Lab 10: Creating and Using ActiveX Documents

For background information on this lab, click each of these topics:

Objectives

By the end of this lab, you will be able to:

- Use the ActiveX Document Wizard to convert forms into an ActiveX document server.
- Add properties to an ActiveX document.
- Enhance the user interface of an ActiveX document.
- Create a document container.

Prerequisites

Before working on this lab, you should be familiar with the following concepts:

- ActiveX document project fundamentals

Lab Setup

To complete this lab, you need the following:

- Visual Basic 5.0 or later
- Internet Explorer 3.0 or later

The exercises in this lab use Internet Explorer, but they will also work with any valid document container.

To see a demonstration of the completed lab solution, click this icon.

Estimated time to complete this lab: **45 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 10.

Exercise 1: Converting Forms to ActiveX Documents

In this exercise, you will use the ActiveX Document Wizard to convert an existing forms-based project to an ActiveX document server.

Exercise 2: Adding Properties to ActiveX Documents

In this exercise, you will enhance ActiveX documents.

Exercise 3: Extending User Interface Functionality

In this exercise, you will enhance the user interface of the container based on the requirements of the document object. You will also modify one of the documents so that it is always centered within the container.

Exercise 4: (Optional) Creating a Document Container

In this exercise, you will modify the project to allow the executable file to run as a stand-alone application and to display the document objects within their own context.

Exercise 1: Converting Forms to ActiveX Documents

In this exercise, you will use the ActiveX Document Wizard to convert an existing forms-based project to an ActiveX document server.

You will convert the Main form and Order Details form in an existing database application to ActiveX documents. You will then test the new ActiveX document files by associating them with Internet Explorer, and invoking them.

▶ Run the ActiveX Document Wizard

1. In the Visual Basic environment, open the project in Lab10.
2. Use the ActiveX Document Wizard to create an ActiveX EXE project and convert the form frmHome to an ActiveX document.

   In the Project Explorer, the project should now consist of two forms (frmOrders and frmOrderDetails) and one user document. Modify the name of the user document to docHome.

▶ Modify the user document

1. Remove the **Exit Application** button from the docHome document.

   The **Exit Application** button does not provide any appropriate behavior in this context. If you view the underlying code for the Click event for the button's click event, you will see that the ActiveX Document Wizard has commented out the invalid code.
2. Save the project.

▶ Test the ActiveX Document in the container

1. Associate Visual Basic ActiveX (.vbd) files with Internet Explorer.
   a. Open either **My Computer** or the Windows Explorer.
   b. On the **View** menu, click **Options**, and then select the **File Types** property page.
   c. Add a new file type that associates .vbd files with Internet Explorer (IExplore.exe).
2. Switch to Visual Basic and run the project.

   There are not any visible forms because the output from running this project is the .vbd file.

   Note While running the project in Visual Basic, the .vbd files are created in the Visual Basic folder, and when running exe or .dll files, the .vbd files are created in the .Exe or .Dll folders.
3. In the **My Computer** or Windows Explorer window, switch to the Visual Basic folder and run the ActiveX file docHome.vbd.

   This should cause the file to be loaded into your document container.
4. Once the .vbd file has been loaded properly, exit Internet Explorer, return to Visual Basic, and then explicitly end the program.

▶ Convert the Order Details form to an ActiveX document

1. Add a new module to the project.
2. Declare the public variable sDocPath to hold the document path.
3. In the docHome user document, add code to the Initialize event, and set the sDocPath variable to point to the Visual Basic folder (for example, C:\Program Files\Vb\).
4. Using the ActiveX Document Wizard, follow the same steps as before to convert the frmOrders form into an ActiveX document, and rename the document to docOrders.

▶ Add code for controls to navigate between the two documents

In the new document, the **Close** button is not appropriate in the current context. In this next procedure, you will change this button so that users can return to the docHome user document only if they have first invoked the docOrders document from the docHome document.

1. Change the **Caption** property of the **Close** button (cmdClose) to **Home**.
2. Add code to the user document for the InitProperties event to enable the **Home** button only if the path variable sDocPath has been initialized.
3. In the Click event for the **Home** button, use the **NavigateTo** method of the document's **Hyperlink** object to return to the docHome document.
4. Switch to the docHome document, and then update the Click event for the **Orders** button to navigate to the docOrders document.

▶ **Test the functionality of the application**

1. Click the **Orders** button on the main form to invoke the docOrders document.
2. Make sure that the database functionality still works correctly, including moving through, adding, modifying, and deleting records.
3. Invoke the Details form for any order.
4. Use the **Home** button to return to the docHome document.

Exercise 2: Adding Properties to ActiveX Documents

In this exercise, you will enhance the ActiveX documents that you created in the previous exercise.

You will add a custom property to the Orders document to store the current Order ID and make the current Order ID externally available. You will then add the necessary code to the **UserDocument** object to make that property value persistent. When users open the user document again, they should see the same record that was displayed when the document was closed.

▶ **Create the custom property OrderID for the Orders document**

1. Add the procedure **FindOrderByOrderID** to the Orders document.

   This procedure should take OrderID as a parameter and move to the associated record, if that record exists.
2. Declare the function as **Private**. The function should also take a variant as a parameter and return a **Boolean**.
3. If the Order ID is a negative number, move to the first record in the recordset.
4. Use the **FindFirst** method of the DAO **Recordset** object to find the associated record. If no match exists, use a bookmark to return to the current record, and return a value of **False**.

The Data control has the **Recordset** property.

▶ **Add the OrderID property to the user document**

1. Add the **OrderID** property to the docOrders document, and include code in the **PropertyGet** procedure that returns the current Order ID.
2. Add code to the **PropertyLet** procedure to move to the associated order, if it exists, and then update the document with the new order information.
3. On the **Tools** menu, click **Add Procedure**, and add the public property OrderID.
4. In the **PropertyGet** procedure, simply return the current value as displayed in the Order ID text box.
5. In the **PropertyLet** procedure, call FindOrderByOrderID, and pass the new property value.

▶ **Store properties for the ActiveX document**

1. Add code to store the properties for the Orders document, so when the Orders document is reinitialized, it displays the appropriate record(the record displayed just before exiting and saving the document).

2. In the WriteProperties event for the docOrders document, invoke the **PropertyBag** object's **WriteProperty** method to store the current Order ID value into the .vbd file. Use a default value of –1, as shown in the following code:

```
PropBag.WriteProperty "OrderID", txtOrderID.Text, -1
```

3. In the ReadProperties event for the docOrders document, invoke the **PropertyBag** object's **ReadProperty** method to read the stored OrderID property value. Use –1 as a default value, as shown in the following code:

```
Me.OrderID = PropBag.ReadProperty("OrderID", -1)
```

4. In both the ReadProperties and WriteProperties events, use the **Debug** object to display an appropriate notification in Visual Basic in the **Immediate** window.
5. In the Changed event of the **Order ID** text box, use the **PropertyChanged** method of the user document to prompt the user to save changes.

▶ **Test the document**

1. After running the project in Visual Basic, use Windows Explorer to note the file size of the docOrders.vbd file in the Visual Basic folder.
2. Invoke the Home document, and then switch to the Orders document.
3. Move to the last record in the recordset and note the Order ID.
4. Close Internet Explorer. When prompted with the message "Do you want to save the changes?", click **Yes**.

   Note the file size of docOrders.vbd. (You may need to refresh Windows Explorer to see the updated file size).
5. Launch the Orders document in the browser, and verify that it starts up with the appropriate Order ID.

Exercise 3: Extending User Interface Functionality

In this exercise, you will extend the user interface of the container based on the requirements of the **Document** object. You will also change the position of the document's controls from the upper-left corner of Internet Explorer to the center of a browser, regardless of the size of the window.

You will add a menu associated with the document to the container's menubar when the **Document** object is active. You will also specify the container size of the document so that when either the container's width or height is less than the document size, the appropriate scrollbars will be displayed.

▶ **Center the document in the browser**

In this exercise, you will add functionality to maintain the document's central position in the browser, regardless of the current size of the browser.

1. Select all of the controls on the docHome document, and click **Cut** on the **Edit** menu to move them to the Clipboard.

The best way to select all of the controls is to use the **Pointer** button and create a selection rectangle with the mouse.

2. Maximize the document window, and use a **Frame** control to fill up the window.
3. Name the control fraMain, and set the caption of the frame to NULL.
4. While the frame is selected, paste the controls back onto the document, thus making the frame the parent of the pasted controls. Resize the frame and center the controls.

The frame window should be the parent of the copied controls. If the frame is not selected when you paste the controls, the form will be the parent, rather than the frame.

5. In the Resize event for the **UserDocument** object, add code to center the frame (fraMain) by setting the **Left** and **Top** properties.

The position of a control on a user document is based on the **ScaleMode** property of the user document, just as a control is placed on a form.

The following settings center the **Panel** control on the user document:

```
pnlMain.Left = (ScaleWidth - pnlMain.Width) /2
pnlMain.Top = (ScaleHeight - pnlMain.Height) /2
```

6. Save and test the document for the desired behavior.

▶ Add a Record Navigation menu to the Orders document

1. Using the Menu Editor, add a **Record** menu to the Orders document. Add the commands to the menu for **MoveFirst**, **MovePrevious**, **MoveNext**, and **MoveLast**.
2. Specify the position of the menu to appear in the middle of the menu for the container.
3. Add code to each of the menu items for the appropriate functionality.

▶ Set the document's minimum width and height

1. Use the size of the **Frame** control on the Main document, set the document's **minWidth** and **minHeight** properties.

▶ Test the project

1. Open the Main document and resize the container until the width and/or height are less than the frame control on the document, fraMain.

   The appropriate scrollbars should be displayed in the window of the document.
2. Switch to the Orders document, and verify that the **Record** menu is displayed and functional on the container.

Exercise 4: (Optional) Creating a Document Container

In this exercise, you will modify a project so the executable file can run as a stand-alone application, and display document objects within their own context.

The **WebBrowser** control included with Visual Basic lets you add browsing capabilities to your application. This includes the ability to act as a **Document** object container.

▶ Add capabilities to a Document object container

1. Create a form as the parent of the **WebBrowser** control. Display this form only if the application is being started in stand-alone mode.
2. Add a form named frmContainer to the project.
3. Add the **WebBrowser** control to the form.

   If the **WebBrowser** control is not available in the Toolbox, click **Components** on the **Project** menu, and then select **Microsoft Internet Controls** on the **Controls** tab.
4. In the Form_Load event, use the **WebBrowser** control to navigate to the docHome document object in the Visual Basic folder, as shown in the following code:

   ```
   WebBrowser1.Navigate "c:\program files\devstudio\vb\docHome.vbd"
   ```

5. In the form's Resize event, move the **WebBrowser** control to fill the client area of the form.

▶ Determine the Startup mode of the application

Add a **Sub Main** procedure to the Standard module of a project to test whether or not the application is being started in stand-alone mode, and take an appropriate action.

1. In stand-alone mode, display the application in the new container form.
2. Add a **Sub Main** procedure to the Standard module of the project.
3. In the procedure, use the App's **StartMode** property to check whether the application is starting in standalone mode, in which case you should display the container form.

▶ **Test the project in stand-alone mode**

1. Open the Project Settings property page, and then modify the project so it will run in stand-alone mode.
2. On the **General** tab, set the Start Object to **Sub Main**.
3. On the **Components** tab, set the Start Mode to **Standalone**.
4. Run and test the application.