

# **Lab: AS/400 Toolbox for Java**

**Fall 1999 COMMON**

15LF/403627

25LF/403627

---

# Lab: AS/400 Toolbox for Java

<b>Fall 1999 COMMON</b>	1
<b>Introduction</b>	4
<b>Overview</b>	4
The Java™ Language	4
AS/400 Toolbox for Java	5
The Lab	6
Lab Setup	7
<b>Exercise 1: Command Call</b>	8
Introduction	8
Goals of this exercise	8
Part 1: Create an AS400 object	9
Part 2: Create a CommandCall object	10
Part 3: Run the command	10
Part 4: Retrieve AS400Message objects	11
Run the application	12
<b>Exercise 2: SQL Result Set Table Pane</b>	13
Introduction	13
Goals of this exercise	13
Part 1: Create an SQLConnection object	14
Part 2: Create an SQLResultSetTablePane object	15
Part 3: Run a query and load the results	15
Part 4: Setup an error handler	16
Run the application	17
<b>Exercise 3: GUI Builder</b>	19
Introduction	19
Goals of this exercise	19
Part 1: Start GUIBuilder	20
Part 2: Set the panel title	22
Part 3: Add text boxes to the panel	23
Part 4: Add buttons to the panel	24
Part 5: Create a PanelManager object	26
Part 6: Show the PanelManager object	26
Run the application	27
<b>Exercise 4: VisualAge for Java</b>	28
Introduction	28
Goals of this exercise	28
Part 1: Start VisualAge for Java	29
Part 2: Create a project	30
Part 3: Create a JFrame object	31

---

<b>Conclusion</b>	43
<b>Appendix A: Solutions</b> .....	44
Exercise 1: Command Call .....	44
Exercise 2: SQL Result Set Table Pane .....	46
Exercise 3: GUI Builder .....	48

# Introduction

## Overview

### The Java™ Language

Java is a simple, object-oriented, network-aware, portable, interpreted, robust, secure, architecture neutral, high-performance, multithreaded, dynamic language. Java is object-oriented from the ground up. Java organizes code into a collection of classes. Each class is made up of methods and data. Classes can be grouped together and placed in packages.

- **Packages** are similar to AS/400 ILE RPG service programs. They enable you to divide your pieces into easily reused units. Packages are Java language constructs. They may contain multiple classes.
- **Classes** are similar to AS/400 ILE RPG modules. They enable you to divide your source code into functions (“methods” in Java, procedures and subroutines in RPG) and the variables those functions need. Classes are typically self-contained groupings. They normally contain multiple fields (variables) and methods.
- **Methods** are similar to AS/400 ILE RPG procedures and subroutines. They contain all the actual code your program will run. In Java, unlike RPG, executable code can only exist in methods, and methods can only exist inside classes.



## AS/400 Toolbox for Java

Java programs can access AS/400 data and resources from any client platform (including the AS/400) using the AS/400 Toolbox for Java. The AS/400 Toolbox for Java contains the infrastructure to access the following AS/400 data and resources:

- JDBC and record-level access to DB2/400 data
- print resources
- integrated file system
- data queues
- program calls
- command calls
- user lists
- job lists
- job logs
- message queues
- *and many others!*

The AS/400 Toolbox for Java provides Java Beans that can be used for visual application development. Developers can use the classes directly from Java code or in a visual application builder. The classes are 100% Pure Java and will run on any JVM that supports JDK 1.1.6 or later.

See <http://www.ibm.com/as400/toolbox> for more information.

## **The Lab**

This lab consists of exercises that demonstrate various components of the AS/400 Toolbox for Java. In the exercises, you will create and run several Java applications. In most cases, you will start with an existing source file. For each exercise, your task is to complete the application by adding the AS/400 Toolbox for Java code. You will then compile and run each application.

You need the following hardware to complete this lab:

Server - AS/400 - OS/400 V4R2 or later

Client - PC - Intel based; 32 MB Memory; 200 MB hard drive  
Windows 95, Windows 98 or Windows NT

Actually, any computer with a Java Virtual Machine can be used (except for the last exercise). However, the lab instructions explain the instructions using Windows terminology and tools.

You need the following software on the client to complete this lab:

- Java Developers Kit 1.1.6 or later
- Swing 1.0.3
- Modification 2 of the AS/400 Toolbox for Java
- VisualAge for Java

## **Lab Setup**

During this lab you will need a userid and password for the AS/400. You will also need to know the name of the AS/400 system. This information will be given to you by the instructor. Please fill in this information below so that you have it for reference during the lab.

My AS/400 **system** is \_\_\_\_\_

My AS/400 **userid** is \_\_\_\_\_

My AS/400 **password** is \_\_\_\_\_

The PC **source code directory** is \_\_\_\_\_

You are now ready to begin the exercises.

---

## Exercise 1: Command Call

## Part 1: Create an AS400 object

Note that all sections that need code written by you start with the comments:

```
// -----  
//           Lab Exercise #1 Part #x - Insert code here.  
// -----
```

and end with the comments:

```
// -----  
//           End of code.  
// -----
```

Type the code for each exercise between the beginning comments of Exercise #1 Part #1 and before the ending comments.

### Setup

1. Start a DOS prompt using the Start menu: select “Programs”, “Command Prompt”.
2. Change the current directory to the directory which contains the lab source code. The lab instructor will tell you the name of this directory:

***cd directory***

3. Edit the CommandCallExample.java source file. You can use any editor you like. For this lab, we will use Windows Notepad. In the DOS prompt, type:

***notepad CommandCallExample.java***

4. Locate the section for Lab Exercise #1 Part #1.

### Procedure

1. Create an AS400 object named *system* and specify your assigned AS/400 system name. Some prototypes that you may need are listed below. (The complete set of prototypes is provided in the documentation that is shipped in soft copy form with the AS/400 Toolbox for Java.) This AS400 object represents the connection to the AS/400 system.

*Remember that at any time during this lab, if you get stuck, you can either ask a lab attendant for help or consult Appendix A for the solutions.*

### Prototypes

#### **class AS400**

- `public AS400()`
- `public AS400(String systemName)`
- `public void setSystemName(String systemName)`

## Part 2: Create a CommandCall object

### Setup

1. Continue editing the CommandCallExample.java source file.
2. Locate the section for Lab Exercise #1 Part #2.

### Procedure

1. Create a CommandCall object named *command* and specify the AS400 object that was created in Part 1. Again, some prototypes that you may need are listed below. This CommandCall object represents a command call, although at this point, no command has been called.

### Prototypes

#### class CommandCall

- public CommandCall()
- public CommandCall(AS400system)
- public void setSystem(AS400 system)

## Part 3: Run the command

### Setup

1. Continue editing the CommandCallExample.java source file.
2. Locate the section for Lab Exercise #1 Part #3.

### Procedure

1. The lab already has code that creates a String named *commandString*, which is made up of the command line arguments passed by the user. Add the code to run this command.
2. Notice that the run() method returns a boolean, which indicates whether or not the command was successful. Add code to check this and print the appropriate message, either  
    System.out.println("The command was successful");  
or  
    System.out.println("The command failed");

### Prototypes

#### class CommandCall

- public boolean run(String commandString)

## Part 4: Retrieve AS400Message objects

### Setup

1. Continue editing the CommandCallExample.java source file.
2. Locate the section for Lab Exercise #1 Part #4.

### Procedure

1. Retrieve any messages that were generated by running the command. This is stored as an array of AS400Message objects. Each AS400Message object in the array represents a message that was generated by the command.
2. Loop through the array of messages, and print each message's ID and text to System.out.

### Prototypes

#### class CommandCall

- public AS400Message[] getMessageList()

#### class AS400Message

- public String getID()
- public String getText()

## Run the application

Now it is time to run the CommandCallExampleapplication.

1. Make sure to save the modified CommandCallExample.java file.
2. Compile the application from the DOS prompt:

**javac CommandCallExample.java**

3. Run the application and specify the AS/400 command that you want to run:

**java CommandCallExample CRTLIB FRED**

4. The application will prompt you for a user ID and password. This happens automatically when your application accesses the AS/400 using the AS/400 Toolbox for Java. Enter your assigned user ID and password.



5. Verify that the application output (which appears in the DOS prompt) looks similar to this:

```
The command failed.  
CPF2111:Library FRED already exists.
```



## Exercise 2: SQL Result Set Table Pane

### Introduction

In this exercise, you will use the AS/400 Toolbox for Java to present the results of an SQL database query in a graphical user interface

The `SQLResultSetTablePane` object (part of the AS/400 Toolbox for Java) enables a Java program to present the results of a database query in a `JTable`. A `JTable` is a Java Swing component and can be imbedded inside any graphical user interface.

In this exercise you will use `SQLConnection`, `SQLResultSetTablePane`, and `ErrorDialogAdapter` objects to complete a Java application. Your application will present the results of an SQL database query. The SQL query is entered by the user. The Swing and AWT parts of the application have been provided for you. You will need to write Java code to connect to the AS/400 database, create the `SQLResultSetTablePane` object, and run the queries.

### Goals of this exercise

At the end of this exercise, you should be able to:

1. Create an `SQLConnection` object.
2. Create an `SQLResultSetTablePane` object.
3. Run a query and load the results.
4. Setup an error handler.

## Part 1: Create an SQLConnection object

### Setup

1. Edit the `SQLResultSetTablePaneExample.java` source file. In the DOS prompt, type:

**notepad SQLResultSetTablePaneExample.java**

2. Locate the section for Lab Exercise #2 Part #1.

### Procedure

1. Create an `SQLConnection` object, *connection*, that uses the JDBC URL “`jdbc:as400://systemName`”, where *systemName* is your assigned AS/400 system name. This `SQLConnection` object represents the JDBC<sup>1</sup> connection to the AS/400 database.

### Prototypes

#### class `SQLConnection`

- `public SQLConnection (String URL)`

---

<sup>1</sup> JDBC stands for Java Database Connectivity. JDBC is the Java standard for SQL database access. The AS/400 Toolbox for Java provides a JDBC 2.0 implementation. With JDBC you can access AS/400 databases using standard Java interfaces.

## Part 2: Create an `SQLResultSetTablePane` object

### Setup

1. Continue editing the `SQLResultSetTablePaneExample.java` source file.
2. Locate the section for Lab Exercise #2 Part #2.

### Procedure

1. Create an `SQLResultSetTablePane` object called *tablePane*. This represents the graphical user interface component which presents the contents of the query to the user.
2. Use `setConnection()` to set *tablePane*'s connection to the `SQLConnection` object that you created in Part 1. This tells *tablePane* which JDBC connection to use for executing the query and gathering results.

### Prototypes

#### class `SQLResultSetTablePane`

- `public SQLResultSetTablePane ()`
- `public void setConnection (SQLConnection connection)`

## Part 3: Run a query and load the results

### Setup

1. Continue editing the `SQLResultSetTablePaneExample.java` source file.
2. Locate the section for Lab Exercise #2 Part #3. **Note:** This section is located in the `keyPressed(KeyEvent)` method.

### Procedure

1. The query text that the user types is stored in a `String` named *queryText*. Use this value to set the query string to be run by *tablePane*.
2. Use `load()` to run the query and load the results into *tablePane*. By calling `load()`, the *tablePane* object will actually run the query (using JDBC), load its results, and present them in the table. If you forget to call `load()`, the table will still appear, but it will be empty.

### Prototypes

#### class `SQLResultSetTablePane`

- `public void setQuery (String query)`
- `public void load ()`

## Part 4: Setup an error handler

### Setup

1. Continue editing the `SQLResultSetTablePaneExample.java` source file.
2. Locate the section for Lab Exercise #2 Part #4.

### Procedure

1. Any errors that occur when accessing the AS/400 are not automatically displayed to the user. You need to set up an `ErrorListener` to handle errors. For this example, we will use an `ErrorDialogAdapter`, which is an `ErrorListener` that handles errors by displaying them in a message box for the user to see. You can also implement your own custom error handler if you have different requirements.
2. Create an `ErrorDialogAdapter` object called *errorHandler* and specify *tablePane* for the component. This initializes the error handler and tells it to use *tablePane* to determine the parent frame for any message box dialogs that it displays.
3. Use `addErrorListener()` to add *errorHandler* as an `ErrorListener` to *tablePane*. This sets up the error handler to “listen” to *tablePane*. Now, whenever an error occurs in *tablePane*, this error handler will display a message box.

### Prototypes

#### class `ErrorDialogAdapter`

- `public ErrorDialogAdapter ()`
- `public ErrorDialogAdapter (Component component)`
- `public void setComponent (Component component)`

#### class `SQLResultSetTablePane`

- `public void addEventListener (EventListener listener)`

## Run the application

Now it is time to run the `SQLResultSetTablePaneExample` application.

1. Compile the application from a DOS prompt.

**`javac SQLResultSetTablePaneExample.java`**

2. Run the application.

**`java SQLResultSetTablePaneExample`**

3. The application displays the graphical user interface below. It includes a text field at the top, where you can type in SQL queries. It also displays an empty table. The table is empty since we have not yet run any queries.



4. Enter an SQL query in the text field. A good one to try is:

**SELECT \* FROM QIWS.QCUSTCDT**

5. The application will prompt you for a user ID and password. This happens the first time you run a query because this is when the physical connection to the AS/400 database is made. Enter your assigned user ID and password.
6. Verify that the results in the table look similar to this:



7. Close the window using the “X” in the upper right corner.

---

## Exercise 3: GUI Builder

<sup>1</sup> XML stands for eXtensible Markup Language. XML is a language for describing other specific languages and is popular for describing data in a format that is easily readable by humans and easily parsable by computers.

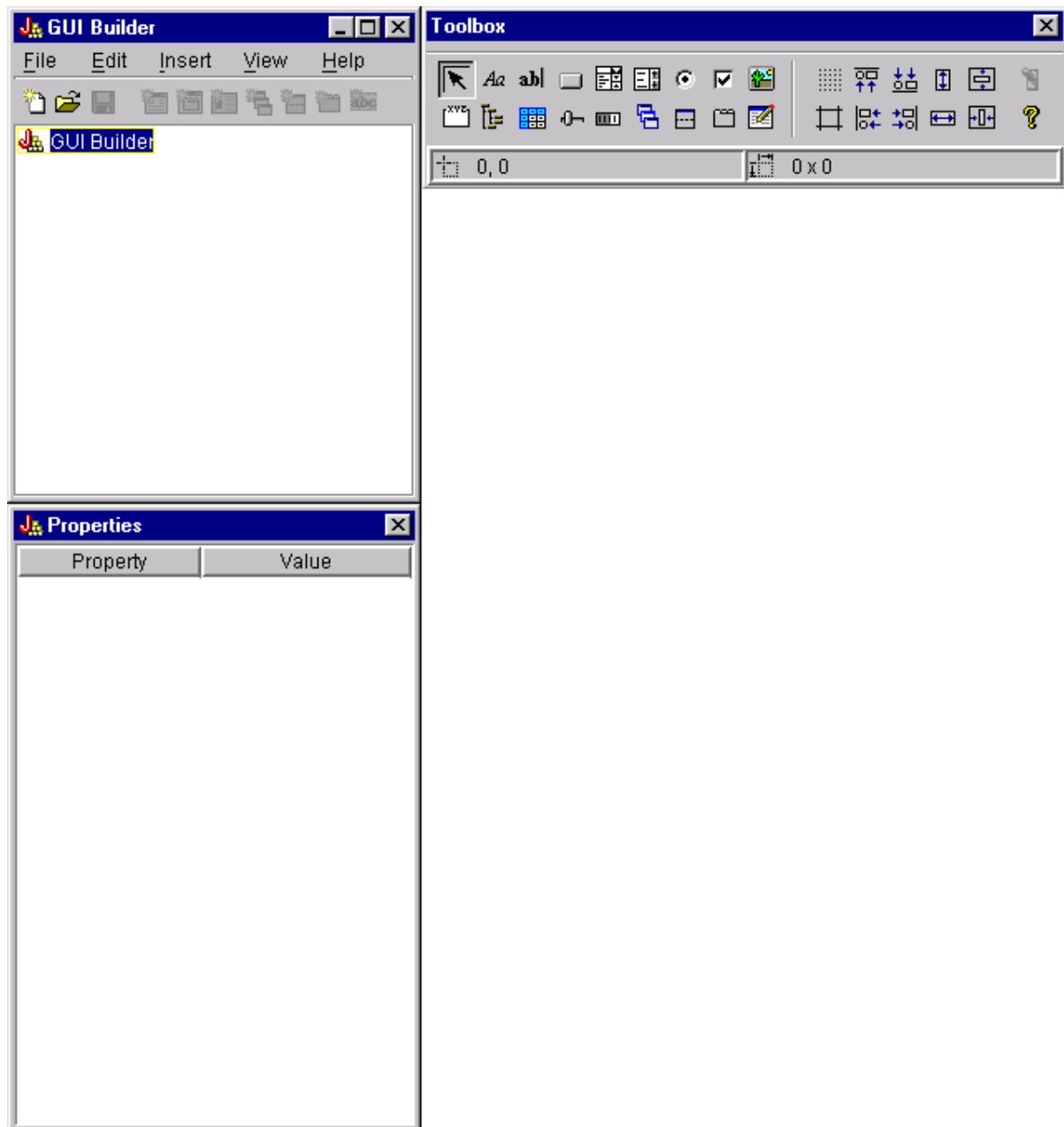
## Part 1: Start GUIBuilder

### Procedure

1. In the DOS prompt, type:

**java com.ibm.as400.ui.tools.GUIBuilder**

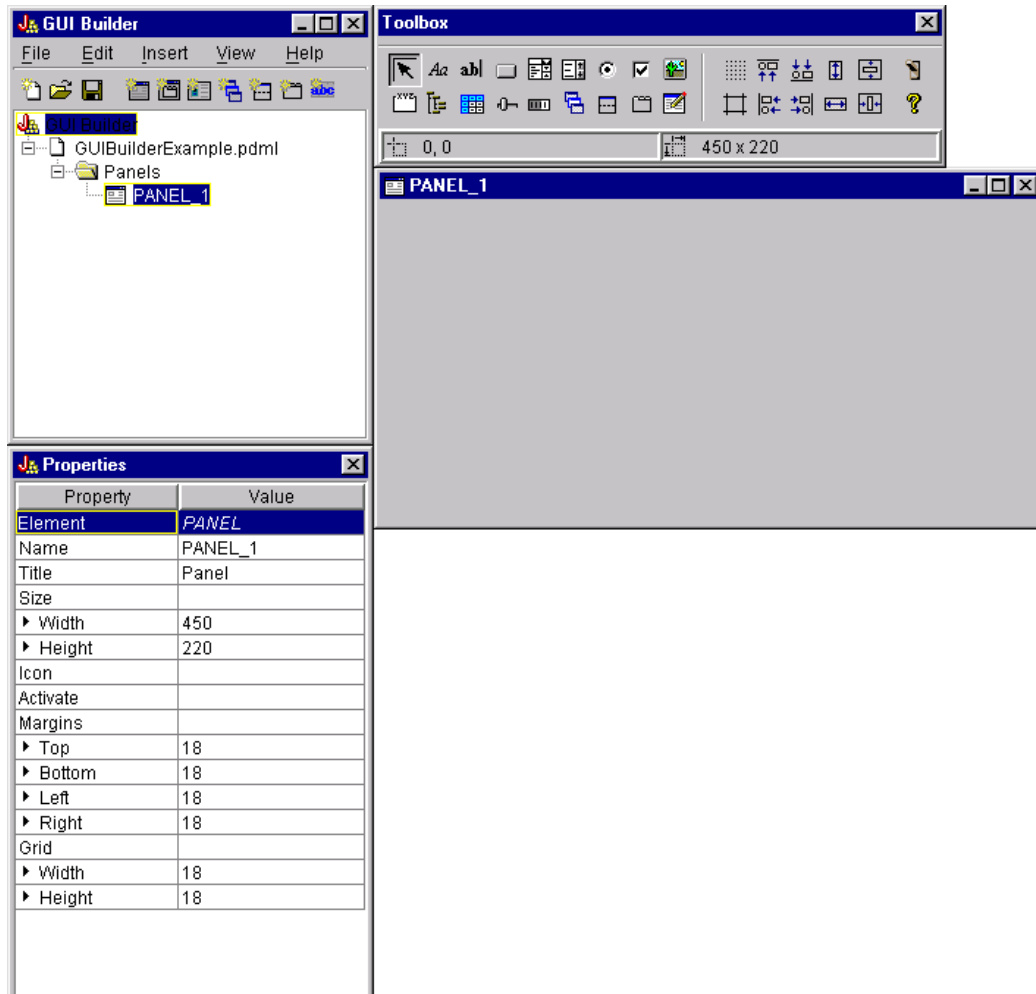
2. Verify that you see a window similar to this:





This is the main application window for GUIBuilder. There are three windows:

- The **GUIBuilder** window shows a hierarchical list of the components in your GUI. There is currently nothing in this list.
  - The **Toolbox** window shows the set of components and tools that you can use to design your GUI.
  - The **Properties** window shows the properties defined for the selected component. Since there are no components currently selected, there is nothing in this window.
3. In the **GUIBuilder** window, select the **File - New File** menu. This creates a new file for your GUI.
  4. Locate the **File1** icon in the **GUIBuilder** window. This represents the file where your GUI definition will be stored.
  5. Right click on the **File1** icon and select the **Save As** menu. This prompts for a file name. Enter **GUIBuilderExample.pdml**. Now your GUI definition will be saved in this file.
  6. Next, you need to create a new blank panel. Right click on the **GUIBuilderExample.pdml** icon and select **Insert - Panel**. This creates a blank panel, called **PANEL\_1**.
  7. Verify that your windows look like this:

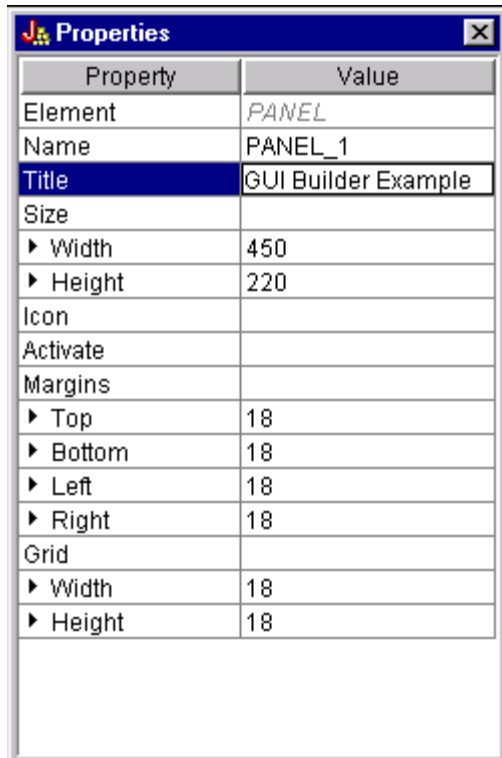


Now you are ready to design the GUI.

## Part 2: Set the panel title


### Procedure

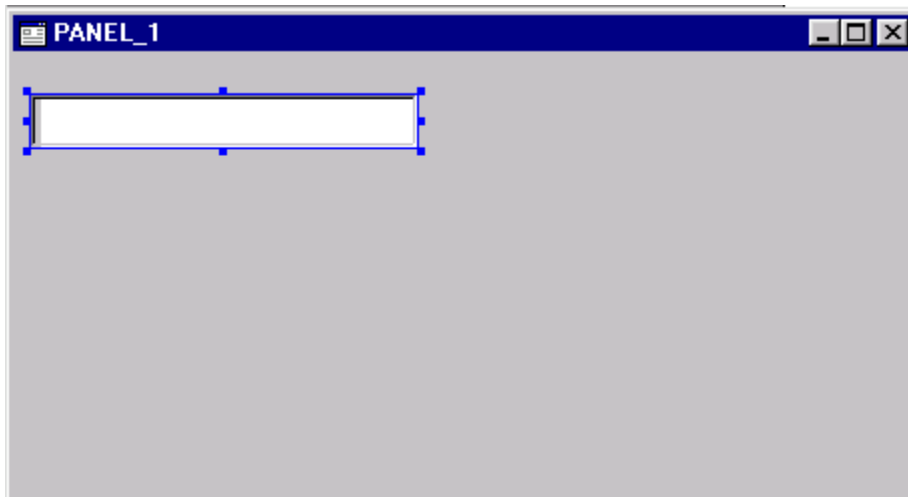
1. In the **GUIBuilder** window, select the **PANEL\_1** icon. The **Properties** windows will show the properties of the panel.
2. In the **Properties** window, locate the **Title** property. Change its value to “GUI Builder Example”.



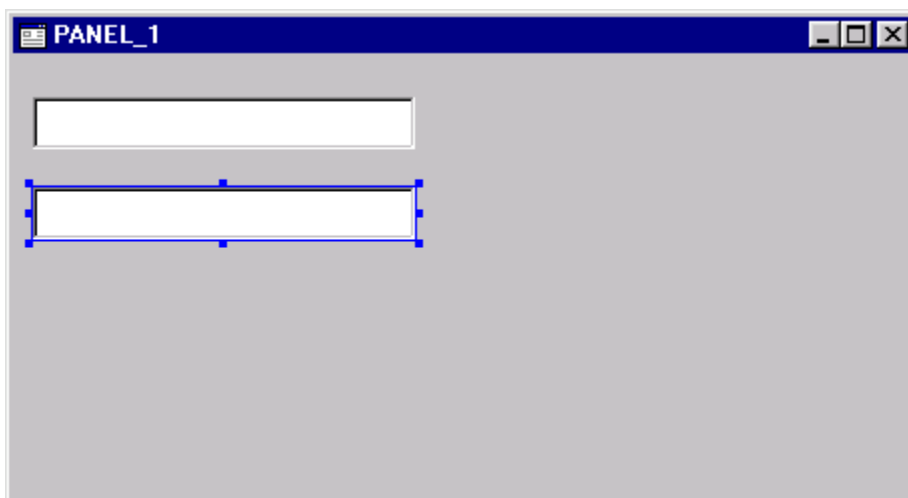
## Part 3: Add text boxes to the panel

### Procedure

1. Locate the  icon in the **Toolbox** window. This represents a text box. Left click on this icon.
2. Left click anywhere in the **PANEL\_1** window where you want to place the text box. For this example, we will place it near the upper-left corner of the panel.
3. You can make the text box wider than its default width. Left click on the small dot on the right side of the text box. Drag the right side to the desired width.




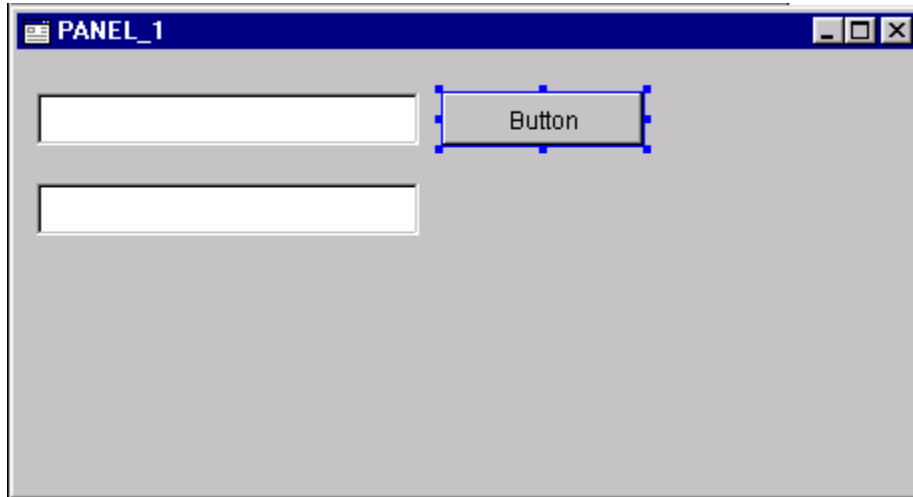
4. Repeat steps 1.-3. to add another text box to the panel.



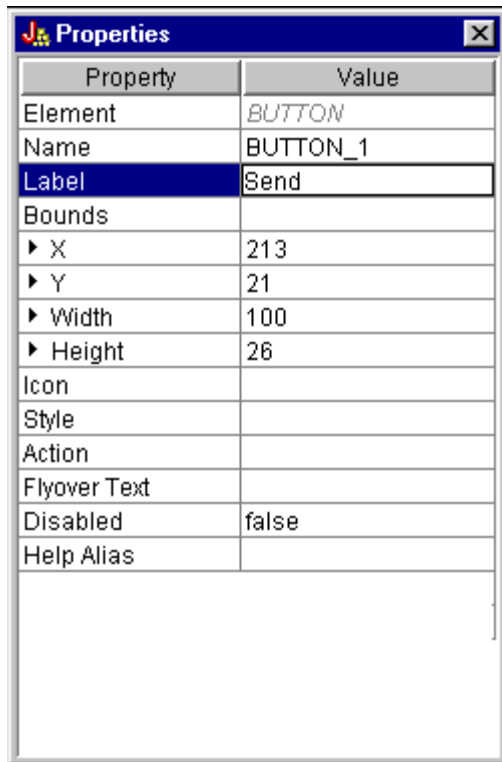
## Part 4: Add buttons to the panel

### Procedure

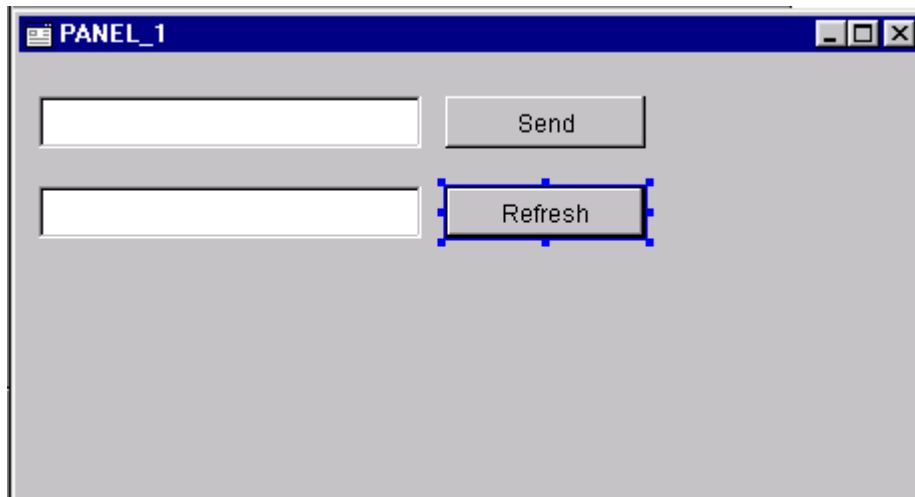
1. Locate the  icon in the **Toolbox** window. This represents a button. Left click on this icon.
2. Left click anywhere in the **PANEL\_1** window where you want to place the button. For this example, we will place it to the right of the first text field.



3. Notice that the **Properties** window shows the properties of the button. Locate the **Label** property. Change its value to “Send”. Notice that the button’s label changes in the **PANEL\_1** window, too.



4. Repeat steps 1.-3. to add another button with the label “Refresh”.



5. In the **GUIBuilder** window, select the **File - Save** menu. This will save the GUI definition in a file called GUIBuilderExample.pdml.
6. In the **GUIBuilder** window, select the **File - Exit** menu. This will exit GUIBuilder.

## Part 5: Create a PanelManager object

The GUI definition that you have just completed does not work by itself. All you have done is described what the GUI looks like. You need to write Java code that displays the GUI and implements the behavior of the program. For this part of the lab, most of the Javaapplication code is provided for you. You will need to write Java code to load the GUI definition and display it on the screen.

### Setup

1. Edit the GUIBuilderMain.java source file. In the DOS prompt, type:

**notepad GUIBuilderMain.java**

2. Locate the section for Lab Exercise #3 Part #5.

### Procedure

1. Create a PanelManager object named *pm*. In the constructor, specify “GUIBuilderExample” as the base name of the GUI definition, “PANEL\_1” as the name of the panel, and *null* for the data beans parameter.

### Prototypes

#### class PanelManager

- public PanelManager(String baseName, String panelName, DataBean[] dataBeans)

## Part 6: Show the PanelManager object

### Setup

1. Continue editing the GUIBuilderMain.java source file.
2. Locate the section for Lab Exercise #3 Part #6.

### Procedure

1. Show the GUI by calling the setVisible(true) method on *pm*, the PanelManager object.

### Prototypes

#### class PanelManager

- public void setVisible(boolean show)

## Run the application

Now it is time to run the GUIBuilderMain application.

1. Compile the application from a DOS prompt.

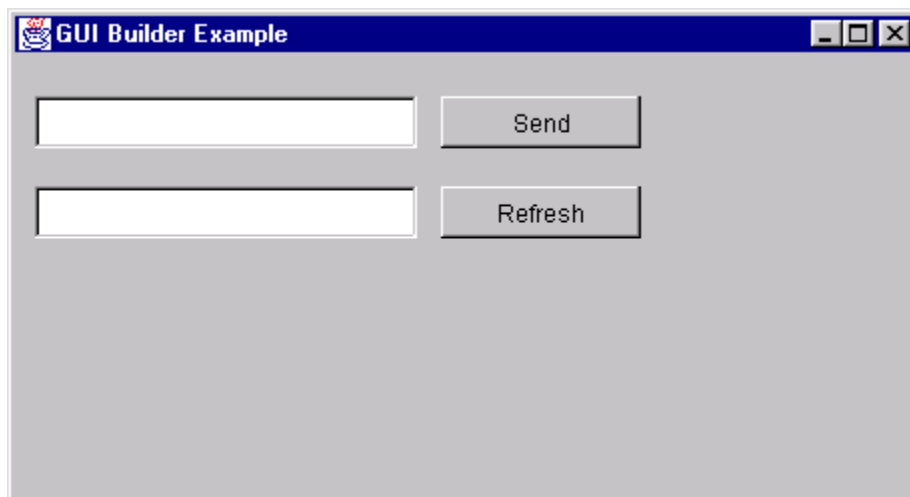
**javac GUIBuilderMain.java**

2. Run the application.

**java GUIBuilderMain**

You may see a `java.util.MissingResourceException` message. This is just a warning message that no help text was found. (GUIBuilder allows you to define help text for the panel, but we did not define any for this exercise.) You can ignore this message.

3. The application displays the user ID and password prompt. Once signed on, it will display the GUI that you designed.



4. Enter some text into the first text box and click the **Send** button. This will write the text as an entry to an AS/400 data queue.
5. Click the **Refresh** button. This will retrieve the most recent data queue entry from the AS/400 and display it in the second text box. Remember that this data queue entry may have been added by someone else in the lab.
6. Close the window using the "X" in the upper right corner.

---

## Exercise 4: VisualAge for Java



## Part 1: Start VisualAge for Java

### Procedure

1. Select the following menus: **Start - Programs - IBM VisualAge for Java for Windows - IBM VisualAge for Java.**
2. Verify that you see a window similar to this:



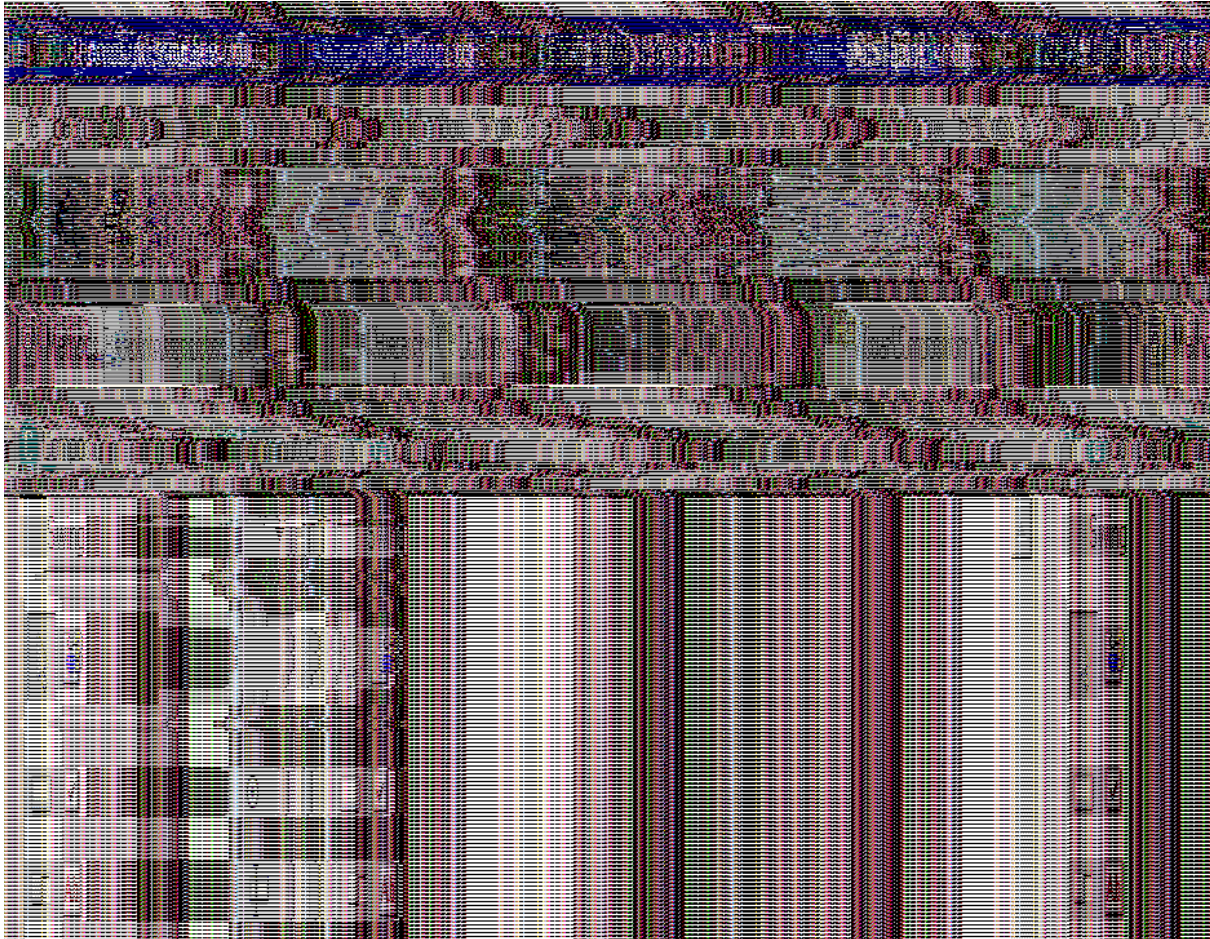
This is the main application window for VisualAge for Java. Note that there may be different projects listed when you run this.



## Part 2: Create a project

### Procedure

1. Select the following menus: **Selected - Add - Project...**
2. Under **Create a new project named:** type “RecordListFormPaneExample”. This is the name of our project.
3. Click **Finish**.
4. If VisualAge for Java tells you that there is already a project with this name in the repository, click **OK** to replace the old project.
5. Verify that there is now a project named “RecordListFormPaneExample”.
6. Right click on this project and select: **Add - Class...**
7. Under **Class name:** type “RecordListFormPaneExample”.
8. Make sure that **Compose the class visually** is checked.
9. Click **Finish**.
10. Verify that the Visual Composition Editor started automatically:




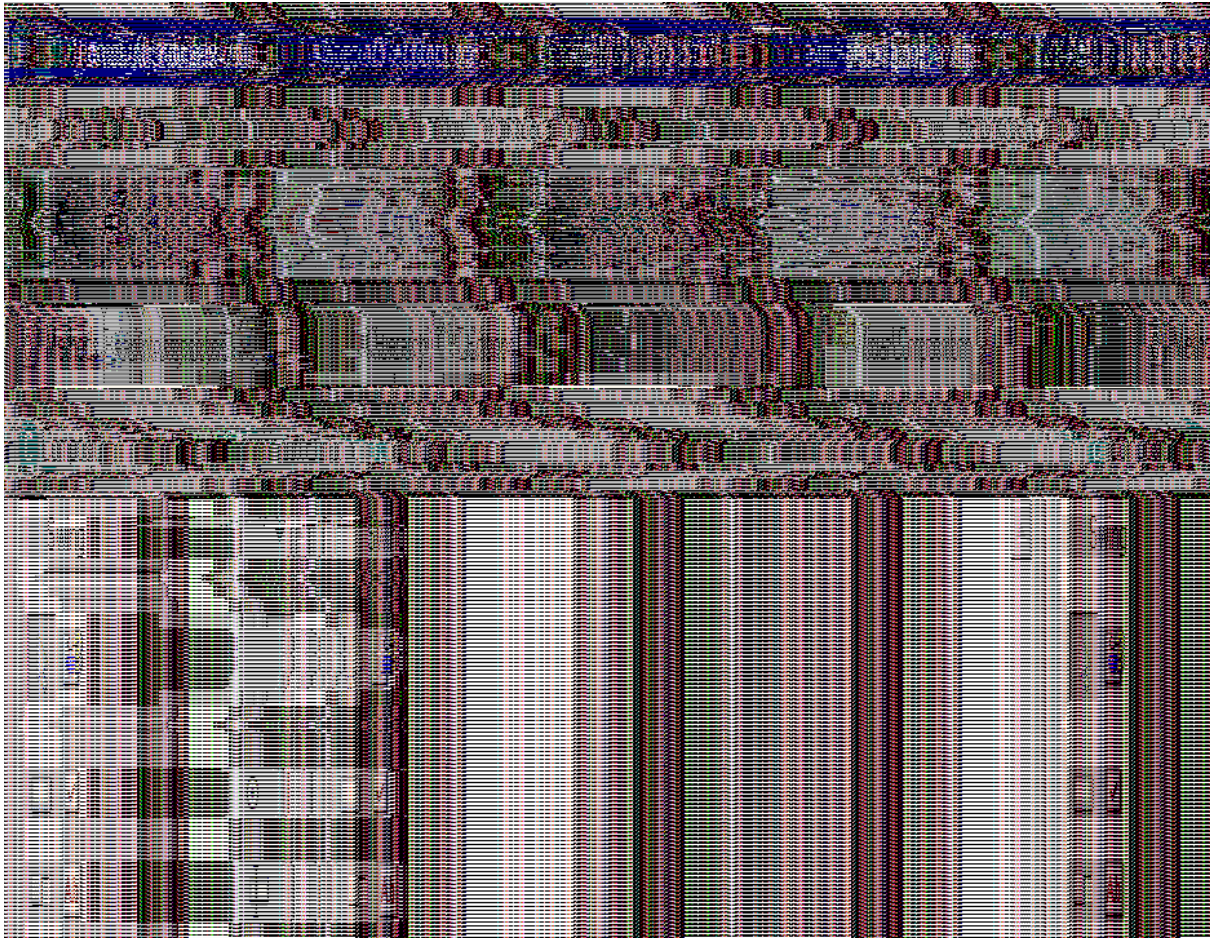
This is the window where we will develop our application.



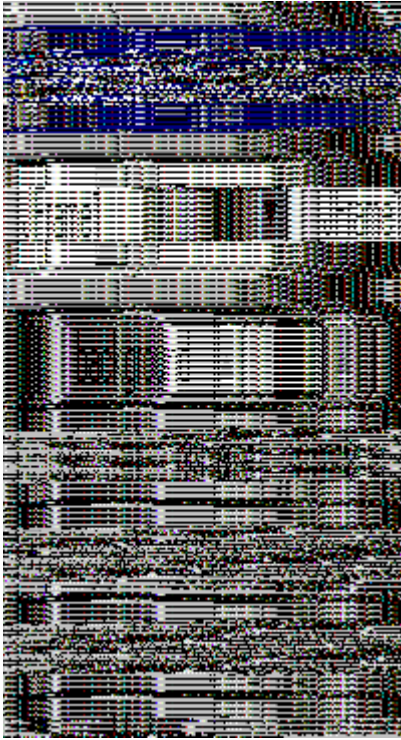
## Part 3: Create a JFrame object

### Procedure

1. Notice the palette on the left side of the Visual Composition Editor. This shows icons for all of the Java Beans currently loaded into VisualAge for Java. Whenever you need a new object in your application, you can select it from the palette and then click on the Visual Composition Editor where you want the object to be located.
2. The first object you need for your application is a **JFrame** object. This is a Swing object which represents the application's main window. At the top of the palette, there is category listed. You can choose from several categories of Java Beans. Select **Swing**, since JFrame is a Swing object.
3. Select on the JFrame object  from the palette to the Visual Composition Editor. If you have trouble finding JFrame in the palette, remember that holding your mouse over any of the icons for a few seconds will cause its name to appear briefly. This is helpful to choose among several icons that look similar. (You may need to scroll through the list of icons.)
4. Click anywhere on the Visual Composition Editor to indicate where to drop the new JFrame object. Verify that the JFrame object appears in the Visual Composition Editor:



5. There are a few properties of the JFrame object that you should set. The first is its title, which will show up in the top bar of the window. Right click on the *top bar* of the JFrame object and select **Properties**.
6. This brings up a window which lists some of the properties of the Frame object. Click on **title** and enter in a title for your application's main window: "Record List Form Pane Example".



7. Click on the "X" in the upper right corner of the Properties window to make it go away. Your JFrame object should now reflect the title that you assigned.
8. Another property that you need to set is the JFrame object's layout manager. The layout manager is responsible for positioning and sizing components correctly. Right click on the *middle* of the JFrame object and select **Properties**.
9. This brings up some more properties of the JFrame object (actually the JFrame object's "content pane"). Click on **layout** and select "BorderLayout" from the list of choices. The BorderLayout is an object in charge of arranging the components within the JFrame.



10. Click on the “X” in the upper right corner of the Properties window to make it go away.  
Your JFrame object should now be completely initialized.

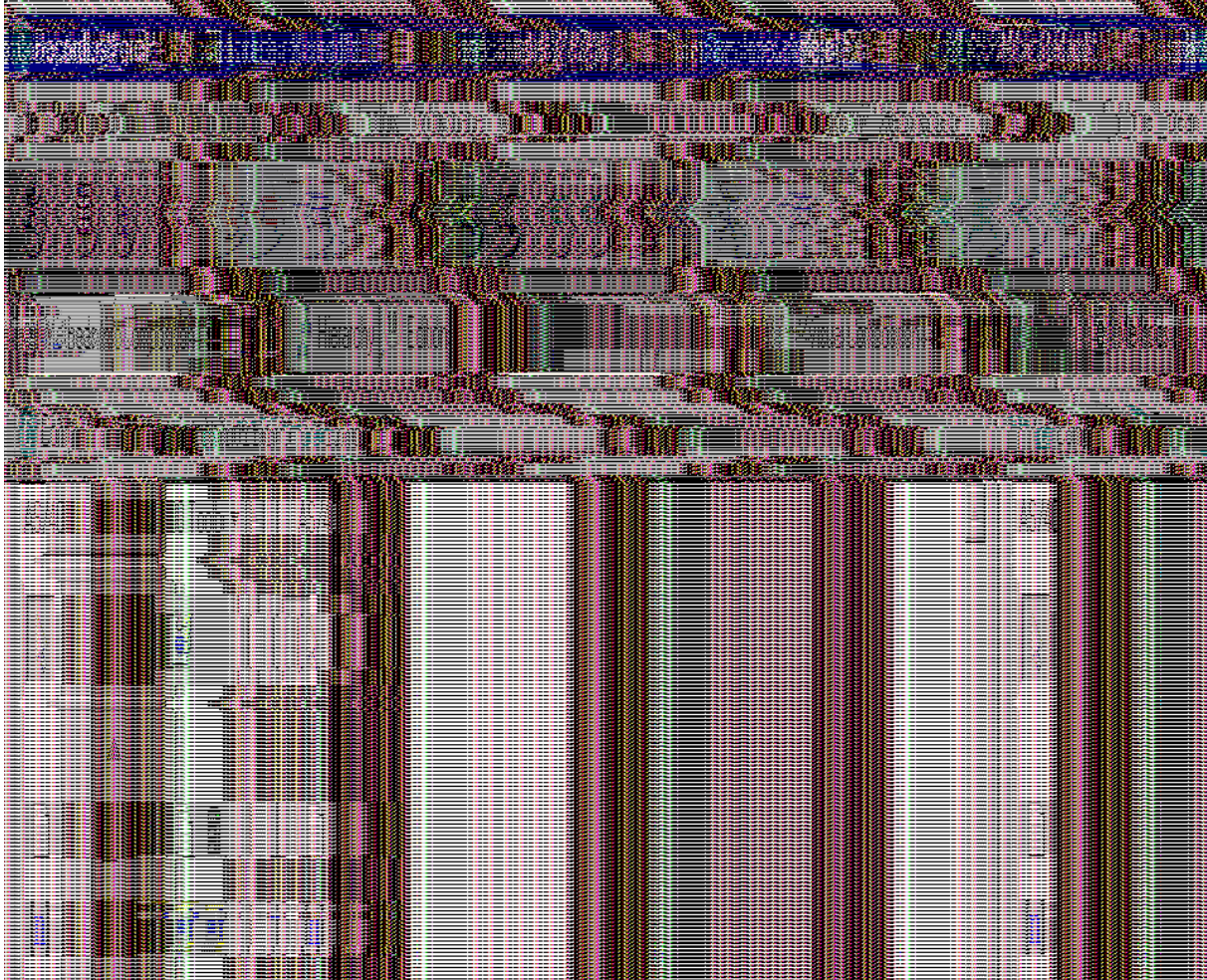



---

## Part 4: Create an AS400 object

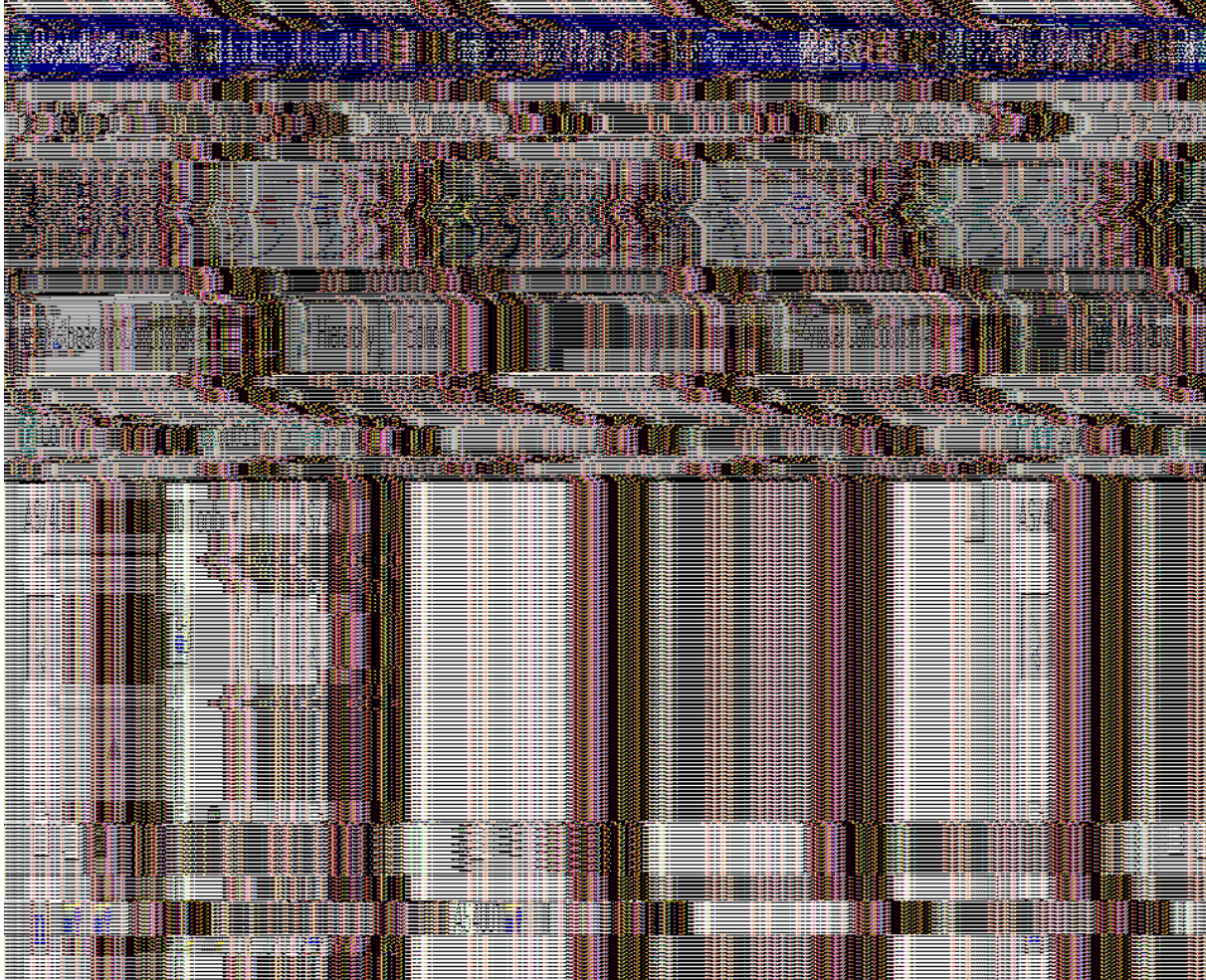
### Procedure

1. At the top of the palette, select **AS/400 Toolbox for Java**. This will cause the palette to display all of the AS/400 Toolbox for Java's Beans. Remember that if you cannot determine what Bean is using the icon, then hold the mouse over each icon and VisualAge for Java will briefly display the name of the Bean.



2. You will need an AS400 object to enable this application to communicate with an AS/400 system. You must create an AS400 object by selecting the AS400 object  from the palette






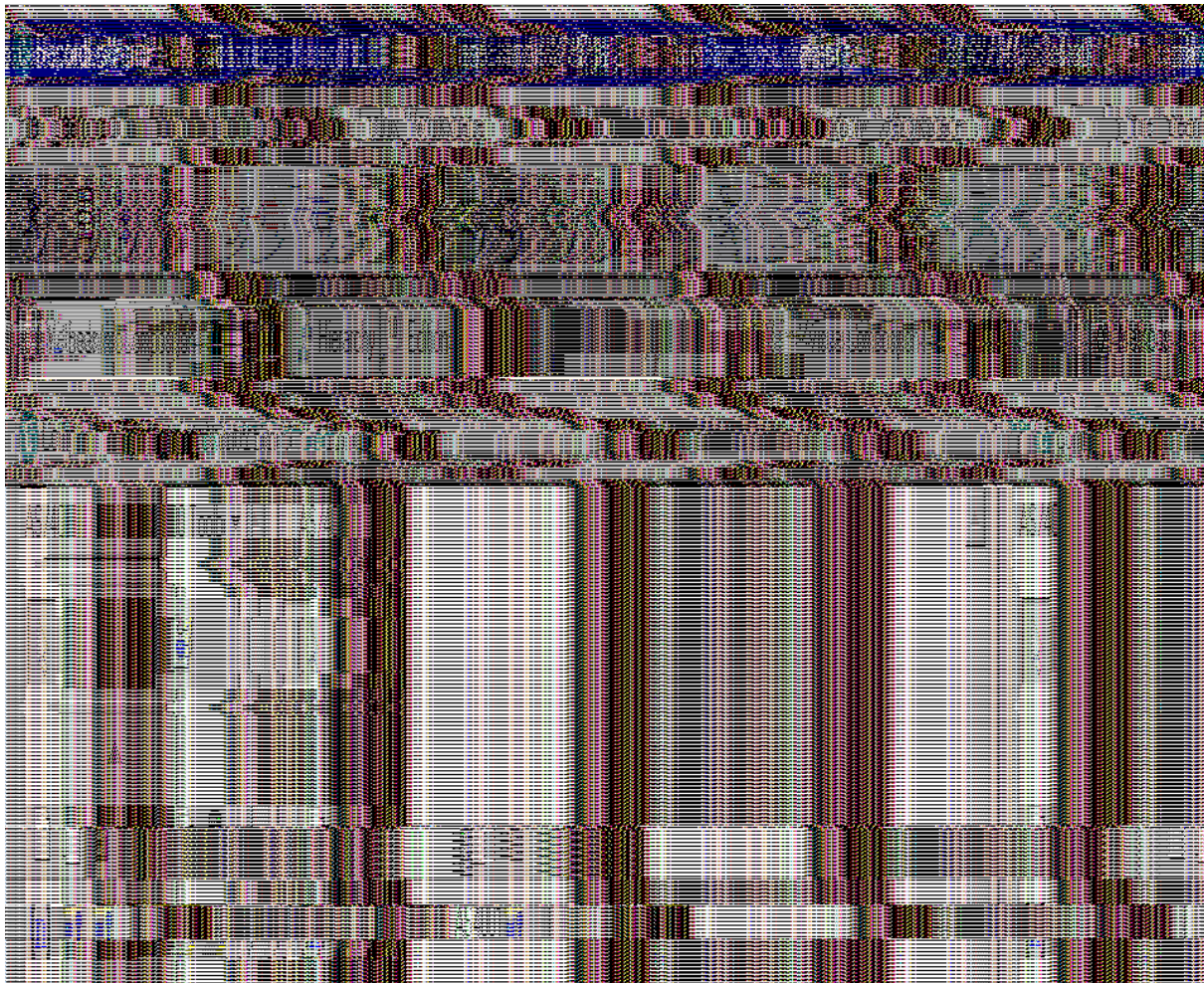
3. Right click on the AS400 object and select **Properties**. Set the **systemName** property to be the name of the AS/400 that you are using for this lab.
4. Click on the “X” in the upper right corner of the Properties window to make it go away. Your AS400 object should now be completely initialized.



## Part 5: Create a RecordListFormPane object

### Procedure

1. Select on the RecordListFormPane object  from the palette to the Visual Composition Editor.
2. Click inside the center of the JFrame. This will place the RecordListFormPane in the JFrame in the Visual Composition Editor. RecordListFormPane is an AS/400 Toolbox for Java Bean that will display the contents of an AS/400 database file, one record at a time. It provides several buttons that allow the user to move to different records. Verify that the contents of the Visual Composition Editor now look similar to this:

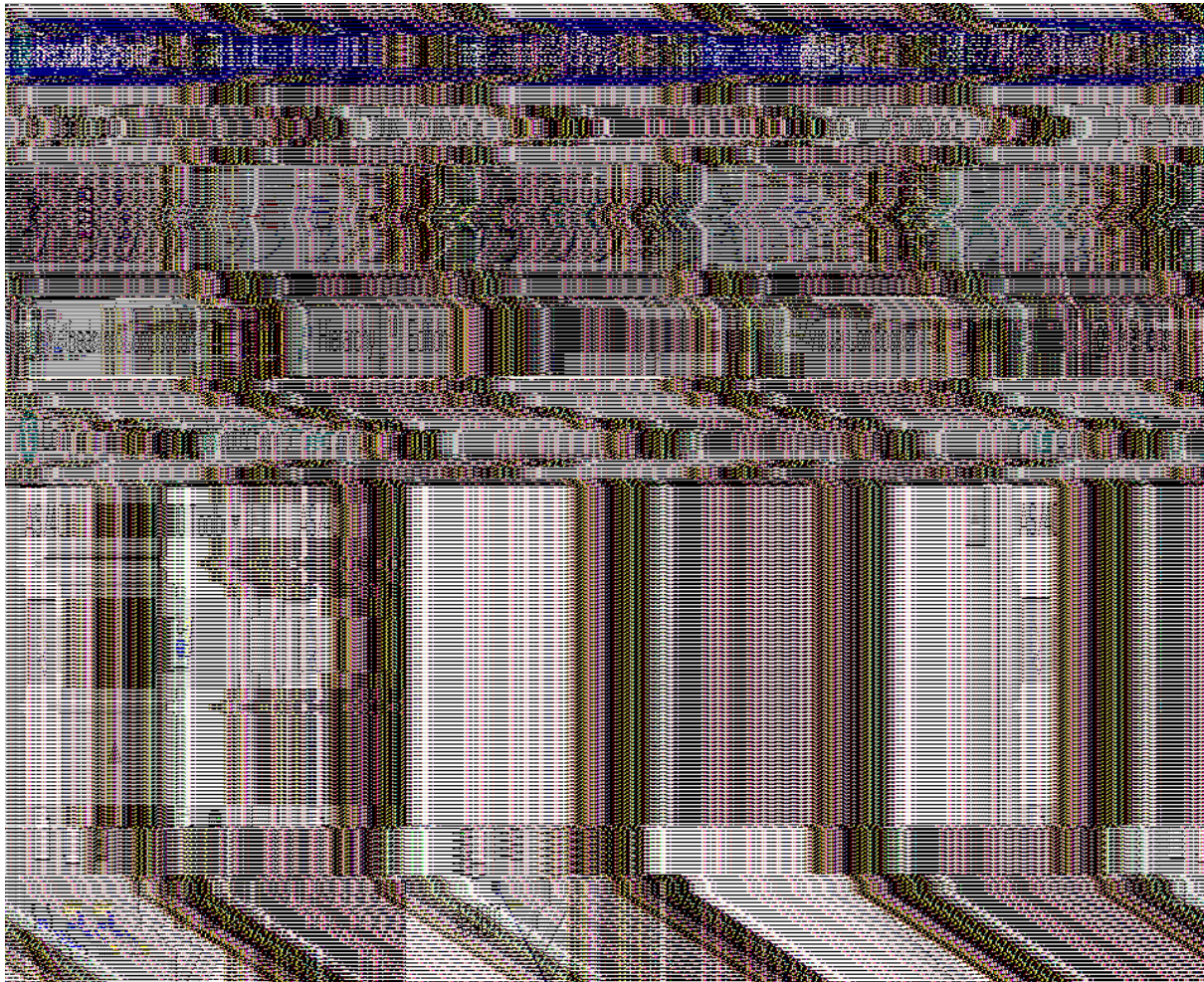


3. Note that at this point, the RecordListFormPane object does not refer to any particular AS/400 database file yet. For this, you need to set some properties. The first is the system where the file resides. The system is just an AS400 object that you created in the previous part. In the Visual Composition Editor, you define object relationships by drawing “connections” between two or more objects. Try not to confuse this use of the word “connection” with the AS/400 connection. You need to specify the **system** property of the



RecordListFormPane object to be equal to the AS400 object that you just defined. Right click on the RecordListFormPane object and select **Connect - Connectable Features...**

4. In the *Property* listing, choose **system** and click on **OK**. This defines which property you want to set and the start of a connection. The connection is represented by a dashed line and there is a “spider” on the end, which means that you must click on the object which should be the other end of the connection.
5. Click on the AS400 object, since it is the value to which we want to set the property.
6. Select **this** on the popup menu that appears. This means that the value of the property is set to the entire AS400 object. You have just set the RecordListFormPane object’s **system** property to be equal the AS400 object, and you still have not written any code... Verify that the contents of the Visual Composition Editor now look similar to this:



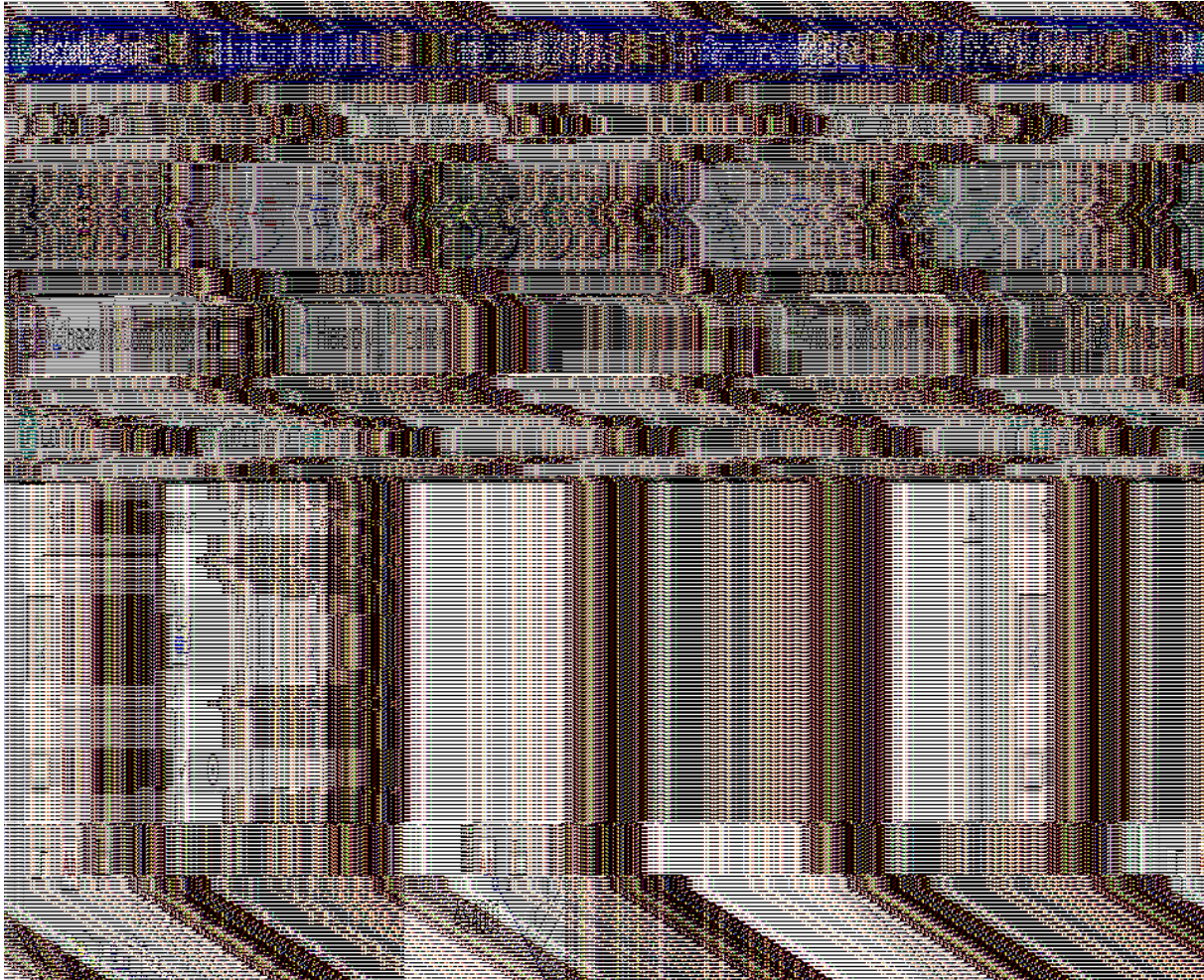
7. You still need to define which database file the RecordListFormPane object will display. Right click on the RecordListFormPane object and select **Properties**. Set the **fileName** property to “/QSYS.LIB/QIWS.LIB/QCUSTCDT.FILE”.
8. Click on the “X” in the upper right corner of the Properties window to make it go away. Your RecordListFormPane object should now be completely initialized

## Part 6: Load the contents of a database file

### Procedure

1. Another thing that you need to do is use the Visual Composition Editor to define when your application should actually load the contents of the database file. For this example, you will load the contents as soon as the application initializes. You can define this with another connection. This time you will connect the `initialize()` event for the application to the `load()` method of the `RecordListFormPane` object. Right click on any part of the white space, which represents the overall application. Select **Connect...**
2. You should be presented with a window title “Start Connection From (`RecordListFormPaneExample`).” This window lists all of the properties and events for which you can start a connection relating to the overall application. Click on **Event** to list the events. Select **initialize()** and click on **OK**. This defines the start of a connection. Your cursor will change to a “spider,” so it is time again to click on the target object of the connection.
3. Click inside the center of the `RecordListFormPane` object. You will see a popup menu that gives you some choices for the other end of the connection. Select **Connectable Features...**. You should now see a window which gives you the choice of all properties and methods for which you can end this connection relating to the `RecordListFormPane` object. Click on **Method** to list the methods. Select **load()** and click on **OK**. This defines the end of the connection. You have just made a connection that means that when the application initializes, it should call the `load()` method of the `RecordListFormPane` object. Verify that the contents of the Visual Composition Editor now look similar to this:





## Part 7: Pack and show the JFrame object

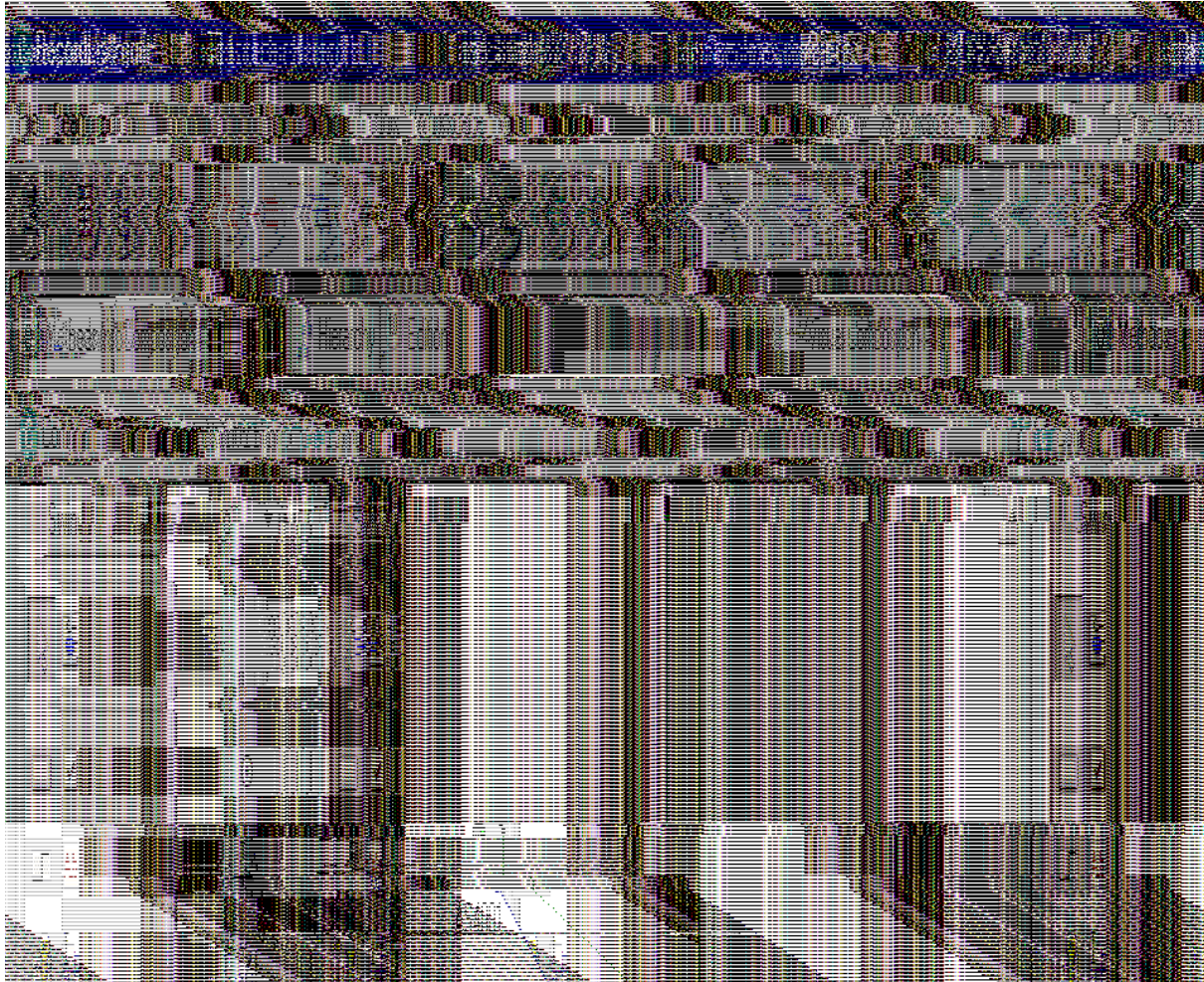
### Procedure

1. Your application is almost complete. The graphical user interface is initialized and ready to go. The last thing you need to do is use the Visual Composition Editor to define when your application should present (or “show”) the graphical user interface to the user. For this example, you will do this as soon as the application initializes. You can do this with yet another connection. This time you will connect the `initialize()` event for the application to the `pack()`<sup>1</sup> and `show()` methods of the `JFrame` object. Right click on any part of the white space, which represents the overall application. Select **Connect...**
2. You should be presented with a window title “Start Connection From (RecordListFormPaneExample)”. This window lists all of the properties and events for which you can start a connection relating to the overall application. Click on **Event** to list the events. Select **initialize()** and click on **OK**. This defines the start of a connection. Your cursor will change to a “spider”, so it is time again to click on the target object of the connection.
3. Click on the top bar of the `JFrame` object. You will see a popup menu that gives you some choices for the other end of the connection. Select **Connectable Features...**. You should now see a window which gives you the choice of all properties and methods for which you can end this connection relating to the `JFrame` object. Click on **Method** to list the methods. Select **pack()** and click on **OK**. This defines the end of the connection. You have just made a connection that means that when the application initializes, it should call the `pack()` method of the `JFrame` object. Verify that the contents of the Visual Composition Editor now look similar to this:

---

<sup>1</sup> `pack()` tells the `BorderLayout` and `JFrame` objects to compact themselves as much as reasonable to make the overall graphical user interface look good. It is a good idea to always call `pack()` before `show()`.






Note that some of the connection arrows may be positioned differently depending on where you dropped the JFrame object. This is fine as long as the correct objects are connected.

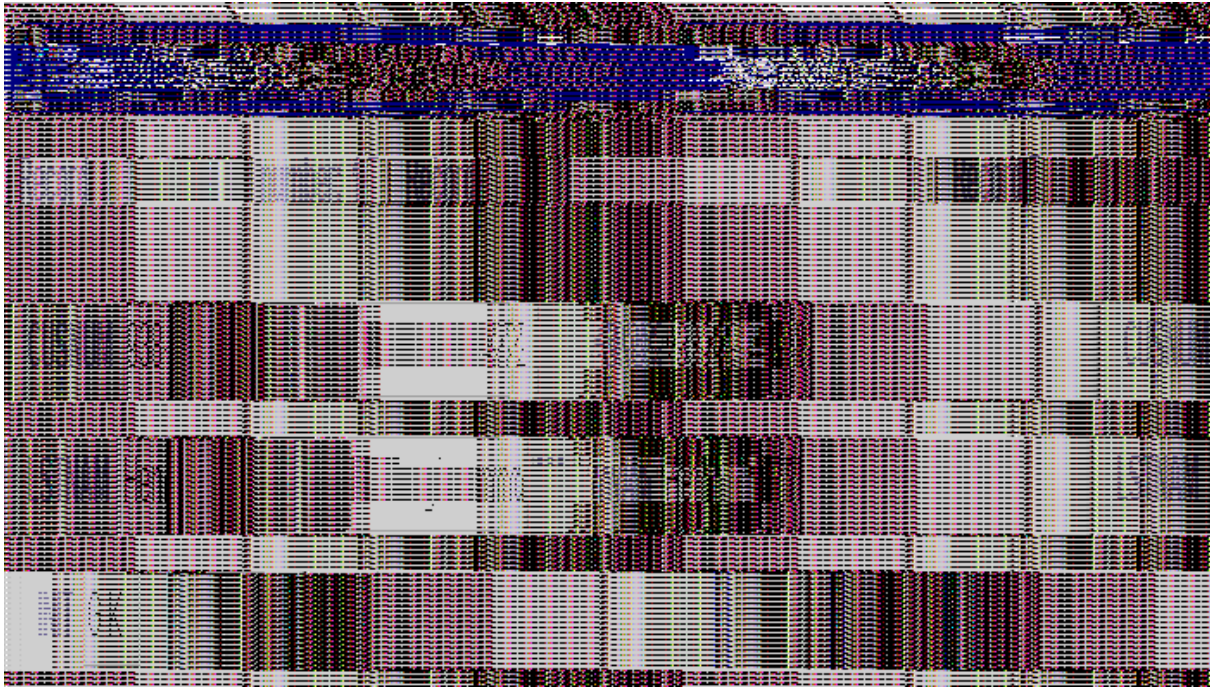
4. Right click on any part of the white space again, which represents the overall application. Select **Connect....**
5. You should be presented with a window title “Start Connection From (RecordListFormPaneExample)”. Click on **Event** to list the events. Select **initialize()** and click on **OK**. This defines the start of another connection. Your cursor will again change to a “spider”, so it is time again to click on the target object of the connection.
6. Click on the top bar of the JFrame object. Select **Connectable Features....** You should see the window which gives you the choice of all properties and methods for which you can end this connection relating to the JFrame object. Click on **Method** to list the methods. Select **show()** and click on **OK**. This defines the end of the connection. You have just made a connection that means that when the application initializes, it should call the show() method of the JFrame object.



## Run the application

Now it is time to run the RecordListFormPaneExample application.

1. In the Visual Composition Editor, click on the Run button . VisualAge for Java will save your application, generate Java code based on the objects and connections that you defined, and run the application. This may take a few minutes.
2. When the application runs, you will get the AS/400 Toolbox for Java signon prompt. Enter the user ID and password that you were assigned for this lab. This should all look familiar, since this is just another Java using the AS/400 Toolbox for Java. The only difference is that VisualAge for Java generated the Java code instead of you!
3. After signing on you should see the JFrame object with the RecordListFormPane object inside:



4. You can step through the records using the buttons at the bottom of the graphical user interface.
5. Close the window using the “X” in the upper right corner.

---

## Conclusion

## Appendix A: Solutions

### Exercise 1: Command Call

```
import com.ibm.as400.access.AS400;
import com.ibm.as400.access.AS400Message;
import com.ibm.as400.access.CommandCall;

public class CommandCallExample
{
    public static void main(String[] args)
    {
        try
        {
            // -----
            //           Lab Exercise #1 Part #1 - Insert code here.
            // -----

            AS400 system = new AS400("mySystem");

            // -----
            //           End of code.
            // -----

            // -----
            //           Lab Exercise #1 Part #2 - Insert code here.
            // -----

            CommandCall command = new CommandCall(system);

            // -----
            //           End of code.
            // -----

            // Gather the command line arguments passed to this .
            StringBuffer buffer = new StringBuffer();
            for (int i = 0; i < args.length; ++i)
            {
                buffer.append (args[i]);
                buffer.append (" ");
            }
            String commandString = buffer.toString ();

            // -----
            //           Lab Exercise #1 Part #3 - Insert code here.
            // -----

            if (command.run(commandString))
                System.out.println("The command was successful.");
            else
                System.out.println("The command failed.");

            // -----
            //           End of code.
            // -----
        }
    }
}
```



```
// -----  
//           Lab Exercise #1 Part #4 - Insert code here.  
// -----  
  
AS400Message[] messageList = command.getMessageList();  
for (int i=0; i < messageList.length; i++)  
{  
    System.out.println (messageList[i].getID() + ":" +  
                        messageList[i].getText());  
}  
  
// -----  
//           End of code.  
// -----  
  
}  
catch (Exception e) {  
    System.out.println ("Error: " + e);  
}  
  
System.exit (0);  
}  
}
```

**Exercise 2: SQL Result Set Table Pane**

```
import com.ibm.as400.access.AS400JDBCDriver;
import com.ibm.as400.vaccess.ErrorDialogAdapter;
import com.ibm.as400.vaccess.SQLConnection;
import com.ibm.as400.vaccess.ResultSetTablePane;

import com.sun.java.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.sql.*;

public class ResultSetTablePaneExample
extends KeyAdapter
{

    private static ResultSetTablePane    tablePane_;
    private static JTextField            textField_;

    public static void main(String argv[])
    {
        try {
            // Register the AS/400 Toolbox for Java JDBC driver.
            DriverManager.registerDriver(new AS400JDBCDriver());

            // -----
            //                      Lab Exercise #2 Part #1 - Insert code here.
            // -----

            SQLConnection connection = new SQLConnection("jdbc:as400://mySystem");

            // -----
            //                      End of code.
            // -----

            // -----
            //                      Lab Exercise #2 Part #2 - Insert code here.
            // -----

            ResultSetTablePane tablePane = new ResultSetTablePane();
            tablePane.setConnection (connection);

            // -----
            //                      End of code.
            // -----

            // Store the table pane in a static variable.
            tablePane_ = tablePane;

            // Initialize the text area.
            textField_ = new JTextField ("Enter an SQL query here.");
            textField_.addKeyListener (new ResultSetTablePaneExample ());
```

```
// Initialize the frame.
JFrame frame = new JFrame ("SQLResultSetTablePane example");
frame.getContentPane ().setLayout (new BorderLayout ());
frame.getContentPane ().add ("North", textField_);
frame.getContentPane ().add ("Center", tablePane_);

// When the frame closes, exit the .
frame.addWindowListener (new WindowAdapter ()
{
    public void windowClosing (WindowEvent event) { System.exit (0);}
});

// -----
//                      Lab Exercise #2 Part #4 - Insert code here.
// -----

ErrorDialogAdapter errorHandler = new ErrorDialogAdapter (tablePane);
tablePane.addErrorListener (errorHandler);

// -----
//                      End of code.
// -----

// Display the frame.
frame.pack();
frame.show();
} catch (Exception e) {
    System.out.println ("Error: " + e);
}
}

// This gets called whenever a key is pressed in the text area.
public void keyPressed (KeyEvent event)
{
    try {

        // Check for the Enter key.
        if (event.getKeyCode () == KeyEvent.VK_ENTER) {
            String queryText = textField_.getText ();

            SQLResultSetTablePane tablePane = tablePane_;

            // -----
            //                      Lab Exercise #2 Part #3 - Insert code here.
            // -----

            tablePane.setQuery (queryText);
            tablePane.load ();

            // -----
            //                      End of code.
            // -----

        }
    } catch (Exception e) {
        System.out.println ("Error: " + e);
    }
}
}
```

**Exercise 3: GUI Builder**

```
import com.ibm.as400.access.*;
import com.ibm.as400.ui.framework.java.*;
import com.sun.java.swing.*;
import com.sun.java.swing.text.*;
import java.awt.*;
import java.awt.event.*;

public class GUIBuilderMain
implements ActionListener
{

    private static DataQueue      dq;
    private static JTextField     sendField;
    private static JButton        sendButton;
    private static JTextField     refreshField;
    private static JButton        refreshButton;

    public static void main(String[] args)
    {

        try {

            // -----
            //              Intro Lab Exercise #3 Part #5 - Insert code here.
            // -----

            PanelManager pm = new PanelManager("GUIBuilderExample",
                                              "PANEL_1",
                                              null);

            // -----
            //              End of code.
            // -----

            // Initialize the components as defined in GUIBuilderExample.pdml.
            sendField      = (JTextField)pm.getComponent("TEXT_1");
            sendButton     = (JButton)pm.getComponent("BUTTON_1");
            refreshField   = (JTextField)pm.getComponent("TEXT_2");
            refreshButton  = (JButton)pm.getComponent("BUTTON_2");

            // Initialize the AS400 object.
            AS400 system = new AS400();

            // Initialize the data queue object.  Create the data queue if
            // it is not already created.
            dq = new DataQueue(system, "/QSYS.LIB/COMMON.LIB/COMMON.DTAQ");
            try {
                dq.create(500, "*ALL", true, false, false, "");
            }
            catch(Exception e) {
                // Ignore.  This means that the data queue is already
                // created.
            }
        }
    }
}
```

```
// Add a listener which makes the Send button write
// an entry to the data queue whenever it is pressed.
sendButton.addActionListener(new GUIBuilderMain());

// Add a listener which makes the Refresh button
// peek the data queue whenever it is pressed.
refreshButton.addActionListener(new GUIBuilderMain());

// -----
//                      Intro Lab Exercise #3 Part #6 - Insert code here.
// -----

pm.setVisible(true);

// -----
//                      End of code.
// -----
}
catch(Exception e) {
    e.printStackTrace();
}
}

public void actionPerformed(ActionEvent event)
{
    try {

        // Write an entry to the data queue when the send
        // button is pressed.
        if (event.getSource() == sendButton) {
            dq.write(sendField.getText());
        }

        // Peek the data queue when the refresh button
        // is pressed.
        else {
            String text = dq.peek().getString();
            refreshField.setText(text);
        }
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
}
```