Abacus GUI User Guide

+

Alpha JFreeChart Components

2005-July-20
AbaGuiBuilder Ver. 1.6

1

About The Java Builder

The Abacus GUI builder is a tool aimed to aid application developers deliver their applications faster to market by removing the Java layout complexity with a WYSIWYG and simple XY layout.

Abacus Research developed the GUI Builder and Renderer to aid their 50+ application developers eliminate frustrating hours of complex layout user interface logic and thus simplifying each screen to an XY coordinate plane.

In addition to the XY layout, Abacus Research developed the Java GUI Builder, a WYSIWYG tool that allows the applications developer to place UI Java swing components on the canvas and have it render exactly as you see on the screen. Due to the various Java layouts, other Java GUI Builders are cumbersome and complicated, our goal is to simplify the UI task and help the application developer speed the UI development in order to concentrate in that application.

2 Revision 1.1

A few changes had to be made in order to fix/enhance the Abacus GUI Builder Open Source version. First, we changed the default Look & Feel to **Windows** and enhanced some parts of the UI. Second and most important was to change the output and how class are packaged, in version 1.0 we outputted class files for event handling and stored the project definition in binary format in a .PROZ and we supported a PROJ format. PROJ format is an XML based project file that defines all the objects within the project, the PROZ was in fact a binary format of the same file, along with some of the compiled classes. In fact, the handlers were left out of the PROZ file and only the main file was stored.

2.1 Abacus JDBC Components

These components work in conjunction Sun's latest Rowset object and SwingSet classes and their code (http://sourceforge.net/projects/swingset) we were able to implement our own database classes for the Abacus designer.

3 What's New in 1.6

1. Importing Beans and components from IDE
2. Dynamic loading for metadata.
3. First step to Plug-in model (Editor only for this version will release details later).
4. Add new Code editor with component function searching.
5. Enhanced property table.
6. Added the following JFreeChart custom components (Alpha Version):

    PieChart
    PieChart3D
    BarChart
    StackedBarChart
    BarChart3D
    StackedBarChart3D
    AreaChart
    StackedAreaChart
    LineChart
    LineChart3D
    GanttChart
    XYLineChart

4 Introduction

Java Builder 1.X requirements (all included in the distribution zip):

- Java JDK 1.4.X
- Merlin (Font Chooser)
- CF (Code Formatter)
- Electric XML (EXML)
- Rowset 1.0 (This will become part of the J2SE 1.5)
- swingset-bin_0.7.0_beta

Optional for demo: MySql database.

Java JDK

Required for compiling auto created code.

Merlin JAR

Required for font choosing actions.

CF JAR

Component written by IBM , and used in the builder to format auto created code.

Electric XML

Component written by Electric Mind and use in the builder to parse XML configuration and the builder meta data.

RowSet

Component written by Sun will become part of the J2SE 1.5 and it is required for the database visual components for Abacus GUI builder 1.1.

SwingSet

Code/Classes as foundation for the database visual components in the GUI builder 1.1.

4.1 Starting the GUI Builder

Once you have downloaded the Abacus Java Builder zip file, make sure to uncompress it and run the start up batch **run.bat.**

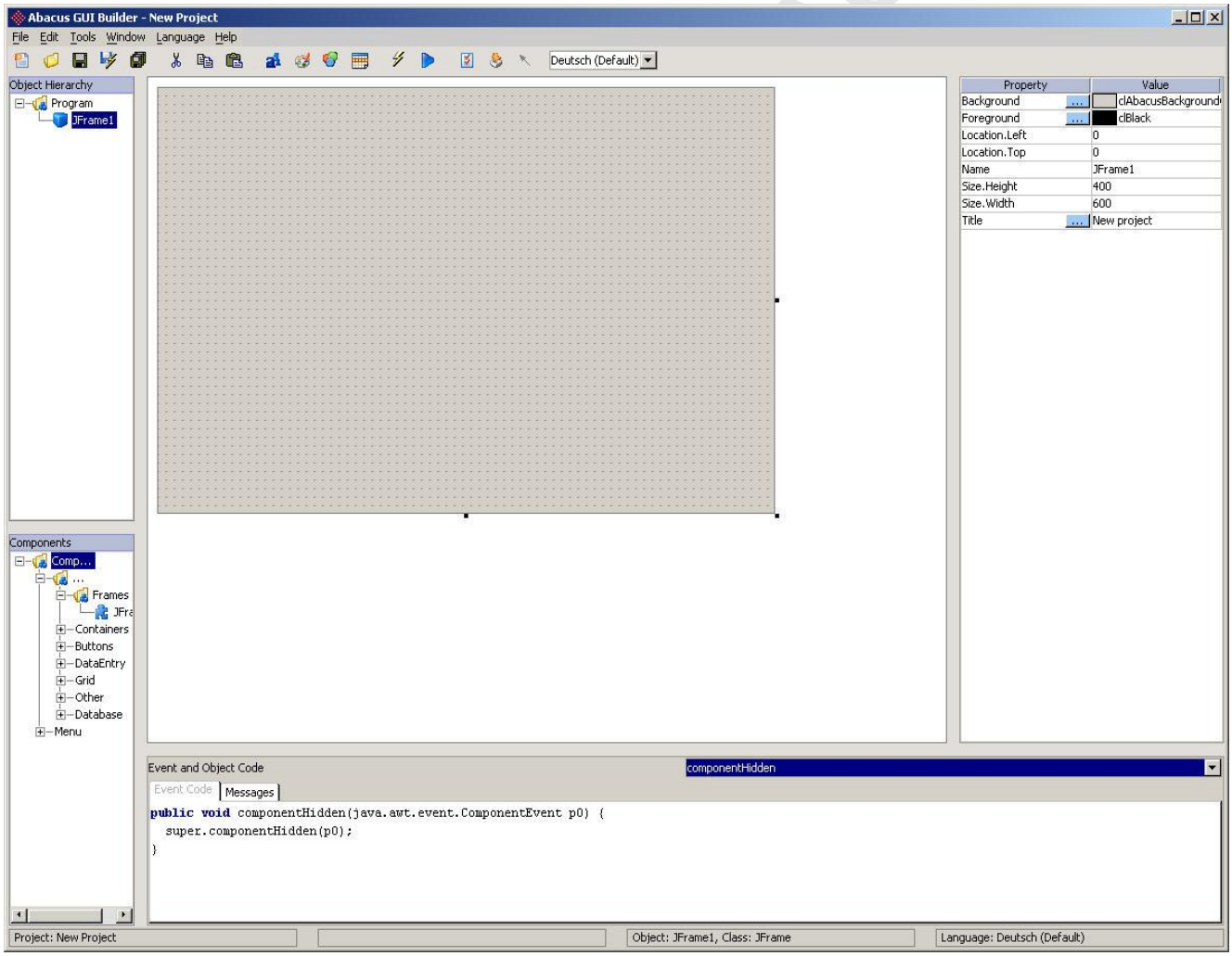


Figure 1

Once you have executed the "run.bat" batch file, the Java GUI builder should appear as in Figure 1. The GUI builder is divided into six areas:

4.2 Active Object Panel

On the top right, the active object hierarchy tree displays all active and visually available objects for the project, in addition, the hierarchy tree indicates the currently selected object.

The other important function of the object panel is to offer a visual representation of the relationship between objects, more specifically containership, as when an object is contained or is a “child” of another object, as in Fig. 2

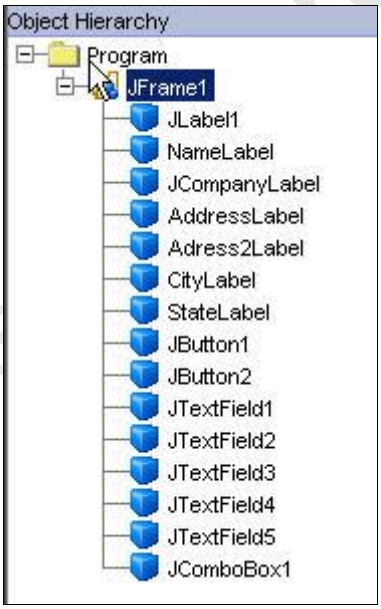


Figure 2

4.3 Object Canvas

Located at the top center of the screen it shows the visual representation of the project and how the screen will look when the Java unit is executed (Fig. 3). This is the basis of the GUI builder WYSIWYG concept, when the GUI Builder project is executed the objects on the canvas will look exactly as you see them in the canvas.

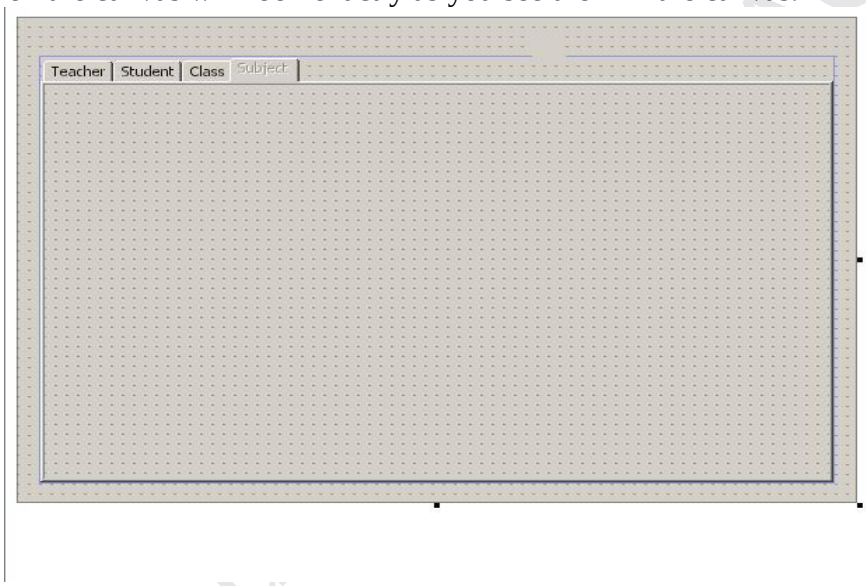


Figure 3

4.4 Property Panel

This panel is located at the top right of the GUI Builder and its function is to display all the available properties for the current selected object. This panel allows the user to set and change property values for each object on the object canvas.

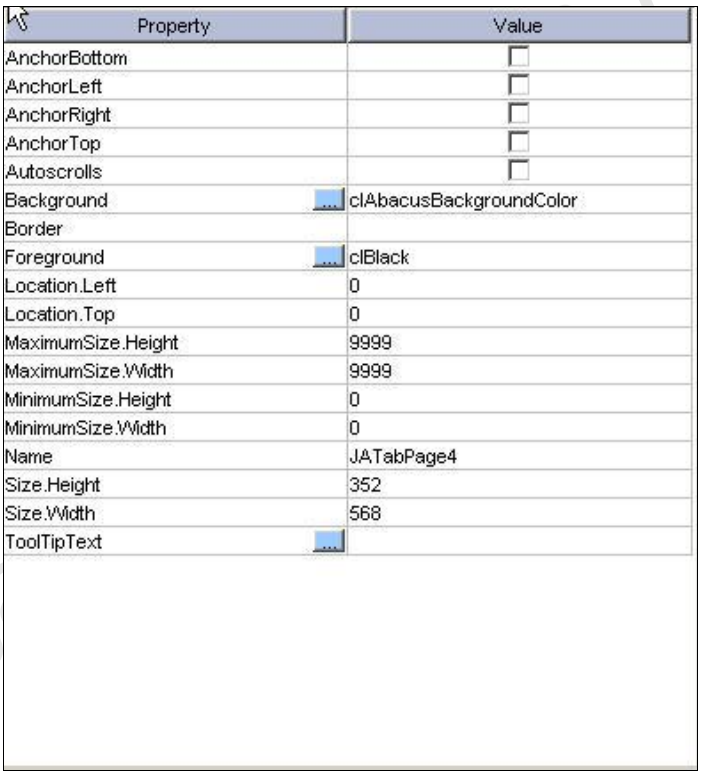


Figure 4

4.5 Component Tool Box

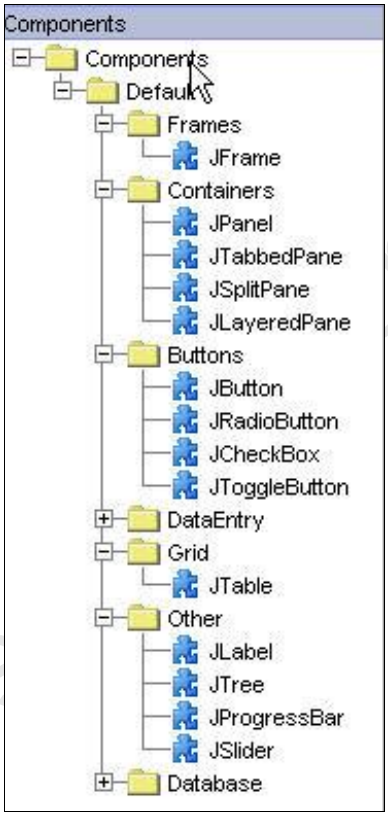
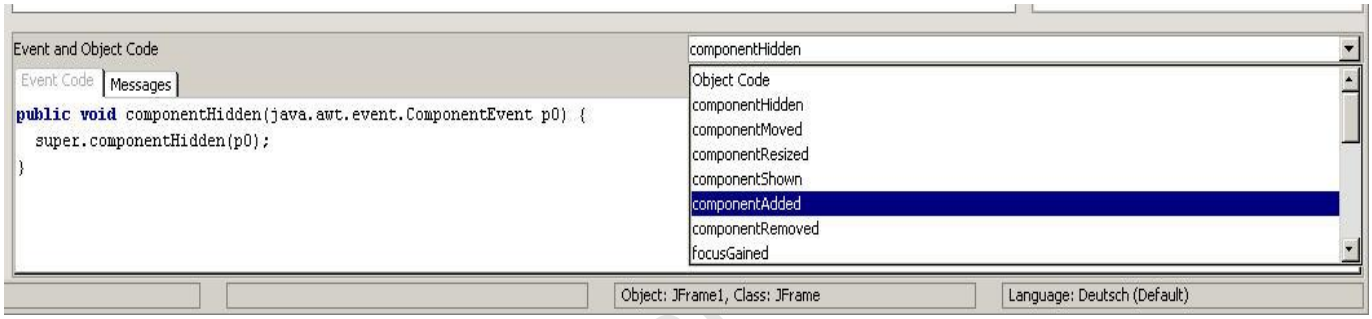


Figure 4

The component tool box is a tree containing all available Swing classes ready for instantiation on the canvas. In order to instantiate an object on the canvas, point and click on the Swing object class you wish to create then click on the canvas, thereafter , you have instantiated a Java object on the canvas at the desirable XY location.

4.6 Event List and Event Code Panel



The event list and event code panel work in tandem to provide a list of available events per object and its corresponding object event code, this is a standard paradigm in today's UI builders such VB and Delphi. The event code panel helps the application developer code business logic to events without the having to add Java listeners and, therefore it presents the developer with an easier paradigm.

5 Our First Attempt

Fist start the GUI builder, if you have not started it yet and, click on the new project button  , at this point you will have a empty object panel and a closed component list on the Component Panel.

Next click on the Components item and, the Swing classes should display as in Figure 5.

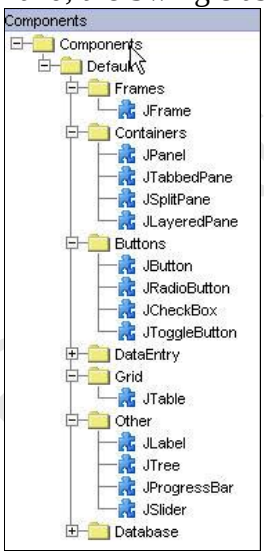
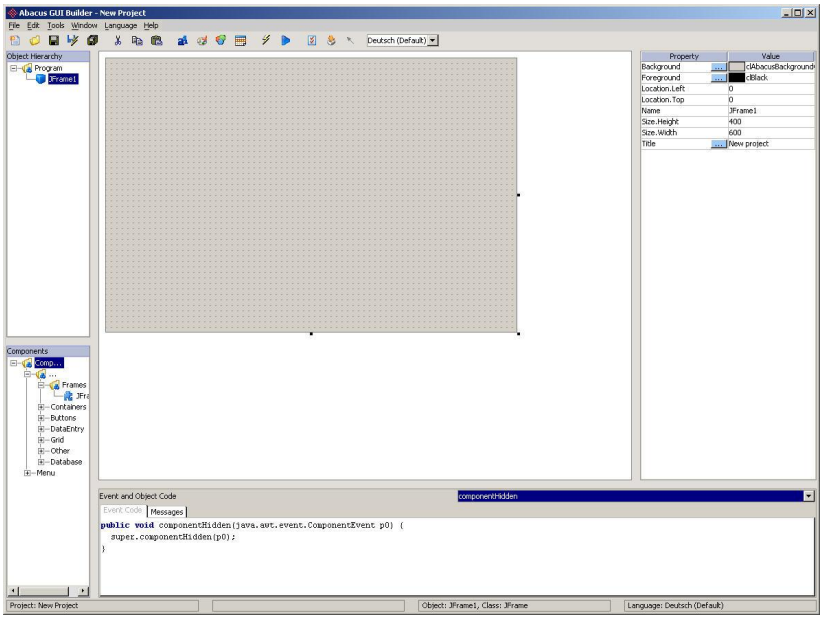
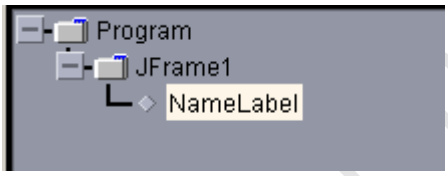


Figure 6

Select the **JFrame** item and click on the canvas at this point a **JFrame** object should be created on the canvas panel, most GUI screens created with the AbaGuiBuilder will be JFrame based.



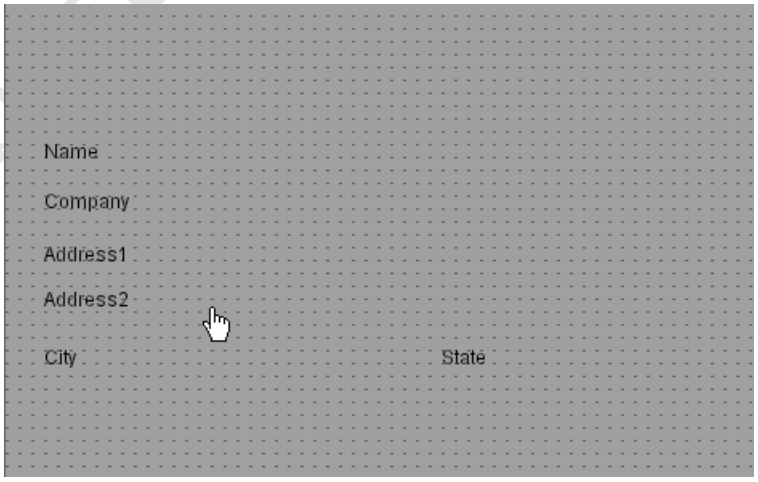
Next select JLabel and click on the canvas, you should have an instance of a JLabel object , named "JLabel" , at this time click on the properties box , select the property **Name**, and changed its value to **NameLabel** press the Enter key in order to update the property value . You changed the object unique to NameLabel, next we will change the display text for the object by clicking on the property **Text** and, changing its value to "Name" and pressing the Enter key. The Active Object Panel and the Object Canvas should look like the following images.



Active Object Panel

Object Canvas

Next make your canvas look like the following figure using the object properties from **Table 1**:

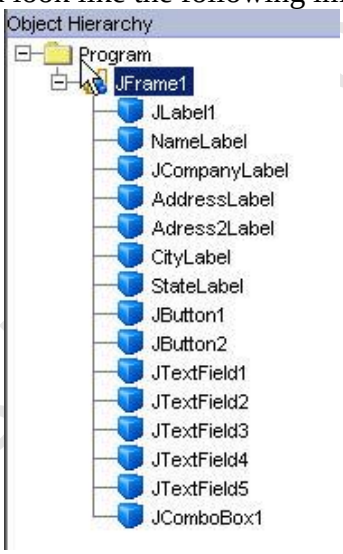


Sample Properties

Object Name	Object Text
CompanyLabel	Company
Address1Label	Address2
CityLabel	City
StateLabel	Label

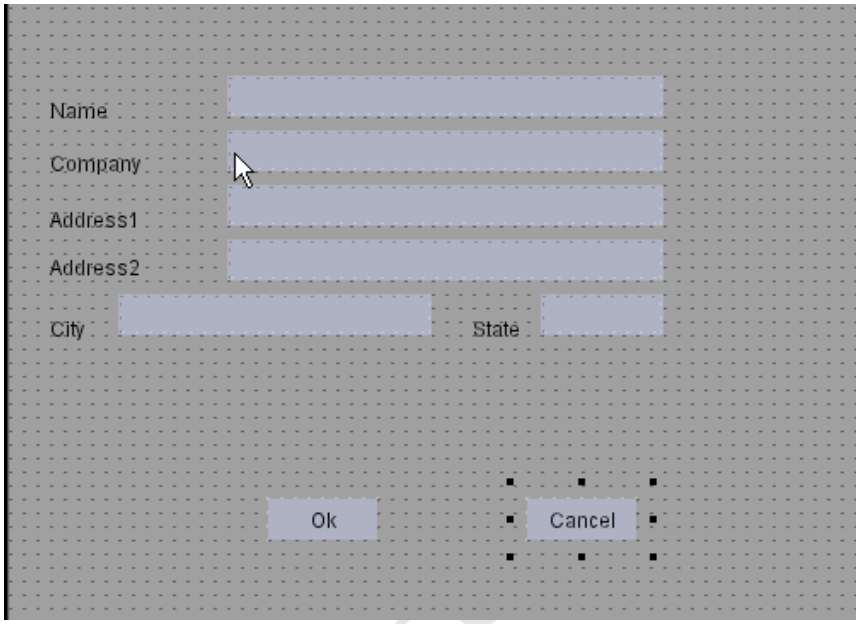
Table 1

The Active Object Panel should look like the following image:

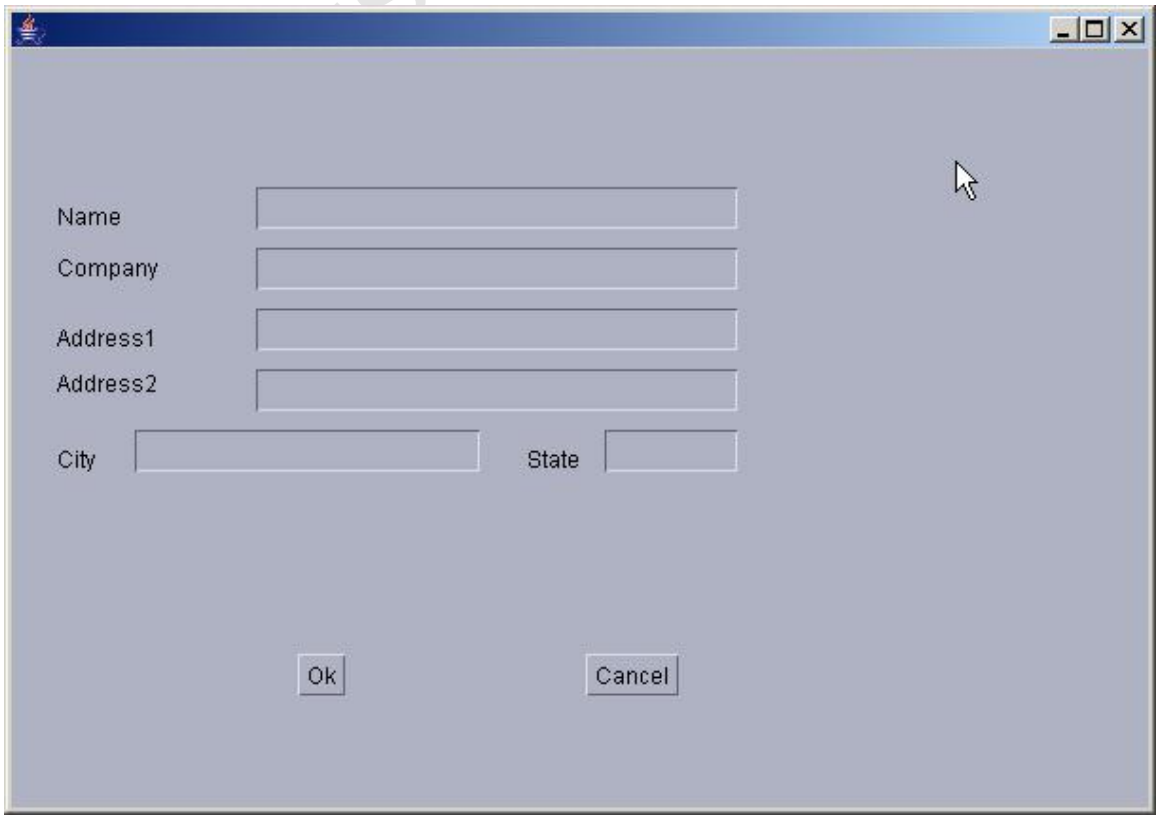


At this point, we should save our project by clicking on  , typing **cityscreen** in the "Enter File Name" edit box and, by clicking on the Save button. Now you have saved your work on a project named **cityscreen** with default extension **proz.**

Next step, you should add six JTextFields and two JButtons and make the canvas look like the following image:



Make sure to save your project once again and, let's try to render the application in order to preview what the application will look, so click on the Render button  and you will see the how the screen will look when you run it.



Once you saved the project, you may instantiate the project screen from another Java program by creating an AbaRenderer object and loading the project file with the render. If you explore the directory where you saved the project, you should find a file with extension **.decl**. The file contains Java code and object references that you will use to execute the screen from another Java program.

cityscreen.proz.decl

```
    private JLabel citiscreen_NameLabel;
    private JLabel citiscreen_CompanyLabel;
    private JLabel citiscreen_Address1Label;
    private JLabel citiscreen_Address2Label;
    private JLabel citiscreen_CityLabel;
    private JLabel citiscreen_StateLabel;
    private JTextField citiscreen_NameEdit;
    private JTextField citiscreen_CompanyEdit;
    private JTextField citiscreen_Address1Edit;
    private JTextField citiscreen_Address2Edit;
    private JTextField citiscreen_CityEdit;
    private JTextField citiscreen_StateEdit;
    private JButton citiscreen_OkButton;
    private JButton citiscreen_CancelButton;


// Assignments for this user interface

public void getReferences(){
    citiscreen_NameLabel = (JLabel) m_AbaRenderer.getObject("NameLabel");
    citiscreen_CompanyLabel = (JLabel) m_AbaRenderer.getObject("CompanyLabel");
    citiscreen_Address1Label = (JLabel) m_AbaRenderer.getObject("Address1Label");
    citiscreen_Address2Label = (JLabel) m_AbaRenderer.getObject("Address2Label");
    citiscreen_CityLabel = (JLabel) m_AbaRenderer.getObject("CityLabel");
    citiscreen_StateLabel = (JLabel) m_AbaRenderer.getObject("StateLabel");
    citiscreen_NameEdit = (JTextField) m_AbaRenderer.getObject("NameEdit");
    citiscreen_CompanyEdit = (JTextField) m_AbaRenderer.getObject("CompanyEdit");
    citiscreen_Address1Edit = (JTextField) m_AbaRenderer.getObject("Address1Edit");
    citiscreen_Address2Edit = (JTextField) m_AbaRenderer.getObject("Address2Edit");
    citiscreen_CityEdit = (JTextField) m_AbaRenderer.getObject("CityEdit");
    citiscreen_StateEdit = (JTextField) m_AbaRenderer.getObject("StateEdit");
    citiscreen_OkButton = (JButton) m_AbaRenderer.getObject("OkButton");
    citiscreen_CancelButton = (JButton) m_AbaRenderer.getObject("CancelButton");
}

// Using the renderer

// Step 1: Declare a variable 'm_abaRenderer' that is a reference to the AbaRenderer
// For example:  AbaRenderer m_AbaRenderer = new AbaRenderer(sDocumentName, false, theGlobalInterfaceObject);
//
//              First parameter is name of the project document.
//              Second parameter is false (unless rendering inside design cockpit.
//              Third parameter is the global interface (if NULL, second parameter must be true
//
// Step 2: Load the project.
//
//              boolean bTestLoad = m_AbaRenderer.load();
//
// Step 3: You can set the language of the renderer like this:
//
//              m_AbaRenderer.setLanguage(HammerLanguagePresentation.DEUTSCH);
//
// Step 4: Render the interface.
//
//              m_AbaRenderer.renderInterface();
```

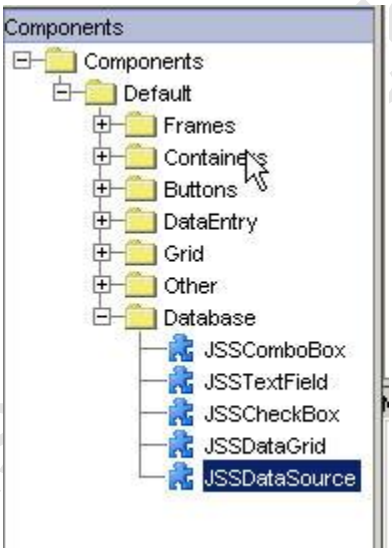
AbaRenderer Sample Code CityRendererSample.java

```
import java.io.*;
import java.awt.*;
import java.util.*;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.plaf.*;
import java.lang.reflect.*;
import java.awt.event.*;
import javax.swing.event.*;

import ch.abacus.lib.ui.*;
import ch.abacus.lib.ui.layout.*;
import ch.abacus.lib.ui.renderer.abaRenderer.*;

public class CityRendererSample {

 public AbaRenderer m_AbaRenderer;

// Declarations of variables for this user interface.

    private JLabel citiscreen_NameLabel;
    private JLabel citiscreen_CompanyLabel;
    private JLabel citiscreen_Address1Label;
    private JLabel citiscreen_Address2Label;
    private JLabel citiscreen_CityLabel;
    private JLabel citiscreen_StateLabel;
    private JTextField citiscreen_NameEdit;
    private JTextField citiscreen_CompanyEdit;
    private JTextField citiscreen_Address1Edit;
    private JTextField citiscreen_Address2Edit;
    private JTextField citiscreen_CityEdit;
    private JTextField citiscreen_StateEdit;
    private JButton citiscreen_OkButton;
    private JButton citiscreen_CancelButton;
// Assignments for this user interface

public void getReferences(){
    citiscreen_NameLabel = (JLabel) m_AbaRenderer.getObject("NameLabel");
    citiscreen_CompanyLabel = (JLabel) m_AbaRenderer.getObject("CompanyLabel");
    citiscreen_Address1Label = (JLabel) m_AbaRenderer.getObject("Address1Label");
    citiscreen_Address2Label = (JLabel) m_AbaRenderer.getObject("Address2Label");
    citiscreen_CityLabel = (JLabel) m_AbaRenderer.getObject("CityLabel");
    citiscreen_StateLabel = (JLabel) m_AbaRenderer.getObject("StateLabel");
    citiscreen_NameEdit = (JTextField) m_AbaRenderer.getObject("NameEdit");
    citiscreen_CompanyEdit = (JTextField) m_AbaRenderer.getObject("CompanyEdit");
    citiscreen_Address1Edit = (JTextField) m_AbaRenderer.getObject("Address1Edit");
    citiscreen_Address2Edit = (JTextField) m_AbaRenderer.getObject("Address2Edit");
    citiscreen_CityEdit = (JTextField) m_AbaRenderer.getObject("CityEdit");
    citiscreen_StateEdit = (JTextField) m_AbaRenderer.getObject("StateEdit");
    citiscreen_OkButton = (JButton) m_AbaRenderer.getObject("OkButton");
    citiscreen_CancelButton = (JButton) m_AbaRenderer.getObject("CancelButton");
}



public CityRendererSample(String paramPath) throws ch.abacus.lib.ui.renderer.common.HammerException
{
            initObject(paramPath);
}

public void initObject(String spath) throws ch.abacus.lib.ui.renderer.common.HammerException {
```

```
            System.err.println("Loading: "+spath);
            m_AbaRenderer = new AbaRenderer(spath, false, null);
            boolean bTestLoad = m_AbaRenderer.load();
            getReferences();
}

public static void main(String[] args) {
            CityRendererSample thisTest = null;
            try {
                        thisTest = new CityRendererSample(args[0]);
                        thisTest.m_AbaRenderer.renderInterface();
            } catch(Exception e) {
                        e.printStackTrace();
                        System.err.println();
                        System.err.println("USAGE: java -cp .;abalib.jar;exml.jar RendererSample <Project file path>");
            }
   }
}
```

6 Database Components

For this beta release, we decided on 5 basic database visual components: A DataSource, a text edit field, an alpha release of a checkbox and a basic grid. Like we pointed before, these classes are either derived directly from SwingSet classes or we have taken their code changed to fit out visual need, for example, the database grid is based on the SwingSet code with additional functionality for compatibility with our GUI builder.



6.1 JSSDataSource

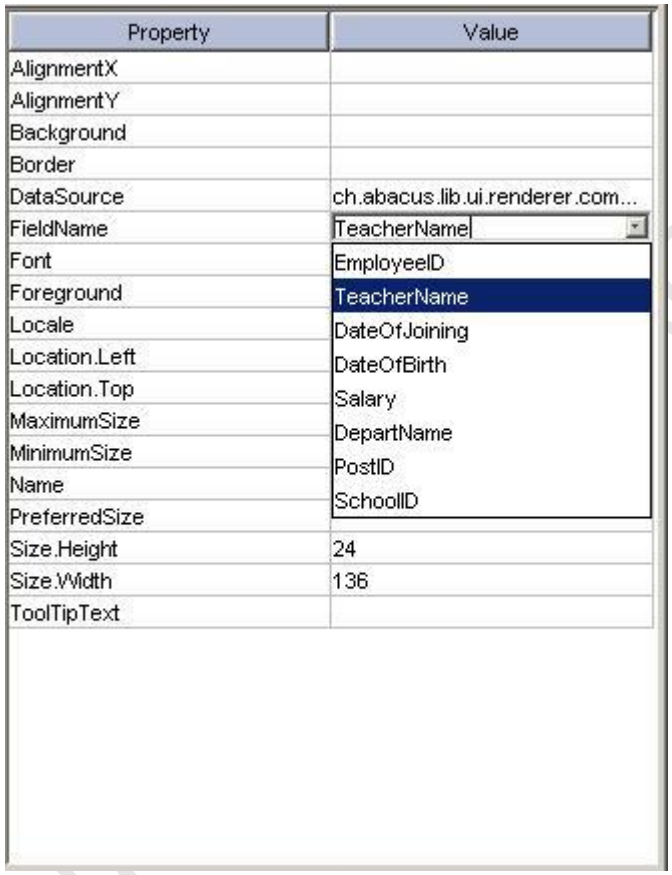
This the database connection and RowSet object handler, this is a non-visual component represented on the canvas by a transparent panel, keep in mind that we do not want to show this object at runtime only at design time. The class keeps a bidirectional cursor to the database using the RowSet class, this class has the following members:

Member	**Description**
setJDBCClassName	This the JDBC driver class name, must be in the classpath. Included as choices: **com.mysql.jdbc.Driver** **org.hsqldb.jdbc.Driver**
setJDBCURL	This is the URL database description for the JDBCdriver. For example: jdbc:mysql://localhost/registrar

setJDBCUserName	The database user with rights to the database.
setJDBCPassword	The user password.
setSQLCommand	The SQL command to execute when the object connects to the database.
execute	
previous	Previous record in the row set
next	Next record in the row set
clear	Clear visual objects and prepare to insert record.
insert	Insert a record into database
update	Update current record
delete	Delete current record
first	Go to first record in the row set
last	Go to last record in the row set

6.2 JSSTextField

This database visual object connects a database field to a text edit on the screen via the data source object, of importance with this object is the data source and the database field since these determine the database data to display.


Property
Value
AlignmentX
AlignmentY
Background
Border
DataSource
ch.abacus.lib.ui.renderer.com...
FieldName
TeacherName
Font
Foreground
Locale
Location.Left
Location.Top
MaximumSize
MinimumSize
Name
PreferredSize
Size.Height
24
Size.Width
136
ToolTipText
EmployeeID
TeacherName
DateOfJoining
DateOfBirth
Salary
DepartName
PostID
SchoolID

7 MySQL Example

7.1 Creating the sample DB with MySql

Login into MySql

Execute: mysql --user=root mysql

1) Create a new test "abacus" user with password "eli" so execute the following command:

*mysql> GRANT ALL PRIVILEGES ON *.* TO 'abacus'@'localhost'IDENTIFIED BY 'eli' WITH GRANT OPTION;*

2) Create demo database

mysql> create database registrar;

3)Execute:

mysql> use registrar;

4)Execute script:

mysql> \. mysql.registrar.sql

Keep in mind you may also use a full path for the script, depending where you installed the demo files.

5) Execute:
runproz drive:\path\registrar_op.jar

You should now see the sample application window. For further reference how to create this demo please take a look at the Flash live demos in LiveDemos.zip.

8 Menu Bar

Starting with revision 1.2, the user may create an application menu bar and menu at top of application window visually. Keep in mind that at design time you maybe able to drop the menu bar anywhere in the canvas, however after compiling the project, the menu bar will be attached to the top of the application window.

8.1 Creating the Menu Bar

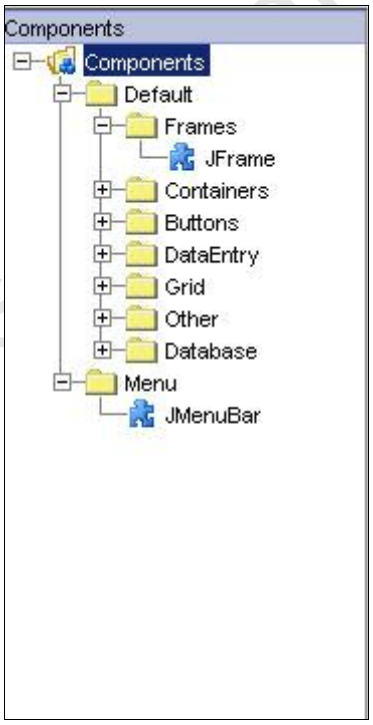


Fig 9.1

First make sure that the JMenuBar (Revision 1.2) is part of the components palette **Figure 9.1**, second select the JMenuBar component click on an existing JFrame within the canvas. Once you dropped the component on the canvas, add the top menus on the menu bar as in **Figure 9.2**.

8.2 Adding Menus to the Menu Bar

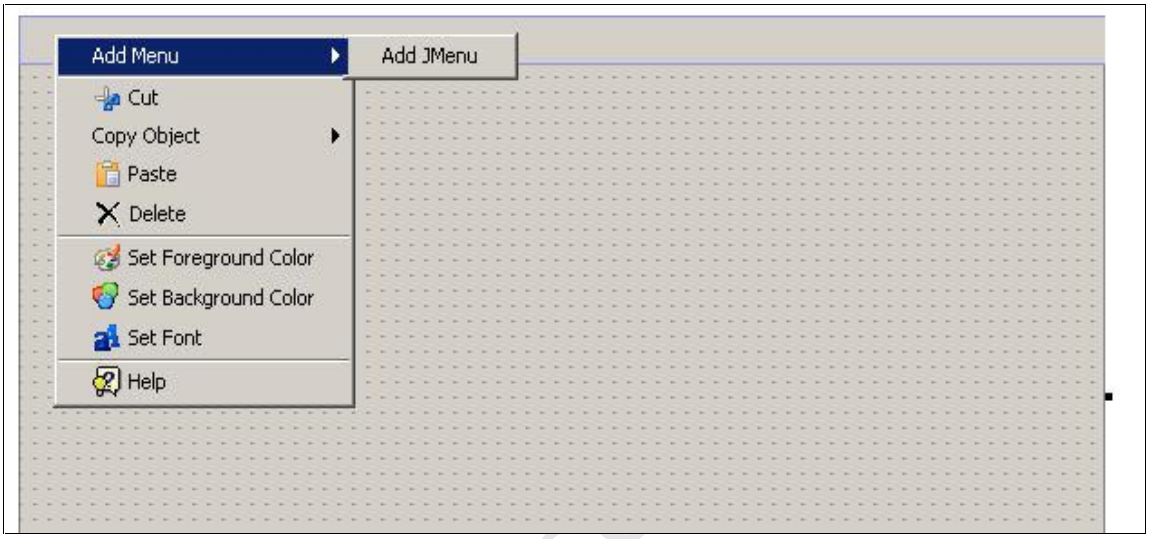


Figure 9.2

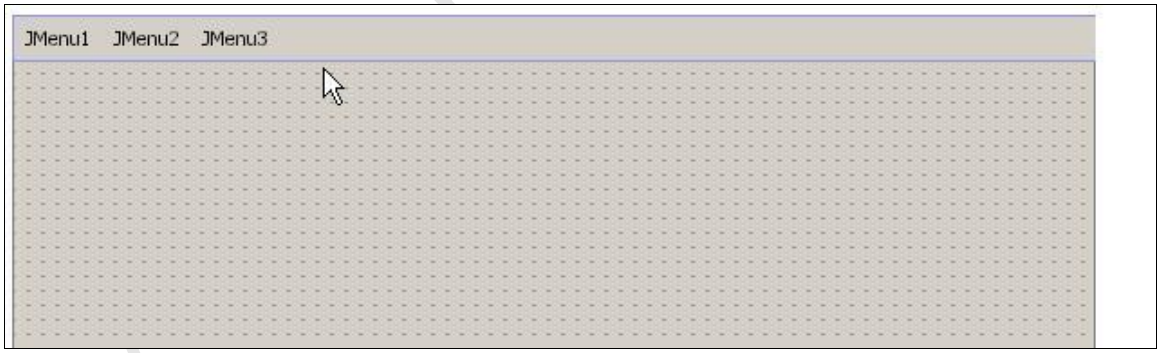


Figure 9.3

8.3 Adding mnemonics to Menu Bar

Figure 9.3 shows a sample screen after adding three JMenu objects to the canvas, once you have added the menus to the Menu bar, you may add the Mnemonics to the individual JMenu objects. Adding Mnemonics is very simple, first select an JMenu object from the object list then, select the "Mnemonic" property from the property table and a dropdown list will appear **Figure 9.4**, at this point you may select the "Hot Key" that will activate this menu, remember Mnemonics or Hot Keys are a combination of the **Alt key** and another key**,** for example (ALT + M).

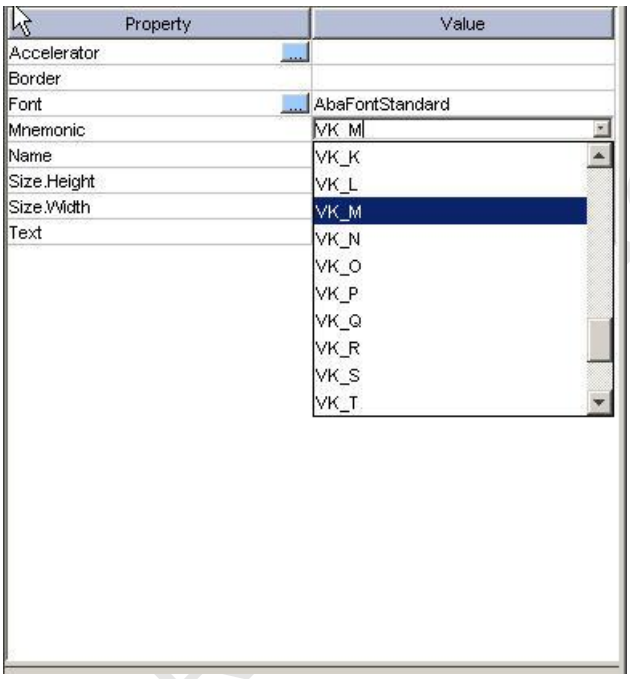


Figure 9.4

After adding a Hot Key to the JMenu object, press the render key to make sure that in fact you added a mnemonic as a result you should see an underscore corresponding the hot key as in **Figure 9.5**.

Figure 9.5

8.4 Adding menu items and separators

Once you have added the JMenu objects to the menu bar, the next step is to add the menu items (JMenuItem) , this is accomplished by selecting the menu bar object on the object list as in **Figure 9.6** and selecting "Add JMenuItem".

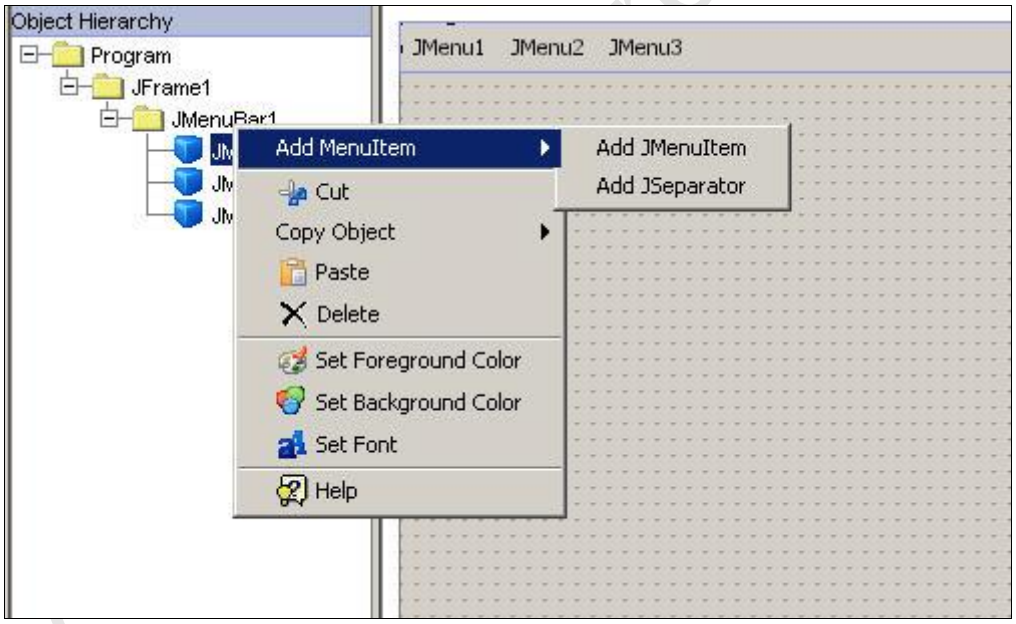


Figure 9.6

At this point you can preview the menu by clicking on the render  button **Figure 9.7.**

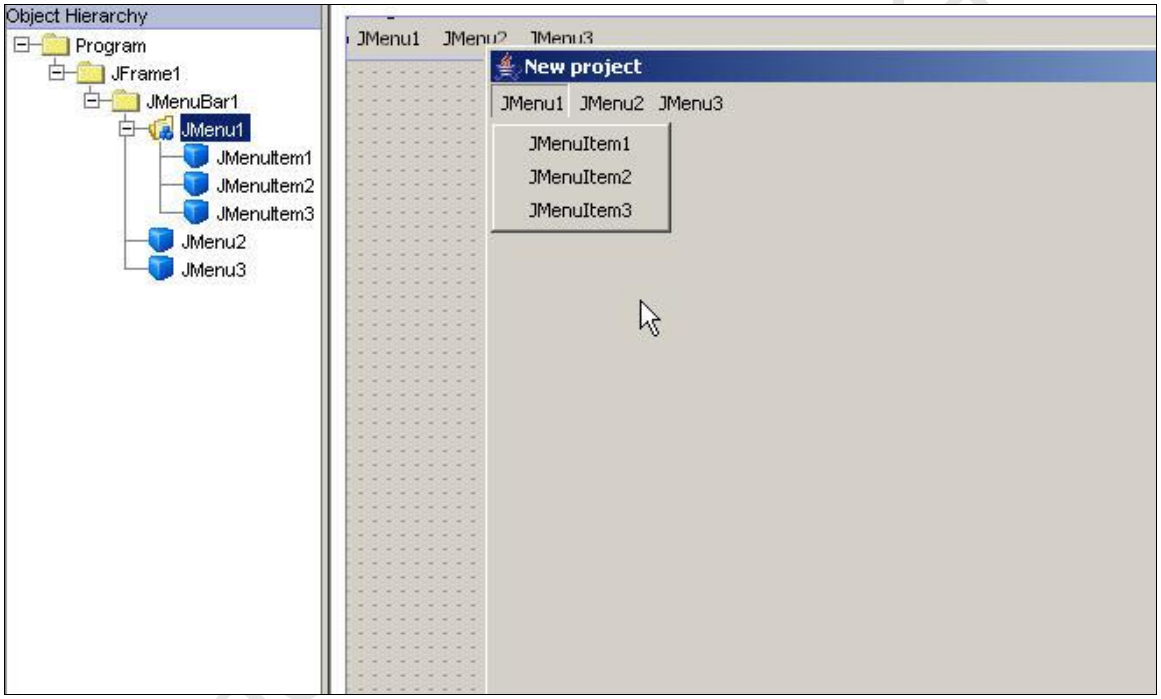


Figure 9.7

Adding menu separators to the menu is also very simple, once again right click on the JMenuItem object and hit "Add JSeparator" **Figure 9.6**, rendering the project should yield something similar to **Figure 9.8.**

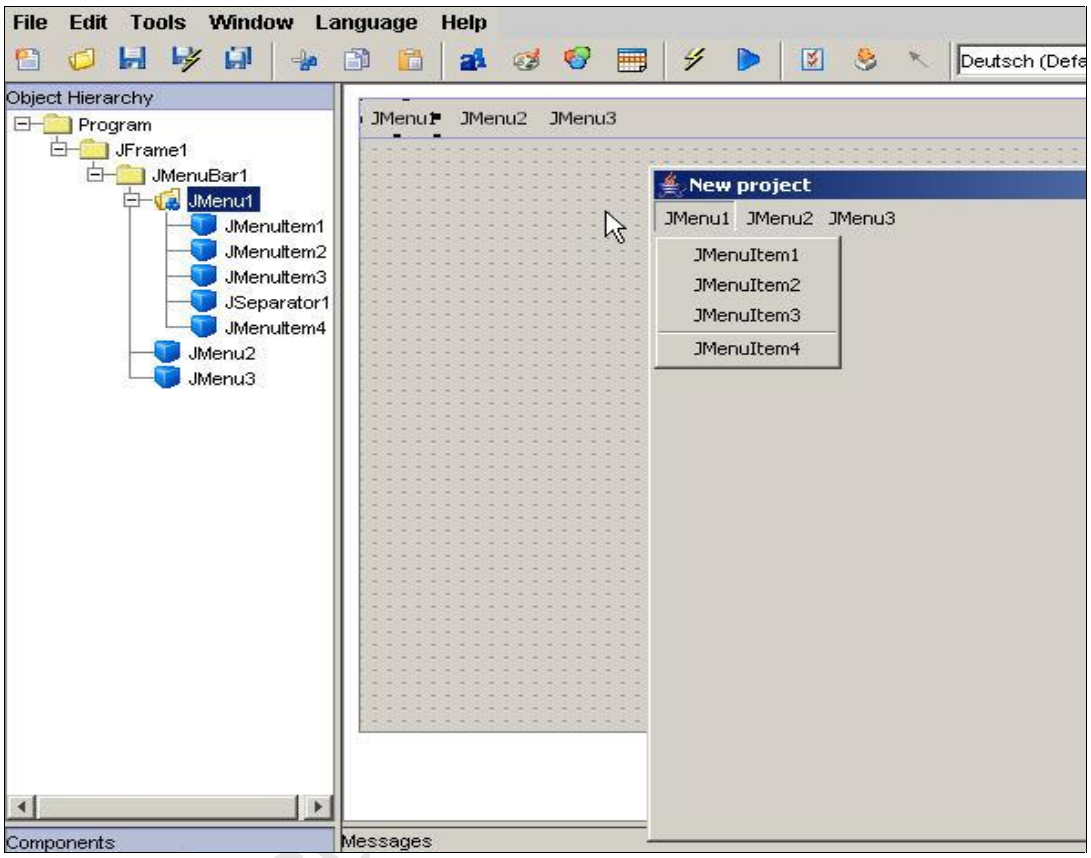


Figure 9.8

8.5 Adding accelerator keys to menu items

To add accelerator keys you need to focus your attention to the property table for the JMenu objects, there you will find the "Accelerator" property **Figure9.9**.

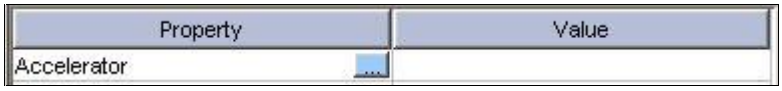


Figure 9.9

Click on the dialog helper dialog button ... to activate the menu item Key Control Editor dialog **Figure 9.10**.

Figure 9.10

The Key Control Editor provides an easy interface to connect hot keys to menu items and the action for each menu item. The **Menu Item** column has all menu items for the corresponding menu, in other words, these are the JMenuItems children objects of the JMenu object.

The **Keys** column activates a drop down list with the available keys for the particular environment, for Windows the default is the "CRTL" key, therefore each letter corresponds to the CTRL + hot key combination, **Figure 9.11**.

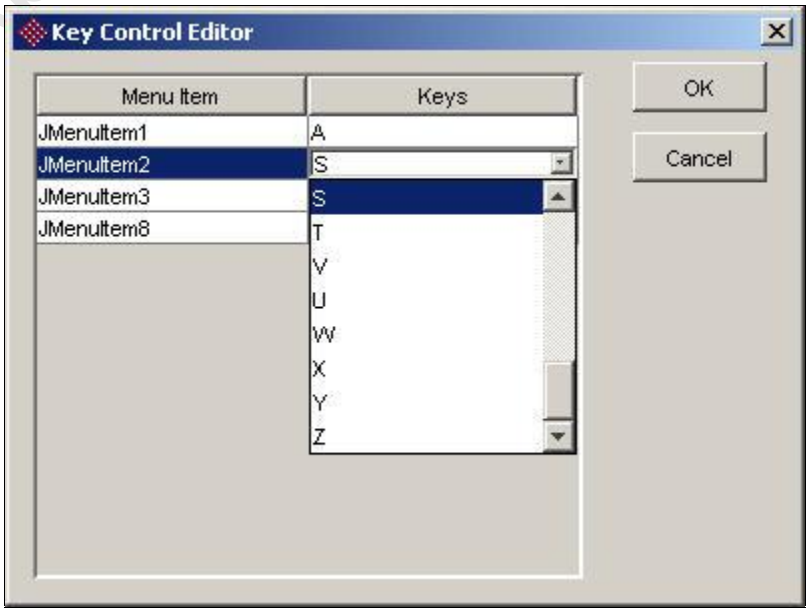


Figure 9.11

After setting the accelerators you may render the application  and you should see the menu with its corresponding accelerator as in **Figure 9.12.**

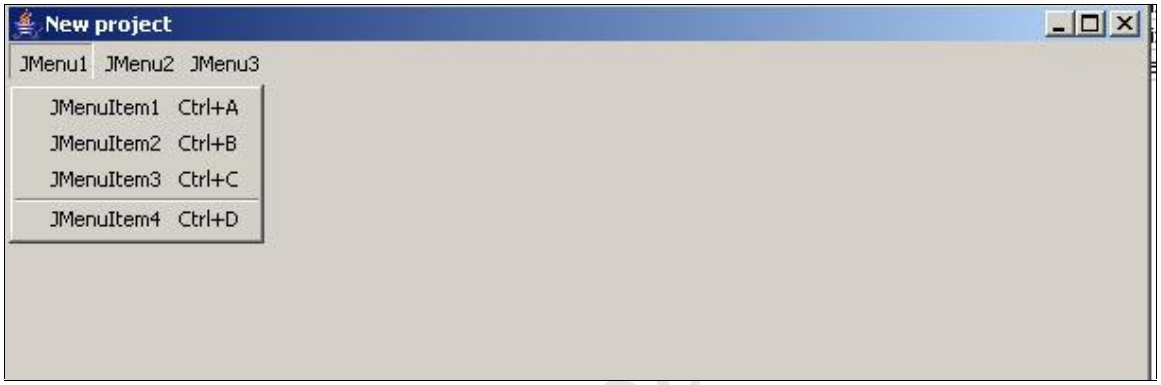


Figure 9.12

8.6 Adding actions to menu items

The last thing is to add "what to do" when a menu item is selected or its accelerator is pressed, this is done via the "Action" event listener **Figure 9.13.** First, make sure to select the desired JMenu and click on the "actionPerfomed" item from the event list, once you done that you may add your Java code like in **Figure 9.13**.

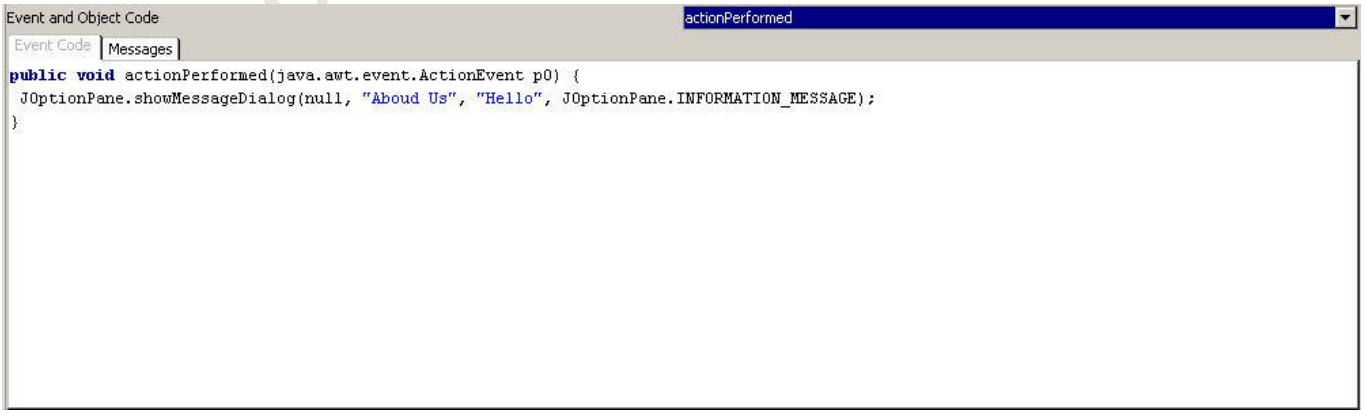


Figure 9.13

This code will be executed when the corresponding item is click or the accelerator is pressed, as in **Figure 9.14**.

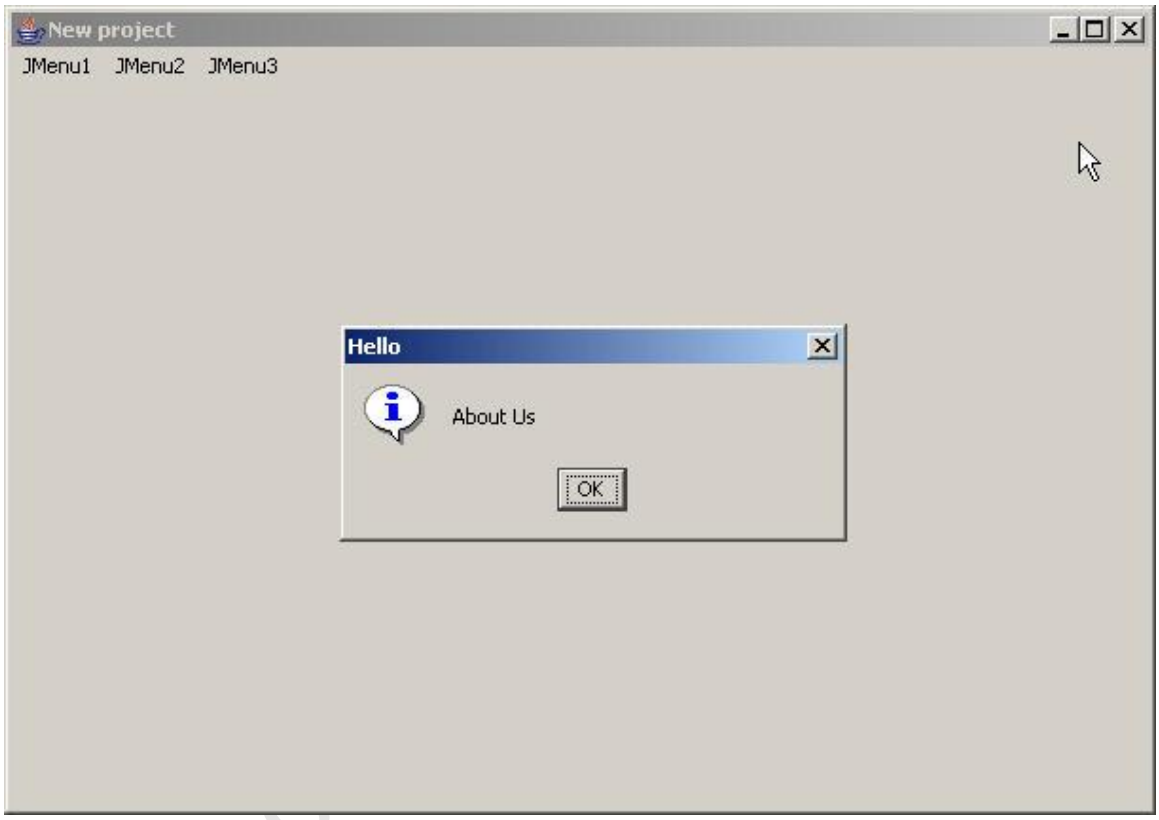


Figure 9.14

9 Outlook Button Bar

The Outlook button bar component visual component was added to give a nicer visual look to the applications **Figure 10.1**. The component makes it very for the application developer to create the tool bar.

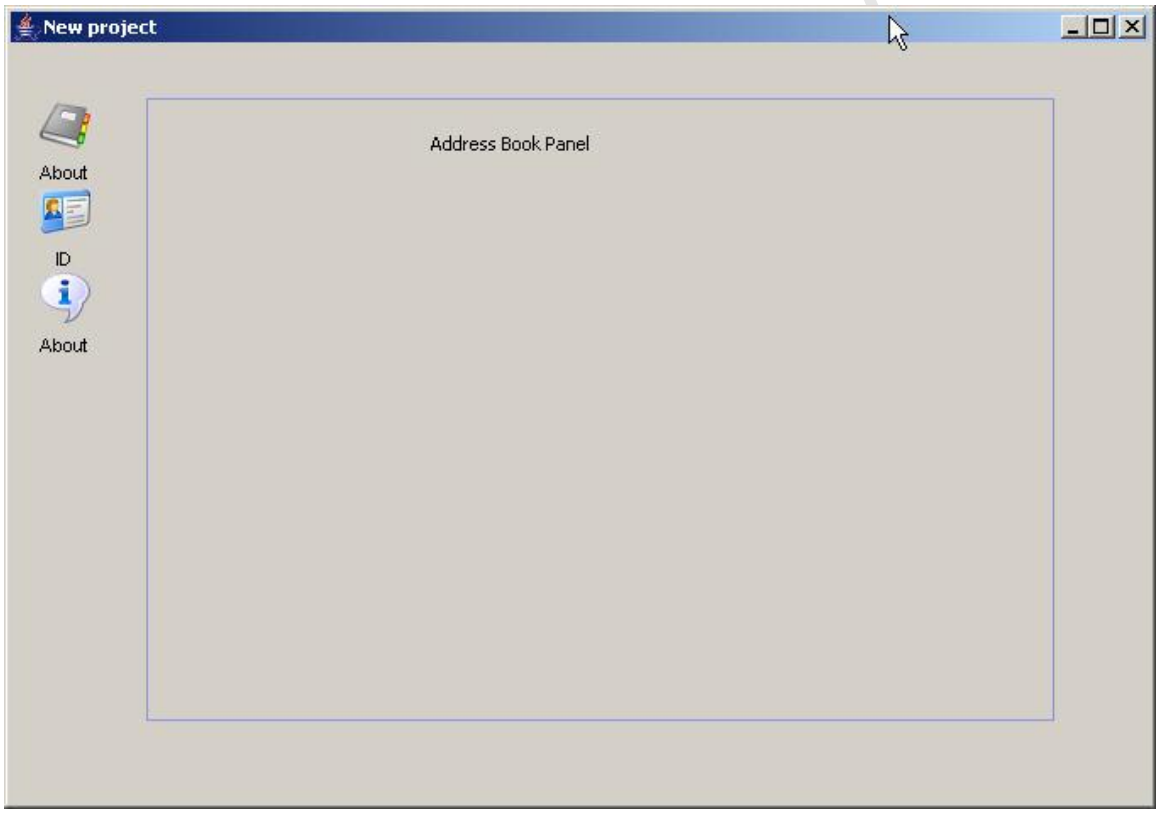


Figure 10.1

9.1 Creating the Outlook Bar

First click on the components panel and select the "Menu" item then click on the **JOutlookBar** component **figure 10.2** and drag the JOutlookbar to the canvas (JFrame).

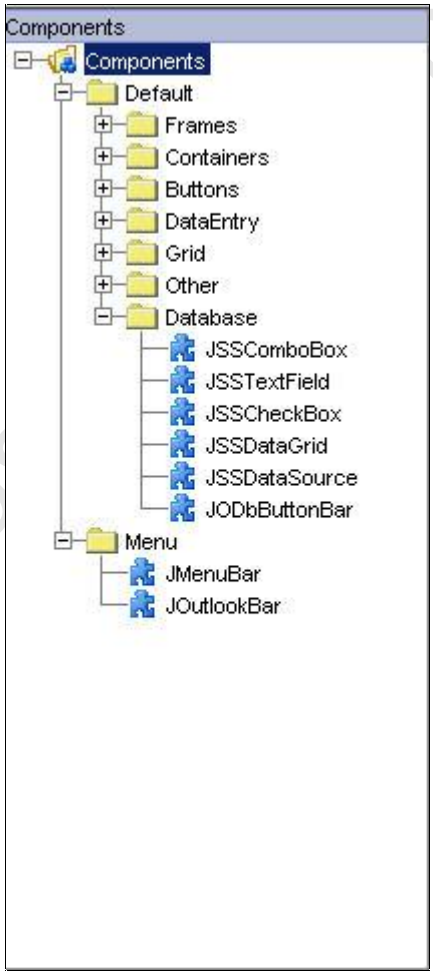


Figure 10.2

You will now see and empty Outlook bar on the JFrame, right click and add a button to the button bar as in **Figure 10.3**.

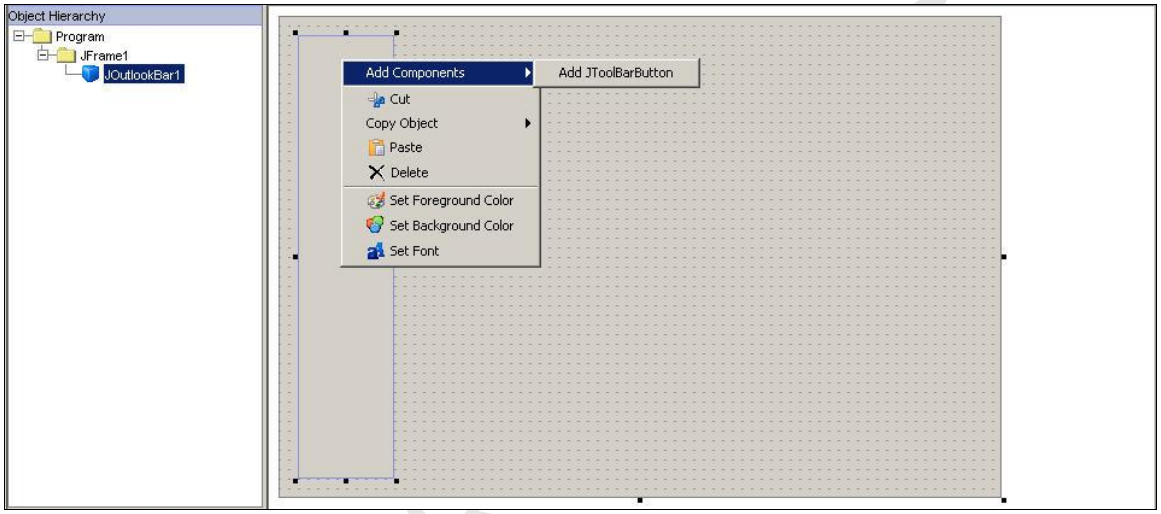


Figure 10.3

Once the button bar has been added to the button bar then button properties need to be set and you will have a workable outlook bar, **Figure 10.4**.

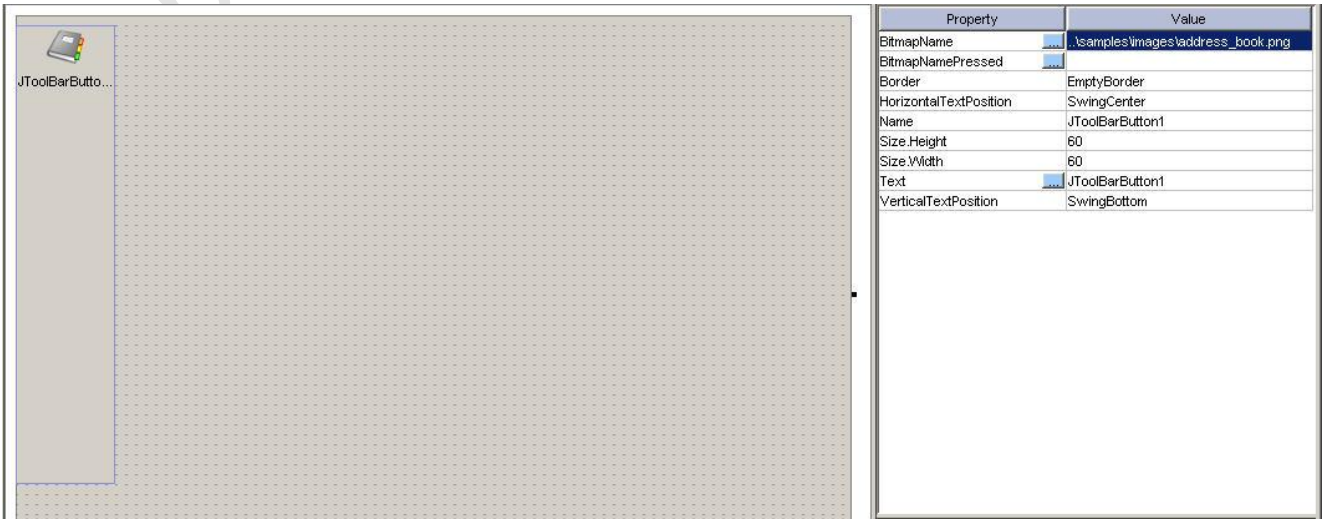


Figure 10.4

9.2 *Editing* ***the Outlook buttons***

The following properties are the most important GUI properties for each button in the button bar **Figure 10.4**.

Member	**Description**
BitmapName	This is the full path for the image render in the button component. A full path is required, if the path is correct then, the image will be rendered on the button.
BitmapNamePressed	This is the full path for the image render in the button component when the user presses the button. A full path is required, if the path is correct then, the image will be rendered when pressing the button.
HorizontalTextPosition	This places the text in the button either: **SwingLeft** – Left side of the button **SwingCenter** – Center of the button **SwingRight** - right side of the button
VerticalTextPosition	This places the text in the button either: **SwingTop** Above the image - **SwingCenter** - on top of the image **SwingBottom** – below the image

9.3 *The Auxiliary Object property*

The auxiliary property is a private property of type ***Object*** and it is controlled externally by the function ***setAuxObject*** and retrieves by the ***getAuxObject*** method. This property is useful when activating panel and there is need to keep track of the active object. In the samples directory starting with revision 1.3, we have included the **outlookbar.proj** sample project in order to illustrate the use of the **Auxiliary Object** property.

First, the code in **actionPerformed** event for each of the three buttons on the Outlook bar:

JToolBarButton1:

```
public void actionPerformed(java.awt.event.ActionEvent p0)
{

  JPanel p = (JPanel)JOutlookBar1.getAuxObject();

  if(p!=null)
          p.setVisible(false);

  JOutlookBar1.setAuxObject(JPanel1);
  JPanel1.setVisible(true);

}
```

Figure 10.5a

JToolBarButton2:

```
public void actionPerformed(java.awt.event.ActionEvent p0)
{

  JPanel p = (JPanel)JOutlookBar1.getAuxObject();

  if(p!=null)
          p.setVisible(false);

  JOutlookBar1.setAuxObject(JPanel2);
  JPanel2.setVisible(true);
}
```

Figure 10.5b

JToolBarButton3:

```
public void actionPerformed(java.awt.event.ActionEvent p0)
{

  JPanel p = (JPanel)JOutlookBar1.getAuxObject();

  if(p!=null)
          p.setVisible(false);

  JOutlookBar1.setAuxObject(JPanel3);
  JPanel3.setVisible(true);
}
```

Figure 10.5c

The **actionPerformed** event is executed whenever the user presses on the individual button on the outlook bar, therefore executing the code above. On all three event handlers, first we retrieve the AuxObject property containing the last panel select and setting to visible so we use the AuxObject as a place holder for the last selected panel this way we know which panel to deactivate then set AuxObject with current selected panel and last we set the current select panel to visible.

Second, it is essential for every application to initialize the AuxObject with a valid value, otherwise it will contain a **null** as default**.** In the outlook bar example, it is initialized with the first JPanel namely *JPanel1* **Figure 10.5**.

JFrame1 focusGained event.

```
public void focusGained(java.awt.event.FocusEvent p0)
{
  super.focusGained(p0);
  JOutlookBar1.setAuxObject(JPanel1);
}
```

Figure 10.6

Open and compile outlookbar.proj.

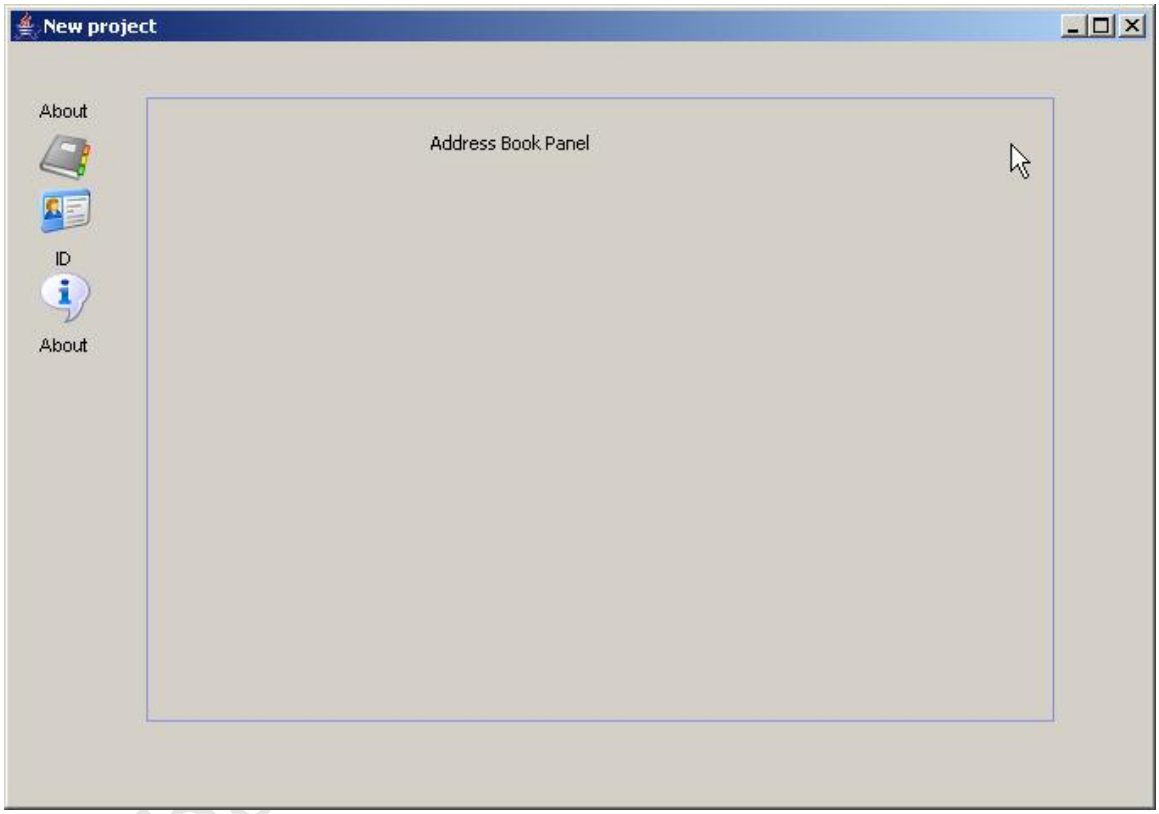
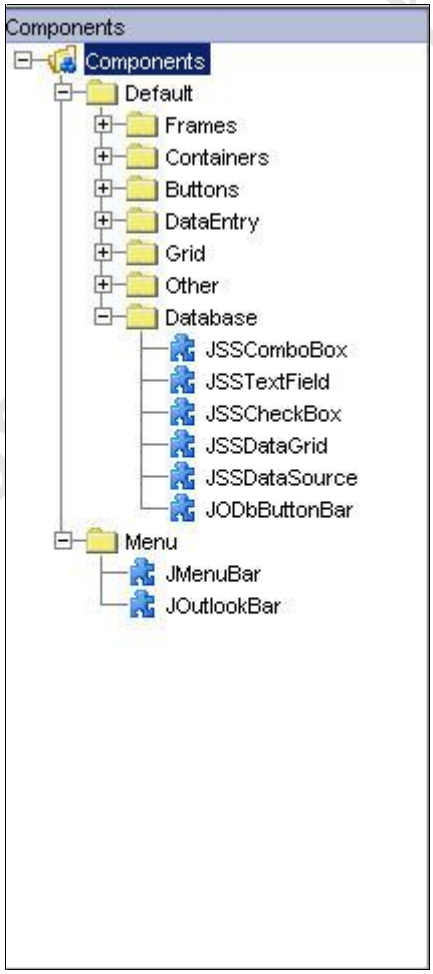


Figure 10.7

Then execute runproz c:\abaguibuilder-1.3\samples\outlookbar.jar **Figure 10.7.**

10 Database Button Bar

This component is very similar to other database "VCR" components that allow the user to move forward/backward, clear , insert and delete. The component requires a JSSDatasource, and thus all the messages are sent to the datasource.



Once the JODbButton is selected and dropped on the canvas, select the datasource and connect to a datasource. **Figure 11.2**

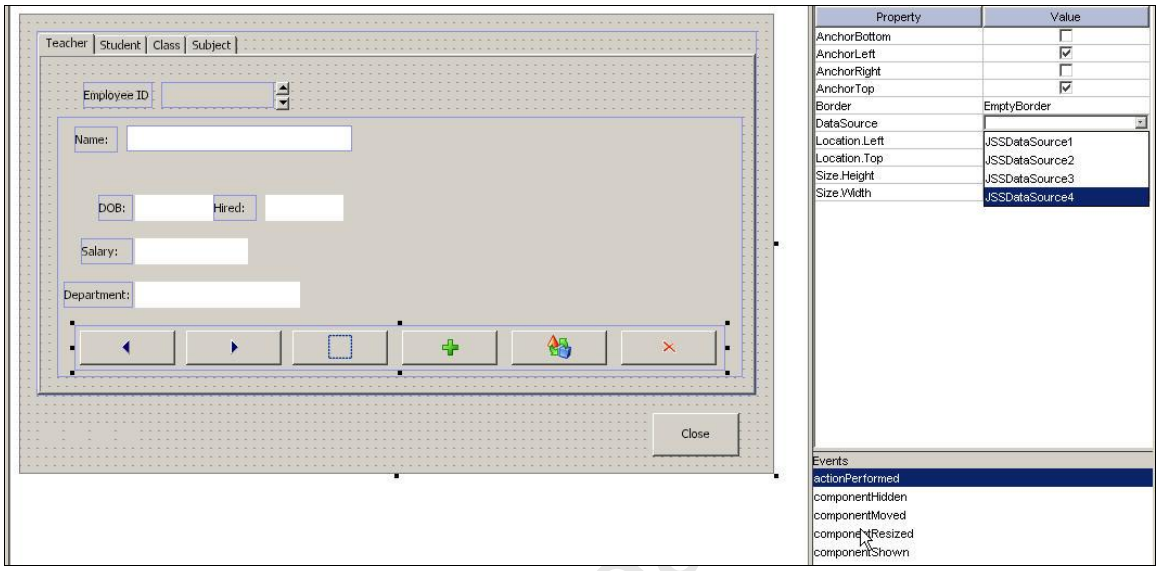


Figure 11.2

11 MetaData Editor

The AbacusGuiBuilder keeps its class components definitions and it properties in an XML file called the **metadata.meta** and to edit this file the user must unzip the abalib.jar, edit the metadata file and add it back to the jar. Usually a developer edits the metadata file in order to add a new visual component, to add or edit a component's property or add a new constant. For example, if a application developer needs to add a new JDBC driver, a new the metadata <JDBC> constant must be added, in fact this is why we decided to add this tool to the Abacus builder.

In order to eliminate the cumbersome process of unzipping, editing and adding the metadata back into the jar, we developed a visual tool called the MetaData Editor to managed the data in the metadata file, the tool is included and you can start it by executing the metaedit.bat in the bin directory.

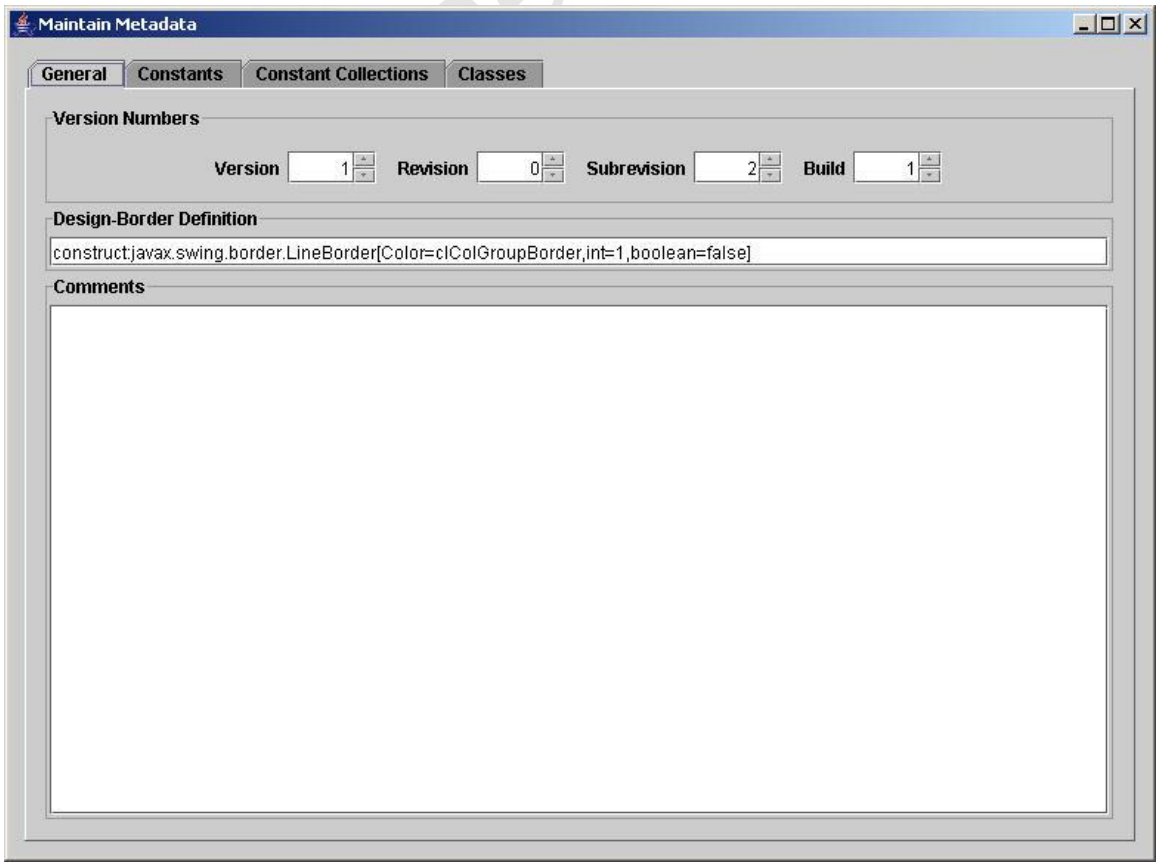


Figure 12.1

11.1 MetaData Editor Constants

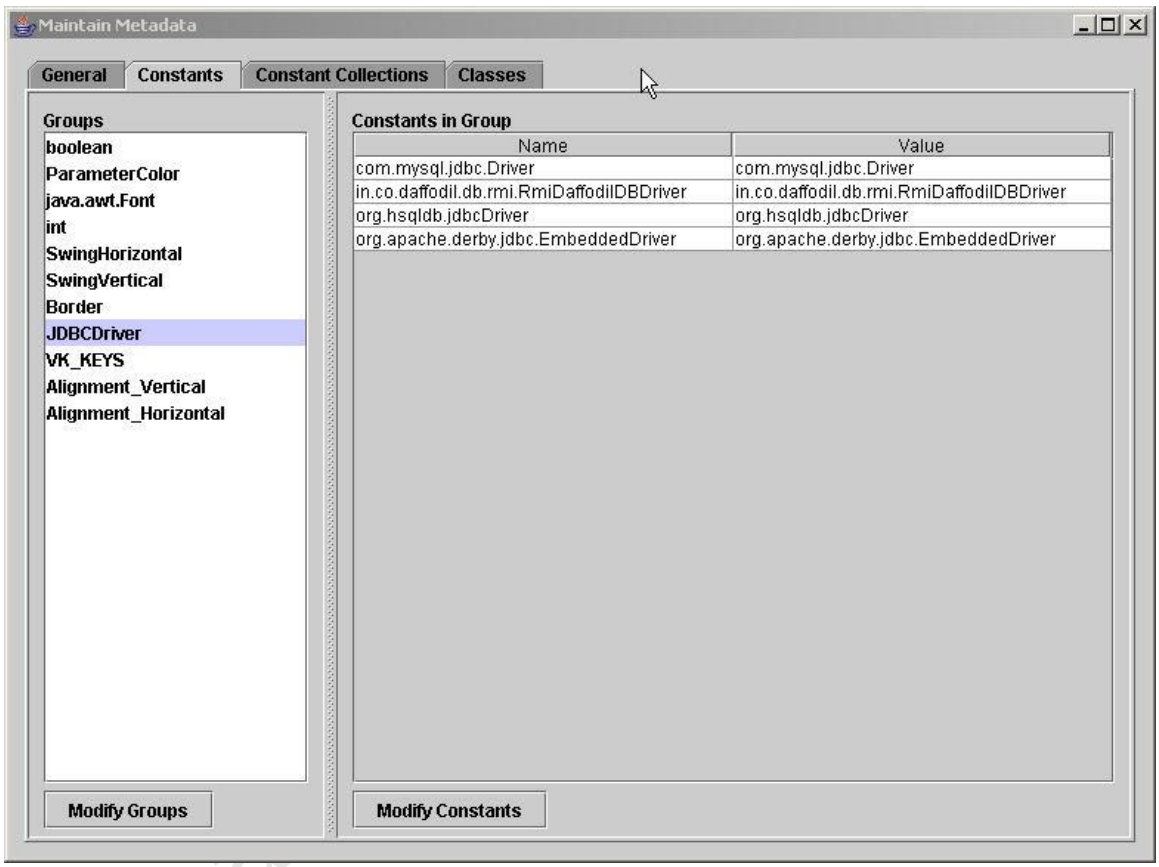


Figure 12.2

The editor contains three sections: ***Constants***, ***Collections*** and ***Classes***. The first section, the ***Constants,*** contains all the data items displayed on the selection boxes on the AbaGuiBuilder's property panel. For example, the JDBC properties on Figure 12.2 show up on the gui builder like in **Figure 12.3**

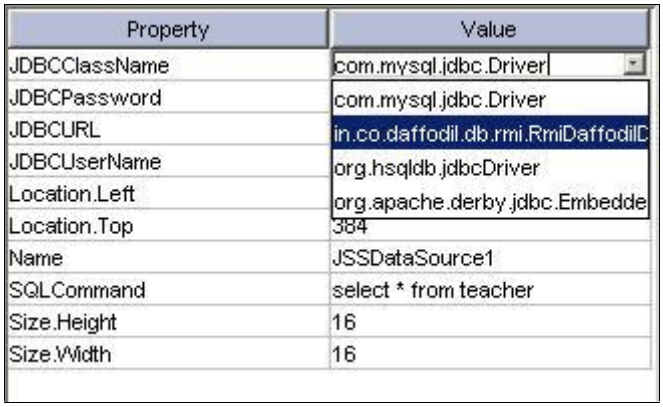
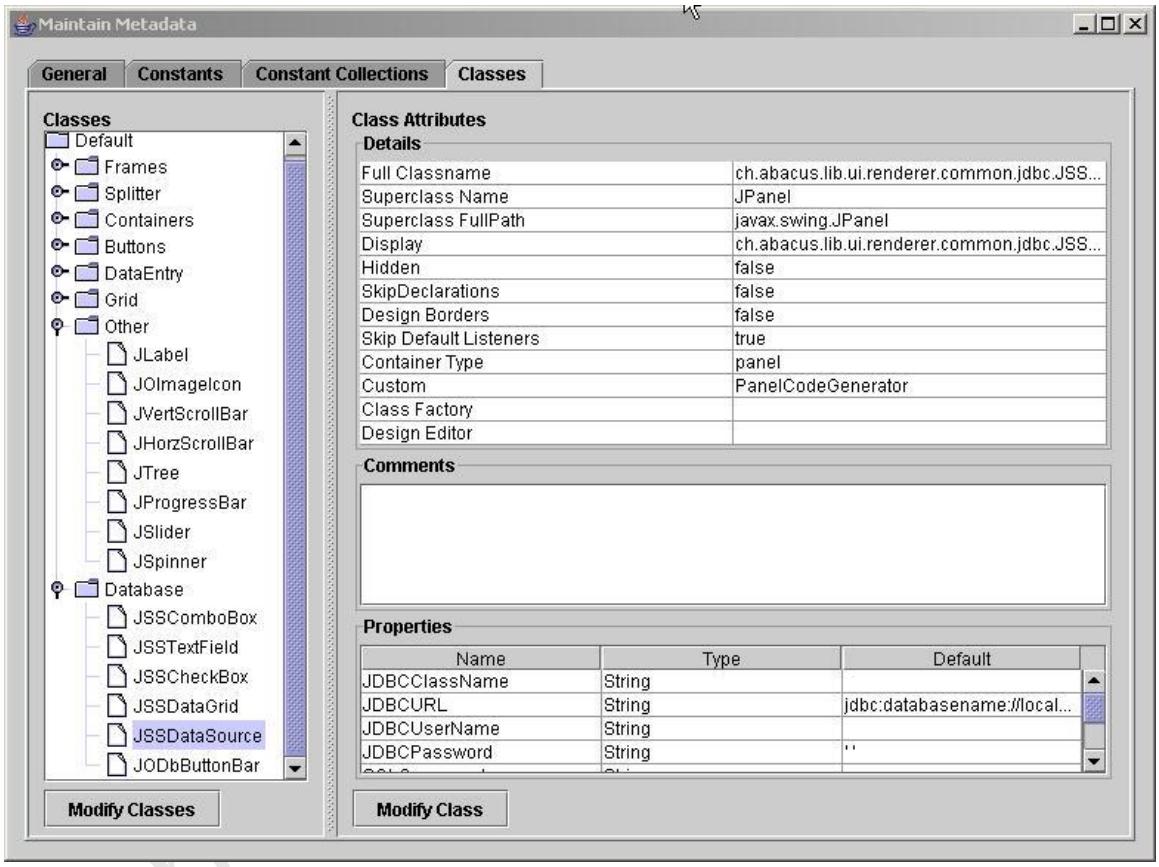


Figure 12.3

11.2 MetaData Editor Classes

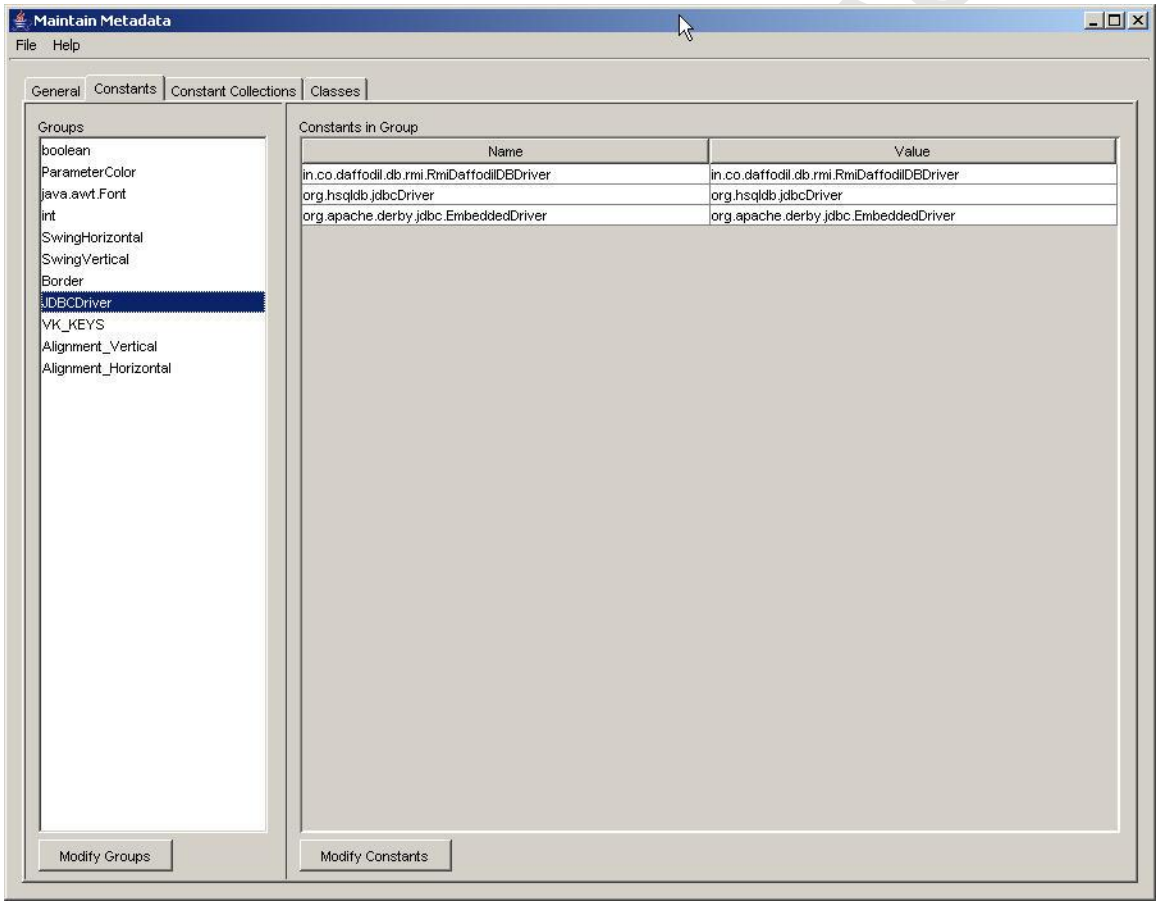


This panel represents all the available visual components in the Abacus Gui Builder and their properties, in future releases the developer we will add editing and adding functionality to this panel, for now it gives the developer an idea to the direction we are moving.

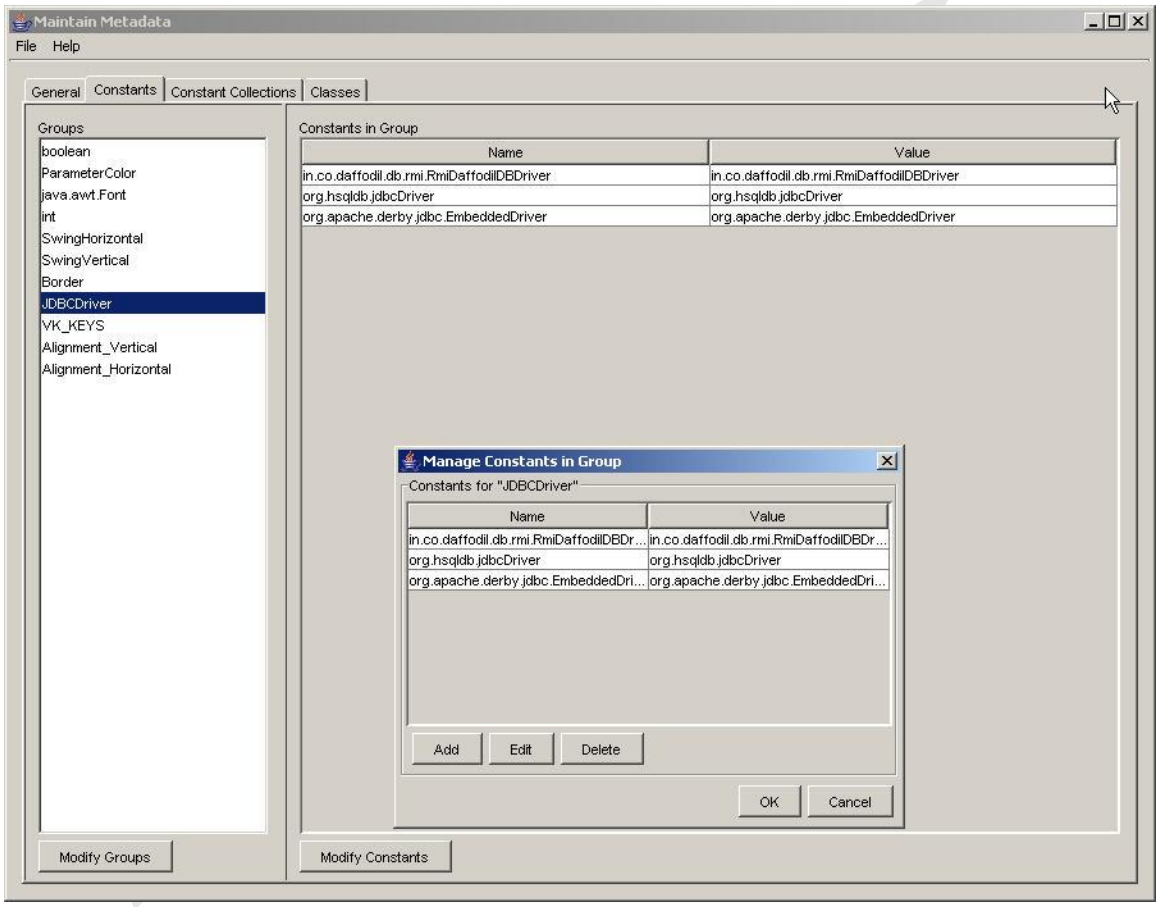
11.3 Adding a new JDBC driver string

For this sample we will use the MS-SQL JDBC driver:

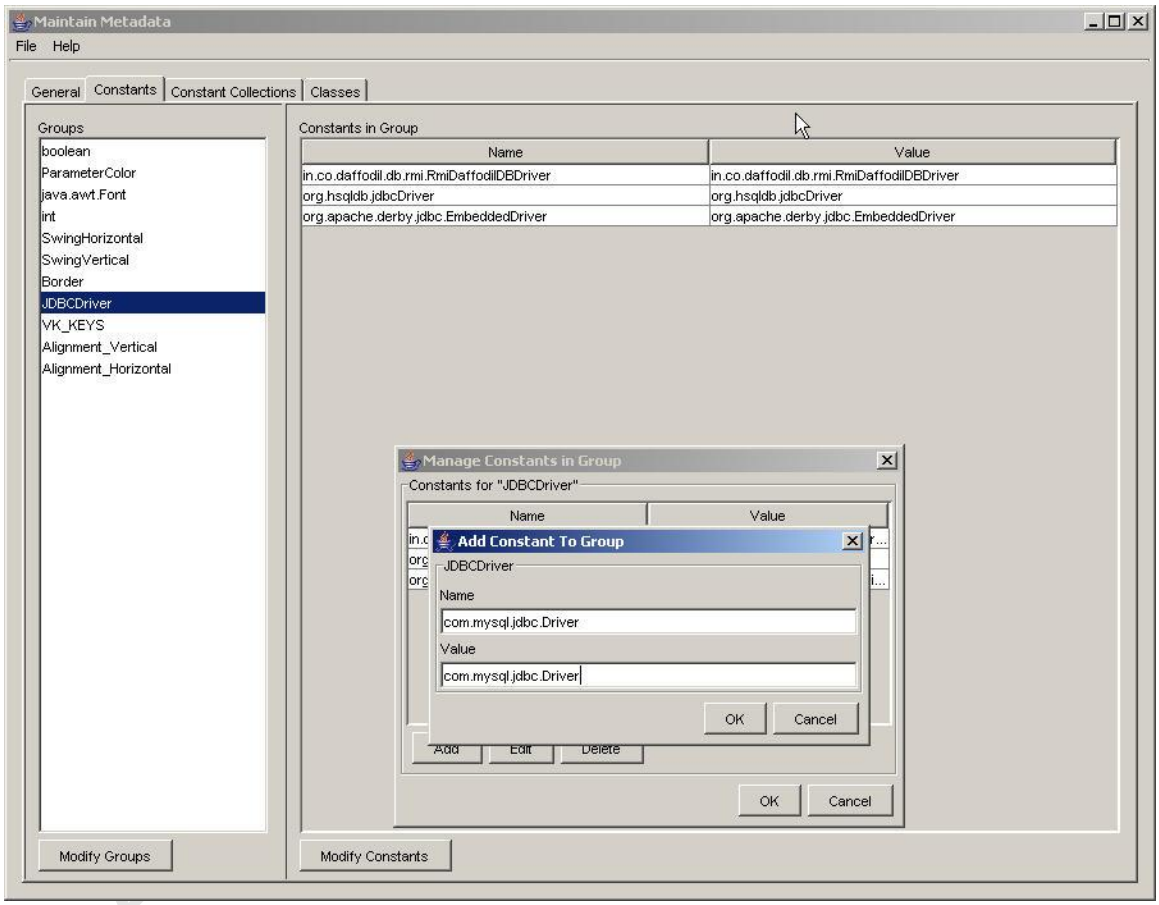
1) Run metaedit (.bat or .sh) , select the **constants** tab and click on JDBCDriver.



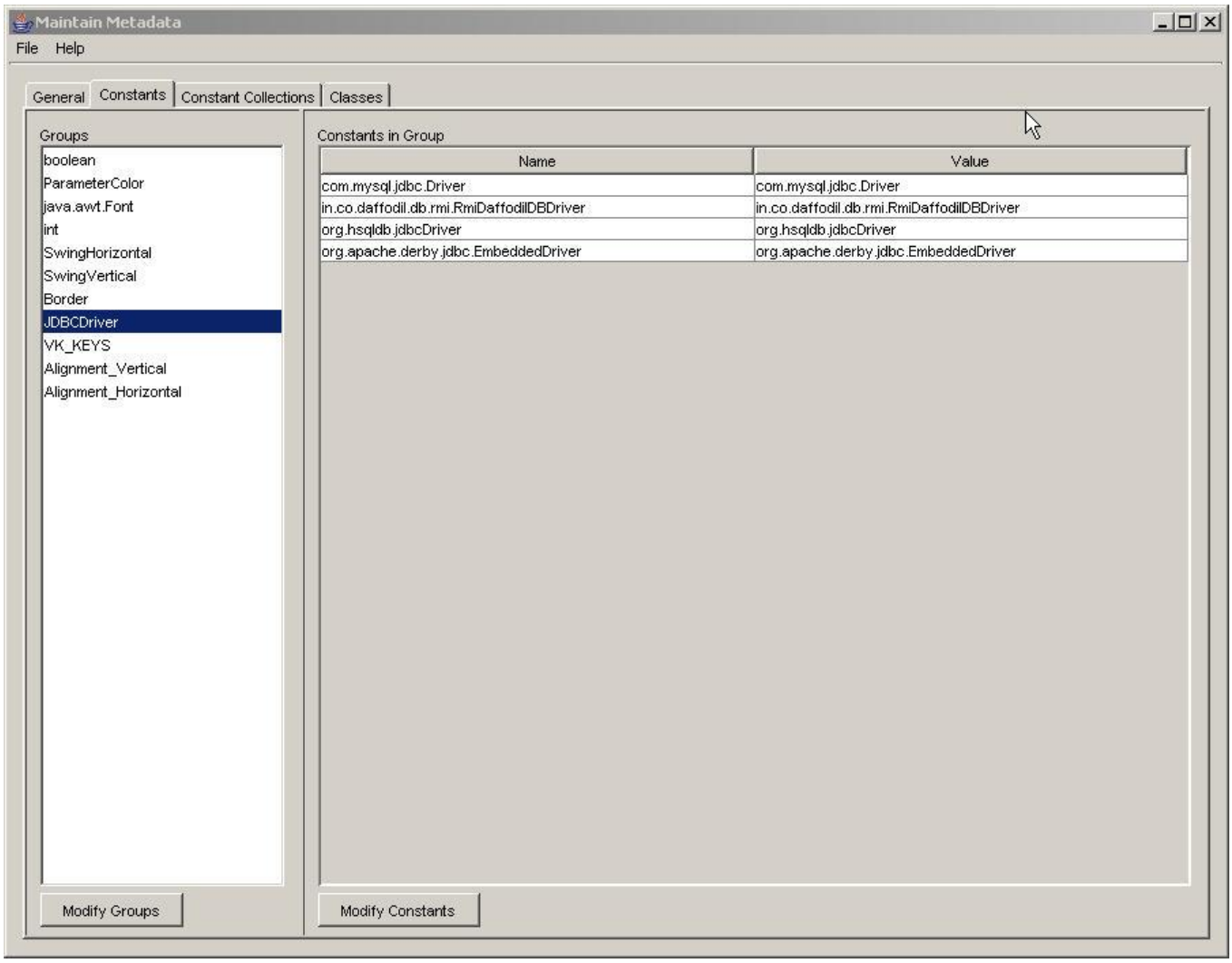
Next click on the “Modify Constants” button and click the “Add” button:



Now type the ClassName to be loaded, for our example, we shall use *com.mysql.jdbc.driver:*



Your list should look like this:



12 UDF Object Editor

12.1 User Defined Functions per Object

Starting with release 1.5, the AbaGuiBuilder adds supports for User Defined Functions (UDF) and data support for each visual object. This means that the developer can add their own functions and/or data to each component via the GUI in addition, we have incorporated support for external imports via the GUI.

12.2 How to add UDFs

First let's load the sample application **customer.proj (figure 11 1)** from the samples directory and we will add a new function "showDialog" to the JFrame1 object. The "showDialog" will be used executed when we click on the either button on the Frame. On previous versions, the developer had to code the functionality twice, on each button's actionPerformed event.

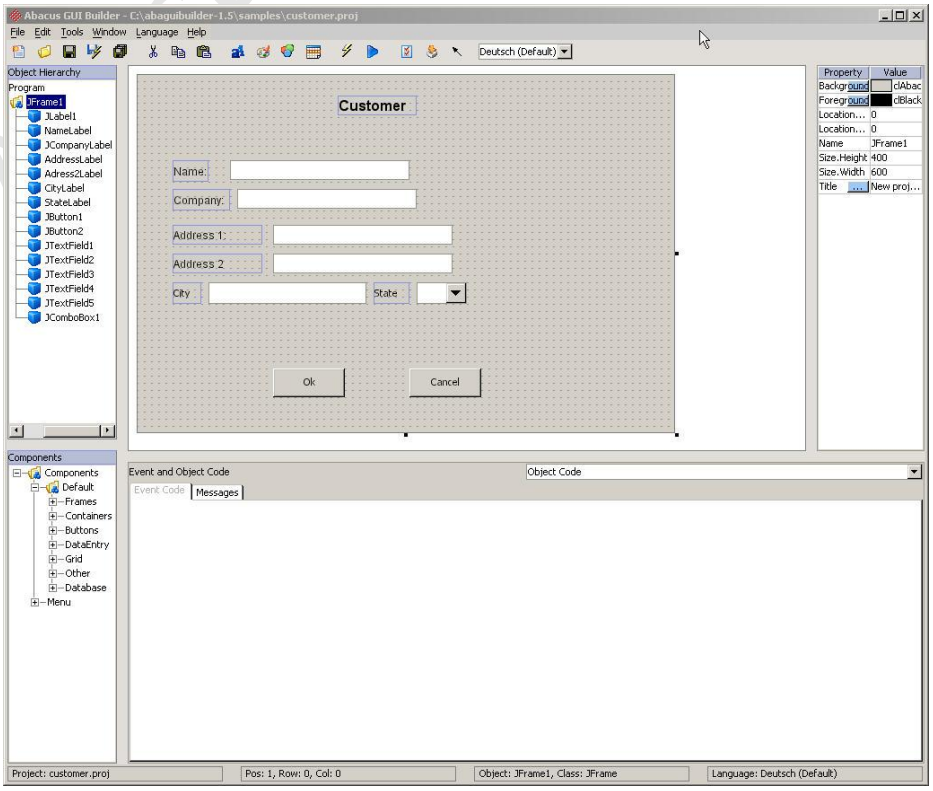


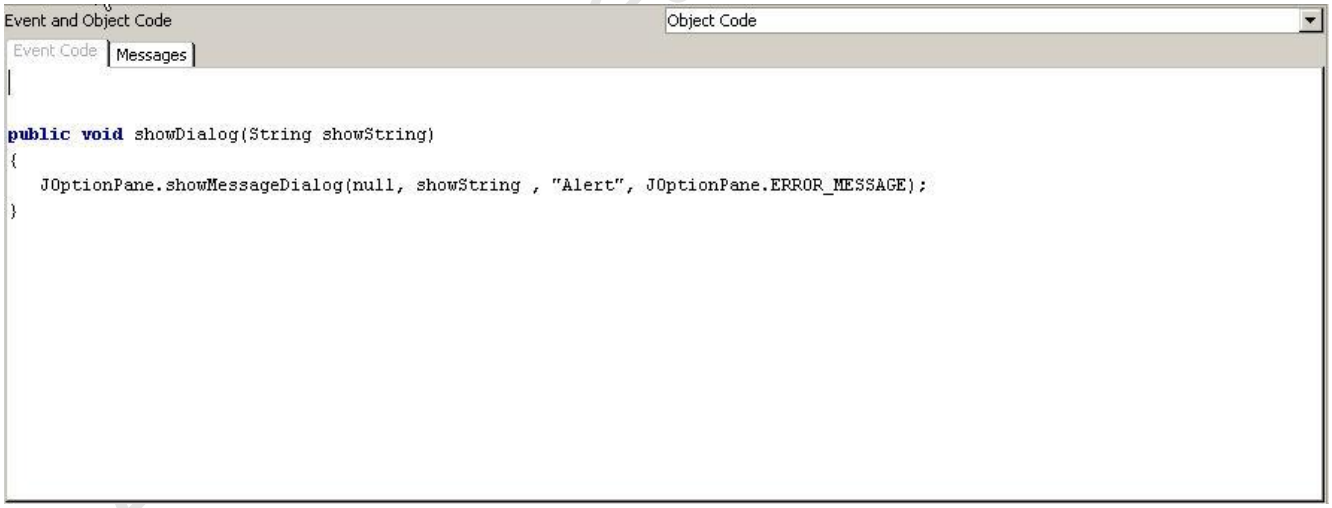
Figure 11 1

12.3 Adding the function to the JFrame

Starting with version 1.5, we can code the function in the container and call the function from the "children" objects. So in our example, we will add the following function to the frame:

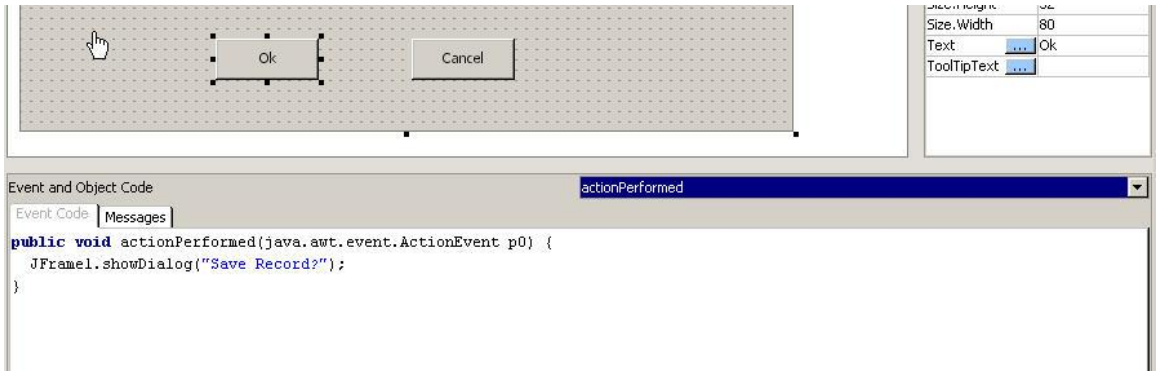
```
public void showDialog(String showString)
{
  JOptionPane.showMessageDialog(null, showString , "Alert", JOptionPane.ERROR_MESSAGE);
}
```

And we do this by first selecting the JFrame1 as the current object then we focus at bottom area labeled "Event and Object Code" and select the Object code on the combox and type the function above in the Event Code editor like in **figure 11.2**.



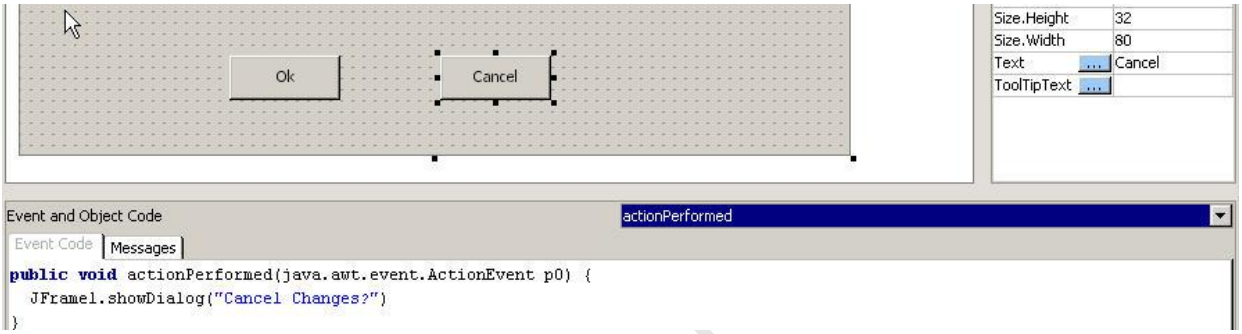
Next select the button **JButton1** and click on the "Event Code" tab and add the following code:

```
JFrame1.showDialog("Save Record?");
```

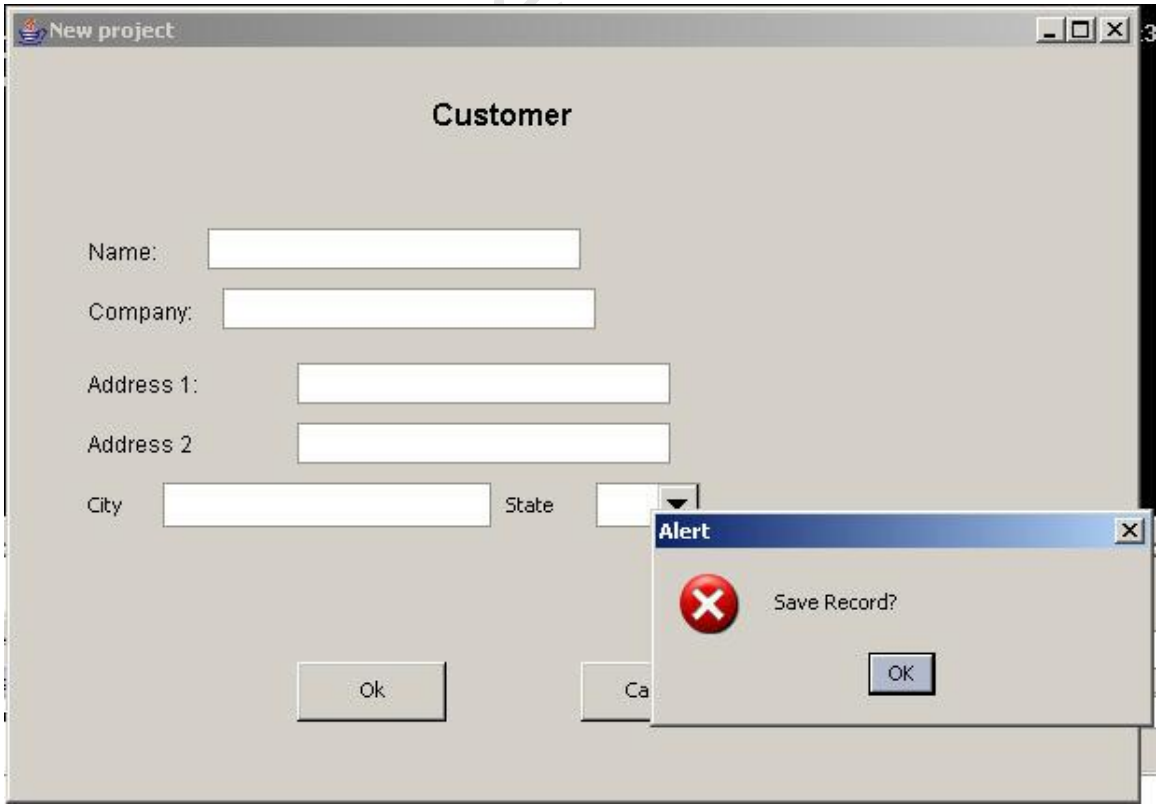


Next select the button **JButton2** and click on the "Event Code" tab and add the following code:

JFrame1.showDialog("CancelChanges?");

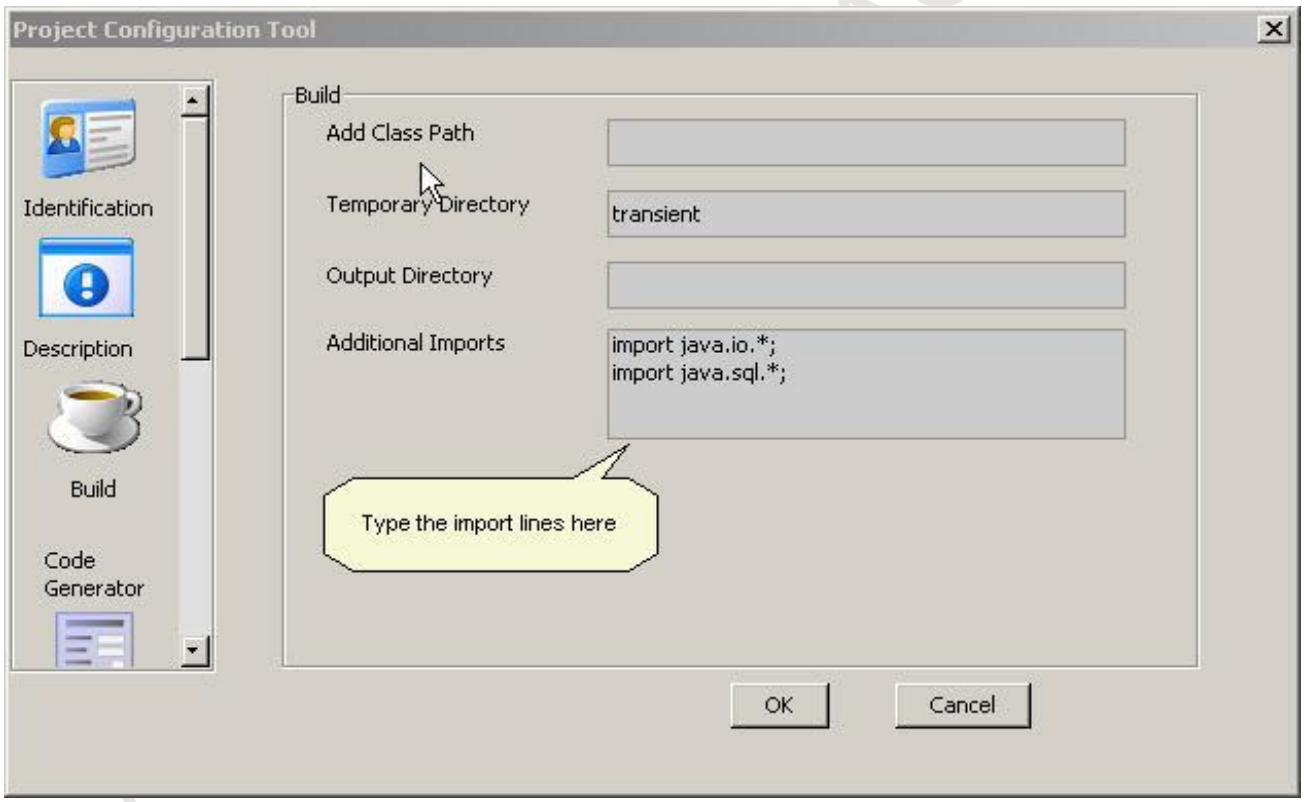


Next select "Save with Compile" (as customer5) and run the project using the command line: **"runproz \abaguibuilder-1.5\samples\customer5.jar"**



12.4 Adding external imports

It became apparent once the **Object Code** feature was implemented that external imports support was necessary, therefore a new option was added to the via **project -> build** options on the UI. **Figure 11.4**



The above dialog comes from the sample project **customer5.proj,** this project demonstrates the use external imports by including the JDBC and IO libraries. The customer5.proj user defined code retrieves the values from the text editors and saves them onto a MySQL table called customer.

The JFrame1 object has three private functions **connect**, **clearItems** and **saveToDatabase** in addition to two private data members *conn* a Connection JDBC object and *st* a Statement JDBC object, **Figure 11.5**. Therefore, in order to compile customer5.proj project the "java.io and java.sql" must be included.

```
// Data members.
// Database connection.
private Connection con = null;.
private Statement st;.
.
public boolean connect().
{.
  boolean isConnected = false;.
  try.
  {.
    Class.forName("com.mysql.jdbc.Driver");.
    con=DriverManager.getConnection(.
    "jdbc:mysql://localhost/test?user=abacus&password=eli");.
    if(con!=null).
    {.
      st = con.createStatement();.
      isConnected = true;.
    }.
  }.
  catch(Exception e).
  {.
    e.printStackTrace();.
  }.
  return isConnected;.
}.
// This function clears all the text fields.
// on the JFrame.
public void clearItems().
{.
.
  String defaultClear = "";.
  JTextField1.setText(defaultClear);.
  JTextField2.setText(defaultClear);.
  JTextField3.setText(defaultClear);.
  JTextField4.setText(defaultClear);.
  JTextField5.setText(defaultClear);.
}.
.
public void saveToDatabase().
{.
  if(st==null).
  connect();.
  if(st!=null).
  {.
    try.
    {.
      String sLineRecord="\"" + JTextField1.getText() + "\"" + "," + "\"".
      + JTextField2.getText() +.
      "\"" + "," + "\"" + JTextField3.getText() + "\"" +.
      "," + "\"" + JTextField4.getText() +  "\"" + "," + "\"".
      + JTextField5.getText() + "\"";.
.
      String insertCmd =.
      "INSERT INTO customer(COLUMN_1, COLUMN_2 ,COLUMN_3,COLUMN_4,COLUMN_5) VALUES(".
      + sLineRecord + ")" ;.
      st.executeUpdate(insertCmd );.
    }.
    catch(Exception e).
```

Figure 11.5

13 Importing Visual Components

Starting with version 1.6 visual components (Beans) may be imported directly from a jar in the path, this process builds the XML component definition and adds to our global component definitions via the custom side-by-side XML files (s-b-s).

The import object MUST have a default constructor for the import facility to work, if there is not a default constructor we recommend you create an intermediate class that provides one, for example, the JFreeChart implementations were created with this method.

First, let's take a look at a very simple visual component in order to illustrate the importing process. For our first example, we will use the class **JOvalButton** derived from the Swing class JButton, **JOvalButton** looks like Figure 1:

Figure 1

The JOvalButton draws an oval in its paint procedure:

```
import javax.swing.*;
import java.awt.*;

public class JOvalButton extends JButton
{
   public JOvalButton()
   {
      super();
      setBorder(BorderFactory.createEmptyBorder());
   }

   public JOvalButton(String text)
   {
      super(text);
      setBorder(BorderFactory.createEmptyBorder());
   }

   public void paint(Graphics g)
   {
      super.paint(g);
      int w= getWidth();
      int h= getHeight();
      g.drawOval(0, 0, w, h);
   }

}
```

You can find the class above in the CastComponents.jar located samples\os.repository.

First, we need to import the ***castcomponents.jar*** into the Abacus GUI Builder using the IDE importer dialog, next activate the dialog by selecting the "import classes" button on the menu bar  (Figure 2) and select the /samples/os.repository.

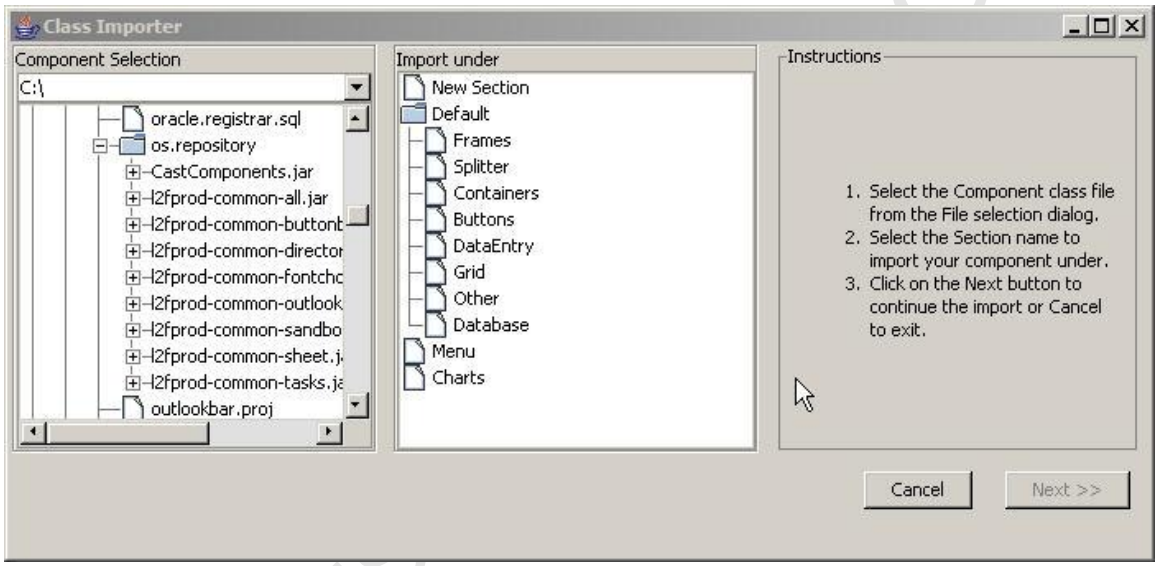


Figure 2

Next, select the JOvalButton.class located in the os.repository directory and select "New Section" for Import under. (Figure 3). "*Import Under*" points to a section in the GUI builder's class palette, we use to insert the class reference into a new palette section.

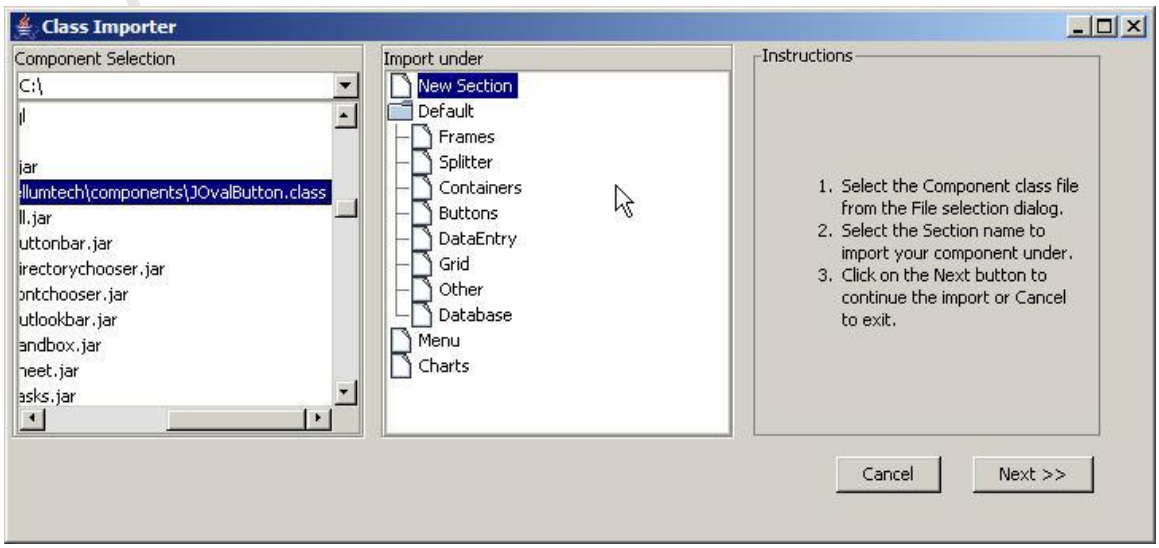


Figure 3

13.1 Defining interface

After selecting a class, we need to choose its properties and events we would like to expose to the GUI builder (Figure 4) and then we select Import.

NOTE: The composite option (upper left check box) is reserved for containers object that have children object which you would like to see activated at the design time.

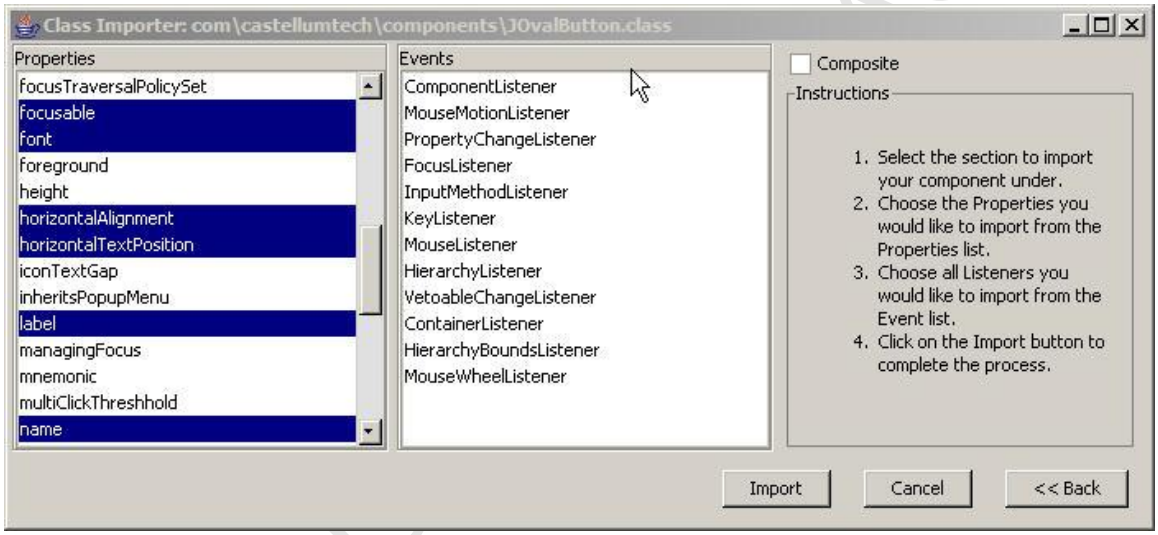


Figure 4

13.2 Adding to a new section in component palette

Next, we create a new component palette section by typing “Castellum” in the name in the Component section dialog (Figure 5).

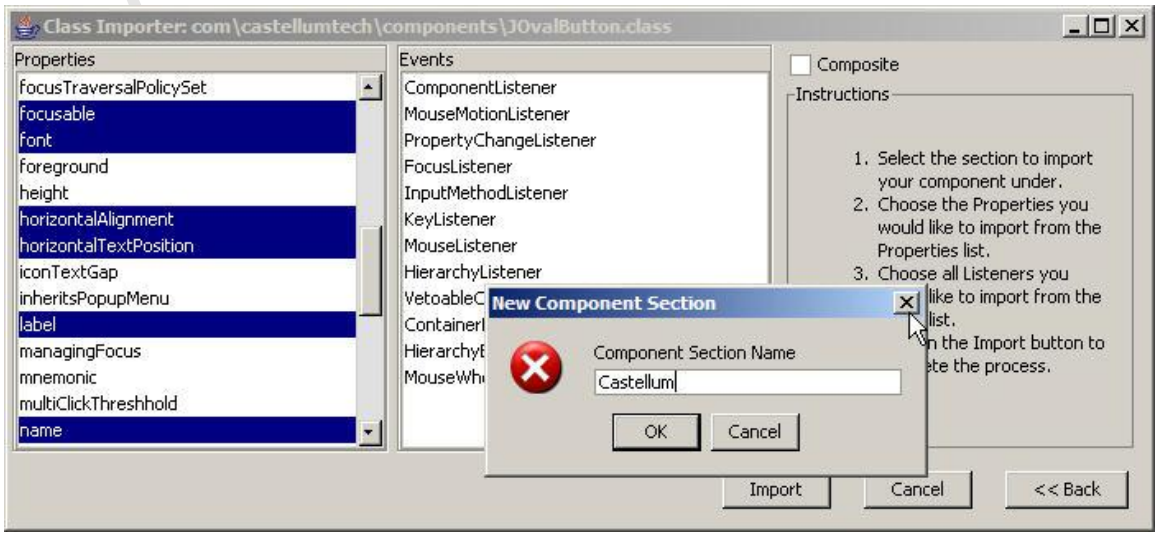


Figure 5

13.3 Storing interface definition

One a palette section has been selected, we need to save the object definition to disk, in our example we shall save it on the /samples/os.repository directory as in Figure 6.

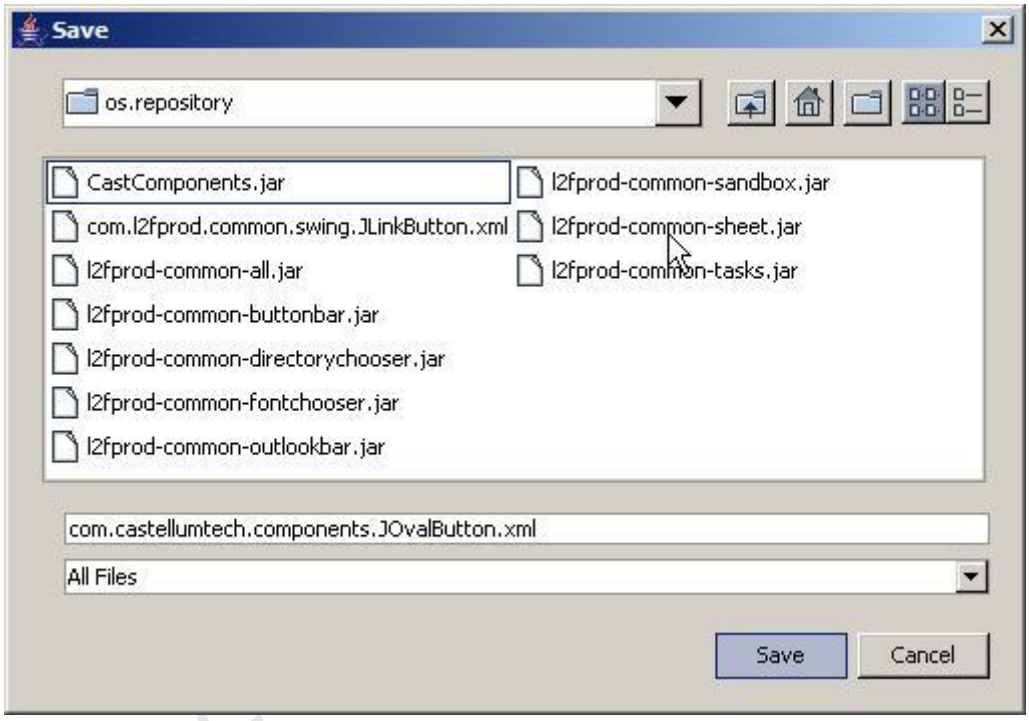


Figure 6

Click on the "Save" button and the class definition is now save in JOVALButton.XML file, this file contains the GUI builder metadata definition for the imported class. At this point, we may use the JOvalButton to create application from within the GUI builder so let's make sure to open the "Components" palette and make sure it looks like Figure 7.

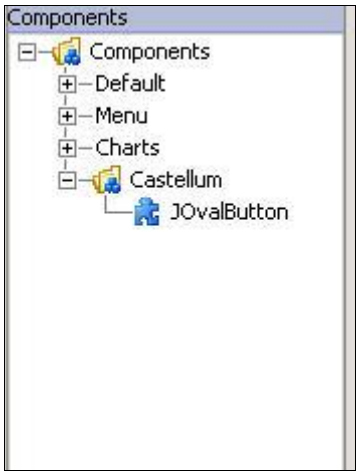


Figure 7

13.4 Creating a sample project

At this point, we are ready to create a form using the imported component:

1. Create a new project.
2. Add a JFrame
3. Drag a new JOvalButton to the canvas
4. “Save and Compile” the project as “OvalButton” (Figure 8).

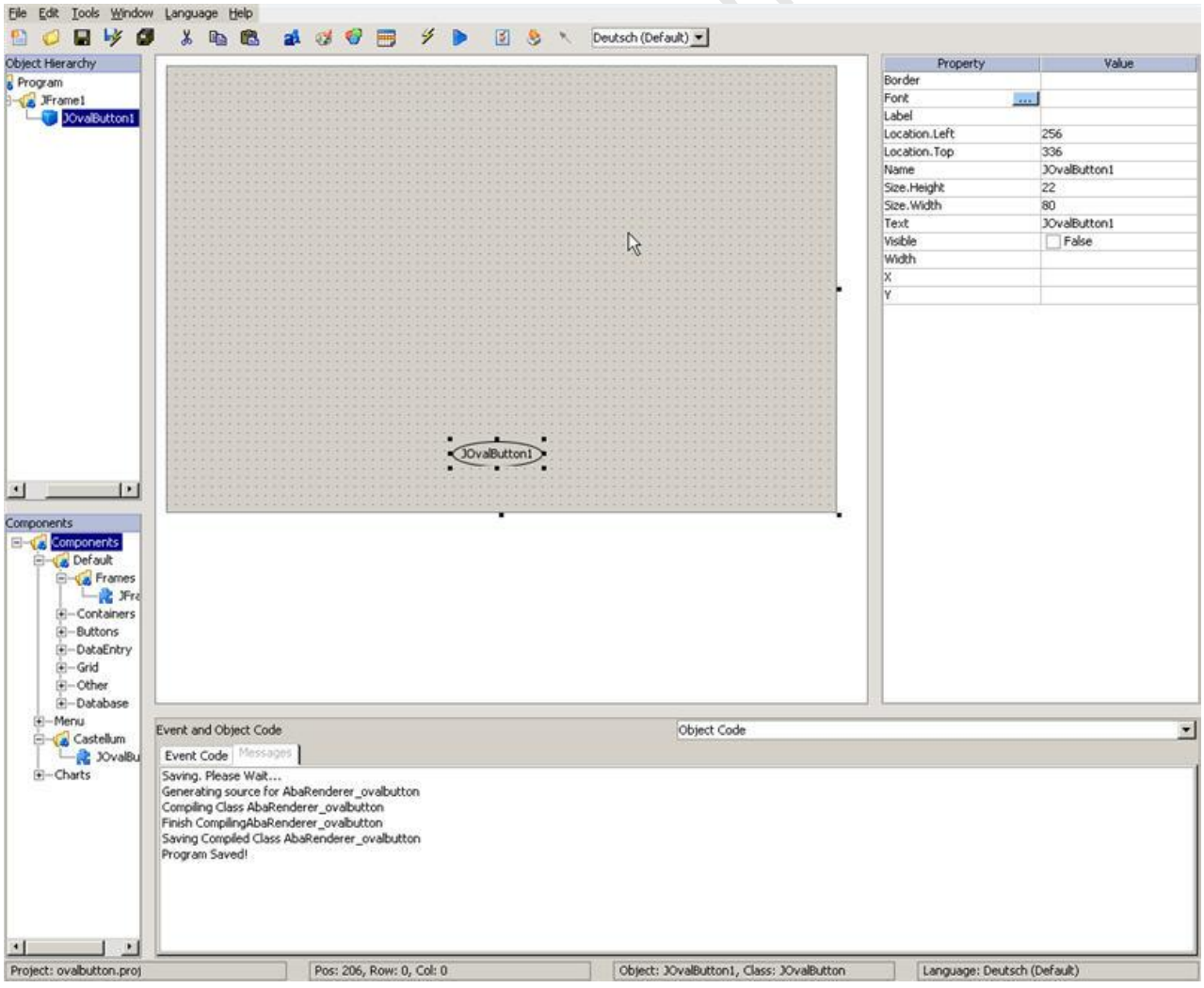


Figure 8

13.5 Rendering Imported Components

When you compile a project with imported components for first time, a .jar file named {location}..*AbacusCostum* is written out, in fact , the jar file is recreated whenever you update the XML class definition either manually or via the import IDE (Note: we recommend the IDE dialog).

For the example in Figure 8, you will the following two files in the os.repository directory:

1. abaguibuilder-1.6.samples.os.repository.AbacusCustom.jar
2. com.castellumtech.components.JOvalButton.xml

The AbacusCustom.jar file is used at rendering to extract the XML stored in the jar while the JOvalButton is used at design time ONLY, therefore when deploying you application you should supply the AbacusCustom(s) jars as well as the imported jars.

For example, to execute the sample application compiled in Figure 8 we need to execute the "runproz.bat or runproz.sh" with the following parameters:

runproz /abaguibuilder-1.6/samples/ovalbutton.jar
/abaguibuilder-1.6/samples/os.repository/castcomponents.jar
/abaguibuilder-1.6/samples/os.repository/abaguibuilder-1.6.samples.os.repository.AbacusCustom.jar

Parameter #1 is the compiled project
Parameter #2 is the jar that contains the JOvalButton component
Parameter #3 is the imported metadata jar (class definition for runtime execution)

When we execute the command line above we should the following form:

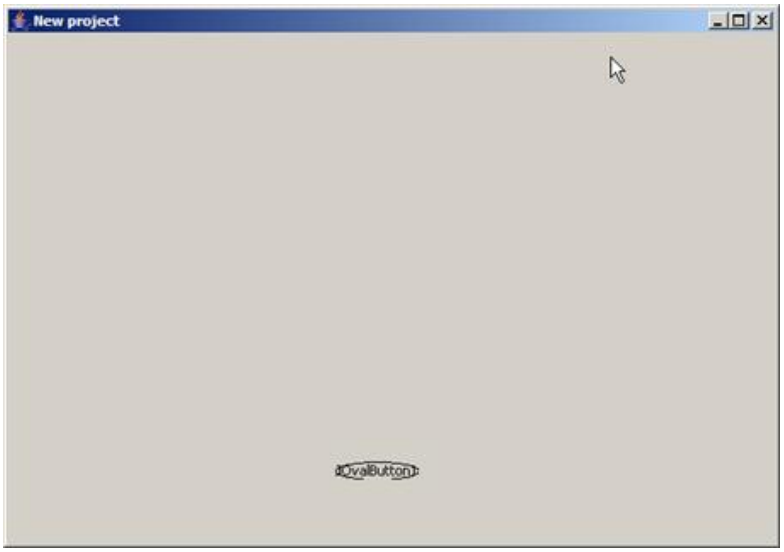


Figure 9

13.6 Example - Importing Components to existing section

This time we will import a calendar class from cbeans.jar included with Abacus GUI builder distribution, this jar contains a handful of visual components that we can import with the GUI builder.

First select the import dialog on the GUI builder  , next point to the \bin directory within our distribution and select cbeans.jar. We will import the CCalendar.class into section "Others" (Figure 10).

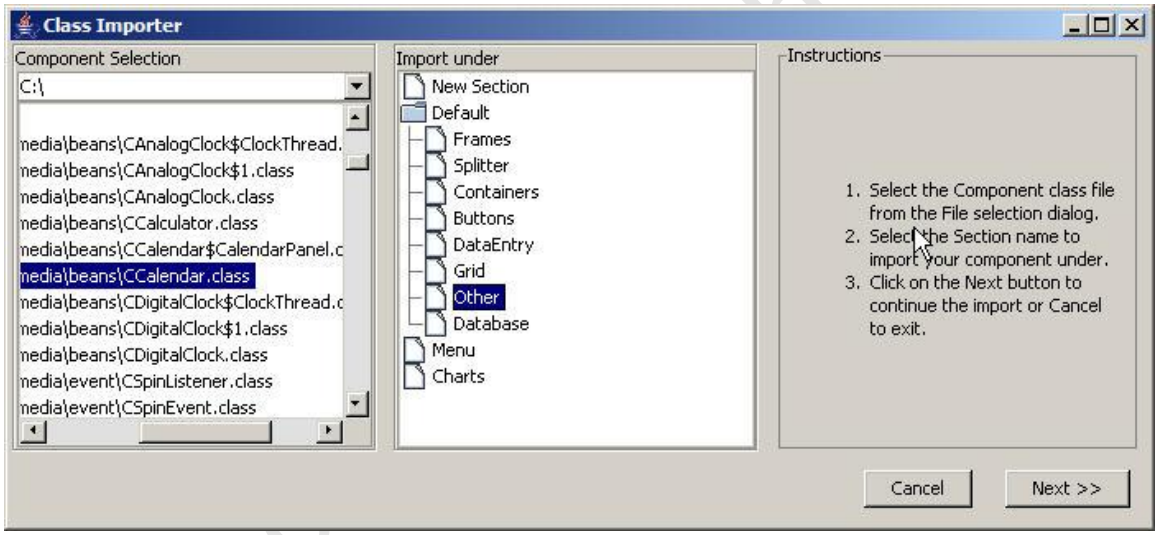


Figure 10

13.7 Defining sample Interface

The next step, we select the interface properties and Events (Figure 11).

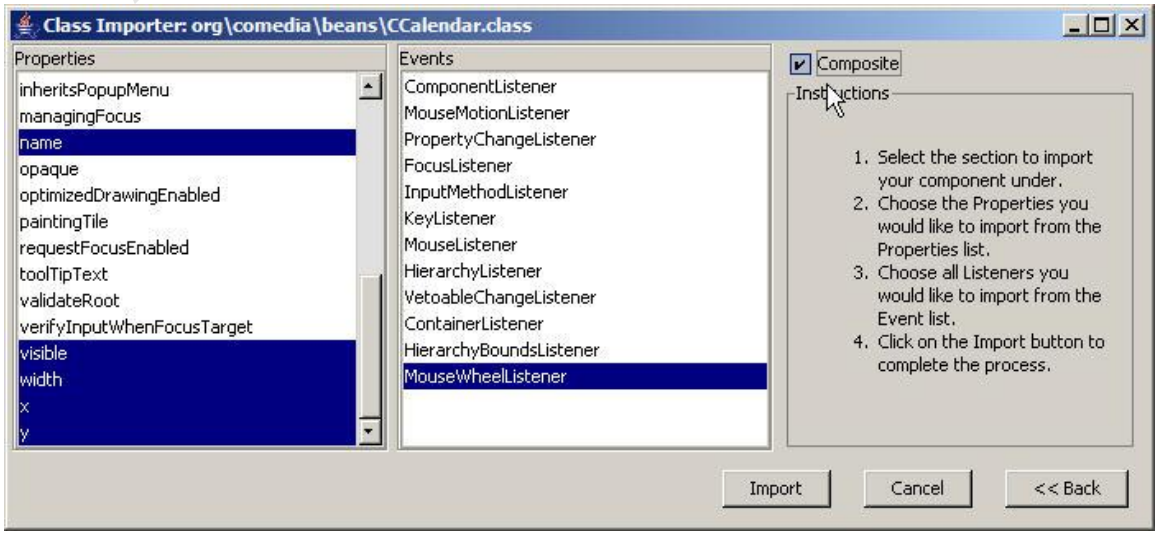


Figure 11

Import the following properties:

Border,font,name,visible,width.

Import the following Event:

MouseWheeListner

13.8 Storing sample interface

Now, we can “Import” the component and create its XML definition and we will save it under \samples\os.repository (Figure 12).

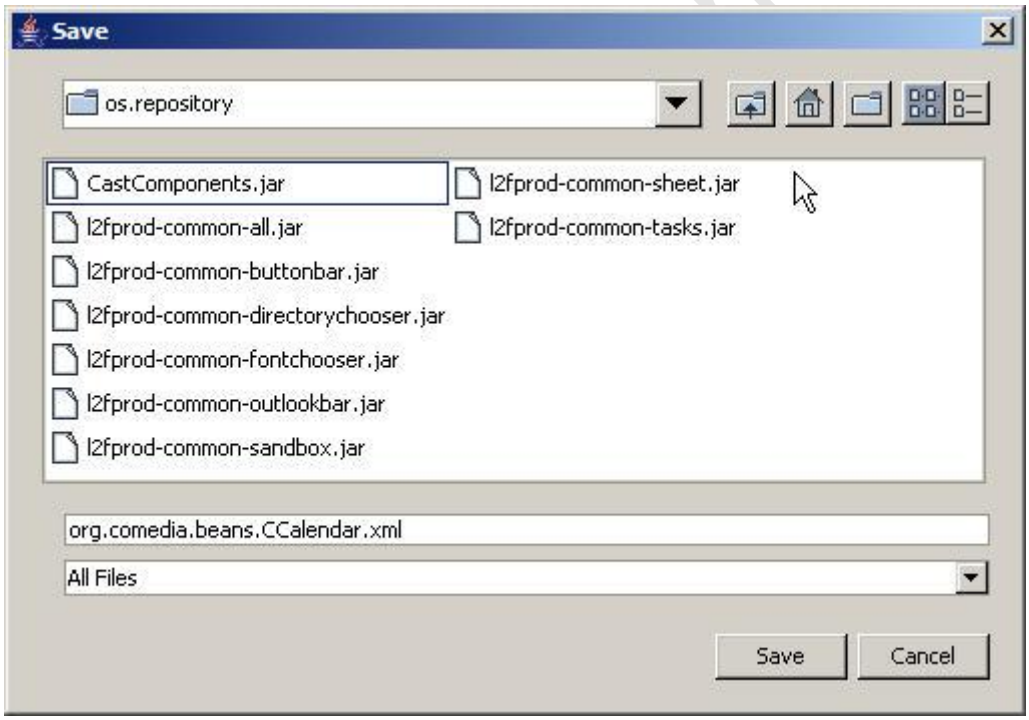


Figure 12

13.9 Component Palette

After saving the definition under “Other” you should see the CCalendar component listed on the “Components” palette (Figure 13).

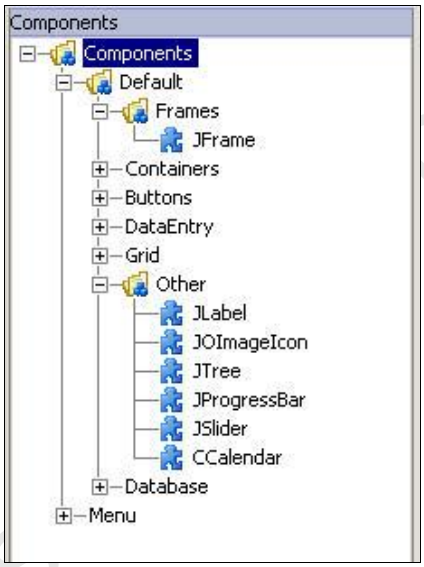


Figure 13

13.10 Saving and Compiling Calendar sample

Let's use the calendar component on a sample application:

1. Create a new project.
2. Add a JFrame
3. Drag the Calendar from "Other" to the canvas
4. "Save and Compile" the project as "CalendarSample" (Figure 14)

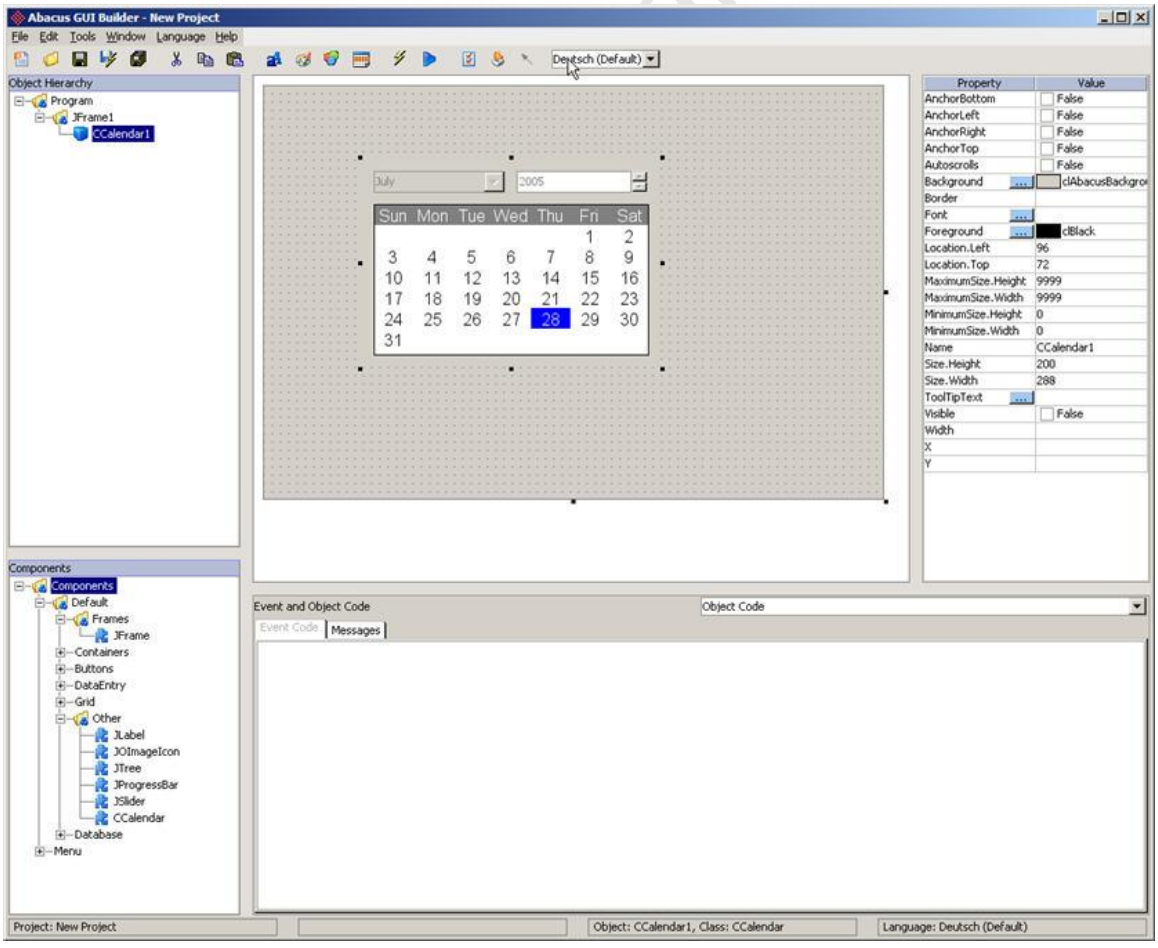


Figure 14

13.11 Rendering Calendar sample

runproz /abaguibuilder-1.6/samples/calendarsample.jar

/abaguibuilder-1.6/samples/os.repository/abaguibuilder-1.6.samples.os.repository.AbacusCustom.jar

Parameter #1 – is the compiled sample project “calendar sample”
Parameter #2 – is the XML definition of the Calendar class at rendering time.

NOTE: Because the runproz batch file adds cbeans.jar to the classpath by default we do not have to pass it to runproz.bat, otherwise the cbeans.jar will be the third parameter in the command line.