Introduction to VisualAge for Java

Lab 2

Fall Common 1998

Pramod Patel, George Voutsinas and VA Java Enterprise Toolkit for AS/400 Team

IBM Canada Ltd

Disclaimer

This package is available for use AS IS. There is no support or service to the documentation and the code shipped with the package. IBM reserves all the rights to the lab material. This self-study material is provided for personal use. Reproduction of the material for commercial use is prohibited unless written agreement is provided by IBM.

Lab 2: Using VisualAge for Java with the AS/400

Overview

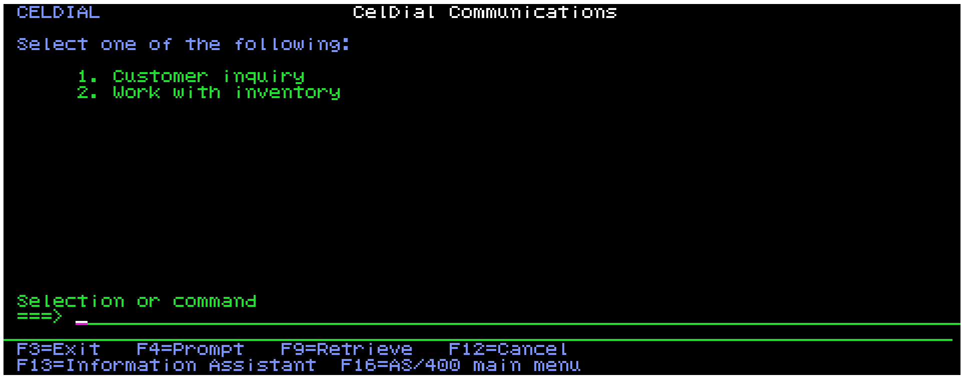
This lab provides an overview of VisualAge for Java. During this lab, you will build a Java part inventory application using the IBM VisualAge for Java integrated development environment (IDE).

Introduction

The Visual Composition Editor of VisualAge for Java allows VisualAge for Java includes a powerful tool called the Visual Composition Editor that is used to visually construct Java applications, applets, and reusable beans. Visual composition is the creation of object-oriented program elements by manipulating graphical representations of components. You can drag and drop JavaBeans to create GUIs, and then "wire them" together to produce Java applications and applets with minimal Java coding. These applications or applets can be used to access your AS/400 data.

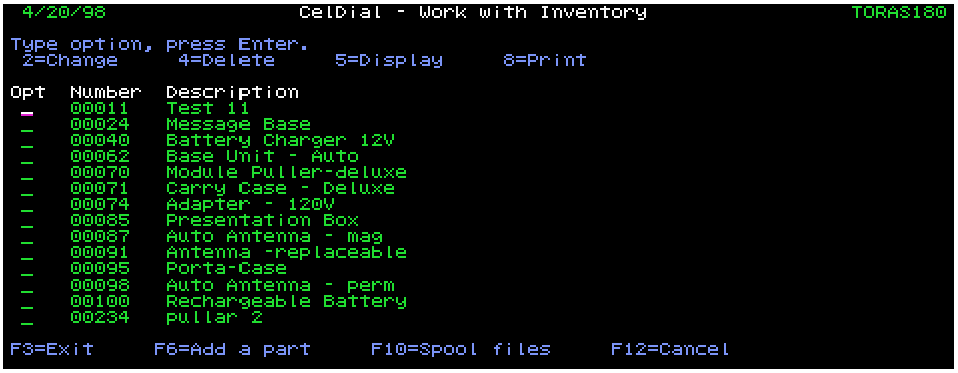
The Parts Inventory application was originally written as a green screen, 5250 based, RPG/400 application. We will convert it to a client-based Java application.

The application accesses an AS/400 database - the parts file. You can view detailed information about a specific part, update that information or delete the part from the inventory. The company is CelDial Communications and the parts in its inventory are cellular telephones and other accessories. The first screen of the application is a menu screen that allows you to either do a customer inquiry or to work with the parts inventory.



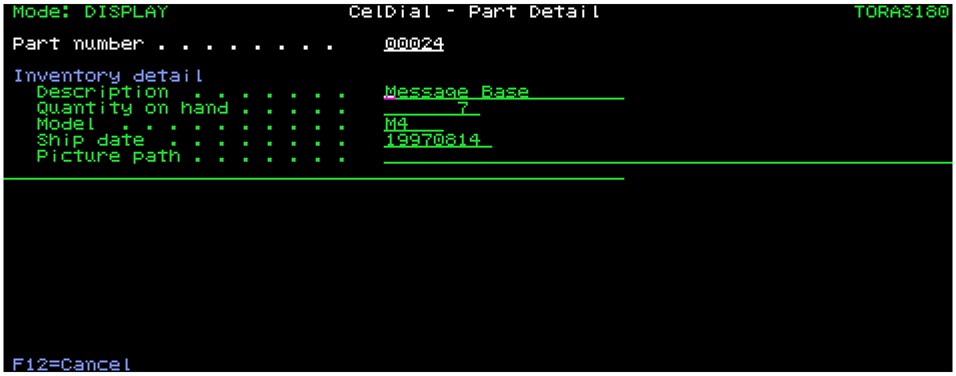
```
CELDIAL                        CelDial Communications
Select one of the following:

     1. Customer inquiry
     2. Work with inventory









Selection or command
===> _

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
F13=Information Assistant   F16=AS/400 main menu
```

The Work with Inventory screen uses a subfile to list the various parts available.



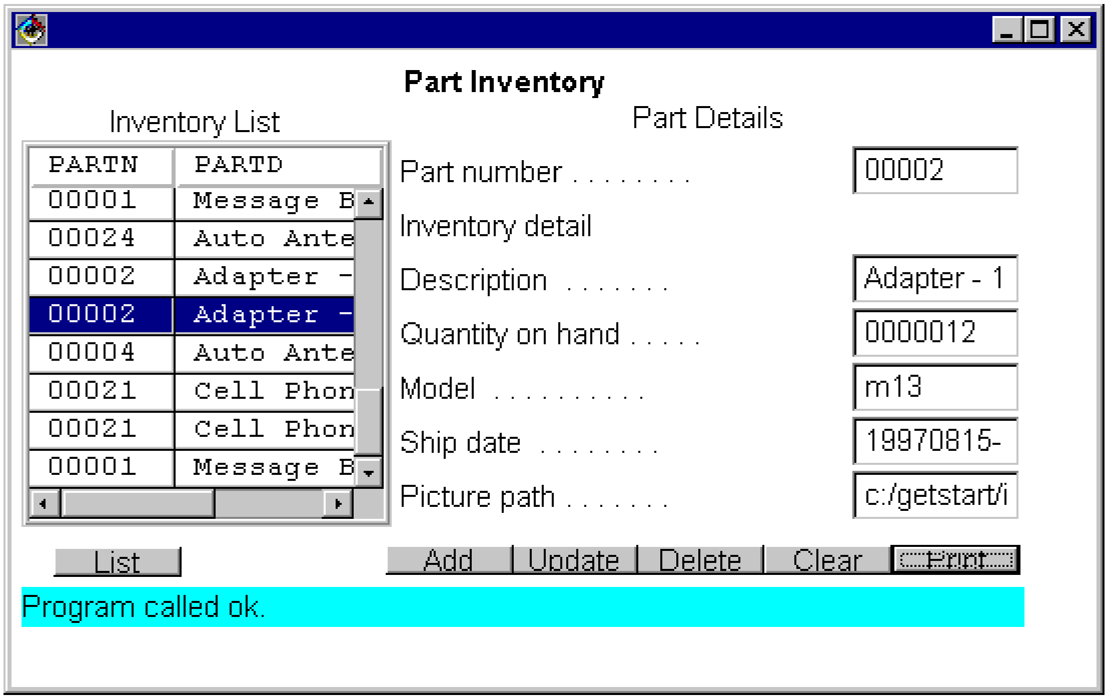
```
 4/20/98                    CelDial - Work with Inventory                  TORAS180

Type option, press Enter.
 2=Change     4=Delete     5=Display     8=Print

Opt  Number  Description
 _   00011   Test 11
 _   00024   Message Base
 _   00040   Battery Charger 12V
 _   00062   Base Unit - Auto
 _   00070   Module Puller-deluxe
 _   00071   Carry Case - Deluxe
 _   00074   Adapter - 120V
 _   00085   Presentation Box
 _   00087   Auto Antenna - mag
 _   00091   Antenna -replaceable
 _   00095   Porta-Case
 _   00098   Auto Antenna - perm
 _   00100   Rechargeable Battery
 _   00234   pullar 2

F3=Exit     F6=Add a part     F10=Spool files     F12=Cancel
```

From this screen you can select the part to be changed, deleted, displayed, or printed. You can also add a part. Another screen displays the information based on your selection.



```
Mode: DISPLAY                CelDial - Part Detail                         TORAS180

Part number . . . . . . . .    00024

Inventory detail
  Description  . . . . . . .   Message Base
  Quantity on hand . . . . .           7
  Model  . . . . . . . . . .   M4
  Ship date  . . . . . . . .   19970814
  Picture path . . . . . . .

F12=Cancel
```

The Java application that is used in the lab exercises is based on this application. To save time the menu screen is not created in the Java version of the application. As well, the two separate screens above are combined into a single screen that maintains all of the original functionality. The Java application, however, is created and runs in a client environment, accessing AS/400 data, programs and data queues. Here is what the finished application looks like in the Visual Composition Editor:



Part Inventory

Inventory List

PARTN	PARTD
00001	Message B
00024	Auto Ante
00002	Adapter -
00002	Adapter -
00004	Auto Ante
00021	Cell Phon
00021	Cell Phon
00001	Message B

Part Details

Part number 00002

Inventory detail

Description . . . . Adapter - 1

Quantity on hand 0000012

Model m13

Ship date 19970815-

Picture path . . . . c:/getstart/i

Program called ok.

Lab Environment

Your Lab Environment and Requirements for the VisualAge Java development environment:

AS/400

- RISC model OS/400 V4R2+ or IMPI V3R2+
- AS/400 Developer Kit for Java (5769-JV1)
- TCP/IP and all AS/400 host servers are started (STRHOSTSVR *ALL and STRTCPSVR *DDM)

PC

- Intel based; 48 Mg Memory (min); 300 Mg hard drive
- Windows 95 or Windows NT
- VisualAge for Java - Enterprise Edition 2.0 (with the Enterprise Toolkit for AS/400 installed)
- AS/400 Toolbox for Java (shipped with VisualAge for Java Enterprise 2.0)

VisualAge for Java - What You Need to Know to Get Started

IBM's VisualAge for Java is a complete **IDE** for the creation of Java applications and applets. It is designed not only to create Java applets, downloadable from an Internet server, but also to create full function business applications. Developers can extend server-based applications to communicate with Java clients on the Internet or intranet. VisualAge for Java creates 100% pure Java compatible applications, applets and Java Beans.

VisualAge for Java is a member of the VisualAge family of products and uses the same Visual Composition Editor as in the other VisualAge products such as SmallTalk, C++, and Generator. This Visual Composition Editor can be considered the 'heart' of VisualAge for Java.

Unlike some other development environments, VisualAge for Java enables a developer to build and run applications, applets and code snippets interactively without the need to compile the code. Along with the execution of the applications and applets, there is a full debug capability.

In fact, VisualAge Java comes with the following core components:

1. **Integrated Development Environment (IDE)**
   a. Hierarchy browser for viewing:
      i. projects
      ii. packages
      iii. classes
      iv. methods
   b. Editor
   c. Debugger
   d. Applet viewer
   e. Team support
   f. Java Class Libraries
   g. Visual Composition Editor

2. **Enterprise Access Builder (EAB)**
   a. Data Access Builder (DAX)
   b. Proxy Builder, J2C++ Builder, CICS Builder

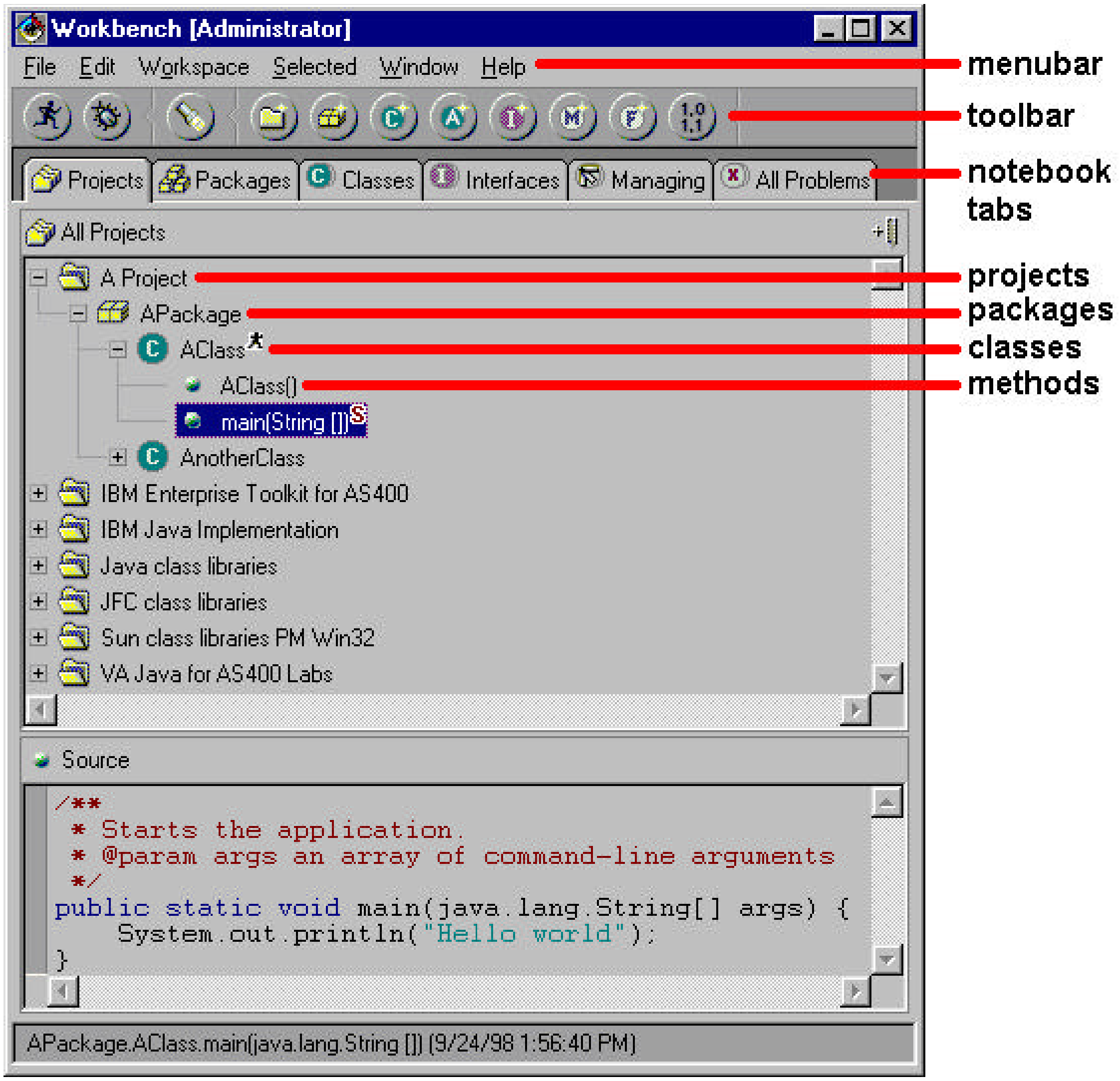
Java Support

Java is a collection of classes built from the ground up, following object-oriented principles. The lowest component (part in some other languages such as SmallTalk) is a class. A class may be joined with other classes and stored as a composite class. These classes are grouped together and placed into packages. The packages can be further grouped into projects. On the VisualAge for Java workbench, the structure of your application, the components that you import and the components that you create are displayed according to this hierarchy.

As an application is created in VisualAge for Java, each object is given some features - properties, events and methods. We will not cover these in detail here, other than to say that when creating the application, it is necessary to connect the features of one component to the features of another component. This is how the application will ultimately execute. The connections (or wires as some developers call them), that are drawn visually by the developer are used to generate some underlying Java code. This code can be viewed in the VisualAge for Java editor.

VisualAge for Java Workbench

The VisualAge for Java **Workbench** is a multiple paned **IDE** for Java development.



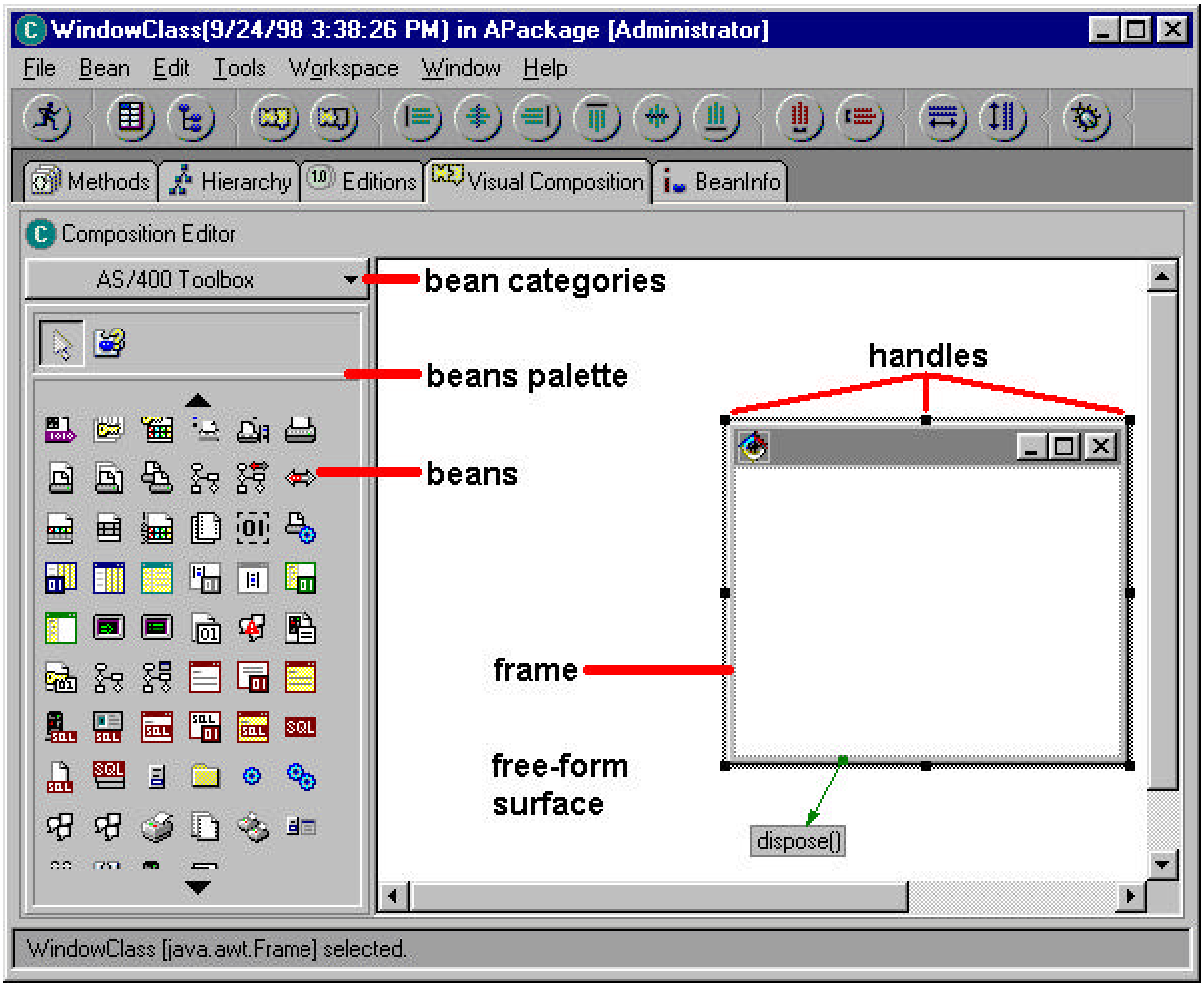
```
/**
 * Starts the application.
 * @param args an array of command-line arguments
 */
public static void main(java.lang.String[] args) {
    System.out.println("Hello world");
}
```

APackage.AClass.main(java.lang.String []) (9/24/98 1:56:40 PM)

The **Workbench** divides your Java applications into:

- **Projects**. These are like AS/400 libraries. They allow you to partition your applications into manageable units. These are VAJava-unique constructs. Projects contain multiple packages.
- **Packages**. These are like AS/400 ILE RPG service programs. They allow you to divide your application pieces into easily reused units. These are Java language constructs. Packages contain multiple classes.
- **Classes**. These are like AS/400 ILE RPG modules that contain multiple fields (variables) and functions. Classes define all the variables and methods (procedures and subroutines in RPG) that are needed to create a Java object.
- **Methods**. These contain all the actual code your program or application will use. They are like AS/400 ILE RPG procedures and subroutines. However, unlike RPG, Java executable code can only exist in methods and the methods can only exist inside classes.

The Visual Composition Editor



WindowClass [java.awt.Frame] selected.

There are various components that make up the Visual Composition Editor:

- **Free-form surface**. This is the large open area in the VCE. It is like a blank sheet of paper or a work area where you can add, manipulate, and connect the beans that you use to create your composite bean. Some of the functions you can perform on the free-form surface include: add visual beans; add nonvisual beans to build the application logic for a composite bean; delete beans; and connect beans to define behavior.
- **Frame**: The window within the free-form surface on which you place all the visual elements of your GUI.
- **Beans palette.** The palette is located on the left side of the VCE. It provides building blocks that you can use to construct a program element. It consists of categories in a drop-down list, each one containing groups of beans. the area on the left containing: Bean categories, a container for beans; Beans, the parts to build your GUI; and the Sticky check box that will allow you to perform multiple drops without returning to click on the bean. To add multiple instances of the same bean, enable **Sticky** by holding Ctrl while selecting the bean.
- **Handles.** Small boxes shown around the frame that you can use to manipulate the size of beans such as a frame and labels.

Setup

During this lab you will need a userid and password for the AS/400 and you will need to know the name of the AS/400 system. The userid and password will be **JAVAxx** where **xx** is the number 01 to 99 (two digits number). This information will be given to you by your instructor. Please fill in this information below so that you have it for reference during the lab.

My AS/400 **userid** is________________________________

My AS/400 **password** is_____________________________

Exercise 1: Creating Classes for the Inventory Application

What This Exercise is About

In this exercise, you are going to take a 5250 screen on an AS/400 and convert it to a Java AWT screen. You will then create a subfile in Java to access and display data from the PRODDTL file on an AS/400. Finally, you create a program call class to call an RPG program on an AS/400 to print a record you selected. All of this is accomplished using the Enterprise ToolKit for AS/400 (ET/400) feature of VisualAge for Java.

What You Should Be Able to Do

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

- Create classes for display file records
- Change the Java class files
- Create a subfile class from a database file
- Create a class to call an AS/400 print function

Starting VisualAge for Java

1. Click the **Start** button on the left side of the taskbar.
2. Select **Program ->IBM VisualAge for Java for Windows->IBM VisualAge for Java**

Converting the Inventory List Display File

You will begin by converting the Inventory List display file record using the Convert SmartGuide.

Select the project **VA Java for AS400 Labs** then the package **VJ400LAB2**. Right click on the package to display a pop-up menu then click on **Tools** then **ET/400** then **Convert Display File...**. On the Convert SmartGuide specify the following information:

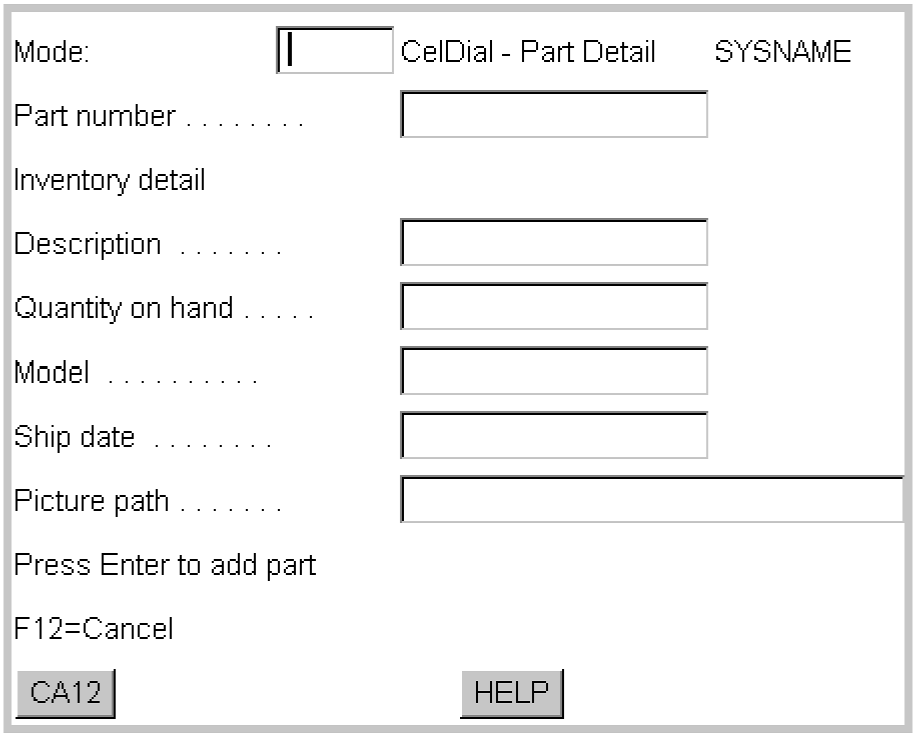
1. A server such as AS400A from the **Server** list
2. A library called ADTSLAB in the **Library** field
3. Expand the tree to see a list of the display files
4. Double-click on display file WRKINVFRM to display its record list
5. Click on the parts detail record INVDETL

Leave all the other fields with their default values and select **Next** until you can select **Finish.** Java classes are generated and placed under the selected project and package in the Workbench.

Modify the Converted Screen

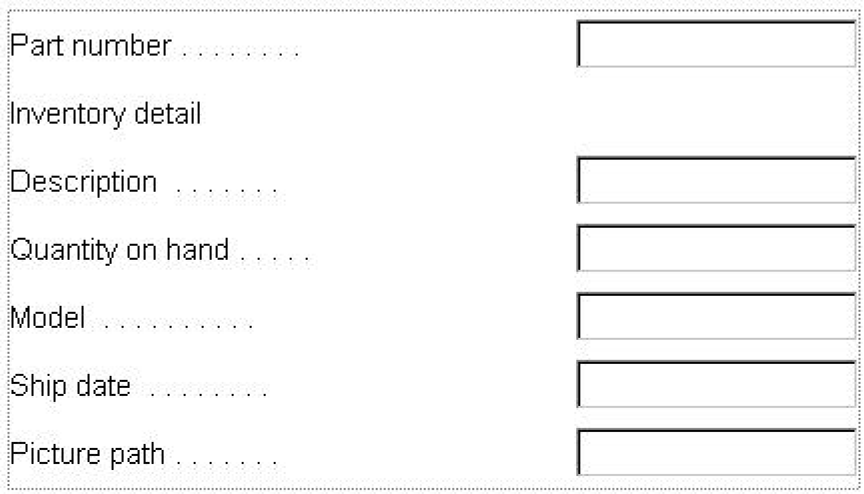
You will take the generated bean and bring it into the Visual Composition Editor (VCE) and visually delete the text and buttons which are no longer needed. You will also change the properties of the entry fields to accept input.

1. Select the generated class INVDETL in the package **VJ400LAB2** under the project **VA Java for AS400 Labs**.
2. Double click on this class. The Workbench disappears and the browser opens on the methods in the new class.
3. Click on the Visual Composition tab. Select **Bean** then **Construct Visuals from Source** from the **Bean** pull-down menu. Click **Proceed** to replace the visual contents. The generated class is presented as separate components. Resize the frame as necessary to show all the components so that you can click on them. You can click on each component and delete the components you do not want to keep.


Mode:
CelDial - Part Detail
SYSNAME
Part number . . . .
Inventory detail
Description . . . .
Quantity on hand . . . . .
Model . . . .
Ship date . . . .
Picture path . . . .
Press Enter to add part
F12=Cancel
CA12
HELP

4. Right click on each of the following components and from the pop-up menu select **Delete**:
   - Mode: and the entry field for Mode
   - CelDial - Part Detail
   - SYSNAME
   - Press Enter to add part
   - F12=Cancel
   - CA12
   - HELP

5. Right click on the **Part number** entry field and select **Properties**. Click the field next to the **enabled** value and select **True** from the list. This enables the text field to be input capable. Leave the Properties window open. Repeat this step for each field.
6. Close the Properties window.
7. Save the work you have done. To save your bean, from the **Bean** menu select **Save Bean**.



Part number . . . .

Inventory detail

Description . . . .

Quantity on hand . . . . .

Model . . . .

Ship date . . . .

Picture path . . . .

8. Close the Visual Composition pane.

NOTE: It's O.K. to find **X** in front of your class. It is a symbol to tell you that it cannot find some methods the class refers to as we have just deleted some of them. This will not affect your code.

Next you will create a subfile class for the Inventory List.

Creating a Subfile from a Database File

Click on the package **VJ400LAB2**. Click on **Tools** then **ET/400** then **Create Subfile...**. On the Create Subfile SmartGuide specify the following information:

w **Create a new subfile class** radio button, press **Next**

w A server __________ from the **Server** list (AS/400 system name)

w A database file PRODDTL in the **File** field

w A library called ADTSLAB in the **Library** field

w Default settings for the class and package names, press **Next**

w Select and add columns PARTNO and PARTD to the columns create list

Leave all the other fields with their default values and select **Next** until you can select **Finish.** Java classes are generated and placed under the selected project and package in the Workbench.

You have finished creating a subfile class for the Inventory List. Next you will create a program call class to call the print inventory function to print an inventory item in the Part Details.

Creating a Program Call Class to Call an AS/400 Print Inventory Function

Select the package, **VJ400LAB2**. Right click on the package to display a pop-up menu then click on **Tools** then **ET/400** then **Create Program Call...**. On the Create Program Call SmartGuide specify the following information:

- **Create a new program call class** radio button, press **Next**
- A server __________ from the **Server** list (AS/400 system name)
- A program called PRTINV in the **Program** field
- A library called ADTSLAB in the **Library** field
- A name for the generated class such as PRTINV in the **Class** field, press **Next**
- Click ‘insert’ button to add a parameter with the name partno, datatype of character and size of 5, press **OK**.

Leave all the other fields with their default values and click **Finish.** Java classes are generated and placed under the selected project and package in the Workbench.

Next you will add the classes generated by the SmartGuides as beans in the Visual Composition Editor to build your client interface visually.

Exercise 2: Adding the Inventory List and Print Function Classes as Beans

What This Exercise is About

In this exercise you add beans for the Inventory List, Part Details and call to the print inventory function visually to the InventoryList class you will be creating. This exercise also introduces you to the concept of visual construction of visual and non-visual parts/beans.

What You Should Be Able To Do

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

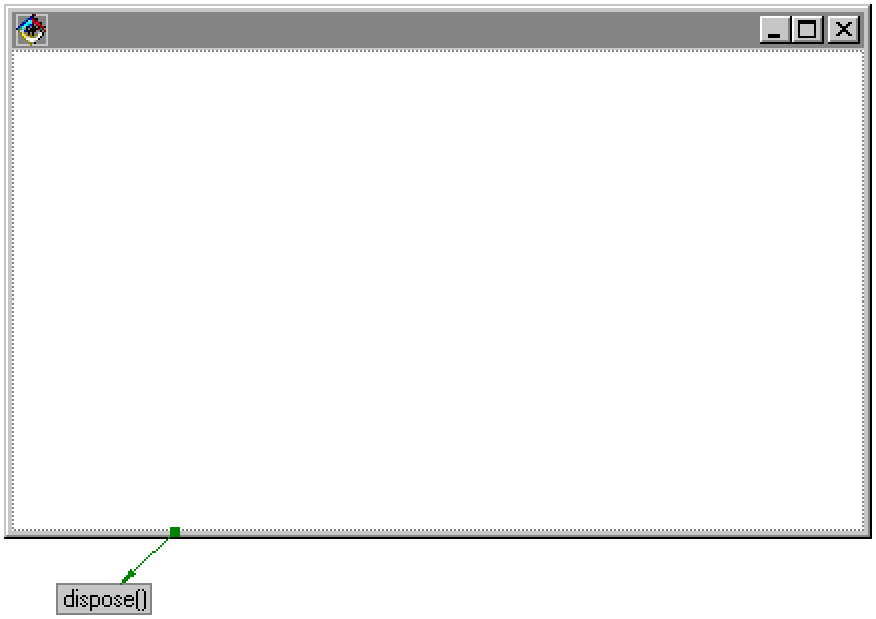
- Create a class
- Create a label
- Create a status label
- Add a push button for the Inventory List
- Add the Inventory List bean
- Create a Label for the Inventory List
- Add the Part Details bean
- Create a Label for the Part Details
- Add the Program Call bean

Creating a Graphical User Interface (GUI) Using VCE

Select the **VA Java for AS400 Labs** project and then the **VJ400LAB2** package. Select Add from the Selected pull-down menu. Then select Class.... On the Create Class SmartGuide specify the following information:

- **InventoryList** in the **Class Name** field
- **java.awt.Frame** in the **Superclass** field
- Select the **Compose the class visually** check box

Click **Finish**. The SmartGuide disappears and the Visual Composition Editor appears.



You can resize the window frame by clicking on the corners of the frame and dragging it.

Creating a Title for the Screen

Use a Label bean to provide a title for the screen.

1. Select the AWT category from the Beans Palette list.
2. Click on the **Label** bean and drop the **Label** bean onto the upper area of the frame by moving the '+' cursor to the desired position and single click using the left mouse button.

   Note: If you move the cursor to any bean on the Beans Palette, help will show you the name of the bean.
3. Right click on the **Label1** bean and select **Properties**.
4. Click the field next to the **text** value and type Part Inventory.
5. Click the field next to the **alignment** value and select **CENTER** from the list.
6. Click the field next to the **font** value and click the list to open the Font window. Select **Bold** from the **Style** list.
7. Click **OK**. Leave the Properties window open.
8. Use the handles to increase the width of the button bean to accommodate the text.

Adding a Status Bar

Use another Label bean to provide a status bar to display information messages for the application.

1. Select the **Label** bean.
2. Place the **Label** bean on the bottom area of the frame.
3. Use the handles to increase the width of the **Label** bean to the size of the frame.
4. Right click on the **Label** bean.
5. On the Properties window, click the field next to the **beanName** value and type LabelStatus.
6. Blank out the field next to the **text** value.
7. Change the background color to cyan. Choose the **Basic** radio button then choose the color cyan. Click **OK**.

Adding a List Button for the Inventory List

1. Click on the **Button** bean and drop the **Button** bean just above the status bar on the left of the frame.
2. Right click on the button.
3. On the Properties window, click on the field next to the **label** value and type List.
4. Click on the field next to the **beanName** value and type ButtonList.

Now it is the time to add the beans generated using ET/400 feature in exercise 1 .

Adding the Inventory List Class as a Bean to the Application

This class is the Inventory List generated by the Convert SmartGuide which contains a parts inventory list.

1. Select **Choose bean...** from the Beans Palette.
2. Click on the **Browse...** button and Choose Bean window appears.
3. Key in **PRODDTL** in the Pattern field. Select the class from the VJ400LAB2 package. Click **OK**. The **VJ400LAB2.PRODDTL** name has been added to the Choose Bean window as the Class name. Do not fill in the Name. Click **OK** to add the bean.
4. Move the crosshairs inside the frame and left click to drop the bean.

The Inventory List PRODDTL bean is added to the frame. Resize the frame as necessary to show the bean. You may need to resize the status label and other components as you resize the frame. Next you will create a title for the Inventory List.

Creating a Title for the Inventory List

Use a Label bean to provide the title Inventory List. Click on the **Label** bean and drop the **Label** bean onto the upper area of the Inventory List bean by moving the '+' cursor to the desired position and single click using the left mouse button. Right click on the **Label2** bean. On the Properties window change the **text** value to Inventory List and the **alignment** value to **CENTER**. Use the handles to increase the width of the label bean to accommodate the text. Leave the Properties window open.

Adding the Inventory Part Details Class as a Bean

This class is the detail form generated by the Create Subfile SmartGuide which contains the details of one inventory part.

1. Select **Choose bean**... from the Beans Palette.
2. Click on the Browse... button and enter INVDETL as the class name. Select the class from the VJ400LAB2 package. Click **OK**. The **VJ400LAB2.INVDETL** name has been added to the Choose Bean window as the Class name. Click OK to add the bean.
3. Move the crosshairs inside the frame and left click to drop the bean.

The Inventory Part Details is now added to the frame. Resize the frame as necessary to show the bean. Now you need to create a title for the Inventory Part Details.

Creating a Title for the Parts Details

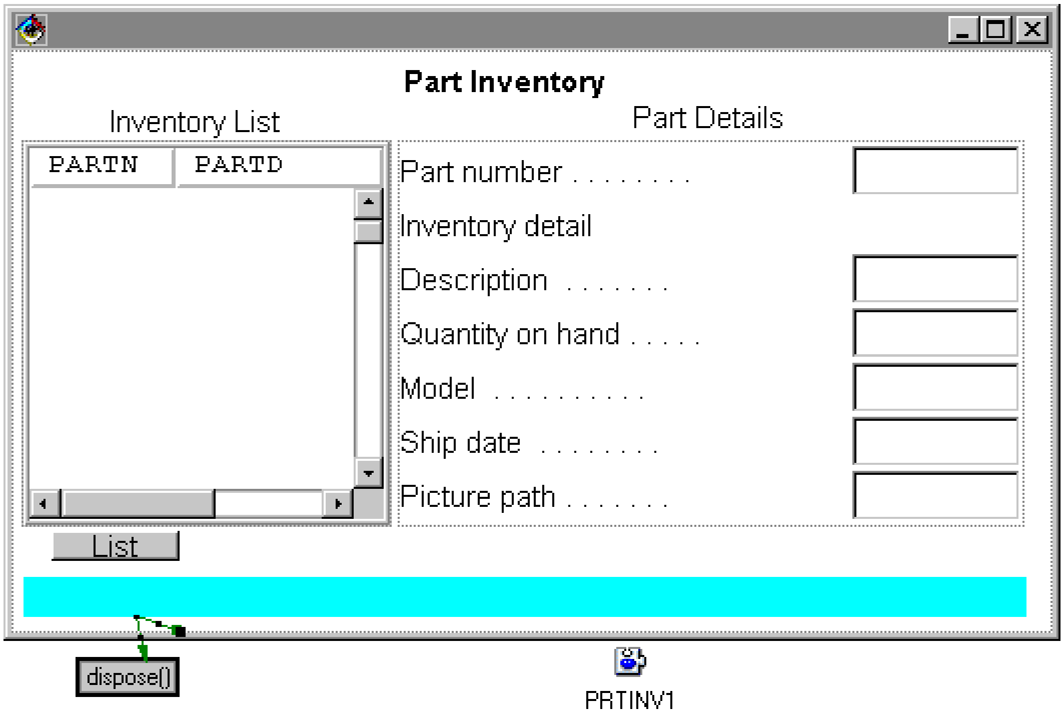
Use a Label bean to provide a title called Part Details. Click the **Label** bean and drop the **Label** bean onto the upper area of the **Part Details** bean by moving the '+' cursor to the desired position and single click using the left mouse button. Right click on the **Label3** bean and on the Properties window change the **text** value to **Part Details** and the **alignment** value to **CENTER**. Close the Properties window.

Resize the label as necessary to show the text. You are now ready to add the Print Inventory Bean.

Adding the Print Inventory Class as a Bean

This class contains the Java code to call the ILE RPG program PRTINV and passes a parameter partno to print the part number detail on AS/400. If the print is successful two spool files result. They are QSYSPRT and INVPRINT. QSYSPRT contains information indicating the program call was successful. INVPRINT contains the part detail.

1. Select **Choose bean...** from the Beans Palette.
2. Click **Browse...** and enter a few characters of the class name PRTINV. Select the class. Click **OK**. The **VJ400LAB2.PRTINV** name has been added to the Choose Bean window as the Class name. Click **OK** to add the bean.
3. Move the crosshairs to the bottom of the frame in the free-form surface and left click to drop the bean. Resize the frame as necessary to show the bean.


Part Inventory
Inventory List
Part Details
PARTN
PARTD
Part number . . . .
Inventory detail
Description . . . .
Quantity on hand . . . . .
Model . . . .
Ship date . . . .
Picture path . . . .
List
dispose()
PRTINV1

Exercise 3: Adding Connections to Enable Buttons and Functions

What This Exercise is About

In this exercise you add connections to open the database file for reading and writing, removing records, populating records and adding and enabling buttons for the Part Details.

What You Should Be Able To Do

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

- Open a database file for reading and writing
- Display messages on the status line
- Remove records from the Inventory List
- Copy methods from another project
- Populate the Part Details
- Add buttons for the Part Details
- Enable all the Part Details buttons

Opening the File for Reading and Writing

When you open the window you want to open the data file automatically and when you close the window the file will be closed as well.

1. Click on the title bar of the application frame, then right click. In the pop-up menu that appears, select **Connect** then **Connectable Features...**. Select the event **windowOpened**. Click **OK**. The mouse pointer changes, indicating that you are in the process of making a connection.

2. Move the mouse pointer to the Inventory List bean then right-click. Select **Connectable Features...** from the pop-up menu that appears. Select the method **OpenSequentialFileReadWrite().** Click **OK**. This connection opens the database file for reading and writing records when the application window opens.

Closing the File

1. Click on the application frame, then right click. In the pop-up menu that appears, select **Connect** then **Connectable Features...**. Select the event **windowClosing**. Click **OK**.
2. Move the mouse pointer to the Inventory List bean, then right click. Select **Connectable Features...** from the pop-up menu that appears. Select the method **closeSequentialFileAfterCommit().** Click **OK**. This connection closes the database file for reading and writing records when the application window closes.

Displaying Message “Getting data from AS/400, please wait..”

1. Select the **List** button, click the right mouse button, and select **Connect** from the pop-up menu.
2. Position the mouse pointer over **actionPerformed** and left click.
3. Make a connection from the **List** button to the Status label bean and click on the left mouse button and a pop-up menu appears. Select **Connectable Features...** and the End connection to window appears. Select the **setText(String)** method. Click **OK**.
4. Right click on the incomplete connection line, and select **Properties**. Deselect the **Pass event data()** check box if visible.
5. Select the **Set parameters...** button.
6. Click on the field next to the **text** value and type **Getting data from AS/400**, **please wait...**
7. Click **OK**. Click **OK** again.

Now when the **List** button is pushed, a message is sent to the status area saying **Getting data from AS/400, please wait...**

Removing All Records from the Inventory List

Before you populate the subfile with data records, you have to remove any records in the list.

1. Connect **actionPerformed** event of the **List** button to the Inventory List PRODDTL bean.
2. Select **Connectable Features...** on the pop-up menu, and select the **removeAllSubfileRecords()** method on the methods list. Click **OK**.

Now when the **List** button is pushed, the records in the subfile InventoryList will be removed.

Populating the Inventory List with Records

Then you populate the subfile with data records from the **PRODDTL** file.

1. Connect **actionPerformed** event of the **List** button to the Inventory List PRODDTL bean.

2. Select **Connectable Features...** on the pop-up menu, and select the **readAllRecords()** method from the method list. Click **OK**.

Copying Methods from Another Project

You need several new methods to populate the Inventory List PRODDTL bean and also to add, update, and clear items from the Parts Detail INVDETL bean and also to enable all the buttons. Usually, you would have to create these methods yourself, however, you can copy the methods from the IBM Enterprise Toolkit for AS400 project.

1. Click on the Workbench pane.
2. Expand the **VA Java for AS400 Labs** project in the project window.
3. Expand the **ZY400APP** package.
4. Expand the **InventoryList** class.
5. Select **addRecord()**, **clearForm()**, **enableAllButtons()**, **showDetails(String[])**, and **updateRecord(int)** methods by holding down the CTRL key and left clicking on each method. Once you have selected all the methods, right click to pull up the pop-up menu and select the **Reorganize** option then **Copy...**.
6. The Copying Methods window appears. Click **Browse...** and select the class name **InventoryList** and the package name **VJ400LAB2**. Click **OK**. The VJ400LAB2.InventoryList name has been added to the Copy Methods window. Deselect the **Rename(copy as)** checkbox. Click **OK**. Click **OK** again.
7. Note that there are red Xs in front of the methods you have just copied. These errors are due to undefined variables and will resolve as soon as you add the appropriate connections. When you add buttons from the Part Details ensure that you use the correct bean name for each button so that the methods work correctly.

Populate the Part Details when the Part is Selected in the Inventory List

Next you can connect the Inventory list PRODDTL bean to Part Details INVDETL bean so that the detail information of the selected part will be shown on the Parts Details screen.

1. Click on the Visual Composition pane. Connect **ItemStateChanged** event of the Inventory List PRODDTL bean to the free form surface. Select **Event to Code...** on the pop-up menu, and on the Event-to-Code Connection from window click on the Method drop down and select the **showDetails(String[])** method. Deselect the **Pass event data()** check box if visible. Click **OK**.
2. Right click on the previous connection and select **Connect** then **rowData**. Connect to the Inventory List PRODDTL bean. Select **Connectable Features...** and on the End connection to window select the Method radio button and then select the **getRecordAsStrings(int)** method. Click **OK**.

3. Right click on the incomplete connection and select **Connect** then **iRecordNumber**. Connect to the Inventory List PRODDTL bean. Select **Connectable Features...** and on the End connection to window select the Method radio button and then select the **getFocusRecordNumber()** method. Click **OK**.

The three connections you have just made allow the parts item selected on InventoryList screen to display its detail information using the showDetails method. The record number of the selected record is passed as an arguement for the method. Then the subfile class gets the record details (using the getRecordAsStrings method) and writes to the Parts Details screen (using the showDetails methods) .

Adding Buttons for the Part Details

1. Click on the **Button** bean and drop it within the frame at the bottom.
2. Repeat this action to add another four push buttons. All the buttons may not fit - just drop them overlapping if necessary and then resize them all. Now you have added all the remaining buttons for the application. Next you add names for each of the buttons.
3. Right click on the first button and select **Properties** and the Properties window for the button appears.
4. Click on the field next to the **label** value and type Add.
5. Click on the field next to the **beanName** value and type ButtonAdd.

   Note: You must use the name ButtonAdd because the method enableAllButtons( ) uses the ButtonAdd bean.

6. Click on **Show Expert Features** checkbox. Select the **False** value for the **enabled** field.
7. Leave the Properties window up. Move it so that you can see all the buttons.
8. Repeat this process for each button, naming them Update, Delete, Clear and Print. Remember to change the beanName accordingly, ButtonUpdate, ButtonDelete, ButtonClear and ButtonPrint.
9. Close the Properties window.
10. To save your work, from the **Bean** menu select **Save Bean**.

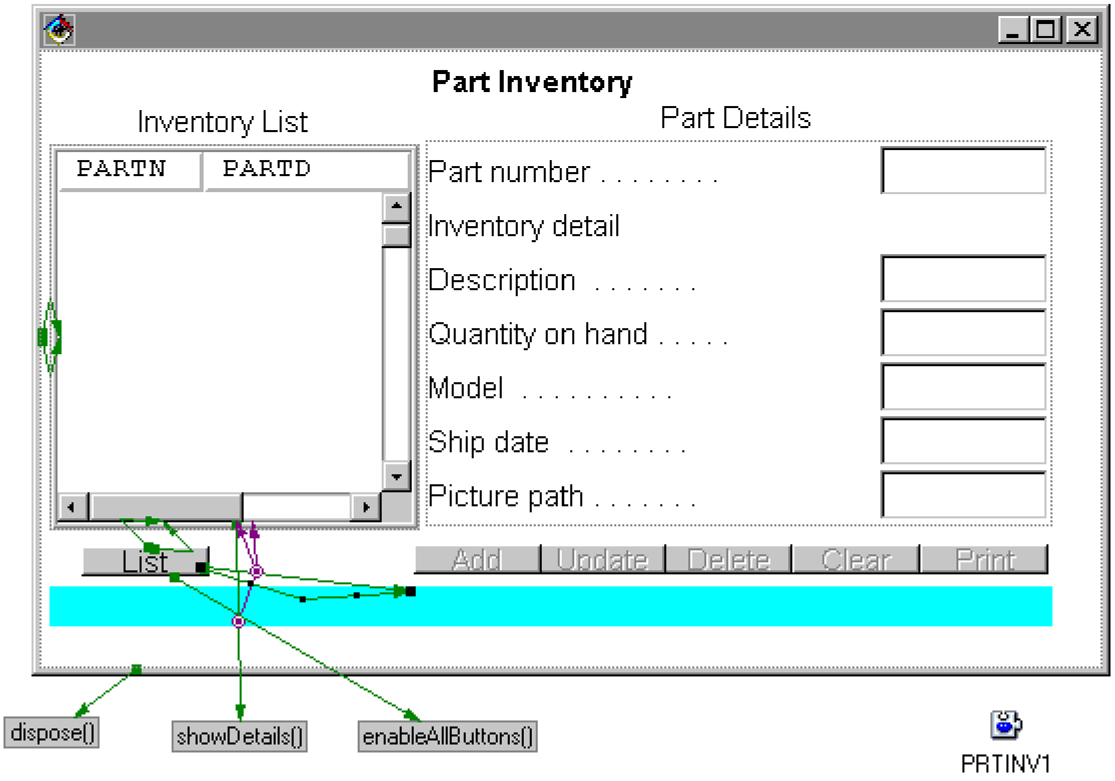
Enabling All the Buttons

1. Connect **actionPerformed** event of the **List** button to the free form surface. Select **Event to Code...** on the pop-up menu, and on the Event Code window select the **enableAllButtons()** method. Click **OK**.

When you click on the **List** button all buttons are enabled. Next you add a connection between the **List** button and the Status label to display a message that data is retrieved from the AS/400 when you click on the **List** button.

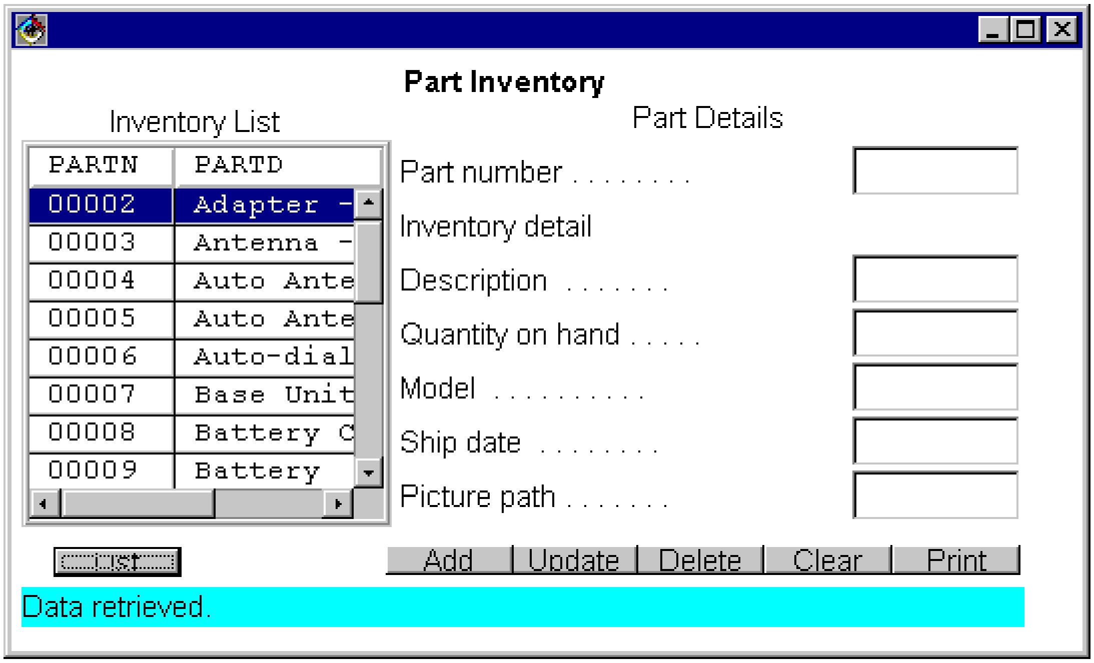
Displaying the Message "Data retrieved"

1. Connect actionPerformed event of the List button to the Status label. Select Connectable Features... on the pop-up menu, and on the End connection to window, select the setText(String) method. Click OK.
2. Right click on the incomplete connection line. Select Properties. Deselect the Pass event data() check box if visible. Then click on the Set parameters... button. Type **Data retrieved** for the text value. Click OK. Click OK again.


Part Inventory
Inventory List
Part Details
PARTN
PARTD
Part number . . . .
Inventory detail
Description . . . .
Quantity on hand . . . . .
Model . . . .
Ship date . . . .
Picture path . . . .
List
Add
Update
Delete
Clear
Print
dispose()
showDetails()
enableAllButtons()
PRTINV1

Testing What You Have Done So Far

1. From the **Bean** pull-down menu select **Run**. Select **Check Class Path...**. Save the beans if prompted. Click **Class Path** tab. Select **Project path**, and click **Edit...** beside it. Select the **IBM Enterprise Kit for AS400** and the **JFC class libraries** projects to the **Class path**. Click **OK**. Click **OK**.
2. Click on the **Run** button on the tool bar. VisualAge for Java saves the bean, compiles the class and runs the compiled bean.
3. A window appears displaying the GUI you have just updated. The Signon to AS400 window appears. Type your user ID and password. Click on the **List** button. You should see "Getting data from AS/400, please wait ...". Once the data has been retrieved from the AS/400, the list box is populated with data. You should see "Data retrieved" on the status bar.
4. Close the window to end the test.


Part Inventory
Inventory List
Part Details
Part number . . . .
Inventory detail
Description . . . .
Quantity on hand . . . . .
Model . . . .
Ship date . . . .
Picture path . . . .
List
Add
Update
Delete
Clear
Print

PARTN	PARTD
00002	Adapter -
00003	Antenna -
00004	Auto Ante
00005	Auto Ante
00006	Auto-dial
00007	Base Unit
00008	Battery C
00009	Battery

Data retrieved.

Exercise 4: Add, Update, Delete Records (OPTIONAL -- ADVANCED SECTION)

What This Exercise is About

In this exercise you make connections between each of the buttons under the Parts Details screen and the methods which performs the different Add, Update, Delete and Clear functions. You enable the Print button in the next exercise.

What You Should Be Able To Do

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

- Use a method to add a record
- Display messages on the status line
- Use a method to update a record
- Use a method to delete a record
- Use a method to clear a record

Enabling the Add Button

The **Add** button first gets the data from the Part Details INVDETL, adds the record to the database, and then refreshes the Inventory List PRODDTL bean.

1. Connect **actionPerformed** event of the **Add** button to the free-form surface.
2. Select **Event to Code...** on the pop-up menu, and on the Event to code window deselect the **Pass event data()** check box if visible. Select the **addRecord()** method. Click **OK**.

Displaying Message “Record added”

You can send a message “ Recod Added” to the status area when the Add button is pushed and a record is added to the database.

1. Right click on the previous connection. Select **Connect** then **normalResult**. Position the mouse over the Status label bean and left click to bring up the pop-up menu. Select **text**.
2. Right click on the incomplete connection line, and select **Properties**. Deselect the **Pass event data()** check box if visible. Click on the **Set parameters...** button. Type **Record added** for the **text** value. Click **OK**. Click **OK** again.

Enabling the Update Button

Now you connect the **Update** button to a method to retrieve the data from the Part Details, update the database and update the Inventory List PRODDTL bean.

1. Connect **actionPerformed** event of the **Update** button to the free-form surface. Select **Event to Code...** on the pop-up menu, and on the Event to code window deselect the **Pass event data()** check box if visible. Select the **updateRecord(int)** method. Click **OK**.
2. Right click on the incomplete connection line, and select **Connect** then **recNum**. Make a connection to the Inventory List PRODDTL bean. Select **Connectable Features...** on the pop-up menu, and on the End connection to window, select the **getFocusRecordNumber()** method. Click **OK**.

The connections you have just completed allow you to take the new parts details information on the Parts Details screen and update the database on the AS/400 using the updateRecord method. Then it refreshes the InventoryList screen to display the updated record.

Enabling the Delete Button

Next you add a connection between the **Delete** button and the deleteRecord method to delete the record from the database. It requires the record number of the record at focus to be passed as a paramenter.

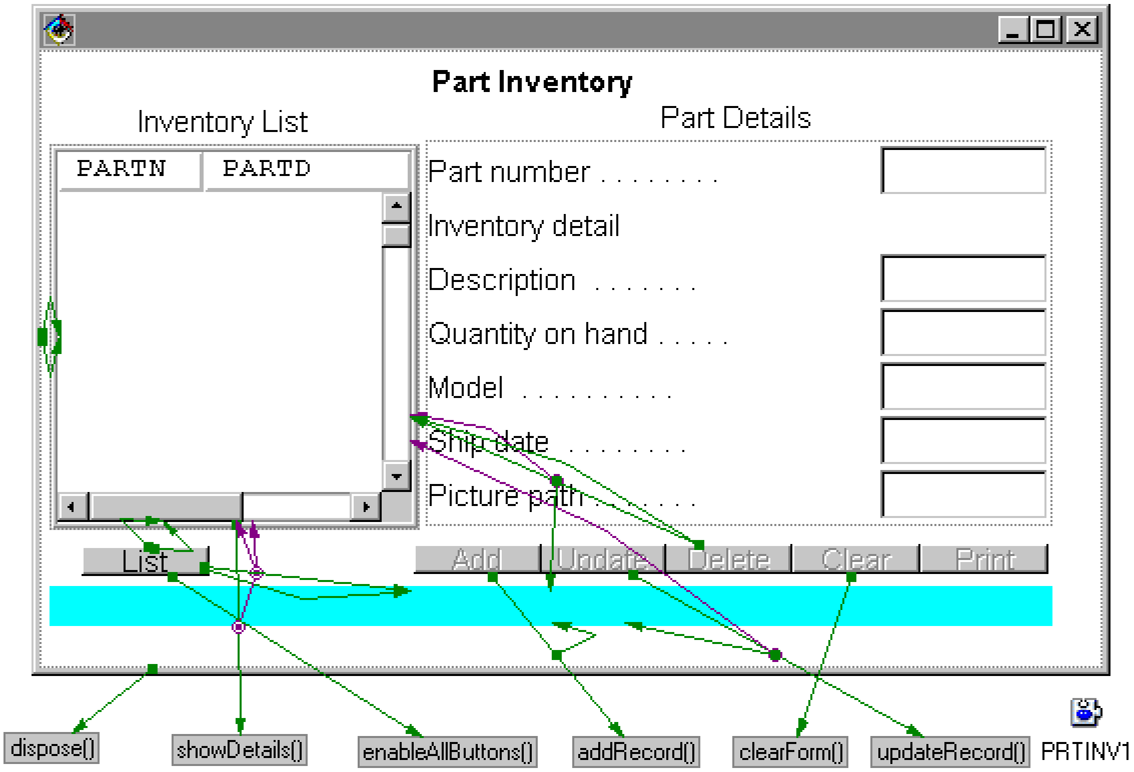
1. Connect **actionPerformed** event of the **Delete** button to the Inventory List bean. Select **Connectable Features...** and on the End connection to window select the **deleteRecord(int)** method. Click **OK**.
2. Right click on the incomplete connection line, and select **Connect** then **recordNumber** and make a connection to the Inventory List bean. Select **Connectable Features...** then the **getFocusRecordNumber()** method.

After you delete the record, you want to refresh the inventory list by reading the record again.

1. Connect **actionPerformed** event of the **Delete** button to the Inventory List bean. Select **Connectable Features...** and then select the **readAllRecords()** method from method list. Click **OK**.

Enabling the Clear Button

Connect **actionPerformed** event of the **Clear** button to the free-form surface. Select **Event to Code...** on the pop-up menu, and on the End connection to window deselect the **Pass event data()** check box if visible. Select the **clearForm()** method. Click **OK**.

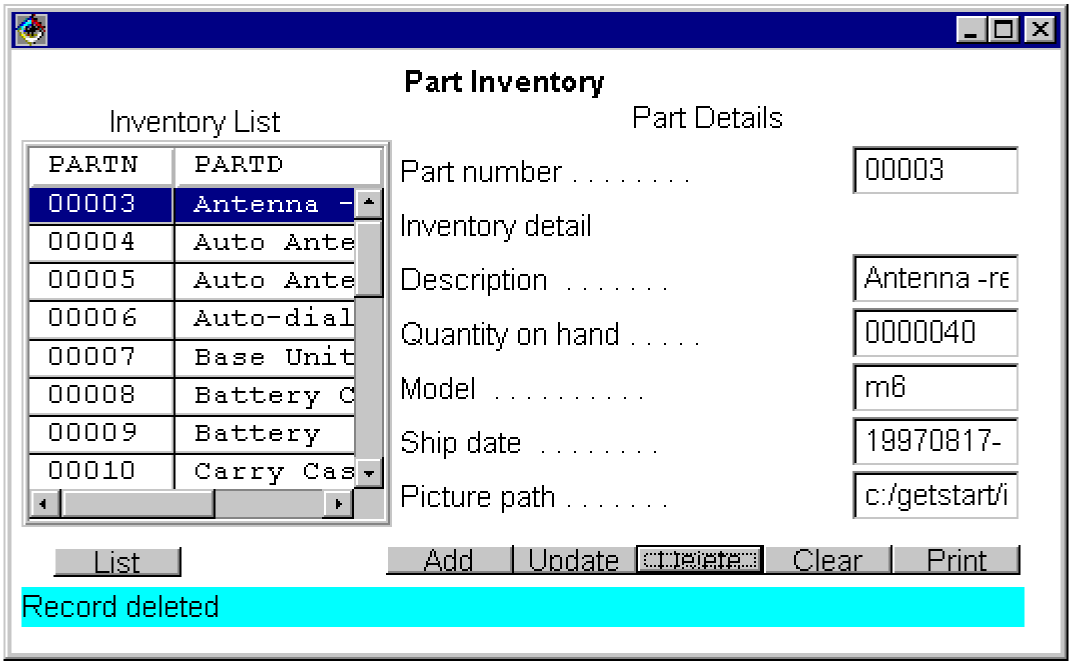


Now when the **Clear** button is pushed the fields in the Part Details screen are cleared.

Testing What You Have Done So Far

You test all the connections and buttons except the **Print** button for the Parts Details.

Select **Run**. A window appears displaying the GUI you have just updated. Test all the buttons currently enabled except **Print**. Close the window to end the test.



Part Inventory

Inventory List

PARTN	PARTD
00003	Antenna -
00004	Auto Ante
00005	Auto Ante
00006	Auto-dial
00007	Base Unit
00008	Battery C
00009	Battery
00010	Carry Cas

Part Details

Part number 00003

Inventory detail

Description . . . . Antenna -re

Quantity on hand 0000040

Model m6

Ship date 19970817-

Picture path . . . . c:/getstart/i

Record deleted

Exercise 5: Print Record Information (OPTIONAL -- ADVANCED SECTION)

What This Exercise is About

In this exercise you are going to add connections between the Print button and the PRTINV1 bean generated using the Program Call Smart Guide in exercise 1. When the Print button is pushed, a program call is made to a RPG program on AS/400 which prints the detail part information of the selected record.

What You Should Be Able To Do

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

- Display messages on the status line
- Call the print function
- Send the part number to the print function
- Run the print inventory program

Making an AS/400 Connection to call the Print Function

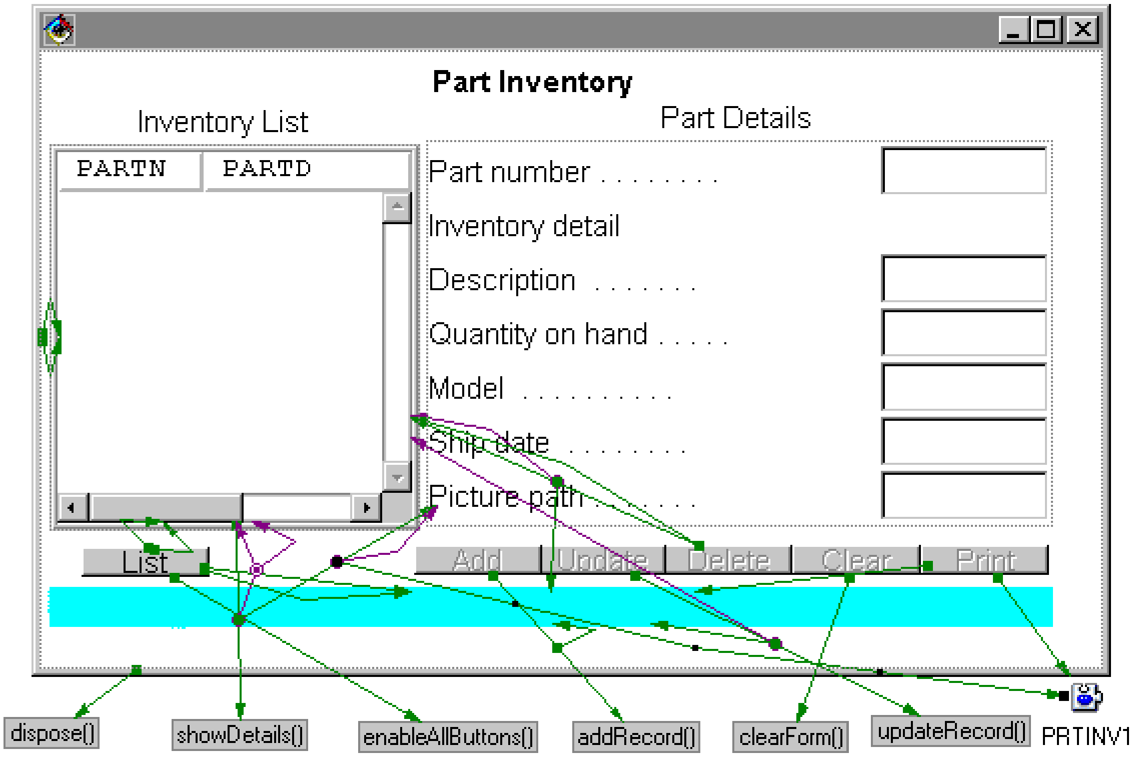
1. Connect **actionPerformed** event of the **Print** button to the PRTINV1 bean. Select **Connectable Features...** on the pop-up menu, and on the End connection to window select the **connect(com.ibm.as400.access.AS400)** method. Click **OK**.
2. Right click on the incomplete connection line, and select **Connect** then **obj**. Connect to the Inventory List PRODDTL bean. Select **Connectable Features...** and on the End connection to window select the **getAS400Server()** method. Click **OK**.

You have just added the connection between the **Print** button and PRTINV1 bean to make an AS/400 connection.

Sending the Part Number to the Print Function

1. Right click on the connection between the Inventory List PRODDTL bean and the showDetails( ) method, and select **Connect** then **normalResult**. Connect to the Part Details INVDETL bean. Select **Connectable Features...** and select the **entryFieldGetStringValue (com.ibm.ivj.as400.util.AS400VisualTextField)** method. Click **OK**.
2. Right click on the incomplete connection line, and select **Connect** then **entryField**. Connect to the Part Details INVDETL bean. Select **Connectable Features...** and on the End connection to window select the **getEntryField1PARTNO()** method. Click **OK**.

Select the previous connection and select **Connect** then **normalResult**. Connect to the PRTINV1 bean. Select **Connectable Features...** and on the End connection to window select the **partnoAsString()** method. Click **OK**.



The connections you just made between Inventory list PRODDTL and the Part Details INVDETL bean allow the data for the part to be assigned to each field in the Part Details screen, when a part is selected in the Inventory list. The record number of the selected record is obtained by getEntryField1PARTNO() method and passes on to the program call bean..

Running the Print Inventory Program on AS/400

Finally, you invoke the runProgram method in the PRTINV1 bean to run the ILE RPG print program on AS/400.

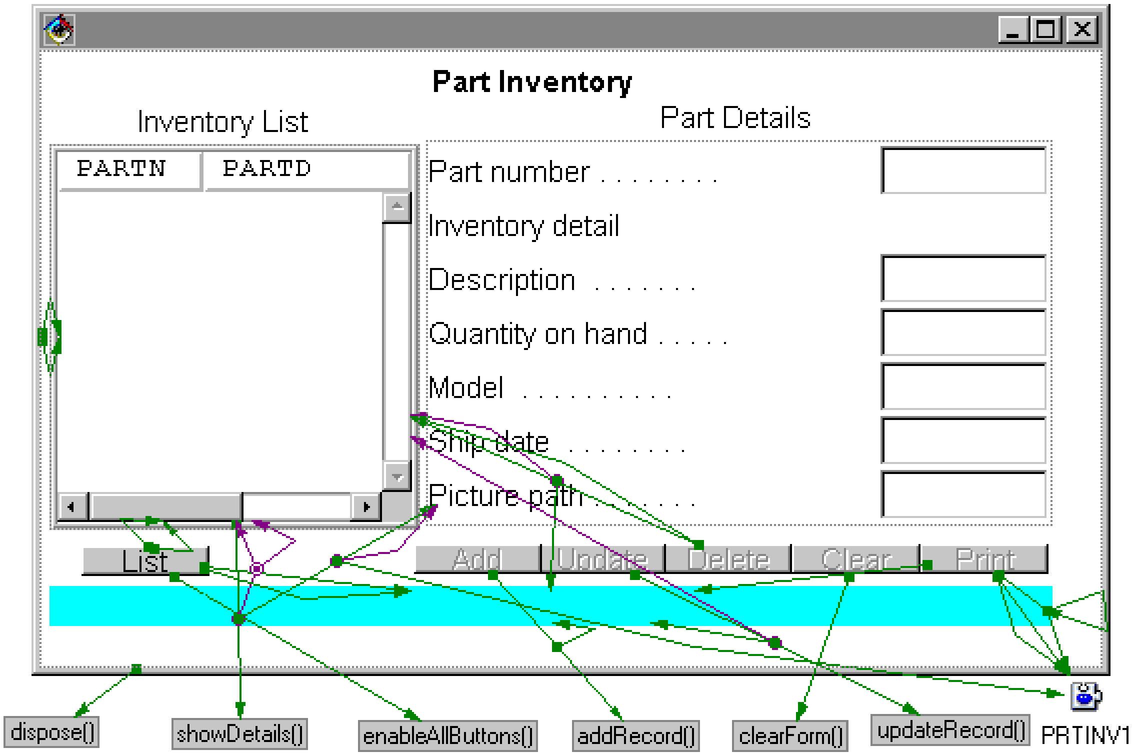
1. Connect **actionPerformed** event of the **Print** button to the PRTINV1 bean. Select **Connectable Features...** on the pop-up menu, and on the End connection to window select the **runProgram()** method. Click **OK**.

Printing the Program Completion Message

Now you want to send a message to the status label after the RPG print program is completed.

1. Connect **actionPerformed** event of the **Print** button to the PRTINV1 bean. Select **Connectable Features...** on the pop-up menu, and on the End connection to window select the **getCompletionMsg()** method. Click **OK**.
2. Right click on the previous connection. Select **Connect** then **normalResult**. Position the mouse over the Status label bean and left click to bring up the pop-up menu. Select **text**.

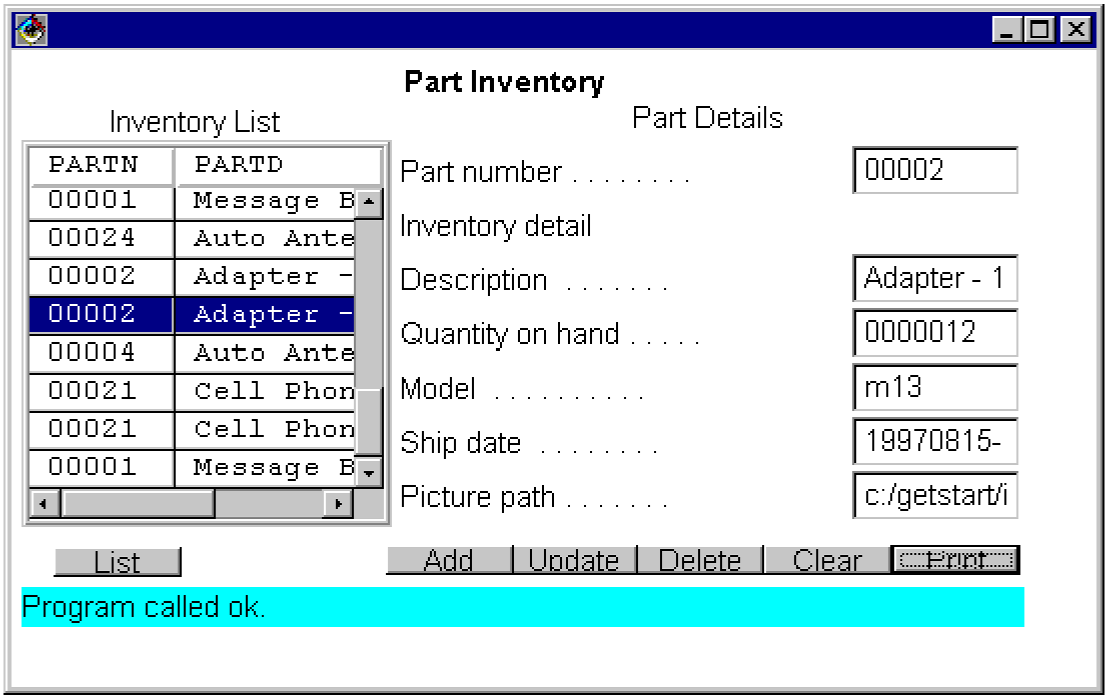
The finished application looks like this:



Next you test the finished application. Check that the **Print** button functions as designed.

Testing Your Completed Application

Select **Run**. A window appears displaying the GUI you have just updated. Click on **List** to retrieve the data from the AS/400. Select a part number from the Inventory List and click on the **Print** button. The Signon to AS/400 window may appear if the server name for the PRTINV1 bean and the INVDETL1 bean are identical. If required, type your user ID and password. The status label shows "Program called ok".



Part Inventory

Inventory List

PARTN	PARTD
00001	Message B
00024	Auto Ante
00002	Adapter -
00002	Adapter -
00004	Auto Ante
00021	Cell Phon
00021	Cell Phon
00001	Message B

Part Details

Part number 00002

Inventory detail

Description . . . . Adapter - 1

Quantity on hand 0000012

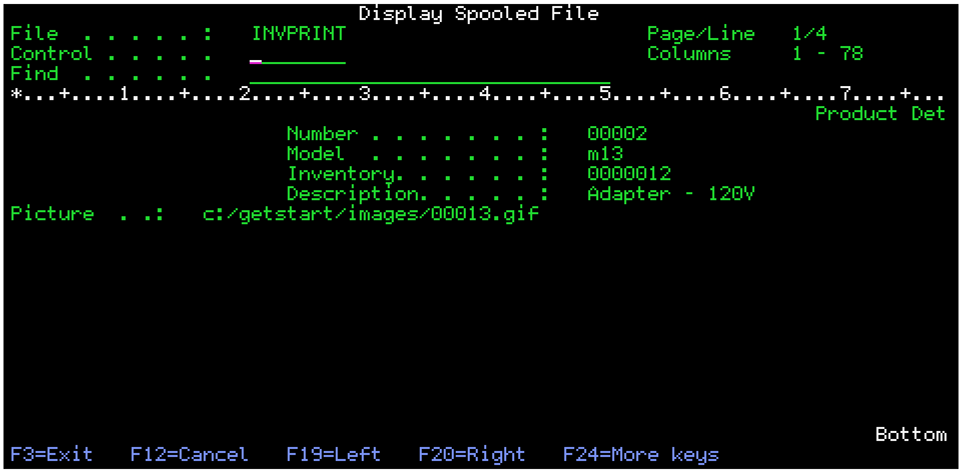
Model m13

Ship date 19970815-

Picture path . . . . c:/getstart/i

Program called ok.

Go to an AS/400 session and check the spool files QSYSPRT and INVPRINT. QSYSPRT should indicate the program was called successfully. INVPRINT should indicate the part number matches the part number selected in the Inventory List.



```
                                Display Spooled File
File  . . . . . :   INVPRINT                          Page/Line   1/4
Control . . . . .   _                                 Columns     1 - 78
Find  . . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+...
                                                                   Product Det
                       Number . . . . . . . :    00002
                       Model  . . . . . . . :    m13
                       Inventory. . . . . . :    0000012
                       Description. . . . . :    Adapter - 120V
Picture  . .:   c:/getstart/images/00013.gif








                                                                        Bottom
F3=Exit    F12=Cancel    F19=Left    F20=Right    F24=More keys
```

Close the window to end the test.