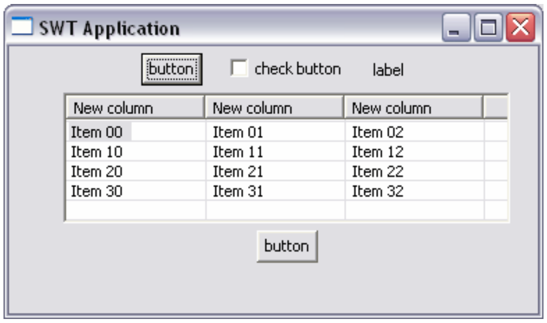


⊟ layout	(FlowLayout)
alignment	center
hgap	20
variable	flowLayout
vgap	5

The alignment property can be set to left, center, right, leading, or trailing. Leading is the same as left on a left-to-right orientation, and trailing is the same as right on a left-to-right orientation. The horizontal and vertical gaps are the number of pixels horizontally and vertically between each control.



A shell using a FlowLayout with center alignment looks like the following:



New column	New column	New column
Item 00	Item 01	Item 02
Item 10	Item 11	Item 12
Item 20	Item 21	Item 22
Item 30	Item 31	Item 32

GridLayout (AWT) (Professional Version Only):

GridLayout is an AWT layout that has been ported over to SWT. The AWT GridLayout is somewhat similar to the SWT GridLayout, but it is more restricted.

Under the layout property in the Inspector, you can set the number of columns, the number of rows, and the gaps between controls. You may notice that the layout property's value is the fully qualified name of GridLayout. This is so that the AWT GridLayout will not conflict with the SWT GridLayout.

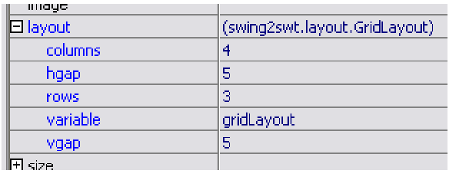
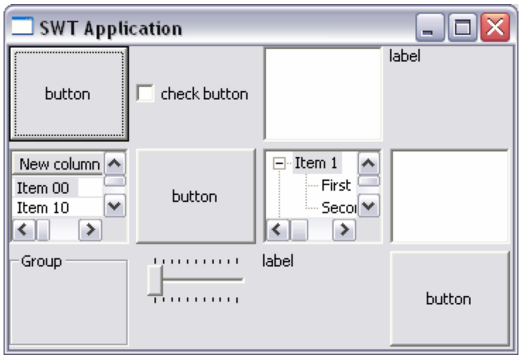


image	
⊟ layout	(swing2swt.layout.GridLayout)
columns	4
hgap	5
rows	3
variable	gridLayout
vgap	5
⊞ size	

The AWT GridLayout forces the columns to have equal width and the rows to have equal height. The largest number of controls that can be visible on a container at one time is the number of columns multiplied by the number of rows. Any more controls that are added will not be visible.

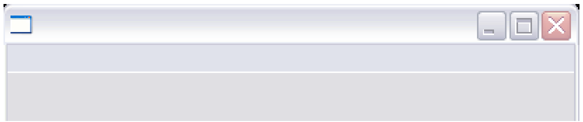
Below is shown a shell using the AWT GridLayout with four columns and three rows. All twelve cells in the grid have a control in them and they are all equal in size.



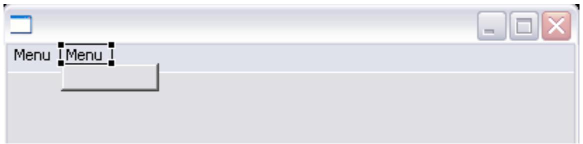
Creating Menus in the SWT Designer (Professional Version Only):

Menus are a must for just about all GUIs. Using the SWT Designer, it is quick and painless to create menu bars and popup menus. The free version of the Designer is able to display already created menus, but it does not have support to create them. We will look at creating menus manually in an example later in this tutorial for users of the free version of the Designer.

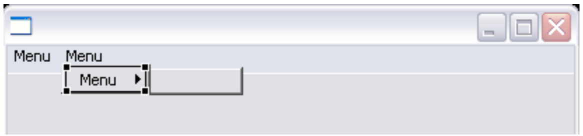
To create a menu bar on your shell, select MenuBar from the Control Palette, and then place it on your shell in the Content Pane. Each shell can have at most one MenuBar, and the only place that you can put a MenuBar is directly on a shell. A particular menu will only be displayed on the shell on which it was created. After the MenuBar is placed, you should see a blank menu bar across the top of your shell in the Content Pane.



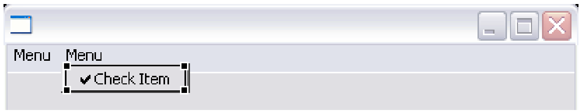
To add individual menus to the menu bar, select Menu from the Control Palette, and then place it on the menu bar. A menu is the same as a menu item that has its cascade style property always set to true.



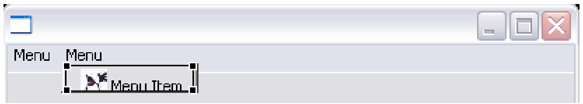
To add a menu item to a menu, select the dropdown menu and add a MenuItem to the dropdown menu that is displayed. If you want to make a submenu, add a Menu to the menu or set a menu item's cascade property to true.



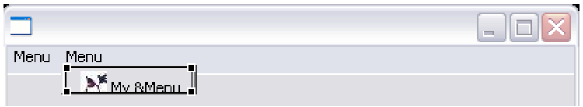
Menu items can also be a check or radio style. To create a checkbox or radio button menu item, select CheckBoxMenu or RadioButtonMenu from the Control Palette and add it to a menu. CheckBoxMenus and RadioButtonMenus are MenuItems with their style set to check or radio. To make a check or radio menu item initially be selected, set the selection property to true in the Inspector for that menu item.



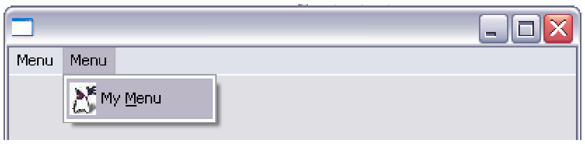
You can also add an image to a menu item. Simply select the menu item, click on the image property in the Inspector, click on the button that appears in the property editor, and select an image. For more information on selecting images, see the Additional Features of the Designer section of this tutorial. The Designer will only show the top part of the image if the image is too big, but when the program is run, it will display the entire image.



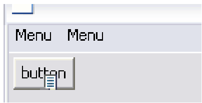
Setting the text of a menu item is the same as setting the text of any other control. In the Inspector, set the text property to the desired text. By placing an ampersand (&) before a letter in the text property, you will create a mnemonic for that menu item. On most platforms, the letter after the ampersand will be underlined when you run the program. When you press this key when the menu is displayed, the menu item will be selected.



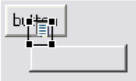
To quickly test what the menu will look like in your application, use the Designer's test feature. To use this feature, click on the test button in the toolbar above the Content Pane or right-click in the Content Pane and select Test.



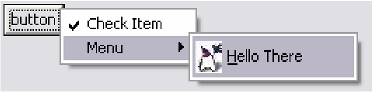
In addition to creating a menu bar, you can also create a popup menu for any control, including the shell. Simply select PopupMenu from the Control palette, move it over to the desired control in the Content Pane and click to place it on that control. Only one popup menu can be placed on a control. After a popup menu is placed on a control, a little menu icon will appear on that control as shown below.



If you click on that icon, the popup menu is displayed.



Adding menu items to the popup menu is identical to adding menu items to the dropdown menus on the menu bar. To see your popup menu at work, use the Designer's test feature and right-click on the control you added the popup menu to. The menu you created will now appear.



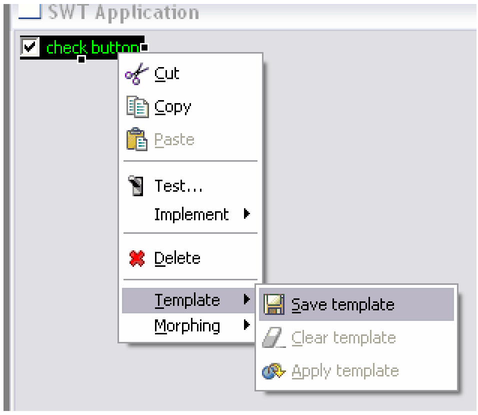
As you have seen above, creating menus in the SWT Designer is very quick and easy. Adding a selection event handler for the menu item is also easy. Simply double-click a menu item and a selection event handler will be created for you.

For more information on menus, see the tutorial *Advanced SWT Widgets*.

Custom Widget Templates (Professional Version Only):

Sometimes you might want to place several controls of the same type, each with the same properties. You could place each control and set their properties individually, or you could select multiple controls and then set all their properties at once as will be shown later in this tutorial, or you can create a custom widget template for that type of control. Custom widget templates allow you to save your own default properties for a control type. When a new control of that type is created, it will have the same properties that you set up. For example, if you wanted all of your check buttons to have a green foreground and a black background, you could set up a template to create all your check buttons like that.

In order to create a template, a control must first be placed. Set the properties of the control to the desired settings, then right-click on the control in either the Content Pane or the Control Tree and select Template > Save template. Now any control of that type that is created will have the properties from the template.



To remove a template for a type of control, right-click on any control of that type and select Template > Clear template. Now if any new controls of that type are added, they will be created with the default properties. Clearing the template will not affect already placed controls.

You can also apply a template to any existing controls by right-clicking on a control whose type has a template and selecting Template > Apply template. The control will now have the properties saved in the template.

There are several properties that are not saved in a template. They are the events, layoutData, variable name, image, and items properties. The scope of a control (whether it is a local or a field) is also not saved.

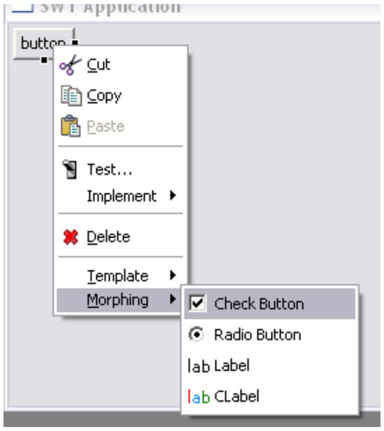
The templates can be used in different files. The templates are saved even if you close Eclipse. You can create at most one template for each type of control.

Widget Morphing (Professional Version Only):

It can be frustrating to suddenly want a group instead of a composite after you have already completed the design of the composite. It is very easy to remedy this using the professional version of the Designer.

The Designer allows you to morph similar controls from one type to another. When a control is morphed from one type to another, the properties that are the same between the two types are kept. This allows quick design changes without having to recreate all the controls.

To morph a control from one type to another, right-click on the control and select a control type to morph to from the Morphing cascading menu as shown below.



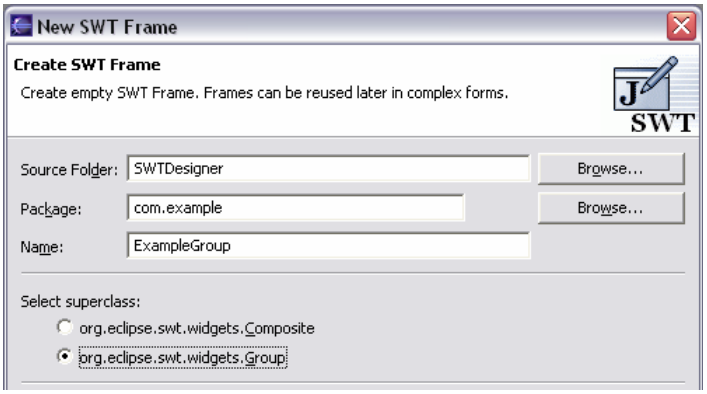
There are numerous control types that are able to morph into other types. Below is a list of the morphs that are possible with SWT controls. A bidirectional arrow means that the morph can go both ways. A list of control types with bidirectional arrows between them means that one of the types can morph to any of the other types.

Composite ↔ Group
TabFolder ↔ CTabFolder
Button ↔ Check Button ↔ Radio Button ↔ Label ↔ CLabel
Text ↔ Styled Text ↔ Combo ↔ CCombo
Combo ↔ CCombo ↔ List
Canvas → Composite
TreeViewer → CheckboxTreeViewer
ListViewer → Combo, CCombo
TextViewer → Text, Combo, CCombo

Custom Composites (Professional Version Only):

The professional version of the SWT Designer allows you to add frames to your application. Another name for a frame is custom composite. A frame is a custom class that extends composite or group. With the SWT Designer, you can easily create your own frame and add it to your application.

To create an SWT Frame using the SWT Designer, select File > New > Other. In the window that appears, select Designer on the left pane, select SWT Frame on the right pane, and then click Next. Enter a package name and the name for the class. The SWT Designer will not be able to use a frame that is in the default package, so you must specify a package. Select the desired frame superclass and click Finish to create your class.



New SWT Frame

Create SWT Frame

Create empty SWT Frame. Frames can be reused later in complex forms.

Source Folder: SWTDesigner

Package: com.example

Name: ExampleGroup

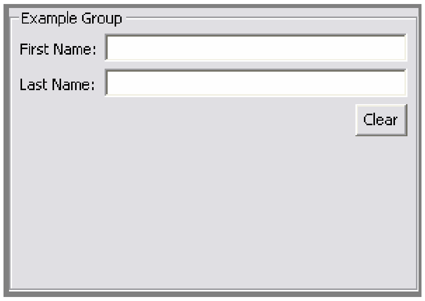
Select superclass:

○ org.eclipse.swt.widgets.Composite

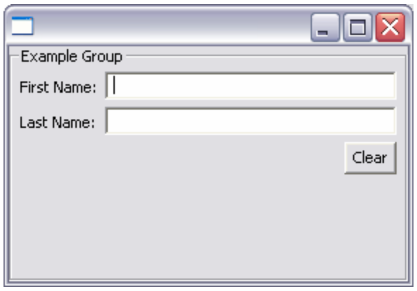
● org.eclipse.swt.widgets.Group

In the Content Pane, you should see a blank composite or group:

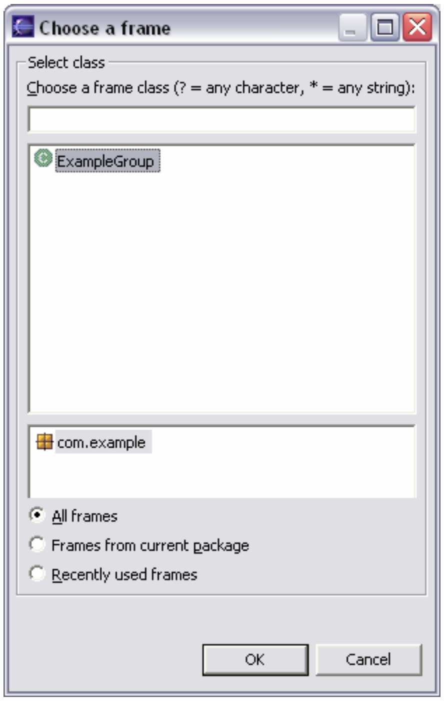
You can add a layout and controls to the frame and edit its properties exactly like you would with a regular composite or group created in an SWT Application.



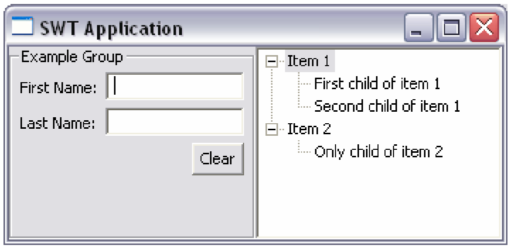
If you click on the Designer's test button, your frame will be displayed in a shell.



After you have created your frame, you can place it in an SWT Application. To add a frame, select Frame from the Control Palette. It is listed close to the top. Select the frame you wish to add from the window that appears and click OK.



Now place your frame just like you would place a normal control. Your frame acts like a singular control. When you click on the frame, you can edit its properties, which are the same properties as that of a composite or group. However, you cannot select controls inside of or add controls to the frame in your SWT Application. You can only edit the controls inside the frame if you open the class for the frame. Below is shown a frame added to an SWT Application.



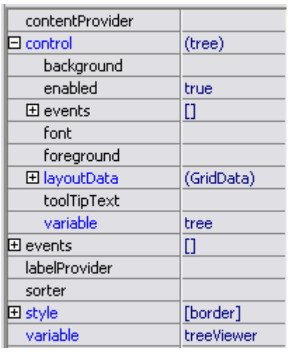
JFace Viewers (Professional Version Only):

JFace extends several controls such as lists and trees to make them even more powerful. You are able to link objects with JFace viewers instead of just displaying strings of text or images. You are also able to easily sort and filter the items to be displayed.

The SWT Designer supports creating all the standard JFace viewers. These are TableViewer, CheckboxTableViewer, TableTreeViewer, TreeViewer, CheckboxTreeViewer, and ListViewer. The SWT Designer also supports the TextViewer, but this viewer is considerably different than the other viewers. To use the TextViewer, jfacetext.jar needs to be added to the project's build path. We will not look at the TextViewer in this tutorial.

To create a JFace viewer in the SWT Designer, select a viewer from the Control Palette under the JFace category and place the viewer on the application window. You can place JFace viewers anywhere you can place a regular SWT control.

The viewers look just like the type of control that they extend. For example, a ListViewer looks like a List and a CheckboxTreeViewer looked like a Tree with its check style set. However, the viewers have several properties in addition to the properties of the control they extend. Below is shown the properties of a TreeViewer.



contentProvider	
⊟ control	(tree)
background	
enabled	true
⊞ events	[]
font	
foreground	
⊞ layoutData	(GridData)
toolTipText	
variable	tree
⊞ events	[]
labelProvider	
sorter	
⊞ style	[border]
variable	treeViewer

The properties of the control that the viewer extends are under the control property. In this case, the properties of a Tree are listed under the control property. The contentProvider, labelProvider, and sorter properties set the classes that contain each provider or sorter.

There are two ways to set the contentProvider, labelProvider, and sorter properties. If you double-click any of these properties in the Inspector, a nested class is created for that property. The nested class will extend and implement any classes and interfaces that are usually used for the selected viewer. Stubs for all the necessary methods are also generated. The providers and the sorter are then set to the viewer by the following code:

```
treeViewer.setSorter(new Sorter());
treeViewer.setLabelProvider(new TreeLabelProvider());
treeViewer.setContentProvider(new TreeContentProvider());
```

If you want to use top-level classes as your providers and sorter, simply change the previous code to pass instances of the top-level classes as parameters in the setSorter(), setLabelProvider(), and setContentProvider() methods.

For more information on creating JFace Viewers, see the *JFace* tutorial. To learn more specifically about TableViewers and TreeViewers, view the articles at the following links.

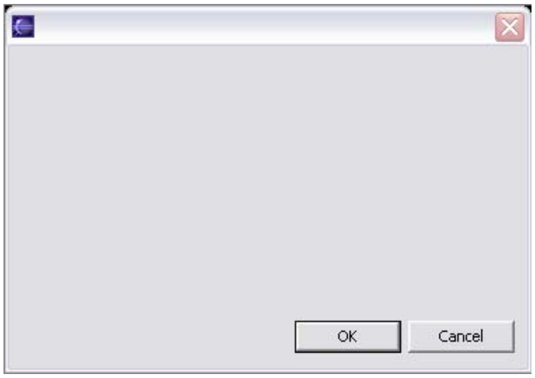
http://www.eclipse.org/articles/Article-Table-viewer/table_viewer.html

http://www.eclipse.org/articles/treeviewer-cg/TreeViewerArticle.htm

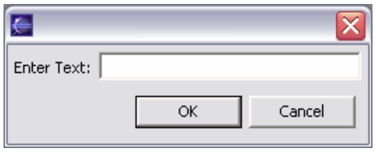
Custom JFace Dialog Creation:

The only JFace element that is fully supported in the free version of the SWT Designer is the JFace dialog. The SWT Designer allows you to graphically create a custom JFace dialog.

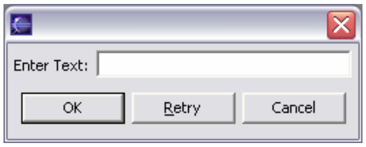
To create a custom dialog select File > New > Other. Select Designer on the left side of the page and JFace Dialog on the right side. Click Next to go to the next page. On the next page enter in the class name for your new dialog and then click Finish. You should now see the following in the Content Pane:



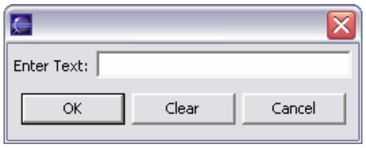
The area between the title bar and the buttons is called the dialog area and the area where the buttons are is called the button bar. Any normal SWT controls can be added to the dialog area. Below is shown a label and a text field added to the dialog area.



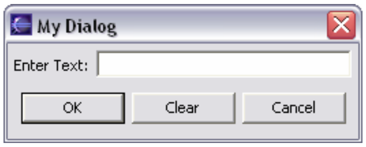
The buttons in the button bar are not normal buttons. They are special JFace dialog buttons. Only JFace dialog buttons can be added to the button bar. You can create buttons in the button bar by selecting Dialog button from the JFace category in the Control Palette and then placing the button on the button bar. You can change the button type in the type property in the Inspector. The text of the button is automatically changed to reflect the type. Shown below is a Retry button added to the button bar.


Enter Text:

You can change the text on a button using the text property in the Inspector. Below, the Retry button's text is changed to "Clear".


Enter Text:

You can change the title of a dialog by selecting the dialog and then changing the title property in the Inspector.


My Dialog

Enter Text:

If we look at the source code now, we see that the controls in the dialog area are declared in the createDialogArea() method, the buttons in the button bar are declared in the createButtonsForButtonBar() method, and the dialog's size and title are set in the configureShell() method.

The abstract class, Dialog, which our dialog class extends, implements the methods buttonPressed(), okPressed(), and cancelPressed(). The buttonPressed() method gets called anytime a button is pressed and the ID of the button is passed as a parameter. The okPressed() method is called every time the OK button is pressed and the cancelPressed() method is called every time the Cancel button is pressed. If you want to perform an action in the dialog when one of the dialog buttons is pressed, you would override one of the methods above.

If we want to make our Clear button clear the text field we have to override the buttonPressed() method. Our buttonPressed() method is shown below.

```
protected void buttonPressed(int buttonID)
{
        if (buttonID == IDialogConstants.RETRY_ID)
                text.setText("");

        super.buttonPressed(buttonID);
}
```

Since our Clear button is really a Retry button, we check if the Retry button has been pressed. If it has, the text field is cleared. Because the superclass, Dialog, already handles presses to the OK and Cancel buttons properly, we can simply call its buttonPressed() method. If you want to explicitly set the return code of the dialog, you can call the setReturnCode() method, passing an integer representing the button ID as a parameter.

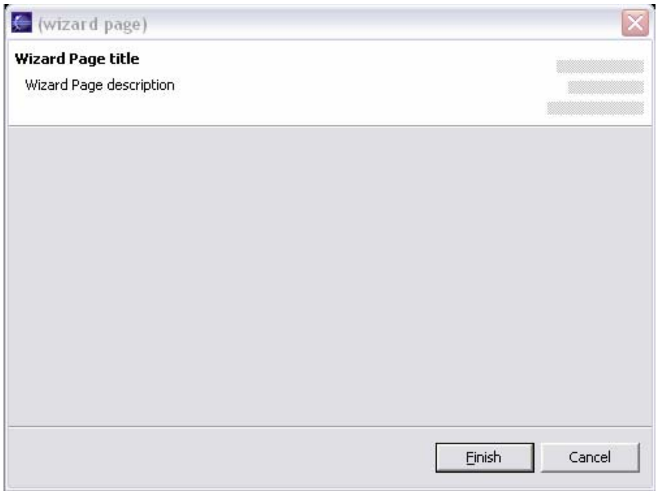
To open our dialog, we need to create an instance of it from another class and then call its open() method.

Inside an SWT application, create an instance of our dialog class, passing the Shell of the application to the constructor of the dialog. You can then call the dialog's open() method whenever you want to open the dialog. The open() method returns an integer representing the return code of the dialog. You can also get the return code of the dialog by calling the dialog's getReturnCode() method. You can use this return code to determine which button was pressed.

JFace Wizard Page Creation (Professional Version Only):

Wizards are a very useful form of user input. They are used when multiple steps are needed to be completed by the user. JFace allows you to quickly and easily set up a wizard user interface. To create a JFace wizard, you need three types of objects. First you will need a container to hold the wizard. For a stand-alone application you would use a WizardDialog object. Then you need a wizard object that extends the Wizard class and finally a number of separate wizard pages, each extending the WizardPage class. To learn how to fully create a JFace wizard, see the JFace Wizards tutorial at http://www.eclipse.org/articles/Article-JFace%20Wizards/wizardArticle.html.

The SWT Designer allows you to create the content for each individual wizard page. To create a wizard page select File > New > Other. Select Designer on the left side of the page and JFace Wizard Page on the right side and then click Next. On the next page enter in the class name for your new wizard page and then click Finish. You should now see the following in the Content Pane.

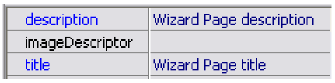


Wizard Page title

Wizard Page description

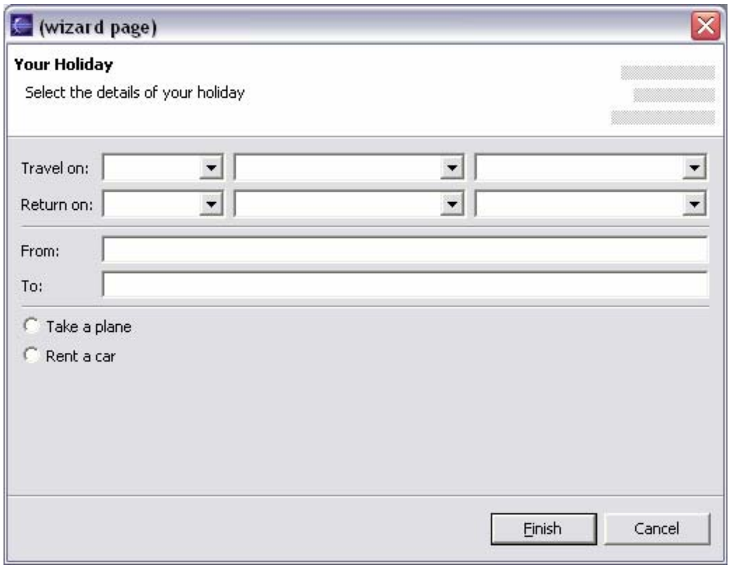
The wizard page's title bar and button bar is the same for all wizard pages in a wizard so you are unable to modify them in the SWT Designer.

If you select the entire wizard page, three properties are displayed in the Inspector. All three properties change the top section of the wizard page. The description property allows you to change the page's description, the imageDescriptor property allows you to set an image to the top-right corner of the page, and the title property allows you to change the page's title.



description	Wizard Page description
imageDescriptor	
title	Wizard Page title

There is a container in the center of the wizard page in which you are able to add controls. This is the unique part of each wizard page. You can add regular SWT controls just like you would when designing a regular SWT Application. It is easy to create a wizard page like the ones in the tutorial from the above link. Shown below is the wizard page, HolidayMainPage, which was created using the SWT Designer.



(wizard page)

Your Holiday

Select the details of your holiday

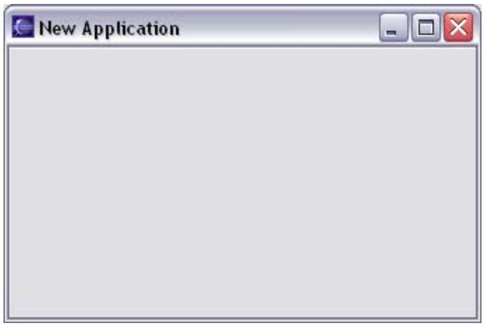
If you look at the source code generated by the SWT Designer, you will see that all of the code for the controls that you placed in the container are in the method createControl() and the code for the top section including the wizard page title and description are in the wizard page's constructor.

After you have created all your wizard pages, you can put them together and add functionality to them. To learn how to do this, look at the tutorial on JFace Wizards by following the link given above.

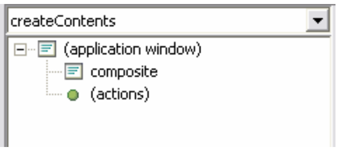
JFace Application Creation (Professional Version Only):

A JFace application is very similar to an SWT application. Both allow you to place SWT controls on the shell, both allow you to have a menu, and both allow you to have a toolbar. However, creating menus and toolbars for a JFace application is quite different than creating menus and toolbars for an SWT application. A JFace application also gives you the option of having a status bar at the bottom of your window.

To create a JFace application, select File > New > Other. Select Designer on the left side of the page and JFace ApplicationWindow on the right side and then click Next. Type in the desired name for your class and click Finish. The Content Pane should now display what looks like a shell for a normal SWT application.



The Control Tree, Method List, and the Inspector, however, contain different items for a JFace application than for an SWT application. The Control Tree's root is an item called "(application window)". It is the equivalent of the shell in an SWT application except you cannot place controls directly on it. The (application window) item has two children, "composite" and "(actions)". You place your controls on the composite which covers the entire area of window. The (actions) item has children which are all the actions for the application. We will look at what actions are further down. The Method List shows us that all the controls are created in the createContents() method.

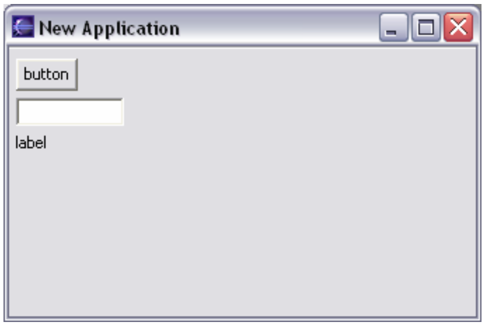


When the (application window) item is selected, the Inspector only shows the properties "size" and "title". The title property allows you to change the text that is displayed on the title bar of the application window and the size property allows you to change the size of the window.

If you look at the source code of your JFace application, you will see that several methods have been created. The more noteworthy ones are createContents(), createActions(), createMenuManager(), createToolBarManager(), configureShell(), and run().

The createContents() method is where all of the controls are created. The createActions(), createMenuManager(), and createToolBarManager() methods are where the actions, menus, and toolbar are created. The configureShell() method is where the window size and the text in the title bar is set. The run() method opens up the application window. It calls the open() method, retrieves the display and shell, and starts the event loop.

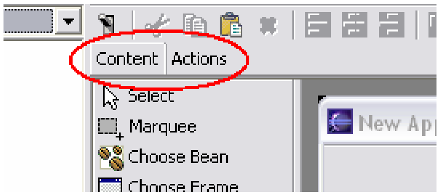
You can place controls on the window the same way you place controls on an SWT window. The only difference is that you are placing the controls on a composite instead of directly onto the shell.



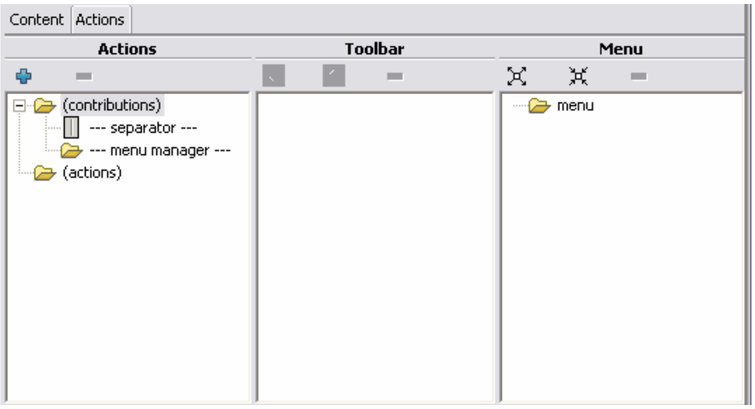
Placing a menu or a toolbar in an SWT application requires you to add menu and toolbar controls to your shell. The items in your menus and toolbars are independent of each other. If you want to have the same function on both a menu and a toolbar, you have to create two separate items with two separate event handlers. In a JFace application, you combine these separate items that have the same function into one item called an action.

An action is a command that is activated by the user when either a menu item or a toolbar item is selected. It allows the same behavior to be used by both a menu item and a toolbar item, reducing redundant code. For example, in Eclipse you can save a file by either selecting Save from the File menu or by clicking on the Save button in the toolbar. Selecting either of these items will execute the same action.

While editing a JFace application in the SWT Designer, you may notice two tabs just above the Control Palette named Content and Actions. Clicking on the Actions tab will turn the Control Palette and the Content Pane into the Actions Pane. Clicking on the Content tab will return you to the Control Palette and the Content Pane.

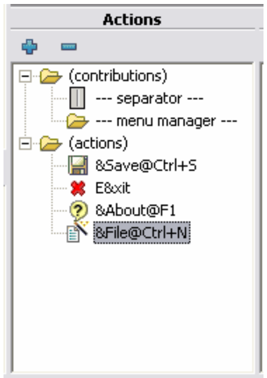


If you click on the Actions tab, you should see the Actions Pane shown below.

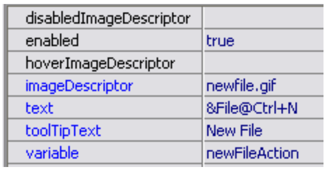


The Actions Pane is split up into three sections: Actions, Toolbar, and Menu. All three sections have a couple of buttons at the top with a tree below.

There are already several items in the actions tree. Listed under "(contributions)" are two items, a separator and a menu manager. The separator can be dragged to the toolbar and menu trees. This creates a visible separator between items in the toolbar and menu. You can place as many separators as you want. The menu manager can only be dragged to the menu tree. Each menu manager is a submenu of the menu it was placed in. You can place as many menu managers as you want. Listed under "(actions)" in the actions tree are all the created actions. You can add and remove actions by clicking on the plus (+) and minus (-) buttons above the actions tree. Below is shown several actions added.



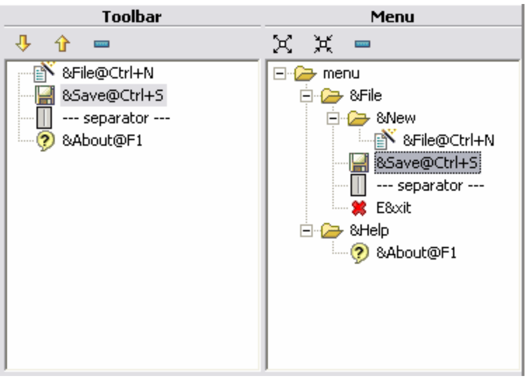
If you select an action, the following is displayed in the Inspector:



disabledImageDescriptor	
enabled	true
hoverImageDescriptor	
imageDescriptor	newfile.gif
text	&File@Ctrl+N
toolTipText	New File
variable	newFileAction

You can set the image for the action that is displayed in the toolbar and menu with the imageDescriptor property. You can also set the disabledImageDescriptor and hoverImageDescriptor properties if you want separate images to be displayed when the action is disabled or when the mouse is moved over the toolbar item. The text property sets the text beside the menu item. The '&' character in front of a letter indicates that the letter is a mnemonic. Anything after the '@' character is the accelerator key. The toolTipText property sets the tool tip that will be displayed if you move your mouse over a toolbar item.

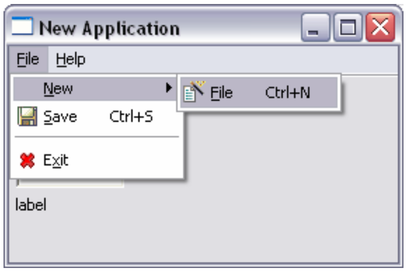
To add actions to the toolbar or a menu, simply click on the action in the actions tree and drag it to the desired position in the toolbar or menu tree. Below, several actions have been added to the toolbar and menu.

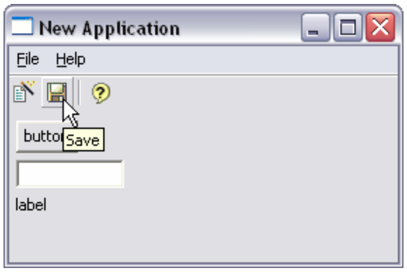


To change the placement of actions, simply drag them to their desired position. In the toolbar section you can also use the up and down buttons above the toolbar tree. To remove actions from the trees, select an action and press the minus (-) button above the tree.

Double-clicking an action in the actions tree will switch to the source view with your cursor in the action's run() method. The run() method is where you place the event-handling code. An action's run() method is executed every time it is selected from either the menu or the toolbar in a running program.

Shown below are the effects of adding actions to the menu and the toolbar.



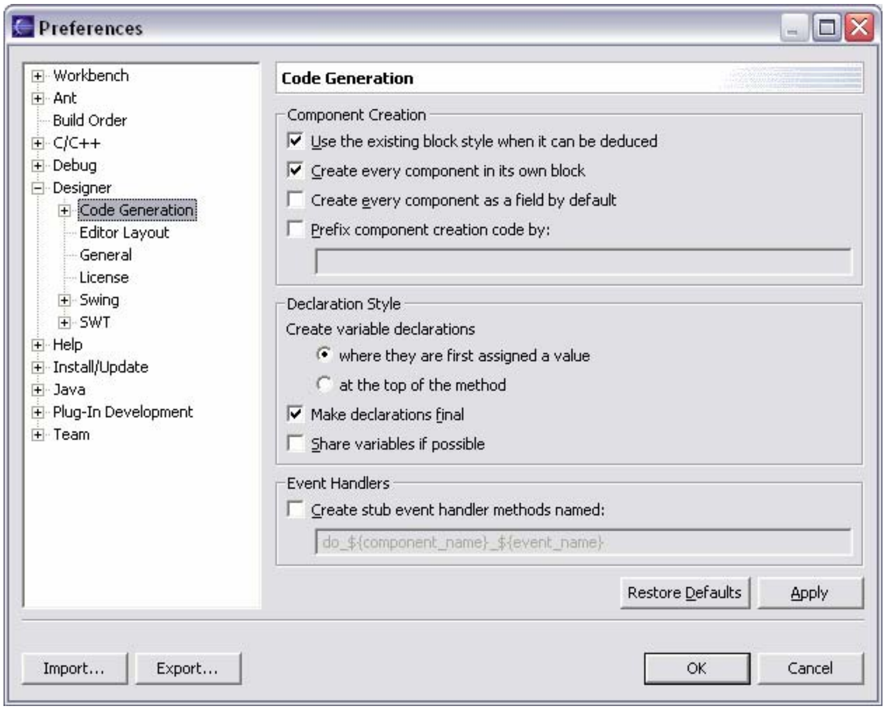


Whether you select the Save item from the menu or the Save item from the toolbar, the same run() method will execute since both items are linked to the same action.

For more information on creating JFace applications, see the *JFace* tutorial.

Code Generation Options:

The Designer supports generating code in numerous ways. Some code generation settings that you can modify are the style of code that is generated, the default variable names of controls, and where variables for controls are declared. To modify the code generation settings, select Preferences from the Windows menu, expand the Designer item and then select Code Generation. The Code Generation preference page for the professional version of the Designer is shown below. The free version of the Designer only includes the "Use the existing block style when it can be deduced" and "Create every component in its own block" settings.



Code Generation

Component Creation

☑ Use the existing block style when it can be deduced

☑ Create every component in its own block

☐ Create every component as a field by default

☐ Prefix component creation code by:

Declaration Style

Create variable declarations

● where they are first assigned a value

○ at the top of the method

☑ Make declarations final

☐ Share variables if possible

Event Handlers

☐ Create stub event handler methods named:

do_${component_name}_${event_name}

When "Create every component in its own block" is selected, the Designer generates code using the block style: a control is declared and instantiated inside its own block of code. All of the control's properties are placed in the block. The block is surrounded by opening and closing braces. Child controls are nested in their parent's blocks. When a control is declared inside its block of code, it is visible only to its children and itself. A control can be made visible to the rest of the class using the "Convert local to field" button. In the examples at the end of this tutorial we will see the block style in more detail. Below is a simple example of what the block style looks like.

```
{
        final Button button = new Button(shell, SWT.NONE);
        button.setText("button");
}
```

When "Create every component in its own block" is not selected, the Designer generates code using the flat style: all controls are created in the same scope in a method making the controls visible anywhere in a method after they have been declared.

When "Use the existing block style when it can be deduced" is selected, the Designer attempts to use the existing code style already in a file to generate code. The Designer checks a file when it is opened in the editor to see if one style or the other is used. If only one of the styles is being used in a file, it will continue generating code in that style, regardless of what the "Create every component in its own block" setting is set to. If no style is detected (no controls have been created yet) or if more than one style is detected, the Designer will generate code as set in the "Create every component in its own block" setting.

When "Create every component as a field by default" is selected, all created controls are automatically created as a field in the class.

When "Prefix component creation code by" is selected, the text field below it is enabled. If the text field is left blank, every new control created will have a blank line above it in the source code. You can also place a custom one-line comment before each new control by putting the comment in the text field. The text field must be blank, start with "//", or start with "/*" and end with "*/". If the text field is not blank and does not hold a comment, new controls will not be created.

By selecting one of the radio buttons under "Create variable declarations", the Designer will place declarations of variables either on the same line as where they are first assigned a value or at the top of the method.

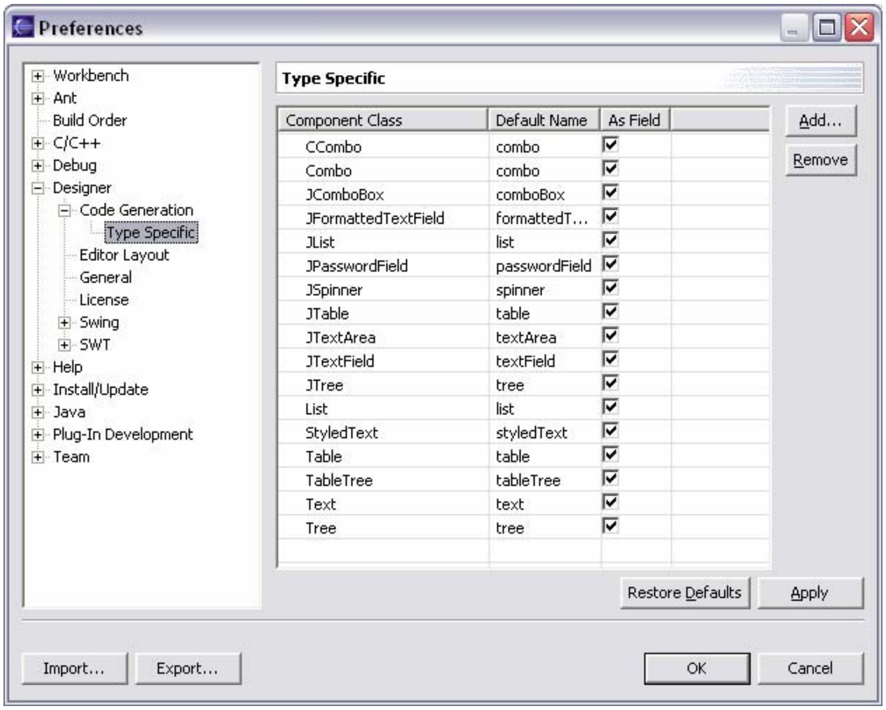
The "Make declarations final" setting makes all declarations final by placing the "final" keyword before each declaration.

The "Share variables if possible" setting makes the Designer reuse variable names like gridData, formData, label, button, etc. The "Make declarations final" and "Share variables if possible" settings cannot both be selected at the same time.

When “Create stub event handler methods named” is selected, the text field below it is enabled. Now whenever an event handler is added, a method stub is created with the name given in the text field. A line of code is also added to the event handler which calls the newly created method. If you move your mouse over the text field, a tool tip will appear which gives you information on how to specify the names of the method stubs generated.

Type Specific Code Generation Options (Professional Version Only):

If you expand the Code Generation item and select the Type Specific item in the Preferences window, the Type Specific Code Generation preference page is shown. In this page you can set code generation settings for individual types.



Component Class	Default Name	As Field
CCombo	combo	☑
Combo	combo	☑
JComboBox	comboBox	☑
JFormattedTextField	formattedT...	☑
JList	list	☑
JPasswordField	passwordField	☑
JSpinner	spinner	☑
JTable	table	☑
JTextArea	textArea	☑
JTextField	textField	☑
JTree	tree	☑
List	list	☑
StyledText	styledText	☑
Table	table	☑
TableTree	tableTree	☑
Text	text	☑
Tree	tree	☑

There are already several types listed in the table. You can add more types by clicking on the Add button and entering in the name of the class you want to add in the dialog box that appears. To remove a type, highlight it in the table and click on Remove.

The “Default Name” column of the table lets you change what variable name is initially used when a control of that type is created. To change the default name, simply click in the table cell that you wish to modify and type in the name. Press Enter to apply the change or Esc to cancel.

The “As Field” column has a checkbox for each table item. If an item’s “As Field” cell is checked, all new controls of that type will be created as a field. When a control is created as a field, it places the declaration of the control in the class outside of any methods.

Additional Features of the Designer:

There is a quick way to edit the text property of buttons (including check and radio buttons), labels, text fields, groups, and table columns. First select a control in either the Content Pane or the Control Tree. Hold down the Alt key and then click on the control in the Content Pane. An edit box will appear in which you can set the control's text property. To use this feature, you must enable the "Allow direct edit" preference. To enable it, select Window > Preferences. On the left side of the Preferences window, expand Designer and click on General. Make sure that "Allow direct edit" is checked on the right pane of the window and click OK.

The Designer allows you to cut, copy, and paste controls. To cut or copy a control, simply select a control and select cut or copy from the toolbar above the Content Pane or from the Edit menu. Alternatively, you can also right-click on a control in either the Content Pane or the Control Tree and select cut or copy from the resulting menu. To paste a control, select paste from the toolbar above the Content Pane, from the Edit menu, from the Control Palette, or from the right-click menu of the Content Pane or the Control Tree, and then place the control in the desired position in either the Content Pane or in the Control Tree. The control that was pasted will then have most of the same properties as the original control. The events and the variable name properties are not copied.

The Designer includes a very handy feature allowing you to view how your Application will look without compiling and running the entire application. There are a couple of ways to run this "test". The first is to click on the Test button on the toolbar above the Content Pane. There is also a Test button on the main Eclipse toolbar when a file is open in the Designer editor. The other way is to right-click anywhere in the Content Pane or in the Control Tree and select Test from the popup menu. Running the test will bring up a window of your application. None of your own code will run, just the code that is generated by the Designer. The main purpose of the test is to see what the controls look like together, and you can test what happens when you resize and move the application window. You can close the test window by click on the 'X' in the top-right corner of the window or by clicking a mouse button on a blank area of the shell.

If your shell window is larger than your content pane, you cannot see the whole shell at once. To allow you to see a small version of what the whole shell looks, the Designer has a thumbnail view. To enable this view, select Window > Preferences, expand Designer and select General. In the preference page on the right select "Show thumbnail in Outline view". Now a thumbnail view of your shell is visible in the Outline view (you will have to close and reopen any currently open files to see this). If your shell is larger than your content pane, a gray box appears outlining which part of the shell is currently visible. You can click and drag the box around to display different areas.

A number of controls, such as buttons, shells and menu items have an image property. Setting this property is the same for all the controls with this property. To set this property, select the control, click on the image property in the Inspector, click on the button that appears in the property editor, and choose an image from the list of images in the window that appears. There are two options in the Select Image window. If the “Relative to class” option is selected, only images that are in the same package or a subfolder of the package that the class is in are shown. If the “Relative to project” option is selected, images in the project folder or any of the project’s subfolders are shown.

A shell has several style properties that can be set. To change a shell’s style, select the shell and expand the style property in the Inspector. By changing the style properties such as border, min, and resize, you can change the look of the shell. The trim property has several options. When default is selected and all the other style settings are their defaults, a regular shell is displayed. If any of the other style properties are set to true when the trim property is set to default, only the style properties set to true are set. The shell_trim value is the same as setting close, title, min, max, and resize to true. The dialog_trim value is the same as setting title, close, and border to true. With the no_trim value set, none of the other styles will be able to be shown except title.

If you want to create several controls of the same type at once, it can be cumbersome to select the control from the Control Palette each time. If you hold down Ctrl as you are selecting a control from the Control Palette, you can place several controls at once. You can keep clicking on the Content Pane and new controls will be added. To stop adding controls of that type, either click on an open spot in the Control Tree or select another control from the Content Pane.

The Designer fully supports multiple level undo and redo. If you wish to undo or redo an operation performed by the Designer such as accidentally deleting a control, you can just choose undo or redo from the Edit menu of Eclipse. Since the Designer supports multiple levels of undo and redo, you can undo and redo multiple operations.

The z-order is a hierarchy of controls. It affects some things in SWT such as which controls can be attached to other controls in the FormLayout, the tab order of the controls, and the location of a control on the FillLayout, RowLayout, and GridLayout. The z-order is shown in the Control Tree and in the source code. A control that is higher up in the Control Tree is higher up in the z-order. There are a few ways to modify the z-order in the Designer. The easiest way is to drag the controls around in the Control Tree. If the FillLayout, RowLayout, or GridLayout is used, you can also drag the controls around on the Content Pane to modify the z-order. You can also manually move the blocks of code around in the source code, but this is not recommended because there are easier ways to modify the z-order. When the z-order is modified, the changes are reflected in the Control Tree, the Content Pane, as well as the source code.

In the professional version of the Designer, you can modify properties on multiple controls at once. There are several ways to select multiple controls in the Designer. If you hold down Ctrl, you can click on multiple controls in the Control Tree or the Content Pane. In the Content Pane, you can hold down Shift to select individual controls. If a control is selected and you hold down Shift and then click on a control in the Control Tree, all the controls between the two controls in the Control Tree will be selected, inclusively. You can also use the Marquee tool, which is at the top of the Control Palette to select multiple controls. When the Marquee tool is selected, the cursor will turn into crosshairs and you can click and drag a box around multiple controls on the Content Pane to select them. To return to the regular cursor, just click on the Select tool at the top of the Control Palette. When multiple controls are selected, all of their shared properties are listed in the Inspector. If not all the selected controls have the same value for a property, the property editor in the Inspector will be blank. You can also move and resize multiple controls when using the NullLayout or FormLayout.

Creating a UI: Two Examples

We are going to go through a couple of examples to demonstrate how to create a GUI using the Designer. In the first example we will create a GUI for a simple Address Book application. We will be designing the first example in SWT only using features from the free version of the Designer. We will also create a GUI that is equivalent to an already existing GUI for a Client Billing Application from a previous tutorial. We will be designing the second example in both SWT and Swing, using the professional version of the Designer.

It will be beneficial to go over both examples regardless of whether you are using the free or professional version of the Designer. If you are going over the second example while using the free version, you may have to use some slightly different methods to achieve the same results, but they are not hard to do.

Creating a GUI for an Address Book Application:

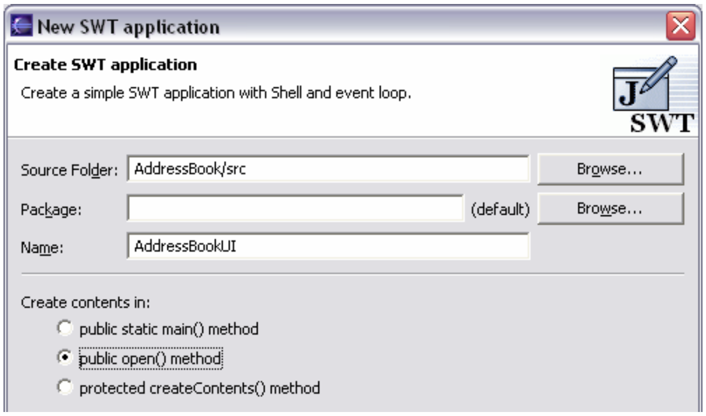
We are going to go through an example of how to create a GUI for an Address Book application. Because this example only shows how to build a GUI, the example application will have no real functionality.

Creating a new Project:

Select File > New > Project, and then select Designer > SWT/JFace Java Project. Set the project name to AddressBook.

Creating a new SWT Application:

Select File > New > Other, then click on Designer > SWT Application and hit Next. Set the name to AddressBookUI, select the “public open() method” radio button under “Create contents in:” and then click Finish.



New SWT application

Create SWT application

Create a simple SWT application with Shell and event loop.

Source Folder: AddressBook/src

Package: (default)

Name: AddressBookUI

Create contents in:

○ public static main() method
● public open() method
○ protected createContents() method

Your generated class' source code should look like the following:

```
public class AddressBookUI {

        public static void main(String[] args) {
                AddressBookUI window = new AddressBookUI();
                window.open();
        }
        public void open() {
                final Display display = new Display();
                final Shell shell = new Shell();
                shell.setText("SWT Application");
                shell.open();
                while (!shell.isDisposed()) {
                        if (!display.readAndDispatch())
                                display.sleep();
                }
        }
}
```

As you can see above, the Designer creates two methods. The open() method contains all your GUI code, and your main() method calls the open() method. In a real application, you may have several classes, and you may want to have your main() method in another class. We will leave it in this class in order to make it easy to run our GUI.

The Designer automatically creates instances of Display and Shell. It sets the name displayed in the title bar of the shell and then opens the shell. The Designer also creates an event loop.

Since the Designer initially generates all the code needed to create a window, we can run the application and see a blank window. While the AddressBookUI.java file is in focus in the editor, select Run > Run As > Java Application. A blank window should appear. Close it by clicking on the X on the top-right corner.

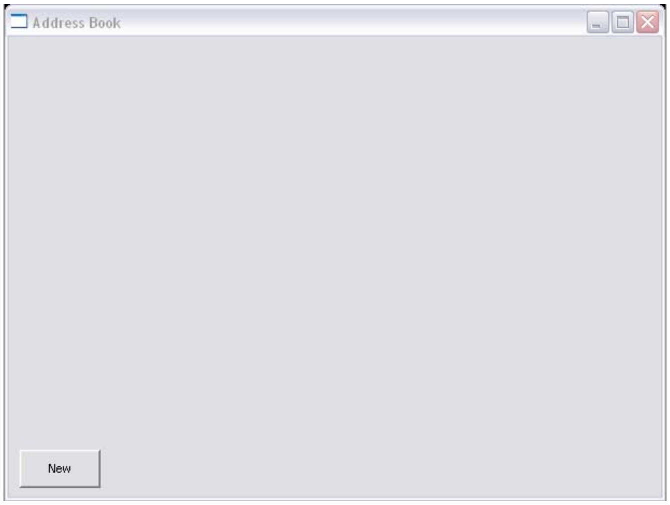
Designing the GUI Window:

Expand the editor window by double-clicking on the tab above the editor that shows AddressBookUI.java. Click on the design tab on the bottom of the screen to enter the Designer's design view.

We will be using a NullLayout for the shell, so make sure that you have enabled it by following the instructions that can be found in the NullLayout section of this tutorial. In a real application, it would be best to use layout managers in order to make the application more portable, but for this simple example a NullLayout will be sufficient.

First, we will change the title from “SWT Application” to “Address Book”. Select shell in the Control Tree and change the text property in the Inspector to “Address Book”.

Next we will create the buttons on the bottom of the window. Select the Button control from the Control Palette. Now on the bottom left corner of the window in the Content Pane, draw out a button. In the Inspector, set the text property to be “New” and set the variable property to be “newButton”. Your window in the Content Pane should look something like the following:



Now we create two more buttons called “deleteButton” and “editButton” with the text as “Delete” and “Edit”, respectively.

Now make sure that a button is selected. Click on the “Open definition” button.

You should see the following lines added to your code:

```
{
        final Button newButton = new Button(shell, SWT.NONE);
        newButton.setBounds(10, 380, 75, 35);
        newButton.setText("New");
}
{
        final Button deleteButton = new Button(shell, SWT.NONE);
        deleteButton.setBounds(85, 380, 75, 35);
        deleteButton.setText("Delete");
}
{
        final Button editButton = new Button(shell, SWT.NONE);
        editButton.setBounds(160, 380, 75, 35);
        editButton.setText("Edit");
}
```

You should notice the opening and closing braces on the top and the bottom of the code for each button. Within these braces is a block of code that creates and configures the button. Go back to the design view and select the "New" button. Then click on the "Convert local to field" button. The "Convert local to field" button's image changes and its tooltip changes to "Convert field to local".

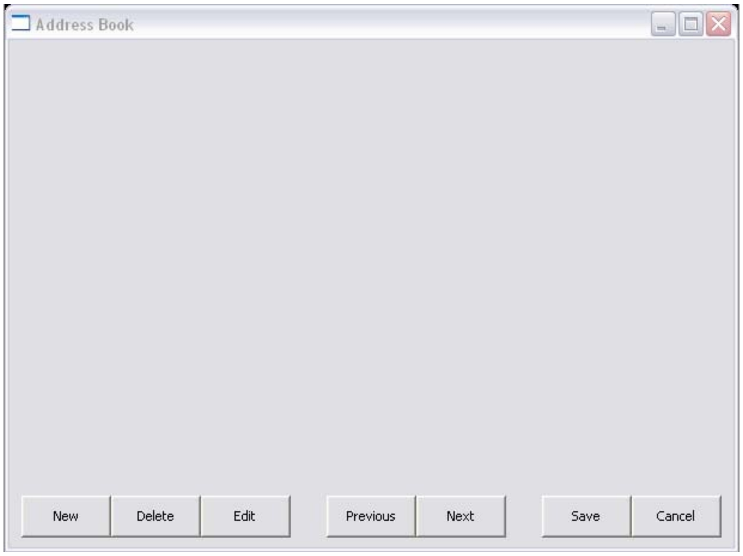
Switch back to the source view and we will see what occurred by clicking on this button. Look at the top of the class, just before the main() method. You should see a new line added:

```
        private Button newButton;
```

Also, "`final Button`" is removed from the beginning of the button instantiation in its block. Declaring the button outside the block and as a field in the AddressBookUI class allows you see and modify its properties anywhere in the AddressBookUI class.

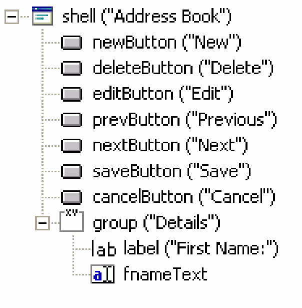
All that you have to do to convert a control to a field of the class is to click on the "Convert local to field" button. To change the control back to local, just click the button again.

Now convert the other two buttons into fields by clicking the "Convert local to field" button for each of them. We will now finish creating the other buttons. Name these buttons "prevButton", "nextButton", "saveButton", and "cancelButton". Set their text to be "Previous", "Next", "Save", and "Cancel", respectively. To quickly set the text on the buttons, select the button, hold down Alt and click on the button in the Content Pane. Make sure you enable the Allow Direct Edit setting in the Designer's properties. Your window should now look something like this:



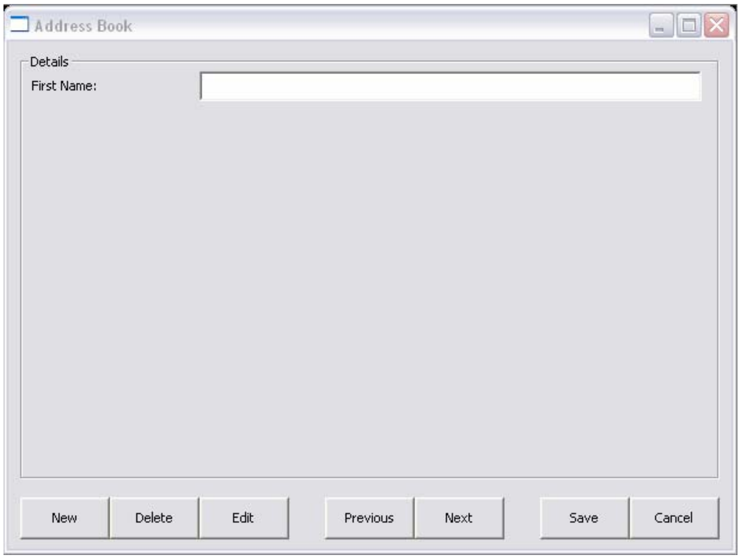
Now create a group in the top part of the shell by selecting Group from the Control Palette, click and hold down the mouse button on the top-left corner of the shell, and then drag the group across the rest of the shell. Set the text of the group to "Details".

We now create a label and a text field inside our newly created group. Make sure that the label and the text field are children of group in the Control Tree. If they are, they will be listed under the group and indented a little bit.



```
shell ("Address Book")
    newButton ("New")
    deleteButton ("Delete")
    editButton ("Edit")
    prevButton ("Previous")
    nextButton ("Next")
    saveButton ("Save")
    cancelButton ("Cancel")
    group ("Details")
        label ("First Name:")
        fnameText
```

Set the label's text to "First Name:" and the text's variable name to "fnameText". Because we will never have to set properties of the label in our code, we will just leave the label as local and keep label's variable name as label. Select fnameText and click on the "Convert local to field" button. Your window should now look something like this:

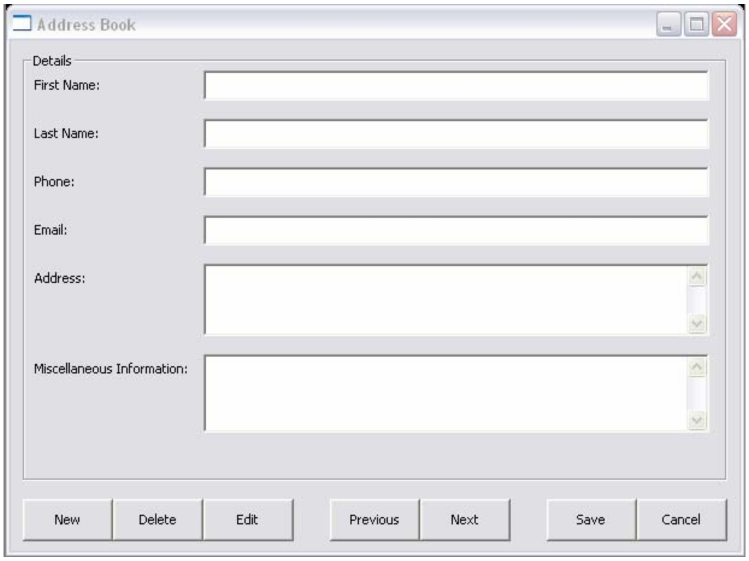


Click on group in the Control tree and then click on the "Open definition" button. You should see the following code added to your class:

```
{
        final Group group = new Group(shell, SWT.NONE);
        group.setText("Details");
        group.setBounds(10, 10, 585, 355);
        {
                final Label label = new Label(group, SWT.NONE);
                label.setBounds(10, 20, 135, 25);
                label.setText("First Name:");
        }
        {
                fnameText = new Text(group, SWT.BORDER);
                fnameText.setBounds(150, 15, 420, 25);
        }
}
```

You can see that the group is in a block just like the buttons are. You may notice that the label and the text field are both nested in the group block. This is because they are both children of group. The Designer nests all control blocks inside their parent container's block. The declaration for fnameText is higher up in the source code with the other field declarations. Now return to the design view.

Create five more labels and five more text fields. Set the text in the labels to "Last Name:", "Phone:", "Email:", "Address:", and "Miscellaneous Information". Set the variable names of the text fields to "lnameText", "phoneText", "emailText", "addrText", and "miscText". Under the style complex property of both addrText and miscText, set the v_scroll property to true. This adds vertical scrollbars to the text fields. For all the text fields, make sure that each is a field by clicking on the Convert local to field button. Now your window should look something like this:



Use the Designer's test feature to see what your window will actually look like by clicking on the Test toolbar button. Close the window by clicking on the X in the top-right corner of the window.

Save and run the application. The window we get when we run the application is probably a different size than the window we got when we used the Designer's test feature.

To make the window the size that we specify, right-click on the shell in the Control Tree or Content Pane, and then select Set shell size > Using the setSize() method. Now make sure you resize the shell window on the Content Pane to the desired size. Select the shell and then click on "Open definition". You should see a line similar to the following added:

```
        shell.setSize(610, 477);
```

Now try compiling and running your application again. You will notice that your application is now the same size as the shell in the Content Pane.

Note: The Designer will add all the appropriate imports into your code that are necessary for your GUI to work. These are the imports that the Designer has put into our application so far:

```
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Group;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
```

Creating Menus:

Though the professional version of the Designer supports building menus, the free version does not. If we wish to design a menu using the free version of the Designer, we will have to code it ourselves.

In order to create the menu, we will create a method in AddressBookUI called setupMenu(). Since we create the menu on the shell, shell will have to be visible to this method. In the design view, select shell from the Control Tree and click on Convert local to field. The remainder of the creation of the menu will be done in the source view. Add the following line to your list of imports:

```
import org.eclipse.swt.widgets.*;
```

Then add the following method to your AddressBookUI class:

```
private void setupMenu() {
        //create the menu bar
        Menu menu = new Menu(shell, SWT.BAR);
        shell.setMenuBar(menu);

        //add the File option to it
        MenuItem file = new MenuItem(menu, SWT.CASCADE);
        file.setText("File");

        //create a menu for the File option
        Menu filemenu = new Menu(file);
        file.setMenu(filemenu);

        //add MenuItems to the File menu
        MenuItem prevItem = new MenuItem(filemenu, SWT.NONE);
        prevItem.setText("Previous");

        MenuItem nextItem = new MenuItem(filemenu, SWT.PUSH);
        nextItem.setText("Next");

        MenuItem seperator = new MenuItem(filemenu, SWT.SEPARATOR);

        MenuItem quitItem = new MenuItem(filemenu, SWT.PUSH);
        quitItem.setText("Quit");

        //add listeners for the actions
        prevItem.addListener(SWT.Selection, new Listener() {
                public void handleEvent(Event e) {
                        System.out.println("Previous menu item selected.");
                }
        });
```

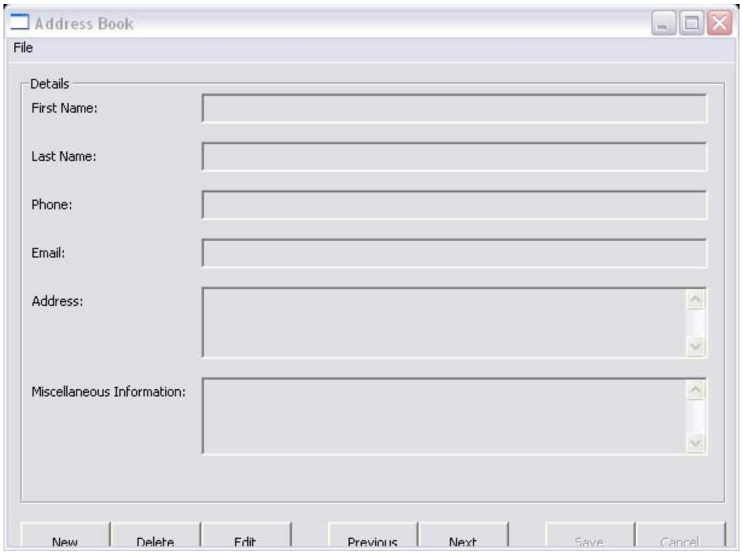
```
                nextItem.addListener(SWT.Selection, new Listener() {
                        public void handleEvent(Event e) {
                                System.out.println("Next menu item selected.");
                        }
                });

                quitItem.addListener(SWT.Selection, new Listener() {
                        public void handleEvent(Event e) {
                                shell.dispose();
                        }
                });
        }
```

Here, as you can see, we do not use the block style. All controls declared in this method are visible throughout the entire method. This style of coding is called flat style.

Now, just before the shell.open() call in your open() method, place a call to the new setupMenu() method.

```
        setupMenu();
        shell.open();
        while (!shell.isDisposed()) {
                if (!display.readAndDispatch())
                        display.sleep();
        }
```

If you return to the design view, you will see the menu on the Content Pane. You can click on the individual menu components and change some of their settings, but you cannot create new menu items. Because the menu takes up some space on the shell, the buttons on the bottom of the shell may have part of their bottoms cut off as shown below. Simply resize the shell to fix this.



Now try compiling and running your application. Make sure you can see the console on the bottom of your screen. Then try selecting Previous and Next from the File menu. Messages should be sent to the console when you select one of the menu items. Quit will close the application.

For more information on creating menus, see the tutorial *Advanced SWT Widgets*.

Creating Event Handlers:

We are now almost done creating our GUI. All that is left to do is to create some functionality. This primarily involves setting up event handlers for the buttons. The Designer will set up the proper framework for the event handlers, but we will have to code what we want the application to do when an event is triggered.

We will now go over what sort of actions and events we want to take place in our application.

First, we want all the text fields to initially be non-editable. They will only be editable if we select either the New or the Edit button. The Save and Cancel buttons will initially be disabled.

If the New button is selected, all the text fields will be cleared and made editable, the New, Delete, and Edit buttons will be disabled, and the Save and Cancel buttons will be enabled. The same will happen with the Edit button, except the text fields will not be cleared.

While editing, if either the Save or the Cancel button is selected, the text fields will return to being non-editable, the New, Delete, and Edit buttons will be re-enabled, and the Save and Cancel buttons will be disabled.

If the Delete button is selected, all the text fields will be cleared of their text.

For all the buttons, a message will appear in the console saying which button was pressed.

First of all we will set up the initial window state. To initially make the text fields be non-editable, you can set their style to be read only. Select each text field and under the style complex property in the Inspector, set the read_only property to true. Disabling the Save and Cancel buttons is also easy. Select each button, and then in the Inspector, set the enabled property to false.

Instead of placing all the lines of code that modify the properties of the controls inside the event handlers, we will create methods for each action. This is because an action may take place on several different occasions. This also makes our code more readable. We will create four new methods called clearText(), setTextEditable(), enableEditButtons(), and enableSaveButtons(). They are as follows:

```
private void clearText() {
        fnameText.setText("");
        lnameText.setText("");
        phoneText.setText("");
        emailText.setText("");
        addrText.setText("");
        miscText.setText("");
}
```

```
private void setTextEditable(boolean editable) {
        fnameText.setEditable(editable);
        lnameText.setEditable(editable);
        phoneText.setEditable(editable);
        emailText.setEditable(editable);
        addrText.setEditable(editable);
        miscText.setEditable(editable);
}

private void enableEditButtons(boolean enable) {
        newButton.setEnabled(enable);
        deleteButton.setEnabled(enable);
        editButton.setEnabled(enable);
}

private void enableSaveButtons(boolean enable) {
        saveButton.setEnabled(enable);
        cancelButton.setEnabled(enable);
}
```

Now double-click the New button. The view should be switched to your source code and a stub for an event handler should have been created. Make your code look like this:

```
{
        newButton = new Button(shell, SWT.NONE);
        newButton.addSelectionListener(new SelectionAdapter() {
                public void widgetSelected(SelectionEvent e) {
                        clearText();
                        setTextEditable(true);
                        enableEditButtons(false);
                        enableSaveButtons(true);

                        System.out.println("New button selected.");
                }
        });
        newButton.setBounds(10, 380, 75, 35);
        newButton.setText("New");
}
```

You can see that when the New button is clicked, the text fields are cleared and made editable, the edit buttons (New, Delete, Edit) are disabled, the save buttons (Save, Cancel) are enabled, and a message is sent to the console saying that the New button was selected.

In the same way, add the following code to each of the buttons.

Delete button:

```
clearText();

System.out.println("Delete button selected.");
```

Edit button:

```
setTextEditable(true);
enableEditButtons(false);
enableSaveButtons(true);

System.out.println("Edit button selected.");
```

Previous button:

```
System.out.println("Previous button selected.");
```

Next button:

```
System.out.println("Next button selected.");
```

Save button:

```
        setTextEditable(false);
        enableEditButtons(true);
        enableSaveButtons(false);

        System.out.println("Save button selected.");
```

Cancel button:

```
        setTextEditable(false);
        enableEditButtons(true);
        enableSaveButtons(false);

        System.out.println("Cancel button selected.");
```

Run the application. Play around with it a little bit to see what happens because of the code that you added to your buttons. Look at the console to view the messages that are printed when the buttons are clicked.

We have now finished creating the GUI for a simple Address Book application. Though this application has no real practical use in its current form, a database could be added to make the application a fully functional address book. For information on how you would add functionality to a GUI by adding a database, see the tutorials *Client Billing Application* and *Time-Tracker Application*.

For more information on events and event-handling, see the tutorials *Basic SWT Widgets* and *Advanced SWT Widgets*.

Creating an SWT GUI for a Client Billing Application:

In order to help you learn the Designer better and to see some differences between the free and professional versions of the Designer, we will look at a Client Billing application that is taken from the *Client Billing Application* tutorial. We will create an equivalent GUI using the Designer.

In this example we will be using the professional version of the Designer. You can still go through this example if you are using the free version of the Designer; however, you will have to use alternate means for some operations such as when we are designing using the FormLayout and when we design menus.

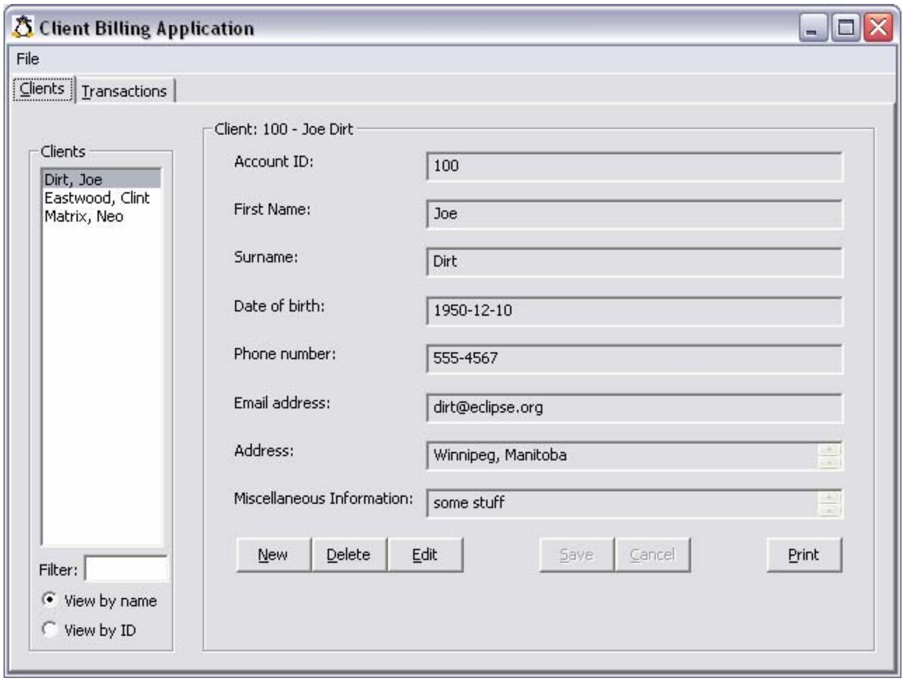
Note: This example is quite a bit more advanced than the previous example. Before going through this example it is suggested that you understand the various controls and layouts by looking at the *Basic SWT Widgets*, *Advanced SWT Widgets*, and *Eclipse Layouts* tutorials. You should also look at the *Client Billing Application* tutorial to better understand how the rest of the application fits together with the GUI.

If you do not already have the example source code for the *Client Billing Application* tutorial, get it now from http://www.cs.umanitoba.ca/~eclipse.

Taking a Look at the Original GUI:

The source code for the GUI of the Client Billing Application is all in ClientBillingUI.java. The main() method is in ClientBilling.java.

Open up ClientBilling.java and select Run > Run As > Java Application. The application should look like the following:



Client: 100 - Joe Dirt

Dirt, Joe
Eastwood, Clint
Matrix, Neo

Account ID: 100

First Name: Joe

Surname: Dirt

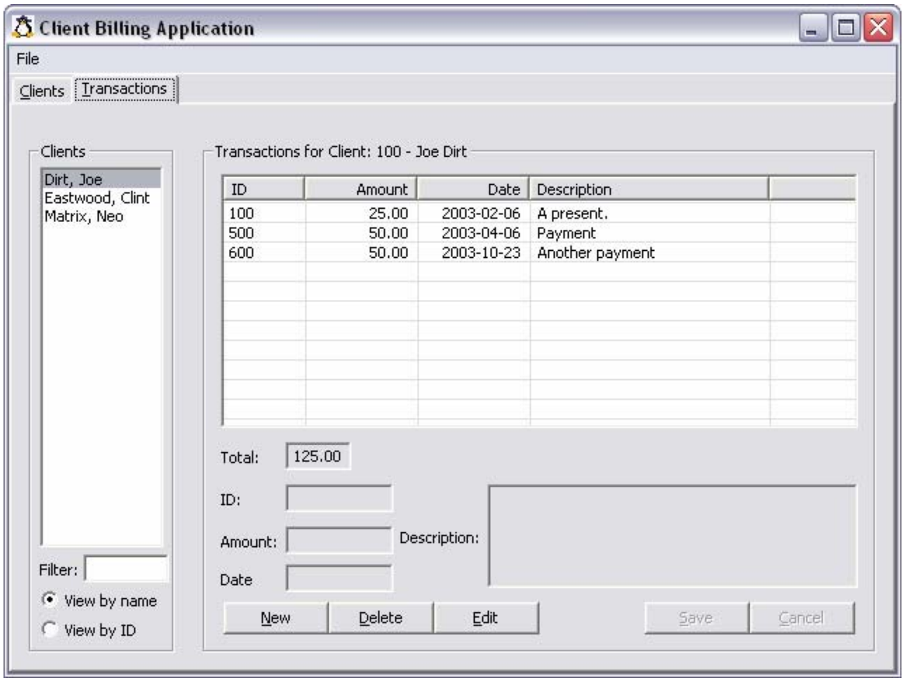
Date of birth: 1950-12-10

Phone number: 555-4567

Email address: dirt@eclipse.org

Address: Winnipeg, Manitoba

Miscellaneous Information: some stuff



Transactions for Client: 100 - Joe Dirt

Dirt, Joe
Eastwood, Clint
Matrix, Neo

ID	Amount	Date	Description
100	25.00	2003-02-06	A present.
500	50.00	2003-04-06	Payment
600	50.00	2003-10-23	Another payment

Total: 125.00

You may notice that the Clients group on the left is in both tabs and does not change. If you look at the source code, the group is created directly on the TabFolder, and not on the TabItems. This is not considered to be a standard practice and because of that the Designer does not support this. For the GUI that we will create in this tutorial we will place the Clients group outside of the TabFolder.

In this tutorial we will not fully integrate the GUI with the rest of the application, but in order to easily do so in the future, we will use the same variable names as the original GUI. You may wish to take a look over the ClientBillingUI class to understand how it works.

Creating Our Own ClientBillingUI:

Now we are ready to create our own GUI for this application. If you are adding to the existing ClientBilling project, be sure to add jface.jar to the project build path. Create a new SWT Application. Name the class ClientBillingUIDesigner and select the option to create the contents in the open() method. After the class has been created, switch to the design view.

In the Inspector, set the text property for shell to "Client Billing Application". Click in the editor of the image property in the Inspector and then click on the button that appears. Set the radio button on the top of the window to "Relative to Project". Select "splash.jpg" from the list of images and click OK. The icon on the top-left corner of the window now displays the image. Set the shell's layout to FormLayout.

If you are using the free version of the Designer, you are still able to use FormLayout. Since you cannot place a FormLayout using the free version, you must set the layout of a container to FormLayout by hand. To do this, use the setLayout() method of the container to set the layout to FormLayout right after the instantiation of the container. For example, if you wish to set the shell's layout to FormLayout, add the line

```
shell.setLayout(new FormLayout());
```

right after the line

```
final Shell shell = new Shell();
```

Then switch back to the design view to add controls to the shell. To change control's layout properties using the free version of the Designer, you will have to change them in the Inspector.

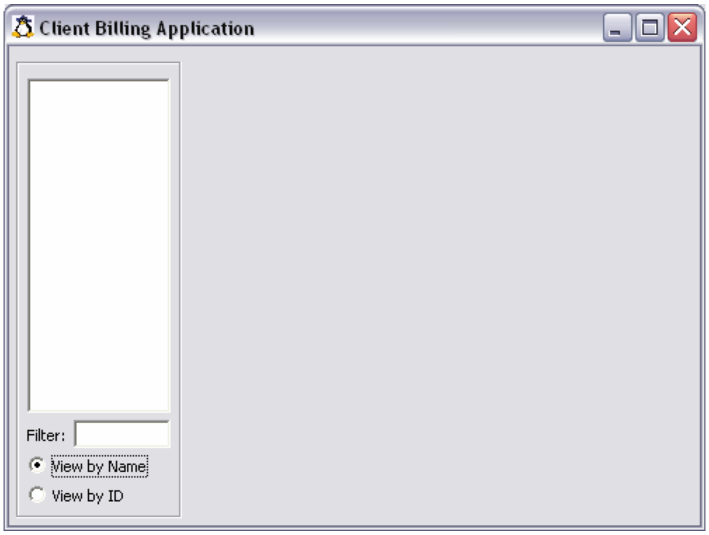
Creating the Client List Group:

Now we will create the group that contains the list. Add a group to the shell. Place your mouse in the top-left quadrant of the group so that the thick lines appear on the top-left corner of the group. Move the group to the top-left corner until the two blue margin lines appear. Drop it so that the top-left corner is close to the top-left corner of the shell. This creates snap points on the shell's margins. Now drag the bottom of the group to the bottom of the shell and snap it to the bottom margin. While the group is selected, click on the small button on the right side of the group. Click on the button with the percent symbol (%). Now drag the right handle of the group to 25%. The tool tip just below the shell tells you what percent you are at while you are dragging it.

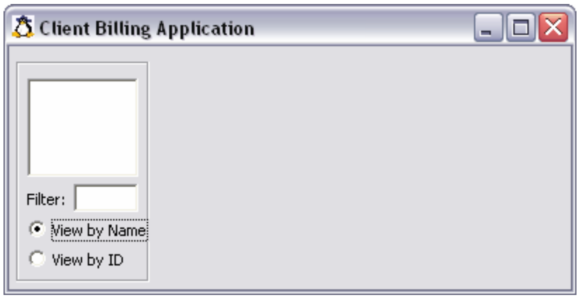
Set the group's text to "Clients". Set the group's layout to GridLayout with two columns. Since a container inherits its parent's layout, you will have to delete the FormLayout first. Simply click on the layout property in the Inspector and press Delete.

Add a list, a label, a text field, and two radio buttons. Name the list "clientList", convert it to a field, and set the horizontal span to 2. Set the label's text to "Filter:". The text field should be named "filterText". The two radio buttons should be named "nameRadio" and "numRadio", and their text should be "View by Name" and "View by ID", respectively. Set the horizontal span for both radio buttons to 2. Set the list to fill horizontally and vertically and to grab horizontally and vertically. Set the text field to fill horizontally.

Test your design by clicking on the Test button on the toolbar. Your shell should look something like this:



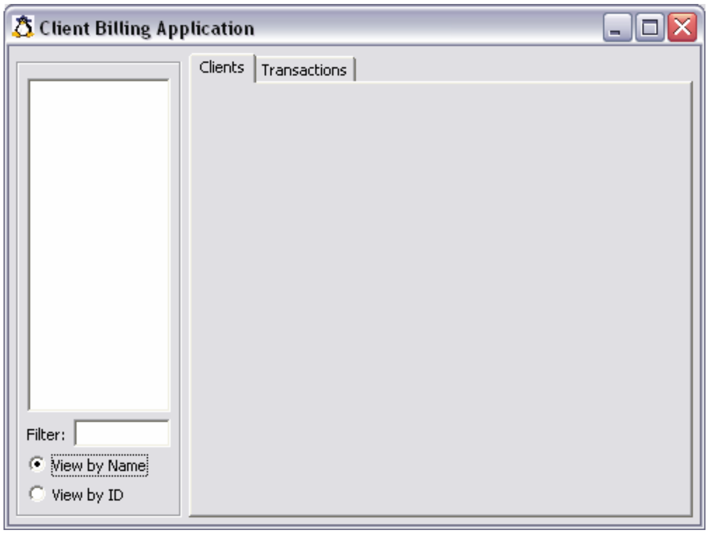
When layouts are used, resizing a window will also resize the controls inside of it. This allows the controls to be properly visible even at different window sizes as shown below.



Creating a Tabbed Folder:

Now create a TabFolder on the shell to the right of the group. Attach the top, right and bottom sides of the TabFolder to the margins like you did with the group. Drag the left side right beside the Clients group until a blue line appears. Now the TabFolder is snapped to the right side of the group. Again, test your shell to see the effects of the layout settings.

Now place two TabItems on the TabFolder. Set the text on them to be "Clients" and "Transactions". To switch between the tabs in the SWT Designer, simply click on the tab you wish to switch to. Your shell should now look like this:



A TabItem only allows one control to be placed directly on it. If you want to place more than one control on a TabItem, you should use either a composite or a group as the one control. Then you can place more controls on that group or composite. For this application we will place a group on both TabItems.

The Clients Tab:

The group in the Clients tab should be named clientInfo and should be converted to a field using the Convert local to field button. Set the group's layout to FormLayout. The group will hold the labels and the text fields and a composite. The composite will hold the row of buttons at the bottom and should have a horizontal FillLayout.

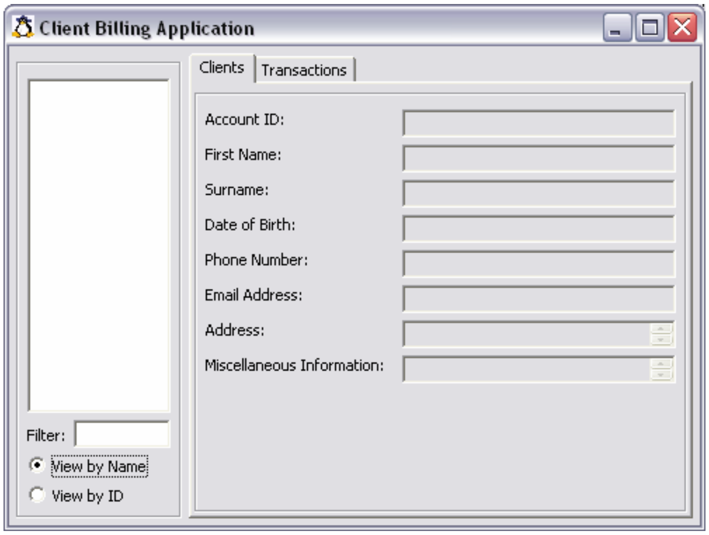
Create eight text fields and name them “acctIDText”, “fnameText”, “snameText”, “dobText”, “phoneText”, “emailText”, “addressText”, and “miscInfoText”. Use the convert local to field button to convert all of the text fields to fields.

The top text field should be snapped to the top and right margins of the group. The other text fields should have their top side snapped to the bottom of the text field above it and their right side to the right margin of the group. Drag the left side of acctIDText a bit to the left so that it is longer than the other text fields. Now drag the left side of all the other text fields to the left and snap them to the left side of acctIDText. Now when acctIDText resizes, the rest will too. Set the read_only style property to true for each text field. Set the v_scroll style property to true for addressText and miscInfoText.

Create eight labels on the group and set their text to “Account ID:”, “First Name:”, “Surname:”, “Date of Birth:”, “Phone Number:”, “Email Address:”, “Address:”, and “Miscellaneous Information:”. Snap the top of each label to its corresponding text field, and snap the left side of each label to the left margin of the group. You can tell when you are attaching to a margin because a blue line appears.

Drag the left side of acctIDText so that it is a little bit to the right of the longest label. Make sure that the snap point button for the left side of acctIDText is a number sign (#) with an arrow pointing left.

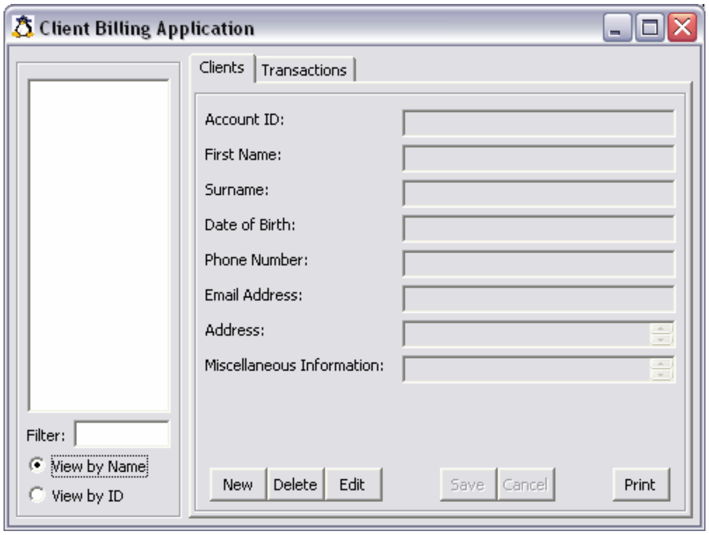
Your shell should now look something like the following:



Add a composite to the group. Set the left, bottom, and right side attachments to the group margins. Click on the button for the top attachment and click on the button with the number sign (#) and a down arrow. Drag the top side till the number in the tool tip to the right of the composite shows approximately 30.

Set the layout on the composite to FillLayout. On the composite, create three buttons, one label, two more buttons, another label, and then another button. The labels are to create space between the buttons. Remove the text in each label. Name the buttons “newClientButton”, “delClientButton”, “editClientButton”, “saveClientButton”, “cancelClientButton”, and “printTransButton”. Set their text to “New”, “Delete”, “Edit”, “Save”, “Cancel”, and “Print”, respectively. Use the convert local to field button to make the buttons visible in the class. Set the enabled property of saveClientButton and cancelClientButton to false.

Your shell should now look like this:



The Transactions Tab:

To modify the Transactions TabItem, click on the tab in the Content Pane. Add a group to the TabItem and name it transInfo. Convert the group to a field and set its layout to GridLayout with one column. Create a table and two composites in this group. The first composite contains the labels and text fields. Set its layout to GridLayout with four columns. The second composite holds the buttons and its layout should be a horizontal FillLayout. Set the horizontal fill property for both composites to true.

Name the table “infoTable”. Set the full_selection property of the table to true. Set the horizontal and vertical grab and fill properties of the table to true.

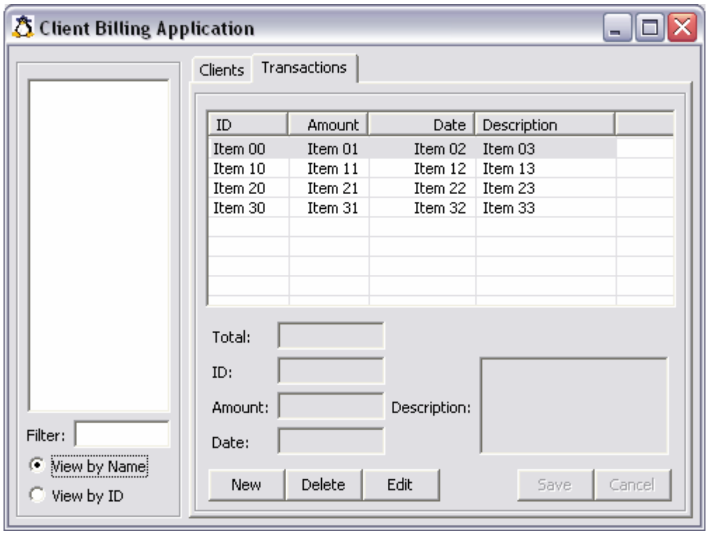
Add four table columns and name them “numberColumn”, “amountColumn”, “dateColumn”, and “descColumn”. Set their text to “ID”, “Amount”, “Date”, and “Description”, respectively. Convert the table and all four columns to fields by using the convert local to field button. Set the align property of amountColumn and dateColumn to right. If the columns do not all fit, shorten the length of the first three columns.

On the first composite, place five labels and five text fields with the labels and text fields alternating. Set the labels' texts to be "Total:", "ID:", "Description:", "Amount:", and "Date". Name the text fields "totalText", "idText", "descText", "amtText", and "dateText". Set all the text fields to read_only and convert them all to fields. Set the wrap property of descText to true.

Set the horizontal span for idText to 3. Set the vertical span for the description label and descText to 3. Set the horizontal grab for descText to true. Set the horizontal and vertical fill for descText to true.

On the second composite place three buttons, one label, and then two more buttons. Clear the text of the label. Name the buttons "newTransButton", "delTransButton", "editTransButton", "saveTransButton", and "cancelTransButton". Set their text to "New", "Delete", "Edit", "Save", and "Cancel", respectively. Set the enabled property of both saveTransButton and cancelTransButton to false. Convert the buttons to fields by using the convert local to field button.

Your shell should now look like this:



ID	Amount	Date	Description	
Item 00	Item 01	Item 02	Item 03	
Item 10	Item 11	Item 12	Item 13	
Item 20	Item 21	Item 22	Item 23	
Item 30	Item 31	Item 32	Item 33	

Creating a Menu:

Creating a menu in the SWT Designer is simple. You are not able to create a menu using the free version of the SWT Designer, so if you are using the free version, you will have to create the menu by hand like we did in the Address Book example.

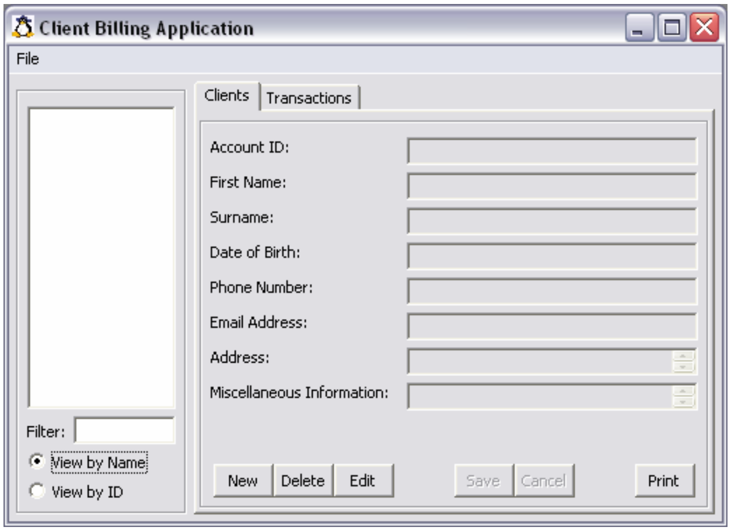
We will create a very simple menu with just one dropdown menu and one menu item.

Place a MenuBar on the shell. Place a Menu on the menu bar you just created. Set its text to "File". Place a MenuItem on the File menu. Set its text to "Quit". That is all you have to do to create a simple menu. The original ClientBillingUI also includes an About menu, but we will not cover creating another shell in this example.

Running the Application:

Run the application. As with the Address Book example, the initial window size is probably different from what is desired. To set up the shell to start with the size you specify, right-click on the shell in the Control Tree and select Set Shell Size > Using the setSize() method. Now resize the shell to the desired size.

Your application should look like the following when run:



Congratulations, you have successfully created a GUI for the Client Billing Application. You have also seen how easy it is to configure layouts and menus in the Designer. The GUI we made has no functionality at all until we add on a database and add some events. This is a task that is left for you. By analyzing the code in ClientBillingUI, it is fairly simple to do and primarily involves copying and pasting.

Creating a Swing GUI for the Client Billing Application:

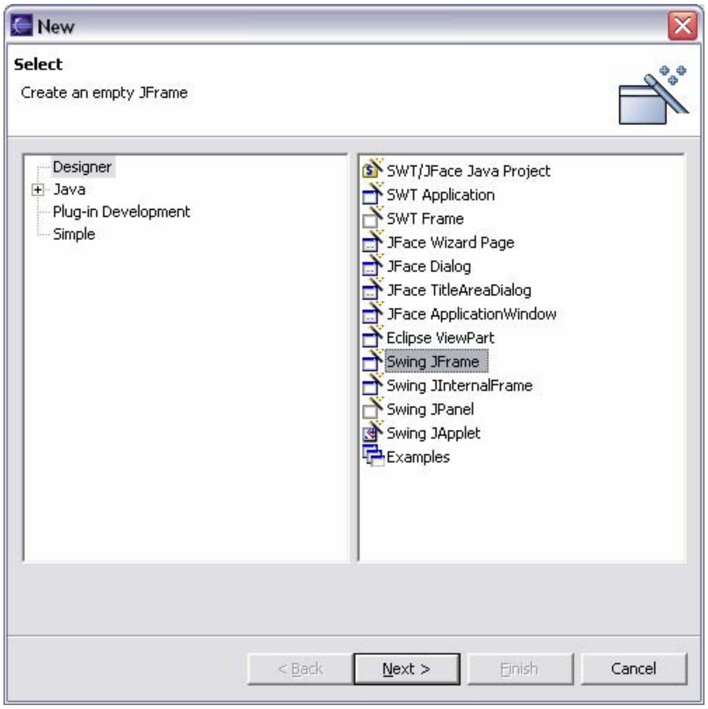
Like the above example, we will be creating a GUI for the Client Billing Application using the Designer. The GUI created in this example, however, will be creating using only Swing components.

If you have not already looked at the SWT example above, I would suggest you look at it first. The GUI we will be creating in this example will look and function almost identically to the GUI created in the previous example. By looking at the previous example, you will be able to see some of the differences and similarities between the SWT Designer and the Swing Designer.

Creating the ClientBillingUI:

We need to create a new class for the Swing GUI that we are about to design. In order to distinguish between the different GUIs, we will name this class ClientBillingUISwing.

To create a new Swing application, select New > Other from the File menu. In the first wizard page, select Designer on the left side and Swing JFrame on the right.

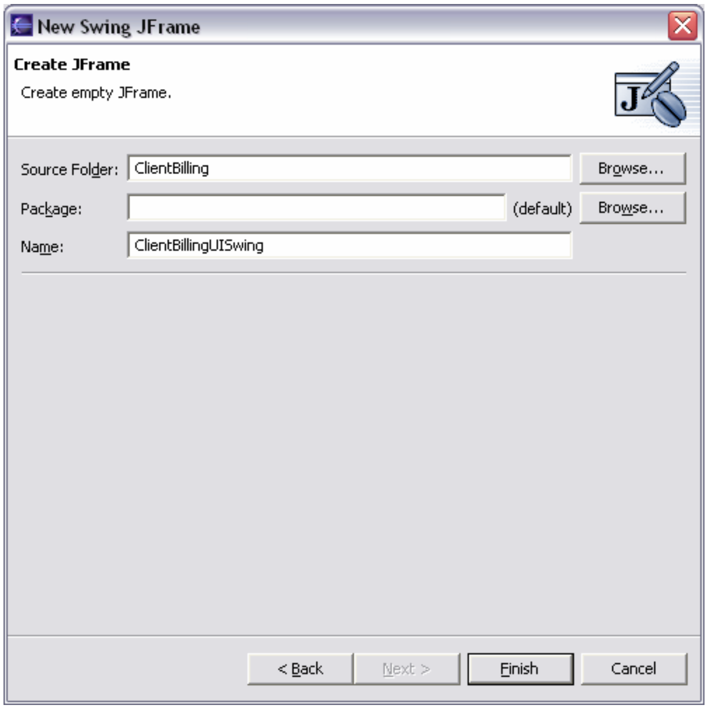


New

Select

Create an empty JFrame

Click Next and name the new class ClientBillingUISwing. Click Finish to create the new class.



New Swing JFrame

Create JFrame

Create empty JFrame.

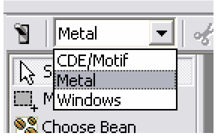
Source Folder: ClientBilling

Package: (default)

Name: ClientBillingUISwing

Swing allows the use of different look-and-feels. This allows a Swing application to look more like an application created for a specific operating system. Although you cannot currently change the look-and-feel of the application using the Designer, you can see what it would look like by selecting a look-and-feel in the design view.

There is a dropdown list above the Content Pane in the design view of the Swing Designer. Since the application uses the Metal look-and-feel by default when it is run, we will use the Metal look-and-feel when we design our application. To change the look-and-feel to Metal, select Metal from the look-and-feel dropdown list.

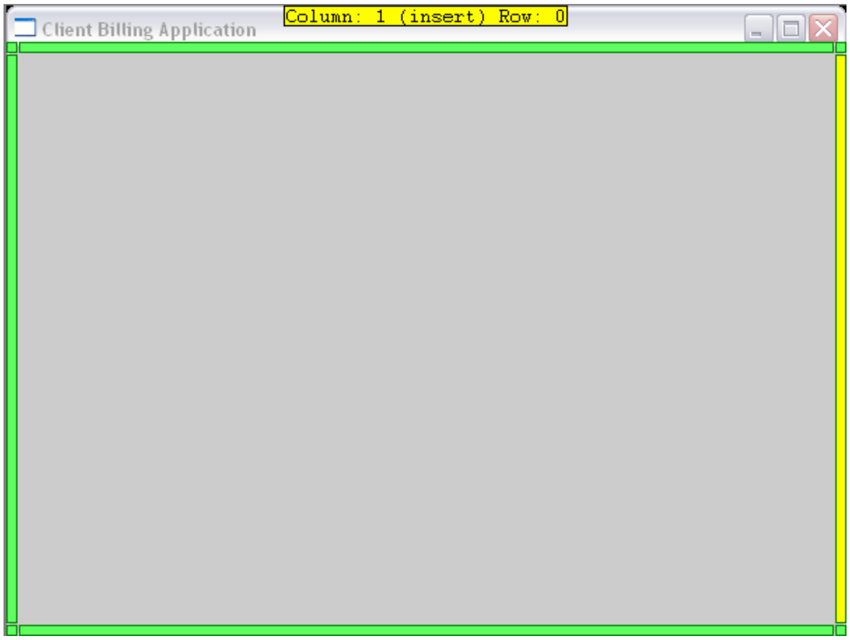


Select the (JFrame) object in the Control Tree and set its title in the Inspector to "Client Billing Application". Then set the contentPane's layout to GridBagLayout by selecting GridBagLayout from the Layouts category in the Control Palette and placing it on the contentPane.

Creating the Client List JPanel:

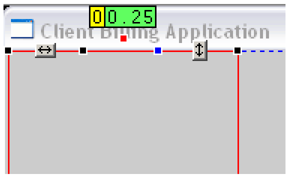
Add a JPanel to the contentPane. Under the constraints complex property in the Inspector, set the weightX property to 0.25 and the weightY property to 1.0. These properties set the proportions of a component's dimensions, which are compared to other component's proportions. A 1.0 value is typically associated with 100%, where 0.25 is associated with 25%. If 0.0 is used, the component's preferred size is used.

Add a JTabbedPane to the right of the JPanel. Add it by placing the mouse cursor on the far right edge of the JPanel. Green bars will surround the JPanel. Click on the far right bar to place the JTabbedPane to the right of the JPanel.



Select the JTabbedPane and set its weightX property to 0.75 and its weightY property to 1.0.

Select the JPanel in the Control Tree. The JPanel should be enclosed in a red border in the Content Pane. There should be two small buttons at the top of the border. Click on the buttons so that arrows appear in them.



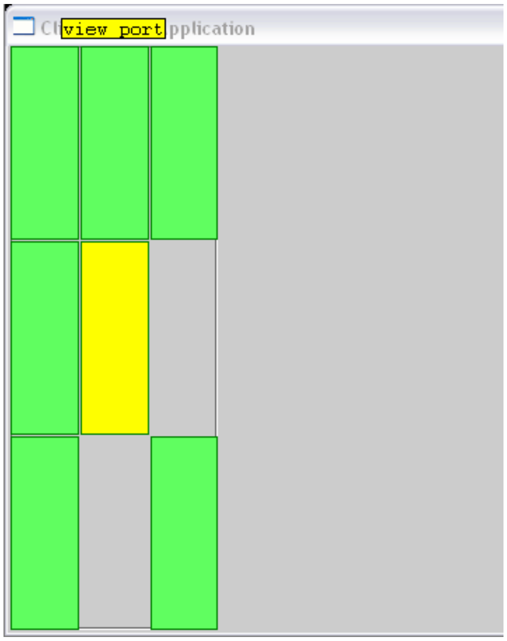
These buttons set the fill property under the constraints complex property in the Inspector. By clicking on both buttons, we have set the fill property to both.

Now do the same thing for the JTabbedPane.

In order to make the JPanel look like a group, we need to add a Titled Border. Select Titled Border in the border property in the Inspector. Now select Etched Border in the nested border property. Set the title property to “Clients:”.

Set the layout of the JPanel to GridBagLayout. Add a JScrollPane to the JPanel. JScrollPanes are needed if you want a component to scroll. Some components that often scroll are JLists, JTextAreas, and JTables. Set both the weightX and the weightY properties of the JScrollPane to 1.0. Set the fill property to both.

Add a JList to the view port of the JScrollPane. Moving the cursor around inside of a JScrollPane while placing a component will highlight the different areas you can place a component. The component that is supposed to scroll is always placed in the view port, which is the center area.



Set the variable name of the JList to “clientList”.

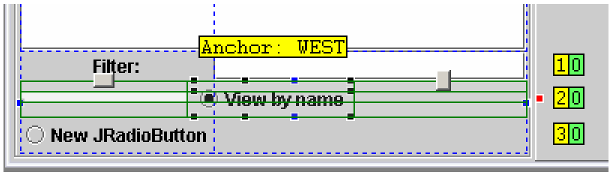
Add a JLabel below the JScrollPane by clicking on the green bar at the bottom of the JScrollPane. Add a JTextField to the right of the JLabel. Add a JRadioButton below the JLabel, and another JRadioButton below the other JRadioButton.

Set the JScrollPane’s gridWidth property under the constraints complex property to 2. The gridWidth and gridHeight properties set the number of columns and rows that a component takes up.

Set the JLabel’s text property to “Filter:”. Expand the insets complex property in the constraints complex property and set the left and right properties to 5. The insets properties put some padding around the component.

Set the JTextField’s variable name to “filterText” and clear its text property. Set its fill constraint property to horizontal.

Set the variable name of the top JRadioButton to “nameRadio”, set the text property to “View by name”, set the selected property to true, set the gridWidth constraint to 2, and set the anchor constraint to West. You can set the anchor constraint by dragging the JRadioButton to its desired position on the Content Pane.



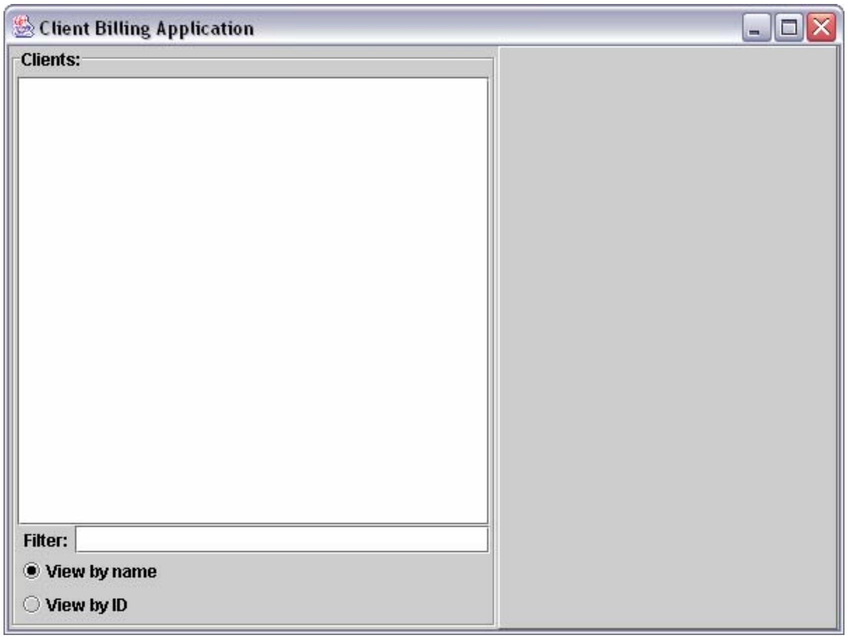
Set the variable name of the bottom JRadioButton to “numRadio”, set the text property to “View by ID”, set the gridWidth constraint to 2, and set the anchor constraint to West.

In order to link the two JRadioButtons together, we have to add them to a ButtonGroup. To create a ButtonGroup, select the ButtonGroup from the Swing Controls section of the Control Palette. Right at the bottom of the Control Tree should be an object called “(button groups)”. Add the ButtonGroup to this object in the Control Tree. A new ButtonGroup named “buttonGroup” should be created.

Select the nameRadio JRadioButton and select “buttonGroup” from the dropdown list of the buttonGroup property. Do the same with the numRadio JRadioButton.

Now when the application is run, selecting one of the JRadioButtons will deselect the other one. This ensures that only one JRadioButton can be selected at one time.

Below is what our completed client list JPanel should look like.

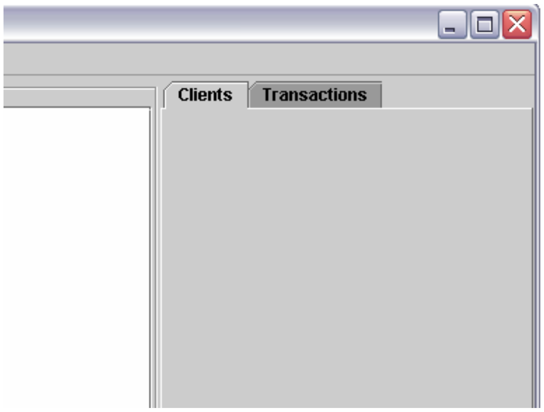


Creating a JTabbedPane:

We had previously added a JTabbedPane to the contentPane. When a component is added to a JTabbedPane, a new tab appears and that tab contains the component that you added to the JTabbedPane.

In order to place several components in a tab, a container must be the component added to the JTabbedPane. Select JPanel from the Control Palette and click anywhere inside the JTabbedPane. A tab appears with a JPanel in it. Create a second tab by selecting JPanel from the Control Palette and add it to the JTabbedPane. If there is already a tab on a JTabbedPane, you must place the cursor beside a previously placed tab to add another tab.

Set the left tab's title to "Clients" and the right tab's title to "Transactions". The tabs should look like the following:

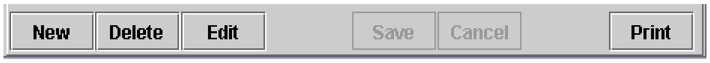


The Clients Tab:

Set the layout of the JPanel in the Clients tab to BorderLayout. Add a JPanel to the south section and then add a JPanel to the center section.

Set the layout of the JPanel in the south section to GridLayout. Add three JButtons, one JLabel, two JButtons, one JLabel, and one JButton, in that order to the south JPanel. The first JButton may have to be added to the JPanel through the Control Tree because the JPanel will initially be very small.

Name the JButtons "newClientButton", "delClientButton", "editClientButton", "saveClientButton", "cancelClientButton", and "printTransButton", in the same order they were placed. Set the text property of the JButtons to "New", "Delete", "Edit", "Save", "Cancel", and "Print", in the same order as above. Expand the margin complex property in the Inspector and set the left and right margins to 4 for each JButton. Set the enabled property for saveClientButton and cancelClientButton to false. Clear the text property for the two JLabels. Select each JButton and click on the "Convert local to field" button. The buttons should look like the following:



Set the center JPanel’s layout to GridBagLayout. Add eight JLabels to the center JPanel. Place them one below each other in one column. Add six JTextFields just to the right of the top six JLabels. All the JTextFields should be in one column. Add two JScrollPanes below the JTextFields and beside the bottom two JLabels. There should now be eight rows and two columns.

Set the text property of the JLabels to “Account ID:”, “First Name:”, “Surname:”, “Date of Birth:”, “Phone Number:”, “Email Address:”, “Address:”, and “Miscellaneous Information:”, from top to bottom.

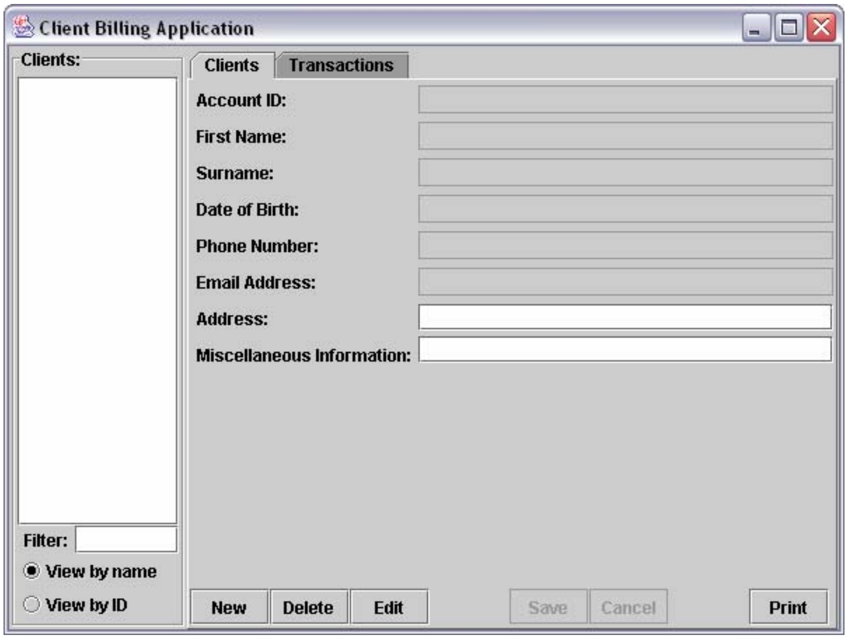
Set all the JLabels’ anchor constraint to West except for the Miscellaneous Information JLabel, which should be set to North. Set the top, left, bottom, and right insets of each JLabel to 5. Set the weightY constraint of the Miscellaneous Information JLabel to 1.0.

Set the variable names of the JTextFields to “acctIDText”, “fnameText”, “snameText”, “dobText”, “phoneText”, and “emailText”, from top to bottom. Clear the text property of all the JTextFields. Set the weightX constraint of all the text fields to 1.0 and the fill constraint to horizontal. Set the editable property of all the JTextFields to false.

Set the fill constraint of both JScrollPanes to horizontal. Set the bottom JScrollPane’s anchor constraint to North.

Add a JTextArea to the view port of both JScrollPanes. Set the variable name of the JTextArea in the top JScrollPane to “addressText” and the other JTextArea to “miscInfoText”. Clear the text property of both JTextAreas. Set the editable property of both JTextAreas to false. In Swing, making a JTextArea non-editable does not color it gray like the JTextFields. If you wish to do this, you will have to do this yourself.

The finished Clients tab should now look like the following:

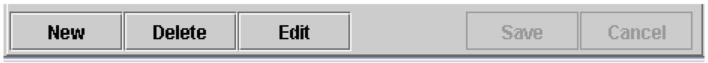


The Transactions Tab:

Double-click the Transactions tab to view the contents of it. Set the layout of the JPanel in the Transactions tab to BorderLayout. Add a JPanel in the south section of the JPanel in the Transactions tab and then add another JPanel in the center section.

Set the layout of the south JPanel to GridLayout. Add three JButtons, one JLabel, and two JButtons, in that order. The first JButton may have to be added to the JPanel through the Control Tree.

Name the JButtons "newTransButton", "delTransButton", "editTransButton", "saveTransButton", and "cancelTransButton", in the same order you added them. Set their text properties to "New", "Delete", "Edit", "Save", and "Cancel", in order. Set the left and right properties in the margin complex property to 4 for each JButton. Set the enabled property to false for saveTransButton and cancelTransButton. Clear the text property for the JLabel. Select each JButton and click on the "Convert local to field" button. The buttons should look like the following:



Set the layout of the center JPanel to GridBagLayout. Add four JLabels, one below the other, all in one column. Add four JTextFields to the right of the JLabels, all in one column. Add one JLabel to the right of the third JTextField from the top. Add a JScrollPane to the right of one cell up from the right-most JLabel. Add a JScrollPane above the top-left JLabel. The reason we did not add the last JScrollPane first was because it would have initially been very small and it would have been hard to add another component below it.

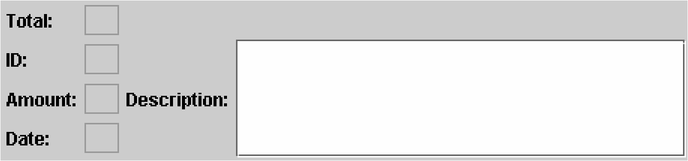
Set the text property of the JLabels to "Total:", "ID:", "Amount:", "Date:", and "Description:", in the order you created them. Set the anchor constraint of all the JLabels to West. Set the top, left, bottom, and right insets of all the JLabels to 5.

Name the JTextFields "totalText", "idText", "amtText", and "dateText", from top to bottom. Clear the text of all the JTextFields. Set the weightX constraint of totalText to 0.25. Set the fill constraint of all the JTextFields to horizontal. Set the editable property of all the JTextFields to false.

Set the gridHeight constraint of the right JScrollPane to 3, its weightX constraint to 0.75, and its fill constraint to both.

Add a JTextArea to the view port of the right JScrollPane. Name it "descText". Clear its text property, set its editable property to false, its lineWrap property to true, and its wrapStyleWord property to true.

The section we just created is shown below. The JTextFields look smaller than they should be. They shrunk when we set the JTextArea's lineWrap property to true. This has to do with how the layout works and should appear correctly after we finish configuring the rest of the components.



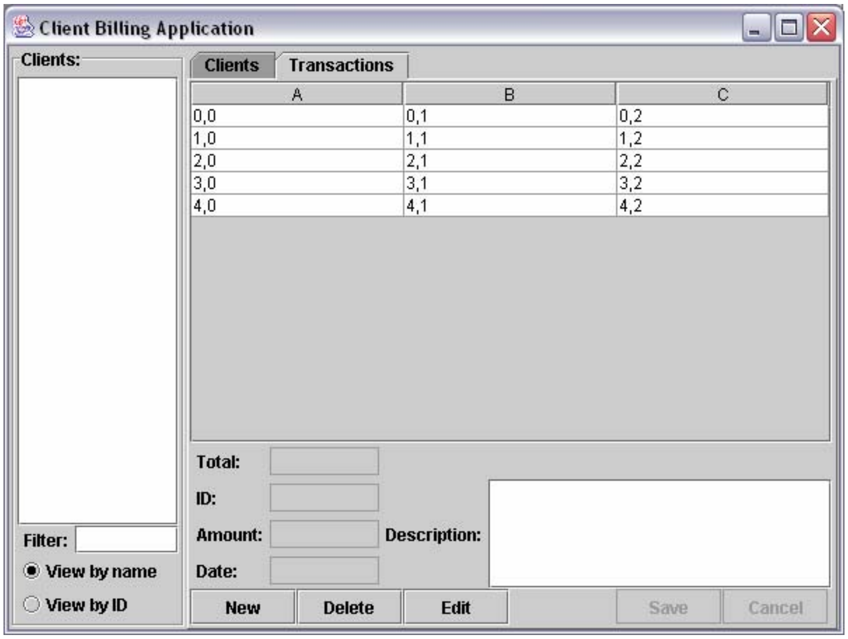
Set both the weightX and weightY constraints of the top JScrollPane to 1.0. Set its fill constraint to both and its gridWidth constraint to 4.

Add a JTable to the view port of the top JScrollPane. Name the JTable "infoTable". Select the JTable and click on the "Open definition" button. Add the following line in the JTable's block of code:

```
        infoTable.getTableHeader().setReorderingAllowed(false);
```

This prevents a user from rearranging the columns. A bit further down we will be using the column number to determine which column is which. If the columns are rearranged, it changes the index of the columns.

The finished Transactions tab should look like the following:



A	B	C
0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2
3,0	3,1	3,2
4,0	4,1	4,2

Filling a JTable with values is not as simple as it is in SWT. In SWT you can assign a String to each cell in a Table, whereas in Swing, a JTable functions similarly to a JFace TableViewer. To create a table in Swing, you must create a model. Fortunately, the Swing Designer creates the skeleton of a table model for us. To create the table model, double-click on the model property of the JTable. This creates a new inner class called InfoTableTableModel. The following is the contents of that class.

```
class InfoTableTableModel extends AbstractTableModel {
        public final String[] COLUMN_NAMES = new String[] {
        };
        public int getRowCount() {
                return 0;
        }
        public int getColumnCount() {
                return COLUMN_NAMES.length;
        }
        public String getColumnName(int columnIndex) {
                return COLUMN_NAMES[columnIndex];
        }
        public Object getValueAt(int rowIndex, int columnIndex) {
                return null;
        }
}
```

The table model does not really do anything. To make the table functional, make the InfoTableTableModel class look like the following:

```
class InfoTableTableModel extends AbstractTableModel {
        private ArrayList transactions = new ArrayList();

        private final String[] COLUMN_NAMES = new String[] {
                "ID", "Amount", "Date", "Description"
        };


        public int getRowCount() {
                return transactions.size();
        }
        public int getColumnCount() {
                return COLUMN_NAMES.length;
        }
        public String getColumnName(int columnIndex) {
                return COLUMN_NAMES[columnIndex];
        }
        public Object getValueAt(int rowIndex, int columnIndex) {
                Transaction transaction = (Transaction)transactions.get(rowIndex);

                switch (columnIndex) {
                        case 0: //ID column
                                return new Integer(transaction.transactionID);
                        case 1: //Amount column
                                return new Integer(transaction.amount);
                        case 2: //Date column
                                return transaction.date;
                        case 3: //Description column
                                return transaction.description;
                        default:
                                return null;
                }
        }
        public void addRow(Transaction transaction) {
                transactions.add(transaction);
                fireTableRowsInserted(transactions.size() - 1, transactions.size() - 1);
        }
}
```

The following also needs to be added to the list of imports.

```
import java.util.ArrayList;
```

The above creates an ArrayList of Transactions to hold the data of the table. Note that the Transaction class from the Client Billing Application must be available. The number of rows is the number of Transactions in the ArrayList. The column names are stored in a String array called COLUMN_NAMES.

The getValueAt() method returns an object based on the row and column indexes that were passed. The table calls this method and then calls the getString() method of the returned object to display the values in the table.

The addRow() method is a method that was added to the class to easily add Transactions to the table. It adds the passed Transaction to the ArrayList and then calls fireTableRowsInserted() to notify the table that rows have been inserted. The parameters of fireTableRowsInserted() are the indexes of the first and last rows inserted, inclusive.

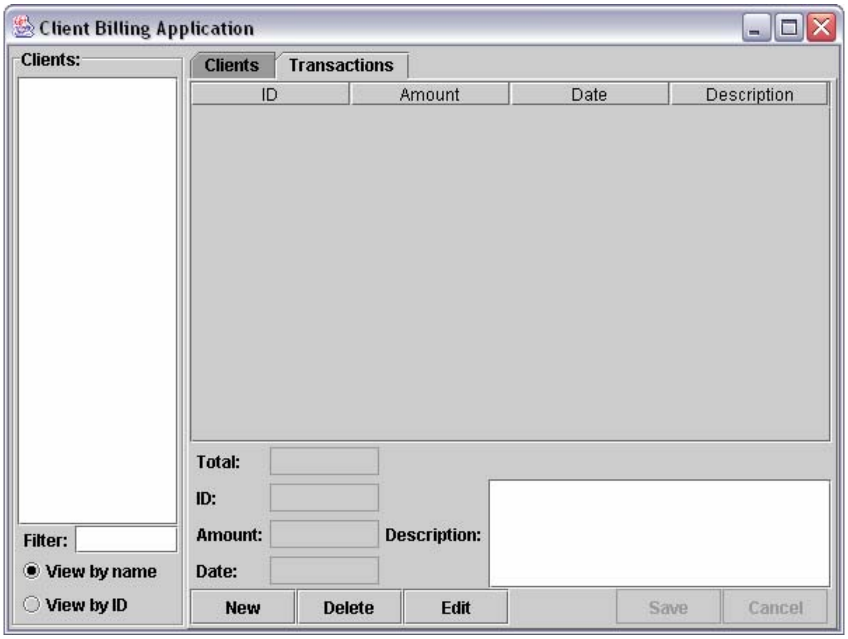
You can add a row to the table by using the following code:

```
((InfoTableTableModel)(infoTable.getModel())).addRow(t);
```

where `t` is the Transaction object that you want to add.

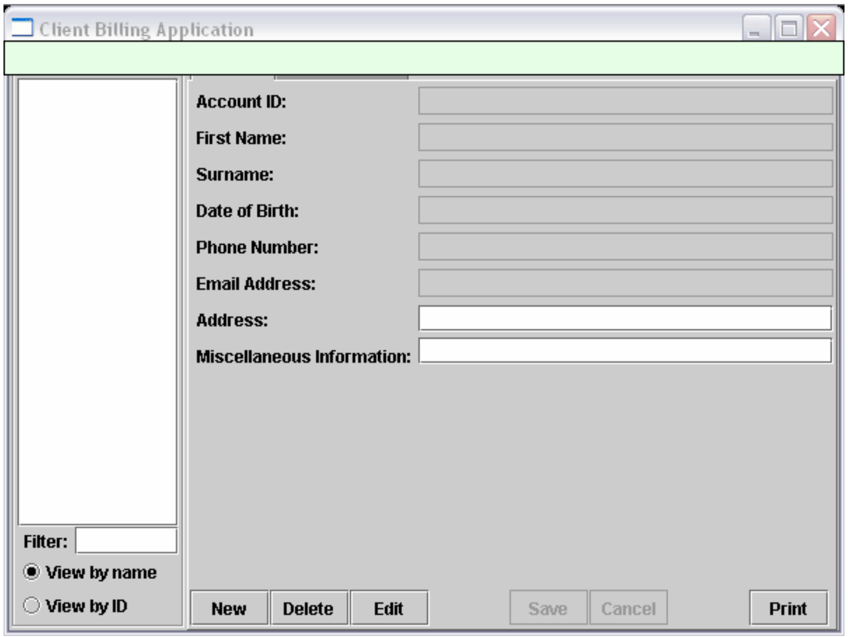
In order to add extra functionality to the table, such as sorting, the table model would simply have to be extended to support the desired functions. We will not cover that here since it is beyond the scope of this tutorial.

When you run the application, you can see the columns that we have created.



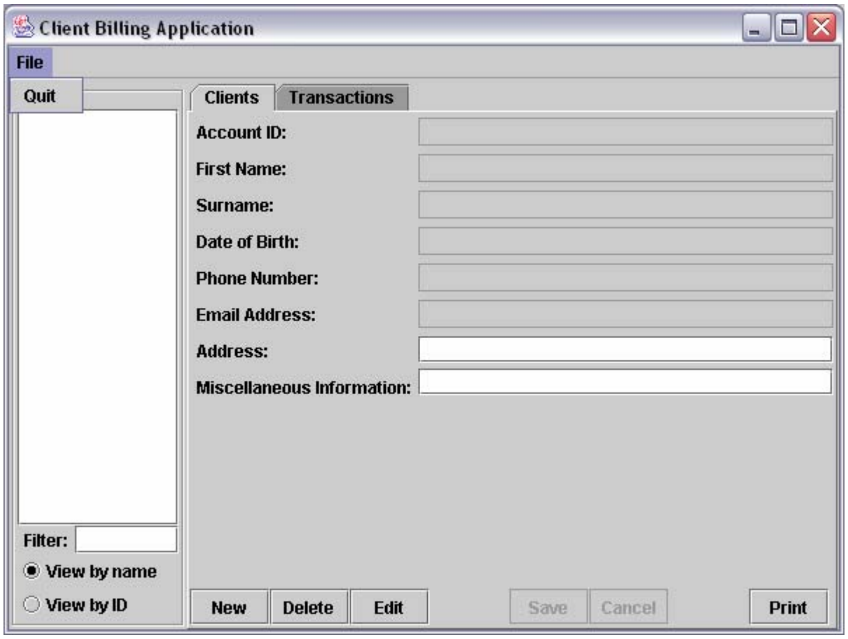
Creating a Menu:

To create a menu bar at the top of the application, we have to add a JMenuBar to the JFrame. Select JMenuBar from the Menu Controls category of the Control Palette and place your cursor over the application title bar. The area the menu bar will go is highlighted.



Add a JMenu to the JMenuBar you just added. Set its text property to "File". Add a JMenuItem to the File menu. Set its text to "Quit".

Shown below is the final GUI, including a menu.



Running the Application:

You can run just the Swing GUI because the Swing Designer creates a main() method. Because the GUI uses just Swing and AWT components, you do not need to add any JAR files to your projects build path.

The GUI currently does not have any useful functionality. No information is displayed and the buttons do nothing when they are clicked. Making this GUI functional would be similar to making the SWT GUI functional. However, there would be some differences. Similar widgets for SWT and Swing may have different properties and may access data differently. An example is a table. Getting a simple table working in Swing requires a lot more effort than getting a simple table working in SWT.

To plug the Swing GUI into an application, you only need to place the code that is in the main() method of the ClientBillingUISwing class into another method in another class that is supposed to open the GUI.

Summary:

This tutorial has described the majority of the features of the free and professional versions of the Designer. We have gone through an example of creating a GUI for a simple Address Book application using the free version of the SWT Designer. Through this example we have learned how to create a GUI as well as how the Designer generates code. We have also gone over a more advanced example in both SWT and Swing using the professional version of the Designer where we used layout managers, tabbed folders, and menus.

The easiest way to get familiar with this easy to use GUI designer is to play around with it. There is additional documentation as well as some demos available at http://www.swt-designer.com.