

# Online Retrieval Database

## Example 3

### Objective:

This example illustrates how to build and access an online Database on your web site. Specifically, it demonstrates retrieving inventory data (such as, Item number, Item Description , etc. for computer parts) upon the user's request. The user may either browse through the inventory items one-by-one or search for the desired computer parts.

### Note:

This example presumes Mojo has been installed in the default directory: "MojoBeta". All directory references in this example are relative to the default directory.

### Goal:

The following (figure 3A.1) is an image of the final interface for this example. The next couple of sections will describe how to design and layout the interface such that it looks like the figure below.

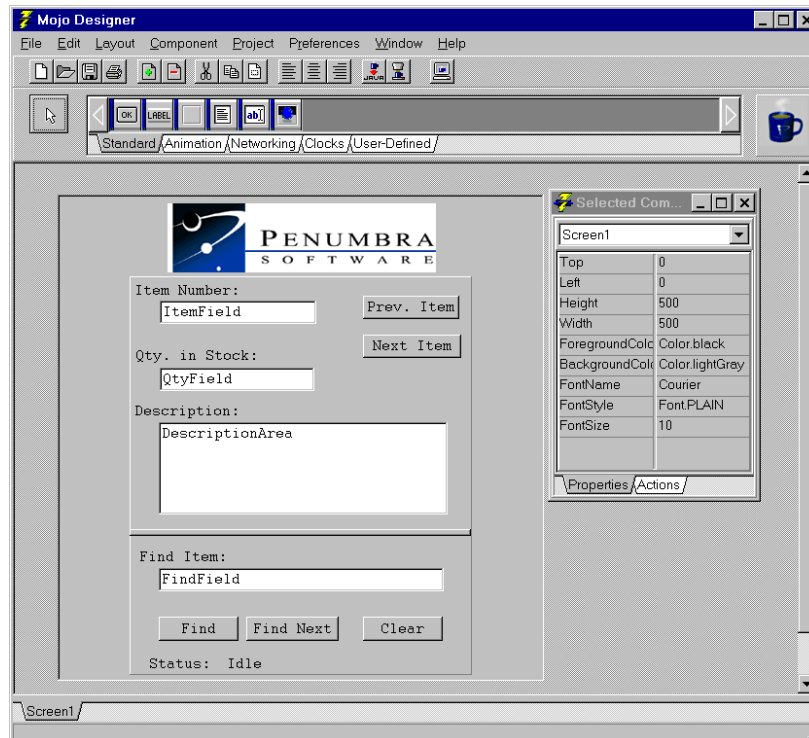


Figure 3A.0

# GETTING STARTED

## Part I. -- Designing the Interface

**Step 1:** Click on the *Picture* component in the *Standard* category and then move the mouse to the Drawing Area and click to place the picture.

See Figure 3A.1 below:

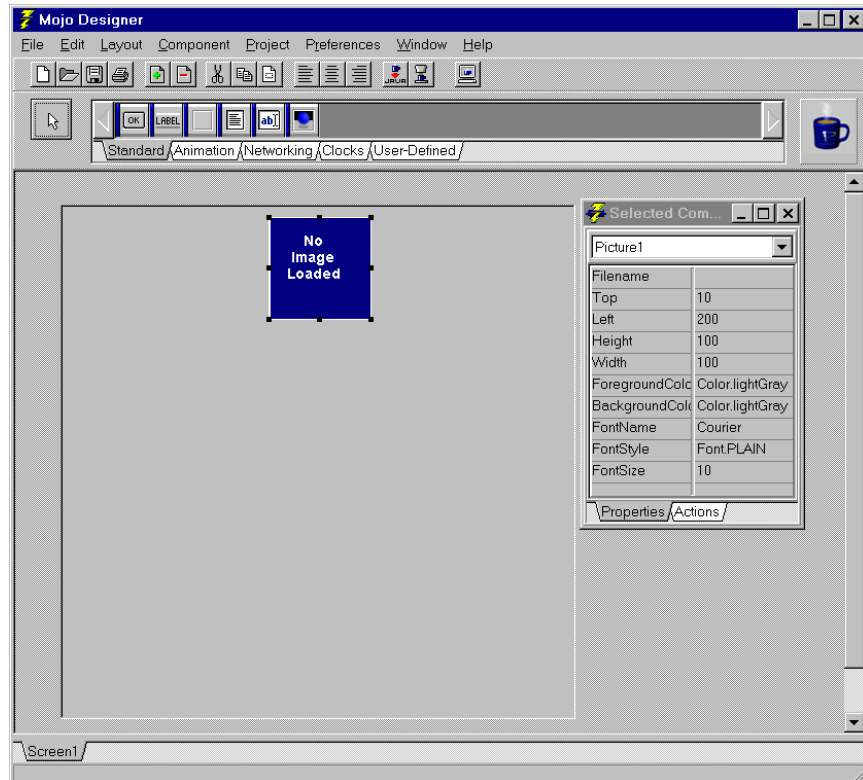
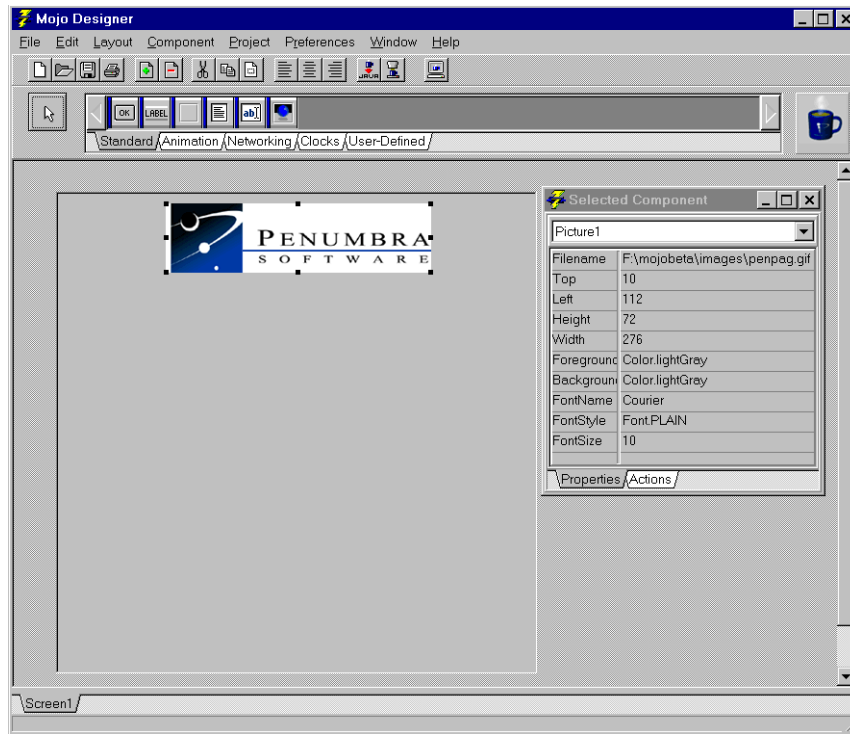


Figure 3A.1

**Step 2:** Click on the *Filename* property (right half) inside the *Selected Component* Property Window and then click on the [...] to browse for the image. Select the *PenPag.Gif* image from the *MojoBeta\images* directory and click *open*.

NOTE: The image may be centered using the *Center* speed button on the top button bar.

See Figure 3A.2 below:

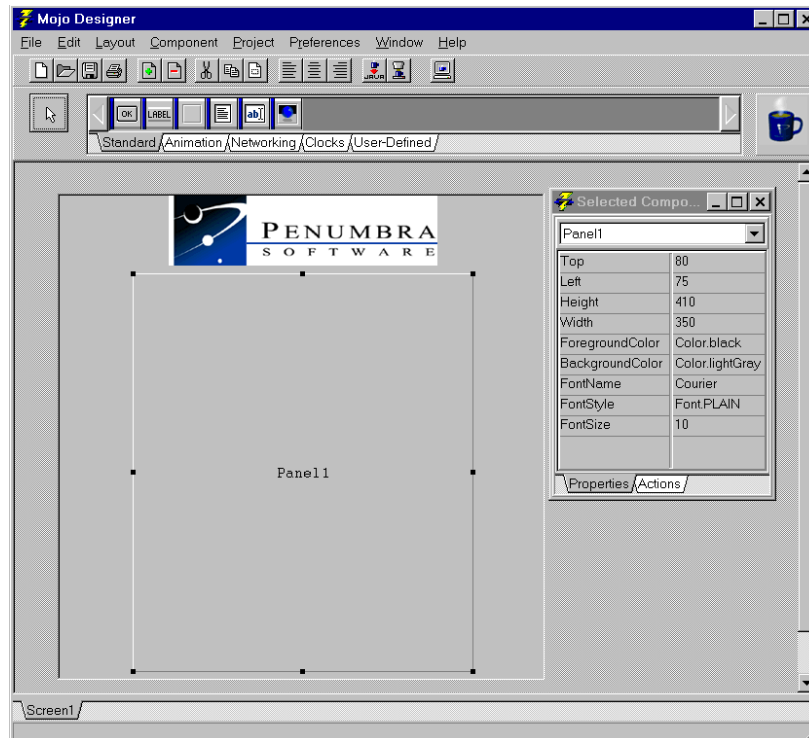


**Figure 3A.2**

**Step 3:** Click on the *Panel* component in the *Standard* category and then place and stretch the panel below the image inside the Drawing Area.

**NOTE:** The Panel1 component just placed should be stretched out close to the bottom edge of the Drawing Area in order for it hold the rest of the components. Any component may be resized any time by just clicking on it and stretching the appropriate edge.

*See Figure 3A.3 below:*



**Figure 3A.3**

**Step 4:** Click on the *Label* component in the *Standard* category and then place the label (named: Label1) inside of the *Panel1* near the upper left corner.

**Step 5:** Click on the *Text* property (right half) inside the *Selected Component* Property Window and then replace the existing text, “Label1”, with “Item Number:” and hit Enter/Return.

**NOTE:** Do not type in the quotes when typing in the text.

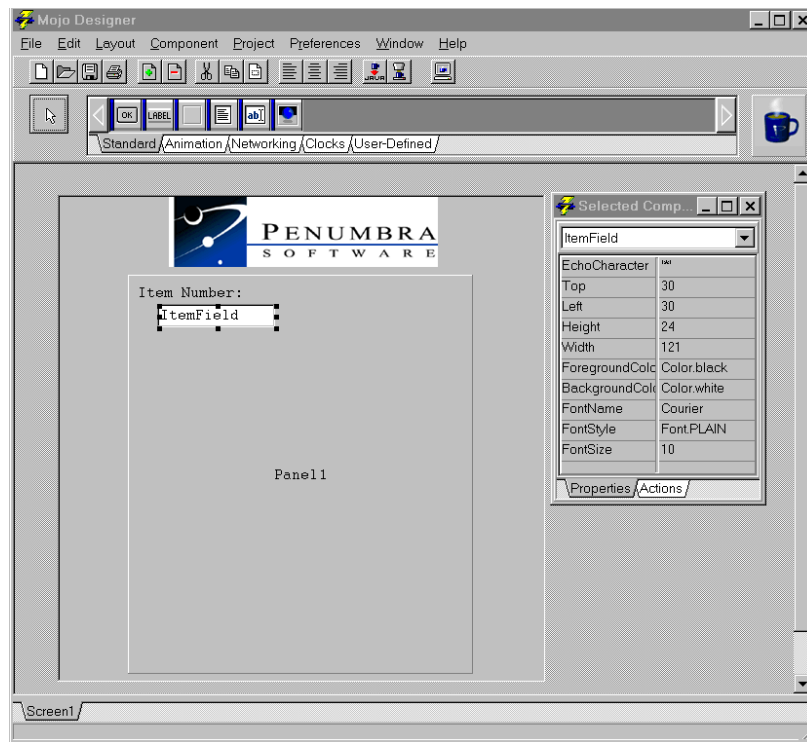
**Step 6:** Click on Label1 (“Item Number:”) and move the cursor over the right-center resize dot and stretch the label to the right until all the letters in “Item Number” appear on the line.

**Step 7:** Click on the *TextField* component in the *Standard* category and then place the text field (named: TextField1) inside of the Panel1 underneath the “Item Number” label. ( See Figure 3A.4)

**Step 8:** Click on the *TextField1* component and then RIGHT Click on it. Highlight *Rename* from the pop-up menu and click on it. Now, replace the existing name, “TextField1” with “ItemField”, and click *OK*.

**NOTE:** Do not type in the quotes when typing in the text.

See Figure 3A.4 below:



**Figure 3A.4**

**Step 9:** Click on the *Label* component in the *Standard* category and then place the label (named: Label2) inside of the *Panel1* underneath the *ItemField* text field. Change the value of the Text property of the label to “Qty. in Stock”. Resize the label appropriately. Refer to steps 4-6 for detailed instructions.

**Step 10:** Click on the *TextField* component in the *Standard* category and then place the text field (named: TextField1) inside of the *Panel1* underneath the “Qty. in Stock” label. Rename “TextField1” to “QtyField”. Refer to steps 7-8 for detailed instructions.

**NOTE:** Do not type in the quotes when typing in the text.

**Step 11:** Click on the *Label* component in the *Standard* category and then place the label (named: Label3) inside of the *Panel1* underneath the *QtyField* text field. Change the value of the Text property of the label to “Description:”. Resize the label appropriately. Refer to steps 4-6 for detailed instructions.

**Step 12:** Click on the *TextArea* component in the *Standard* category and then place the text area (named: TextArea1) inside of the *Panel1* underneath the “Description” label. Rename “TextArea1” to “DescriptionArea”. Refer to steps 7-8 for detailed instructions.

**NOTE:** Do not type in the quotes when typing in the text.

**Step 13:** Click on the *Label* component in the *Standard* category and then place the label (named: Label4) inside of the *Panel1* underneath the *DescriptionArea* Area field. Change the value of the Text property of the label to “Find Item:”. Resize the label appropriately. Refer to steps 4-6 for detailed instructions.

**Step 14:** Click on the *TextField* component in the *Standard* category and then place the text field (named: TextField1) inside of the *Panel1* underneath the “Find Item” label. Rename “TextField1” to “FindField”. Refer to steps 7-8 for detailed instructions.

**NOTE:** Do not type in the quotes when typing in the text.

**Step 15:** Click on the *Button* component in the *Standard* category and then place the button (named: Button1) inside of the *Panel1* underneath the *FindField* text field. Rename “Button1” to “FindButton”. Refer to steps 7-8 for detailed instructions.

**Step 16:** Change the value of the Label property of *FindButton* to “Find”. Resize the button appropriately.

**Step 17:** Click on the *Button* component in the *Standard* category and then place the button (named: Button1) inside of the *Panel1* next to the *FindButton* button. Rename “Button1” to “FindNextButton”. Refer to steps 7-8 for detailed instructions.

**Step 18:** Change the value of the Label property of *FindNextButton* to “Find Next”. Resize the button appropriately.

**Step 19:** Click on the *Button* component in the *Standard* category and then place the button (named: Button1) inside of the *Panel1* next to the *FindButton* button. Rename “Button1” to “ClearButton”. Refer to steps 7-8 for detailed instructions.

**Step 20:** Change the value of the Label property of *ClearButton* to “Clear”. Resize the button appropriately.

**Step 21:** Click on the *Label* component in the *Standard* category and then place the label (named: Label5) inside of the *Panel1* underneath the *FindButton* button. Change the value of the Text property of the label to “Status:”. Resize the label appropriately. Refer to steps 4-6 for detailed instructions.

**Step 22:** Click on the *Label* component in the *Standard* category and then place the label (named: Label6) inside of the *Panel1* to the right of *Label5* (“Status:” label). Change the value of the Text property of the label to “Idle”.

**Step 23:** Rename “Label6” to “StatusLabel”. Refer to steps 7-8 for detailed instructions.

**NOTE:** This is the only label this is renamed in this example. This is done for easier reference purposes later in the example. The other labels do not need to be renamed.

**Step 24:** Resize *StatusLabel* by stretching the right resize dot to the right edge of *Panel1*. This Label will dynamically change as the program runs; thus, extending the label will allow longer text messages to be displayed.

See Figure 3A.5 below:

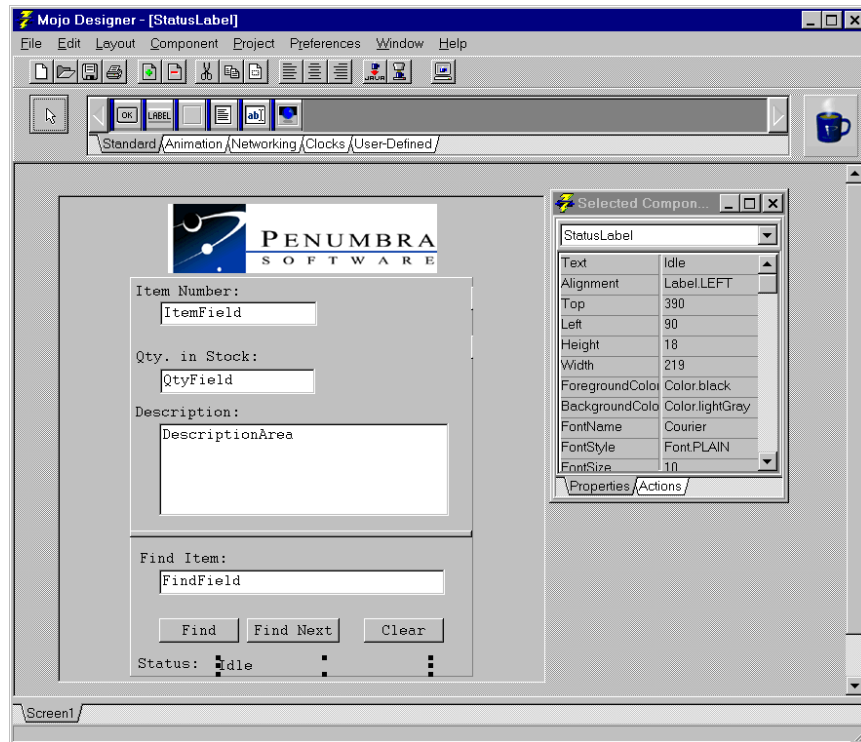


Figure 3A.5

**Step 25:** Click on the *Button* component in the *Standard* category and then place the button (named: *Button1*) inside of the *Panel1* next to the *ItemField* text field (upper right corner). Rename “*Button1*” to “*PrevItemButton*”. Refer to steps 7-8 for detailed instructions.

**Step 26:** Change the value of the *Label* property of *PrevItemButton* to “*Prev. Item*”. Resize the button appropriately.

**Step 27:** Click on the *Button* component in the *Standard* category and then place the button (named: *Button1*) inside of the *Panel1* underneath *PrevItemButton* button. Rename “*Button1*” to “*NextItemButton*”. Refer to steps 7-8 for detailed instructions.

**Step 28:** Change the value of the *Label* property of *NextItemButton* to “*Next Item*”. Resize the button appropriately.

You have now complete the Design part of the interface. The interface should look similar to Figure 3A.6 below:

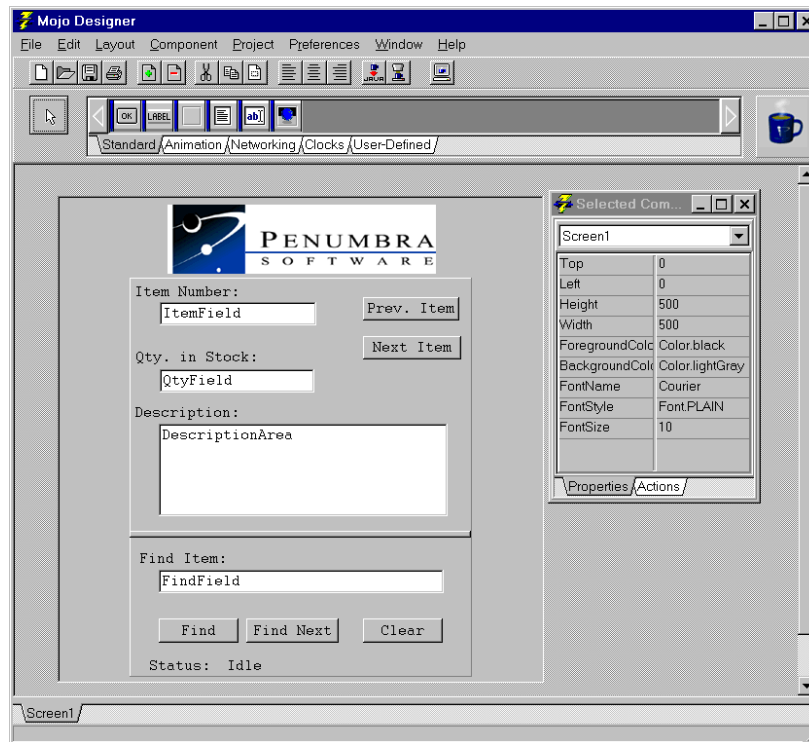


Figure 3A.6

**END OF PART I.**

## **Part II. -- Adding Functionality**

**Step 1:** Switch over to the Mojo Coder using the “Switch to Coder” speed button located on the Speed Button Bar. This can also be done by choosing “Coder” from the “Window” menu off of the Main Menu Bar.

*The Mojo Coder is represented in Figure 3B.1*



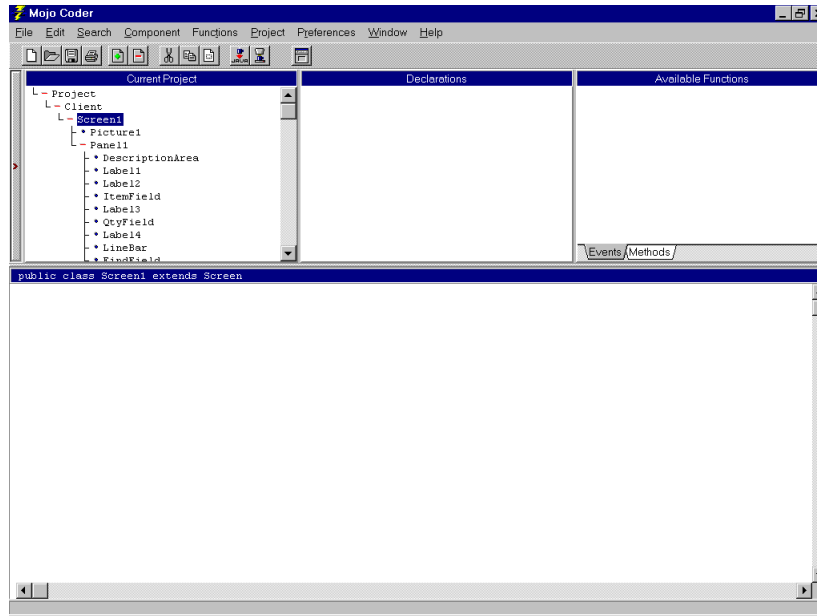


Figure 3B.1

**Step 2:** Click on the *Project* component under the **Current Project** pane. Type the following code into the Code Editor.

```
class InventoryItem extends Object {

    public String item;           // Holds Item number
    public String qty;            // Holds quantity available in stock
    public String description;    // Holds Item Description

    InventoryItem()
    {
        item = "";
        qty = "";
        description = "";
    }
}
```

**NOTE:** The code will automatically update whenever you click anywhere outside of the Code Editor. The code may also be manually updated by either choosing “Update Code” from the “Edit” menu on the main menu bar or by using the keyboard shortcut, CTRL-U.

**Explanation (Step 2):** Here, the class *InventoryItem* is created. The *InventoryItem* object contains three fields of information (Item Number (*item*), Qty. in Stock (*qty*), and Description). The class constructor, *InventoryItem()*, is also defined, used for initialization.

**NOTE:** All user defined Class definition, Class constructors and global variables must be defined here (under the *Project* component).

**Step 3:** Click on the *Panel1* component under the **Current Project** pane. Type the following code into the Code Editor.

```
int index = 0; // used to keep track of which item is
               // currently being viewed
```

**Explanation (Step 3):** “index” is an instance variable used to keep track the currently viewed item. It is initially set to 0.

**Step 4:** Make sure the *Panel1* component is still selected (highlighted). Then, click on the “Component” menu on the main menu bar and choose “Attributes”. Next, click on the “Imports” tab near the top of the “Components Attributes” window.

**Step 5:** Click on the **Add** button and enter in “java.io.\* ” in the “Add Package to Import List” window. Click on OK.

**NOTE:** Do not type in the quotes when typing in the text.

See Figure 3B.2 below:

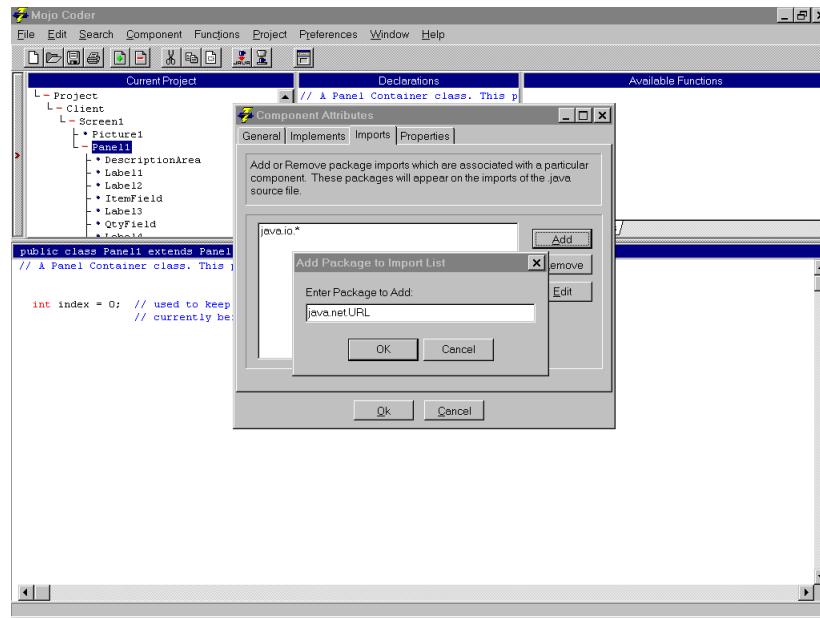


Figure 3B 2

**Step 6:** Repeat **Step 5** and add the following packages to the Import list, individually.

```
java.net.URL
java.net.MalformedURLException
java.util Enumeration
```

**Explanation (Steps 4-6):** The purpose of these steps is to add the needed Java packages to the Import list. They are needed because the Java compiler must know what packages are used in order for it to compile properly.

**Step 7:** Click on *Panel1* again, and then click on the **Hierarchy Toggle Button** located to the left of the **Current Project** pane; which expands the **Class Hierarchy** panes.

**Step 8:** Double-Click on the “java” package under the **Class Hierarchy** pane, and then double-click on the “util” package.

**Step 9:** Scroll through the **Class Hierarchy** pane until the *Vector* component is visible.

**Step 10:** Click on the *Vector* component and drag-and-drop the component over to the **Current Project** pane. (other words: click and hold on the *Vector* component and move the mouse over to the Current Project pane and let go of the mouse button.)

**Step 11:** Type in “InventoryVector” into the **Add Component** dialog box that pops up. This will add a non-visual vector component into the Current Project hierarchy named *InventoryVector*.

See Figure 3B.3 below:

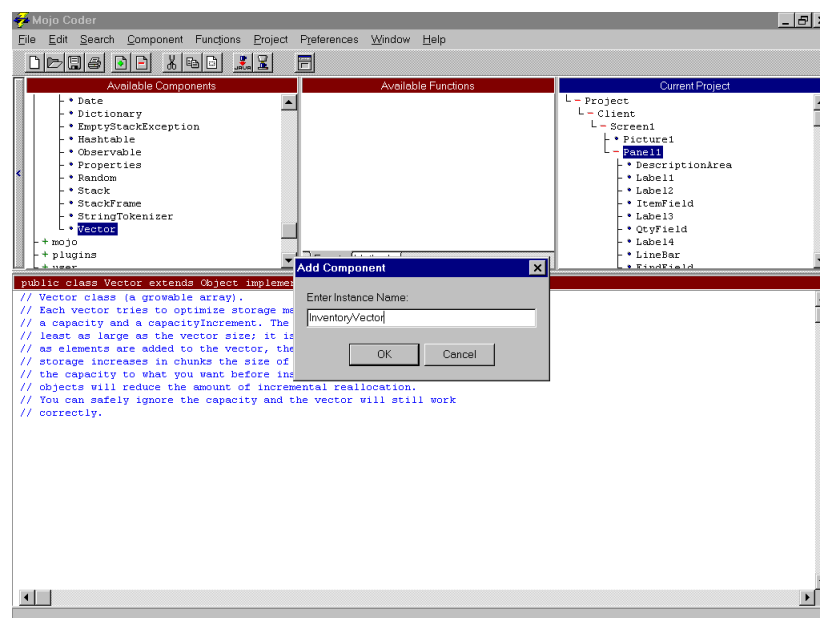


Figure 3B.3

**Step 12:** Click on the **Hierarchy Toggle Button** located to the left of the **Current Project** pane; which hides the **Class Hierarchy** panels and displays all of the current project panels.

**Step 13:** Scroll through the **Current Project** pane until the *InventoryVector* component is visible. Click on the *InventoryVector* component.

**Step 14:** Click on the **Methods** Tab under the **Available Functions** pane in order to highlight it.

**Step 15:** Move the mouse inside the **Available Functions** pane and RIGHT click. Highlight and click on “Add Function” option from the pop-up menu.

**Step 16:** Type in “initialize” for the Method Name to be Added, and choose “none” for the Method Type (radio button).

**Step 17:** Type in the following code inside the **Code Editor**.

**NOTE: Make sure *InventoryVector* is selected in the Current Project and *initialize* is selected in the Available functions.**

```
void initialize ()
{
    // local variables

    URL hostURL = null;
    String aLine = "";
    String filename = "inventoryfile.txt";
    InventoryItem anInventoryItem = null;

    // method body
    try
    { // get inventory file from host URL
        hostURL = new URL(applet.getCodeBase(), filename);
    }
    catch (MalformedURLException e)
    {
        System.out.println("MalforedURLExeption using File: " + filename);
    }
    try
    {
        InputStream hostInputStream = hostURL.openStream();
        // Creating DataInputStream wrapped around BufferedInputStream for better performance.
        DataInputStream dataStream = new DataInputStream(new
            BufferedInputStream(hostInputStream));

        while ((aLine = dataStream.readLine()) != null)
        {
            anInventoryItem = new InventoryItem();
            anInventoryItem.item = aLine;
            anInventoryItem.qty = dataStream.readLine();
            anInventoryItem.description = "";
            while (!(aLine = dataStream.readLine()).equals("END_ITEM"))
            {
                anInventoryItem.description = anInventoryItem.description + aLine + "\n";
            }
            this.addElement(anInventoryItem);
        }
        if (this.isEmpty() == false)
        {
            anInventoryItem = ((InventoryItem) this.elementAt(0));
            parent.ItemField.setText(anInventoryItem.item);
            parent.QtyField.setText(anInventoryItem.qty);
            parent.DescriptionArea.setText(anInventoryItem.description);
        }
    }
    catch (IOException e) {
        System.out.println("IOExeption while reading: " + hostURL.toString());
    }
}
```

**Explanation (Step 17):** The purpose of this code is to retrieve the inventory file (inventoryfile.txt) from the host URL. The entire file is then loaded into a data input stream. The each item is loaded into an InventoryItem object and stored inside of the *InventoryVector*.

**Step 18:** Scroll through the **Current Project** pane until the *NextItemButton* component is visible. Click on the *NextItemButton* component.

**Step 19:** Click on the **Events** Tab under the **Available Functions** pane in order to highlight it.

**Step 20:** Move the mouse inside the **Available Functions** pane and RIGHT click. Highlight and click on “Override Function” option from the pop-up menu, and choose by clicking the “action” event (1<sup>st</sup> event in the list).

See Figure 3B.4 below:

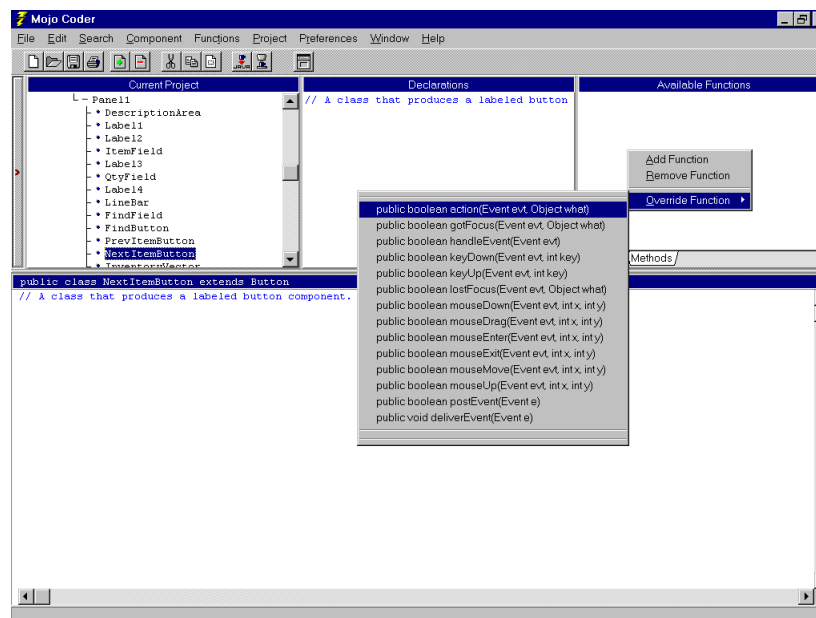


Figure 3B.4

**Step 21:** Type the following code into the Code Editor:

```
public boolean action(Event evt, Object what)
{
    // local variables

    InventoryItem anInventoryItem = null;

    // event body

    if (parent.index < (parent.InventoryVector.size() - 1))
    {
        parent.index++;
    }
}
```

```

        anInventoryItem = ((InventoryItem) parent.InventoryVector.elementAt(parent.index));
        parent.ItemField.setText(anInventoryItem.item);
        parent.QtyField.setText(anInventoryItem.qty);
        parent.DescriptionArea.setText(anInventoryItem.description);
    }
    return(true);
}

```

**Explanation (Step 21):** The purpose of this code is to advance to the next inventory item in the *InventoryVector* and display the new item information in their respective fields. This event occurs whenever the *NextItemButton* is clicked.

**Step 22:** Click on the *NextItemButton* component in the **Current Project** pane, and click on the **Events** Tab under the **Available Functions** pane.

**Step 23:** Move the mouse inside the **Available Functions** pane and RIGHT click. Highlight and click on “Override Function” option from the pop-up menu, and choose by clicking the “action” event (1<sup>st</sup> event in the list).

**Step 24:** Type the following code into the Code Editor:

```

public boolean action(Event evt, Object what)
{
    // local variables

    InventoryItem anInventoryItem = null;

    // event body

    if (parent.index > 0)
    {
        parent.index--;
        anInventoryItem = ((InventoryItem) parent.InventoryVector.elementAt(parent.index));
        parent.ItemField.setText(anInventoryItem.item);
        parent.QtyField.setText(anInventoryItem.qty);
        parent.DescriptionArea.setText(anInventoryItem.description);
    }

    return(true);
}

```

**Explanation (Step 24):** The purpose of this code is to retreat to the previous inventory item in the *InventoryVector* and display the new item information in their respective fields. This event occurs whenever the *PrevItemButton* is clicked.

**Step 25:** Click on the *FindButton* component in the **Current Project** pane.

**Step 26:** Type the following code into the Code Editor:

```

String currentSearchString = ""; // Holds the previously searched String;

int currentIndex = 0;           // Holds current index position to start Searching from.

```

**Step 27:** Click on the **Events** Tab under the **Available Functions** pane. Move the mouse inside the **Available Functions** pane and RIGHT click. Highlight and click on “Override Function” option from the pop-up menu, and choose by clicking the “action” event.

**Step 28:** Type the following code into the Code Editor:

**NOTE:** Make sure *FindButton* is selected in the Current Project and *action* is selected in the Available functions.

```
public boolean action(Event evt, Object what)
{
    // local variables

    String searchItem = "";
    Enumeration elementsList = null;
    InventoryItem anInventoryItem = null;
    boolean found = false;

    // event body

    searchItem = parent.FindField.getText();
    this.currentSearchString = searchItem;
    this.currentIndex = 0;

    if (!(searchItem.equals("")) )
    {
        parent.StatusLabel.setFont(new Font(parent.StatusLabel.getFont().getName(),
                                           Font.BOLD + Font.ITALIC,
                                           parent.StatusLabel.getFont().getSize()));
        parent.StatusLabel.setText("Searching...");
        try { Thread.sleep(50); }
        catch (InterruptedException e) { }

        elementsList = parent.InventoryVector.elements();

        while ((elementsList.hasMoreElements() == true) && !(found))
        {
            anInventoryItem = (InventoryItem) elementsList.nextElement();
            this.currentIndex++;
            if (anInventoryItem.item.toLowerCase().startsWith(searchItem.toLowerCase()))
            {
                found = true;
                parent.index = this.currentIndex - 1;
                parent.ItemField.setText(anInventoryItem.item);
                parent.QtyField.setText(anInventoryItem.qty);
                parent.DescriptionArea.setText(anInventoryItem.description);
            }
        }
    }
    if (found == false)
    {
        parent.StatusLabel.setText("Not Found");
        this.currentSearchString = "";
        this.currentIndex = 0;
        try { Thread.sleep(500); }
        catch (InterruptedException e) { }
    }
    parent.StatusLabel.setFont(new Font(parent.StatusLabel.getFont().getName(),
                                       Font.PLAIN,
                                       parent.StatusLabel.getFont().getSize()));
    parent.StatusLabel.setText("Idle");

    return(true);
}
```

**Explanation (Step 28):** The purpose of this code is to take in a string from *FindField* and search the database (*InventoryVector*), item by item, for a match or a partial match. If a match is found the appropriate item information is displayed on the screen;

otherwise, an error message, “Not Found”, is given on the StatusLabel. This event occurs whenever the *FindButton* is clicked.

**Step 29:** Click on the *FindNextButton* component in the **Current Project** pane. Click on the **Events** Tab under the **Available Functions** pane. Move the mouse inside the **Available Functions** pane and RIGHT click. Highlight and click on “Override Function” option from the pop-up menu, and choose by clicking the “action” event.

**Step 30:** Type the following code into the Code Editor:

```
public boolean action(Event evt, Object what)
{
    // local variables

    String searchItem = "";
    InventoryItem anInventoryItem = null;
    boolean found = false;
    int index = 0;

    // event body

    searchItem = parent.FindField.getText(); // Holds current search string to search for
    index = parent.FindButton.currentIndex; // Holds where to start searching in Database.

    if (! (searchItem.toLowerCase().equals(parent.FindButton.currentSearchString.toLowerCase())))
    { // the Find Text Field has changed since last Find/FindNext; thus calling Find
        parent.FindButton.action(evt, what);
    }
    else
    {
        if (!(searchItem.equals("")))
        {
            parent.StatusLabel.setFont(new Font(parent.StatusLabel.getFont().getName(),
                                                Font.BOLD + Font.ITALIC,
                                                parent.StatusLabel.getFont().getSize()));
            parent.StatusLabel.setText("Searching...");
            try { Thread.sleep(50); }
            catch (InterruptedException e) { }
            while ((index < parent.InventoryVector.size()) && !(found))
            {
                anInventoryItem = (InventoryItem) parent.InventoryVector.elementAt(index);
                index++;
                if (anInventoryItem.item.toLowerCase().startsWith(searchItem.toLowerCase()))
                {
                    found = true;
                    parent.index = index - 1;
                    parent.FindButton.currentIndex = index;
                    parent.ItemField.setText(anInventoryItem.item);
                    parent.QtyField.setText(anInventoryItem.qty);
                    parent.DescriptionArea.setText(anInventoryItem.description);
                }
            }
        }
        if (index >= parent.InventoryVector.size())
        {
            parent.StatusLabel.setText("No More Records");
        }
        else
        { if (found == false)
            {
                parent.StatusLabel.setText("Not Found");
            }
        }
        try { Thread.sleep(500); }
        catch (InterruptedException e) { }
```



```

        parent.StatusLabel.setFont(new Font(parent.StatusLabel.getFont().getName(),
                                            Font.PLAIN,
                                            parent.StatusLabel.getFont().getSize()));
        parent.StatusLabel.setText("Idle");
    } // end OUTER else

    return(true);
}

```

**Explanation (Step 30):** The purpose of this code is to take in a string from *FindField* and search the database (InventoryVector) for the NEXT possible match. If the *FindField* has been changed since the last Find for FindNext, then this function calls the “action” event of *FindButton* and performs a find. If the *FindField* has Not changed, then it continues to search for the next possible match from where it last left off. If a match is found the appropriate item information is displayed on the screen; otherwise, an error message, “Not Found”, or “No more Records” is given on the StatusLabel. This event occurs whenever the *FindNextButton* is clicked.

**Step 31:** Click on the *ClearButton* component in the **Current Project** pane. Click on the **Events** Tab under the **Available Functions** pane. Move the mouse inside the **Available Functions** pane and RIGHT click. Highlight and click on “Override Function” option from the pop-up menu, and choose by clicking the “action” event.

**Step 32:** Type the following code into the Code Editor:

```

public boolean action(Event evt, Object what)
{
    // local variables

    // event body

    parent.FindField.setText("");

    return(true);
}

```

**Explanation (Step 32):** The purpose of this code is to simply clear the *FindField*.

***END OF PART II.***

### **Part III. -- Building the Inventory Text File (inventoryfile.txt)**

This example requires a database file; inventoryfile.txt. This file is a simple text file that can be created using any text editor; i.e. DOS edit.com. The text file must be written in the following format:

```

[ItemNumber]
[Qty. in Stock]
[Description]
[Description]
[...]
END_ITEM
[ItemNumber]

```

[Qty. in Stock]  
[Description]  
[Description]  
[...]  
END\_ITEM  
[...]

**NOTE: ItemNumber must come first (1 line only), followed by Qty. in Stock (1 line only), followed by at least one line of Description (1+ lines), and the item block MUST end with END\_ITEM. There must NOT be any blank lines the inventory file, anywhere.**

**NOTE: This file must be placed in the mojobeta\myproject\output directory.**

*The following is an example inventoryfile.txt:*

```
CASE-BABY
2000
Desktop Computer Case
250 watt Switchable Power Supply
three 5 1/4" Drive Bays
two 3 1/2" Drive
END_ITEM
MB-586-PT5
330
Intel 586 Pentium Main Board
256k Pipeline Cache
4 72-Pin SIMM Slots
3 32-bit PCI Expansion Slots
5 16-bit ISA Expansion Slots
On-board IDE Controller (2s/1p)
END_ITEM
CPU-P100
43
Pentium 100 MHz CPU
END_ITEM
```

**NOTE: This example inventory file contains three (3) items.**

**NOTE: This file must be placed in the mojobeta\myproject\output directory.**

***END OF PART III.***

## **Part IV. -- Generating and Running the Applet**

This is the final part of this example. The Example is now ready to run. From the Coder, Click on "Run Applet" from the "Project" menu on the main menu bar. If there are any compile errors due to typing mistakes, please correct the typo(s) and re-run the applet. When Mojo is done generating the code, AppletViewer will open and display your applet.

*END OF PART IV.*

**DONE!!! Congratulation!!**