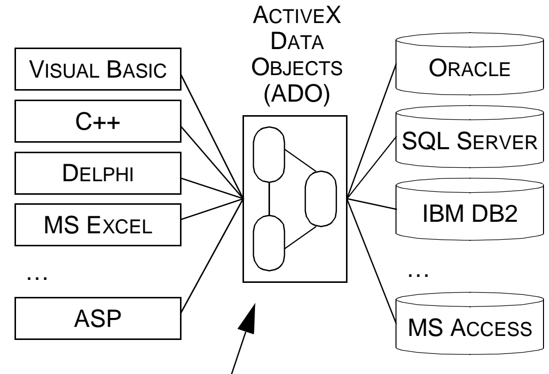
Lesson 28: Server-side data access using ADO

28.1 Introduction: What is ADO?

ACTIVEX DATA OBJECTS (ADO) is a MICROSOFT technology that enables programmers to access data stored in virtually any type of database. The nice thing about ADO is that it provides a *lingua franca* between many different development tools on one side and many different databases on the other. The relationship between development tools, ADO, and databases is shown in Figure 28.1.

ADO is database middleware and subsumes ODBC, which you saw in Lesson 9 and again in Lesson 23. Although ODBC has been widely adopted and remains well supported, ADO is more powerful and provides a more flexible, object-oriented, means of manipulating a database using a programming language.

FIGURE 28.1: The relationship between development tools, ADO, and databases.



The ADO object model consists of three classes of objects. Each object has its own properties and methods that help simplify various aspects of data access:

1. **Connection object** — stores all the information about a connection to a database and provides methods to manage the connection. The database may be file-based (e.g., a MICROSOFT ACCESS .mdb file) or client/server (e.g., ORACLE, SQL SERVER, INTERBASE).
2. **Command object** — stores information about commands sent to the database. Some database commands return data

(e.g., SQL **SELECT** statements) while others perform processing (e.g., ACCESS action queries and stored procedures written using SQL **INSERT**, **UPDATE**, and **DELETE** statements).

3. **Recordset object** — stores the results of a query against a database. Although a recordset is invisible, it has the same structure as a datasheet view of data in ACCESS (records in rows, fields in columns). Changes made to the data in recordset objects can normally be saved to the database.

Since **Recordset** objects contain the data, they are the most important objects in the ADO model for most applications. **Connection** and **Command** objects are used as a means of specifying which data should appear in the recordset.

If you do not have a (recent) version of ADO installed on your machine or if you want to access MICROSOFT's on-line documentation, you should visit the ADO web site at **www.microsoft.com/data/ado**.

28.2 Learning objectives

- understand how ACTIVEX DATA OBJECTS can be used to create dynamic web content
- create ADO Connection, Command, and Recordset objects
- display records from Recordset object
- use ADO to "up-size" a web-based application
- use database look-up for user authorization
- modify values in a database using an HTML form

28.3 Exercises

28.3.1 Creating a Connection object

In this section, you will create a **Connection** object and configure it to point to a web-based version of your ACCESS order entry database.

28.3.1.1 Setting up the source database

You have been provided with an ACCESS database file called **OrderEntryWeb.mdb** in the project package. The database is similar to the one you created in previous lessons for your ACCESS application. However, the **OrderEntryWeb.mdb** file contains a number of queries that are used to help you complete your web-based application in Lesson 29.

➡ Copy the **OrderEntryWeb.mdb** file from the project package to a folder on your web server. Make a note of the full path of the file on the server (e.g., **C:\Documents and Settings\brydon\My Documents\KitchenSupply\OrderEntryWeb.mdb**).

It does not matter where in the web server's directory structure you locate the database file. However, if your web server supports MICROSOFT FRONTPAGE extensions, it is best to locate your database file in a folder that is *not* part of a "FRONTPAGE web".

28.3.1.2 Creating the EmployeeList file

➡ Create a new ASP file and save it as **EmployeeList.asp**.

➡ Add the core HTML tags as well as a title and heading.

➡ In the header section (or anywhere near the top of the file), add the following code to create and configure a new **Connection** object:

```
NL  <HEAD>
NL  …
NL  <%
NL  Set objCon =
    Server.CreateObject("ADODB.
    Connection")
NL  objCon.Provider =
    "Microsoft.Jet.OLEDB.4.0"
NL  objCon.ConnectionString =
    "path\OrderEntryWeb.mdb"
NL  objCon.Open
NL  %>
NL  </HEAD>
```

"*path*" in the **ConnectionString** property refers to the *physical* location of your database on the web server. To use ADO with a MICROSOFT JET (i.e., ACCESS) database, you must specify the complete location (including drive letter and path) of the database file.

➡ To make sure the database connection is opened successfully, add the following verification code to the body of the document:

```
NL  <HTML>
NL  <HEAD>
NL  <TITLE>Employee List</TITLE>
NL  <%
NL  …
NL  objCon.Open
NL  %>
NL  </HEAD>
NL  <BODY>
```

```
NL   State = <%= objCon.State %>
NL   </BODY>
NL   </HTML>
```

The state property can be used to determine whether the connection is open (**State** = 1) or closed (**State** = 0).

➡ Test the **Connection** object, as shown in Figure 28.2.

28.3.1.3 Understanding the ADO code

The code you have just written introduces a few new VBSCRIPT constructs that warrant a brief description.

- The **Set** statement — When you assign an intrinsic data type (e.g., integer, string, etc.) to a variable, you use an equals sign. However, when you set a variable to "point" to an existing object, you must use the **Set** statement. In this case, the variable **objCon** is set to point to a **Connection** object.

FIGURE 28.2: Create and test a ***Connection*** object for a MICROSOFT JET database.

1 Create a new **Connection** object to access your OrderEntryWeb.mdb file.

EmployeeList.asp - Notepad

```
<HTML>

<HEAD>
<TITLE>Employee List</TITLE>
<% Set objCon = Server.CreateObject("ADODB.Connection")
objCon.Provider = "Microsoft.Jet.OLEDB.4.0"
objCon.ConnectionString = "C:\Documents and Settings\bry
objCon.Open
%>
</HEAD>

<BODY>
<H1>Employees</H1>
State = <%= objCon.State %>
</BODY>
```

2 Write the value of the **Connection** object's state to verify that the connection is working.

Employee List - Microsoft Internet Explorer

Address http://localhost/KitchenSupply/EmployeeList.asp

Employees

State = 1

3 View the page and ensure that the connection state is equal to 1 (open).

- **Server.CreateObject("ADODB.Connection")** — This method tells the web server to create a new **Connection** object. The ADODB prefix is required because ACTIVEX components other than ADO may have an object called "connection".
- **objCon.Provider** — The **Provider** property of the **Connection** object is set to a predefined string for accessing MICROSOFT JET databases. You will see in Section 28.3.5 that it is possible to make connections to other types of data providers.
- **objCon.ConnectionString** — The name of the database to open is provided in the connection string.
- **objCon.Open** — The **Open** method opens the connection to the specified database.

In summary, the code in Figure 28.2 tells the server to create a **Connection** object, sets a variable (**objCon**) to point to the new object, sets the properties of the object so that it knows about a MICROSOFT ACCESS database file stored on the server, and opens the database connection.

28.3.2 Creating a Command object

A **Command** object simply stores a command to be sent to the database using Structured Query Language (SQL). The SQL commands are translated by the **provider** (in this case **Microsoft.Jet.OLEDB.4.0**) into a format understood by the data source. Because the database-specific provider takes care of the translation, all you need to know to access data over an ADO connection is a handful of SQL commands.

➡ Create a new variable called **objCmd** and set it to point to a new **Command** object. You can add this to the scripting code at the top of your **EmployeeList.asp** file:

```
NL  <HTML>
NL  <HEAD>
NL  …
NL  <%
NL  Set objCon =
    Server.CreateObject("ADODB.
    Connection")
NL  objCon.Provider =
    "Microsoft.Jet.OLEDB.4.0"
NL  objCon.ConnectionString =
    "path\OrderEntryWeb.mdb"
NL  objCon.Open
NL  Set objCmd = Server.CreateObject
    ("ADODB.Command")
NL  %>
NL  </HEAD>
NL  …
NL  </HTML>
```

Although the code above creates a **Command** object, the object knows nothing about what it

should do, which database it should connect to, and so on. Thus, just as we set the properties of the **Connection** object in Section 28.3.1, we have to set the properties of the **Command** object before using it in subsequent code.

➡ Add the following code to specify the SQL statement and the database connection to be used by the **Command** object:

```
NL  <HTML>
NL  <HEAD>
NL  …
NL  <%
NL  …
NL  objCon.Open
NL  Set objCmd = Server.CreateObject
    ("ADODB.Command")
NL  With objCmd
NL    .CommandText = "SELECT * FROM
    Employees"
NL    .CommandType = 1
NL    Set .ActiveConnection = objCon
NL  End With
NL  %>
NL  </HEAD>
NL  …
NL  </HTML>
```

The resulting **EmployeeList.asp** file is shown in Figure 28.3.

Within the **With objCmd**… **End With** statement, **objCmd** is assumed to be the object being referred to. So instead of writing **objCmd.CommandType**, you can omit the object name and just write **.CommandType**. The **With… End With** construct in VBSCRIPT exists simply to save you some typing.

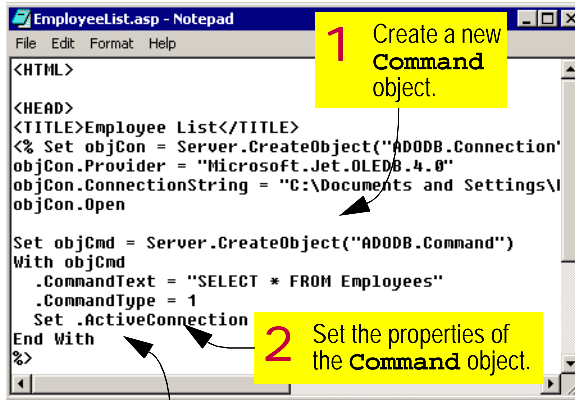
The **CommandText** property is straightforward—it is a SQL **SELECT** statement. The **CommandType = 1** statement tells the target database that the **CommandText** property contains a textual query (versus the name of a table or the name of a stored procedure, etc.). Finally, each **Command** object needs to act on an active database connection. In this case, the **ActiveConnection** property is set to point to the **Connection** object created in Section 28.3.1.

28.3.3 Creating a Recordset object

Once the **Command** object is set up correctly, creating a **Recordset** object based on the results of the embedded SQL query is straightforward.

As is the case with many MICROSOFT technologies, ADO provides many different ways to do the same thing. Although this is powerful and convenient for experienced users, it is frustrating and confusing for new users. The method used

FIGURE 28.3: Create a ***Command*** object based on a SQL ***SELECT*** query.



The **With… End With** statement can be used instead of repeatedly typing “objCmd”.

to create recordsets used here is more complex than it needs to be. However, it permits greater control over the type of **Recordset** returned. Controlling the recordset type becomes important if you need to update the data in the database.

- Create a new **Recordset** object called **rsEmp** using the now-familiar **Server.CreateObject** method (see the code example with the next step if you are stuck).

- Add the following code to set the properties of the **Recordset** object:

```
NL  <HTML>
NL  <HEAD>
NL  …
NL  <%
NL  …
NL    Set .ActiveConnection = objCon
NL  End With
NL  Set rsEmp = Server.CreateObject
    ("ADODB.Recordset")
NL  With rsEmp
NL    .CursorType = 1
NL    .LockType = 3
NL    .Open objCmd
NL  End With
NL  %>
NL  </HEAD>
NL  …
NL  </HTML>
```

Although the **CreateObject** method creates a new **Recordset** object, the **Recordset** object does not contain any data until the **Open** method is invoked.

The **Recordset.Open** method requires that a valid **Command** object be passed as an argument. The **Command** object is used when opening the recordset to determine which records are

retrieved from the database. And the **Command** object uses its **ActiveConnection** property to know how to connect to the database. In a nutshell, this is how ADO works: the **Recordset** object depends on the **Command** object and the **Command** object depends on the **Connection** object.

➨ To ensure that the SQL query in your **Command** object returns data, add the following verification code to the body of your document:

```
NL <HTML>
NL <HEAD>
NL <TITLE>Employee List</TITLE>
NL <%
NL …
NL %>
NL </HEAD>
NL <BODY>
NL <P>State = <%= objCon.State %></P>
NL <P>End of File = <%= rsEmp.EOF %></
   P>
NL </BODY>
NL </HTML>
```

The **EOF** property of a **Recordset** object returns True if the recordset is at the "end of file" marker and False otherwise. When a recordset is opened, its record pointer is set to the first record; however, if the recordset is empty (i.e., there is no first record) the **EOF** property is set to True.

➨ Test your code, as shown in Figure 28.4.

28.3.4 Viewing a Recordset's contents

Although an ADO **Recordset** object is invisible, its logical structure is identical to that of a datasheet in ACCESS. The most important thing to know about using a recordset is that there is a **record pointer** (or **recordset cursor**, if you prefer) which points to the "current row". To access a particular record, you must use the record pointer to navigate through the individual records.

Although an ADO **Recordset** and a DATA ACCESS OBJECTS (DAO) **Recordset** (recall Lesson 21) have many important differences, the basic concept of a recordset is the same in both cases.

Once you have set the record pointer to a particular row in a recordset, you can access the data in the record by using the **Fields** collection. For example, to get the current value of the **emp_lname** field in the **rsEmps** recordset, you use the following syntax:

```
NL … rsEmps.Fields("emp_lname") …
```

FIGURE 28.4: Create and test a **Recordset** object.

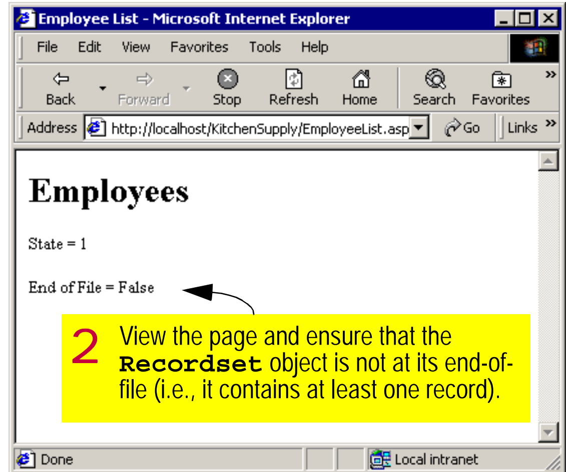
1 Create a new Recordset object by executing a **Command** object containing a SQL **SELECT** query.

EmployeeList.asp - Notepad

```
Set rsEmp = Server.CreateObject("ADODB.Recordset")
With rsEmp
   .CursorType = 1
   .LockType = 3
   .Open objCmd
End With
%>
</HEAD>

<BODY>
<H1>Employees</H1>
<P>State = <%= objCon.State %></P>
<P>End of File = <%= rsEmp.EOF %></P>
```

In the **Open** method, **objCmd** is passed as an argument.



2 View the page and ensure that the **Recordset** object is not at its end-of-file (i.e., it contains at least one record).

Unlike VBA, VBScript does not support use of the “**!**” shortcut. Thus, the following syntax returns an error: **rsEmps!emp_lname**.

- Delete the verification code in the body of the document. It is no longer required.

- Add a loop to cycle through all the records in the **Recordset** object:

```
NL <HTML>
NL <HEAD>
NL <TITLE>Employee List</TITLE>
NL <%
NL ...
NL %>
NL </HEAD>
NL <BODY>
NL <UL>
NL <% Do Until rsEmp.EOF %>
```

```
NL    <LI>
NL    <%= rsEmp.Fields("emp_lname") &
      ", " & rsEmp.Fields("emp_fname")
      %>
NL    </LI>
NL    <% rsEmp.MoveNext
NL  Loop %>
NL  </UL>
NL  </BODY>
NL  </HTML>
```

Note that the record pointer is explicitly moved from one record to the next using the recordset's **MoveNext** method. There are other cursor manipulation methods, such as **MoveFirst**, **MoveLast**, and **MovePrevious**.

If you forget the **MoveNext** method within the loop, **EOF** will never be set to True because the record pointer will never be moved from the first record. In such a case, the web server will become caught in an endless loop.

➡ Test the employee list, as shown in Figure 28.5.

Congratulations: you have just used created a dynamic web page that pulls its content from a database.

28.3.5 Swapping data sources

It is generally *not* a good idea to build a web based application on top of an ACCESS database. The JET database engine (which powers ACCESS) is intended for workgroup-size applications. It cannot scale up to a large volume of concurrent users and its security features are inadequate for storing large volumes of confidential or financial transactions.

It is possible to use ACCESS for prototyping your web-based applications. But once you have your prototype working, it is best to "up-size" to an industrial-strength client/server database system. Fortunately, ADO makes this easy.

28.3.5.1 Demonstration mode

In this section, we will revert to demonstration mode. I will show you how to create an ADO connection to a client/server database (the same SQL SERVER database that I used for the demonstration in Section 9.3.2).

28.3.5.2 Modifying the Connection object

➡ I start by modifying the properties of **objCon** so that it connects to a SQL SERVER database using the TCP/IP protocol. The changes are shown below.

```
NL  <%
NL
```

```
NL  <HTML>
NL  <HEAD>
NL  …
NL  Set objCon =
    Server.CreateObject("ADODB.
    Connection")
NL  With objCon
NL    .Provider = "sqloledb"
NL    .ConnectionString =
      "Server=brydon.bus.sfu.ca;
      UID=<username>;pwd=<password>"
NL    .Open
NL  End With
NL  …
NL  %>
NL  </HEAD>
NL  …
NL  </HTML>
```

Each type of data source requires a different **ConnectionString** property. For example, client/server databases require a server name, a user name, and a

FIGURE 28.5: Loop through the ***Recordset*** object to display the items return by the query.

1 Loop through the **Recordset** object and add the items to an unordered list. The loop stops when the end-of-file (EOF)marker is reached.

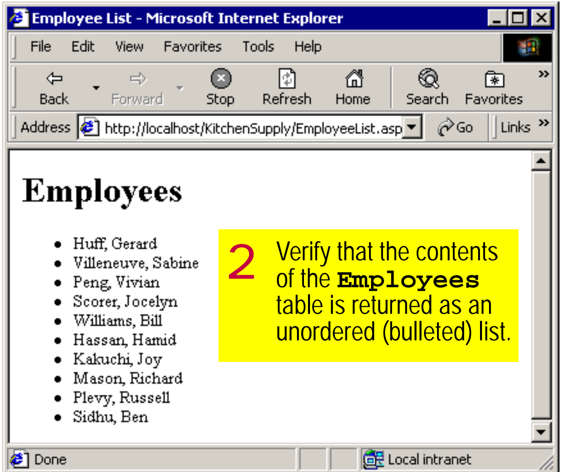
EmployeeList.asp - Notepad

File Edit Format Help

```
<BODY>
<H1>Employees</H1>
<UL>
<% Do Until rsEmp.EOF %>
  <LI>
  <%= rsEmp.Fields("emp_lname") &  ", " & rsEmp.Fiel
  </LI>
  <% rsEmp.MoveNext
Loop %>
</UL>
</BODY>

</HTML>
```

Note the placement of the scripting tags, **<%** and **%>**. They enclose VBSCRIPT code only.



password, whereas ACCESS only requires the location of a *.mdb file.

Once the properties of the **Connection** object have been updated, I am done. It is as simple as that. Since the details of the data source are entirely encapsulated within the **Connection**, no changes have to be made to the rest of my code.[1]

➡ I verify that the new data source is working correctly. The results are identical to those shown in Figure 28.5.

28.3.6 Authentication

In Section 26.3.3.2, you created a "mock" authentication routine. A more useful authentication routine consists of two steps:

1. Search the **Customers** table for a user name that matches the value of **txtUserName** passed to the server by login form. If a match is not found, authorization fails.
2. If a matching user name is found, compare the value of **txtPassword** passed to the server to the user's password in the database. If they do not match, authorization fails; otherwise, authorization succeeds.

[1] In practice, changing databases may be more complex. Since database systems have very different capabilities, problems occur when code that exploits features of one DBMS is executed against a different DBMS that does not support the feature.

In this section, you will create an authorization routine that implements the above logic.

➡ Open **Authorize.asp** for editing. Cut and paste the connection and command information for your ACCESS database from **EmployeeList.asp** into the top part of the **Authorize.asp** file.

➡ To make the code a bit neater, modify the **objCon** section to use the **With… End With** syntax. The results are shown below:

```
NL <%
NL strUserName=Request.Form.Item("txtUserName")
NL strPassword=Request.Form.Item("txtPassword")
NL Set objCon = Server.CreateObject("ADODB.Connection")
NL With objCon
NL   .Provider = "Microsoft.Jet.OLEDB.4.0"
NL   .ConnectionString = "C:\Documents and Settings\brydon\My Documents\KitchenSupply\OrderEntryWeb.mdb"
NL   .Open
NL End With
```

```
NL Set objCmd =
   Server.CreateObject("ADODB.Command")
NL With objCmd
NL   .CommandText = "SELECT * FROM
   Employees"
NL   .CommandType = 1
NL   Set .ActiveConnection = objCon
NL End With
NL If strUserName=strPassword Then
NL   Session("LI") = "True"
NL   Response.Redirect "Menu.asp"
NL Else
NL   Session("LI") = "Fail"
NL   Response.Redirect "Login.asp"
NL End if
NL %>
```

➨ Change the **CommandText** property to a SQL statement that returns the correct row from the **Customers** table:

```
NL <%
NL …
NL Set objCmd =
   Server.CreateObject("ADODB.Command")
NL With objCmd
NL   .CommandText = "SELECT * FROM
   Customers WHERE UserName = '" &
   strUserName & "'"
NL   .CommandType = 1
NL   Set .ActiveConnection = objCon
NL End With
NL …
NL %>
```

➨ Cut and paste the **Recordset** code from **EmployeeList.asp** to **Authorize.asp**.

➨ Change the name of the **Recordset** object from **rsEmp** to **rsCust** wherever **rsEmp** is used.

These changes are shown in Figure 28.6. If the username entered by the user is in the **Customers** table, a recordset consisting of a single record is returned to **rsCust**. Otherwise, an empty recordset is returned.

If you do not make the changes shown in Figure 28.6, the authorization routine will fail. Since debugging ASP scripts is difficult and time consuming, it is important to look very carefully at your code for typos and silly mistakes before publishing the files to the web server.

➨ Modify the **If… Then** statement to recognize the three possible authorization outcomes. The code is shown below and in Figure 28.7.

```
NL <%
NL …
NL If rsCust.EOF Then
NL   Session("LI") = "FAIL_USER"
NL   Response.Redirect "Login.asp"
```

FIGURE 28.6: Create **`Connection`**, **`Command`**, and **`Recordset`** objects in **`Authorize.asp`** to support an authorization routine.

1 Cut and paste the connection and command information. Clean up the code a bit by using the **`With…End With`** construct.

3 Create a **`Recordset`** object. It should consist of one record if the user exists in **`Customers`** table.

? Since the SQL statement requires nested quotation marks, the single quotes are enclosed within double quotes to give a result of the form: **`"SELECT * FROM Customers WHERE UserName = 'sammy'"`**.

authorize.asp - Notepad

File Edit Format Help

```
<%
strUserName=Request.Form.Item("txtUserName")
strPassword=Request.Form.Item("txtPassword")

Set objCon = Server.CreateObject("ADODB.Connection")
With objCon
   .Provider = "Microsoft.Jet.OLEDB.4.0"
   .ConnectionString = "C:\Documents and Settings\brydon\My Documents\
   .Open
End With

Set objCmd = Server.CreateObject("ADODB.Command")
With objCmd
   .CommandText = "SELECT * FROM Customers WHERE UserName = '" & strUs
   .CommandType = 1
   Set .ActiveConnection = objCon
End With

Set rsCust = Server.CreateObject("ADODB.Recordset")
With rsCust
   .CursorType = 1
   .LockType = 3
   .Open objCmd
End With
```

```
NL  Else
NL    If rsCust.Fields("Password") <>
      strPassword Then
NL      Session("LI") = "FAIL_PASSWORD"
NL      Response.Redirect "Login.asp"
NL    Else
NL      Session("LI") = "True"
NL      Response.Redirect "Menu.asp"
NL    End If
NL  End If
NL  %>
```

➡ Modify **`Login.asp`** to provide meaningful error messages when the user enters the wrong username or password:

```
NL  <HTML>
NL  …
NL  <H3>
```

FIGURE 28.7: Implement the authorization logic using nested ***If… Then*** statements.

authorize.asp - Notepad

File Edit Format Help

```
Set rsCust = Server.CreateObject("ADODB
With rsCust
  .CursorType = 1
  .LockType = 3
  .Open objCmd
End With

If rsCust.EOF Then
  Session("LI") = "FAIL_USER"
  Response.Redirect "Login.asp"
Else
  If rsCust.Fields("txtPassword") <> strP
    Session("LI") = "FAIL_PASSWORD"
    Response.Redirect "Login.asp"
  Else
    Session("LI") = "True"
    Set Session("rsCust") = rsCust
    Response.Redirect "Menu.asp"
  End If
```

Invalid user name

User name okay; wrong password

User name and password okay

```
NL  <% If Session("LI")="FAIL_USER"
    Then %>
NL    User name incorrect: please try
      again
NL  <% ElseIf
    Session("LI")="FAIL_PASSWORD"
NL  Then %>
NL    Password incorrect: Please try
      again
NL  <% End If %>
NL  </H3>
NL  …
NL  </HTML>
```

➡ Test the authorization procedure using different user names and passwords. When the valid user name/password combination "sammy"/"sam" is entered, you should be transferred to the menu page.

28.3.7 Updating data

To this point, you have only *displayed* data contained in **Recordset** objects. However, it is possible to use the **Recordset** object to make changes (append, delete, update) to the underlying data. In this section, you will create a "customer profile" form. Your on-line customers can use this form to view and edit certain information about themselves, including their billing address, contact information, and so on.

The **rsCust** recordset you used in the preceding section will be used for two purposes:

1. provide the initial values to show on the form (via the textboxes' **VALUE** attribute), and
2. accept and save the new values of the **VALUE** attribute entered into the form by users.

To save the users' changes to the data, you have to do a bit of plumbing. Specifically, you have to write a script that takes the values from the

form (via the **`Request.Form`** collection) and use them to update the customer record. As a consequence, you will require two ASP files:

- **`Customer.asp`** — the visible form used to display customer data and permit users to make changes; and,
- **`Customer_Process.asp`** — an invisible script that writes any changes made by users to the database and then returns users to the updated **`Customer.asp`** page.

28.3.7.1 Reusing the customers recordset

The **`rsCust`** recordset already contains all the customer information that you need. However, recall that the scope of ASP variables is limited to the page on which the variables were created. In other words, once the user leaves the **`Authorize.asp`** page, the **`rsCust`** object is destroyed.

To get around this, you are have two options:

1. Create a new customer **`Recordset`** object whenever you need it. To create the correct SQL query each time, you need to pass some information about the customer (e.g., **`CustID)`** from page to page using a query string or a session variable.
2. Save the **`Recordset`** object as a session variable.

Neither option is always better than the other. Much depends on how often the **`Recordset`** in question will be used by other pages. However, Option 2 is certainly easier to implement, so we will stick with it.

The amount of memory required to store a simple value (such as **`CustID`**) is many times smaller than the amount of memory required to store a complex object such as an ADO **`Recordset`**. Since each user requires a unique session, the number of ADO objects that the server has to store in memory can become very large if there are many simultaneous users. Since we are more interested in broad concepts at this stage, we will ignore memory optimization. However, you should be aware of the scalability issues that arise when you save objects to session variables.

➨ Add the following code to **`Authorize.asp`** to save a reference to the customers recordset in a session variable:

```
NL  <%
NL  …
NL  If rsCust.EOF Then
NL    Session("LI") = "FAIL_USER"
NL    Response.Redirect = "Login.asp"
NL  Else
```

```
NL    If rsCust.Fields("Password") <>
  strPassword Then
NL      Session("LI") = "FAIL_PASSWORD"
NL      Response.Redirect "Login.asp"
NL    Else
NL      Session("LI") = "True"
NL      Set Session("rsCust") = rsCust
NL      Response.Redirect "Menu.asp"
NL    End If
NL  End If
NL  %>
```

Whenever the user logs in successfully, a session-level pointer to the **Recordset** object is created that is available in all pages in the application for the duration of the session.

As soon as there are no variables pointing to the object, the object is destroyed and marked for garbage collection.

28.3.7.2 Creating a customer profile form

➡ Rename the **Customer.html** file you created in Section 24.5 to **Customer.asp**.

➡ In the header of **Customer.asp**, create a local variable which points to the session-level customer **Recordset** object. This is done to save some typing later.

```
NL  <HTML>
NL  <HEAD>
NL  <% Set rsCust=Session("rsCust") %>
NL  …
NL  </HEAD>
NL  …
NL  </HTML>
```

➡ Add form tags to the body of **Customer.asp**. As always, the **METHOD**, should be "Post". The **ACTION** value should be **Customer_Process.asp** (which you will create shortly).

In order to simplify the layout of multiple textboxes, it is worthwhile to put the form elements within an invisible table.

➡ Nest **<TABLE>… </TABLE>** tags inside of the **<FORM>… </FORM>** tags.

➡ Add the following code to create a row with two columns:

```
NL  …
NL  <FORM ACTION="customer_process.asp"
    METHOD="post">
NL  <TABLE cellPadding=1 cellSpacing=1
    width="75%">
NL  <TR>
NL  <TD>Name: <INPUT TYPE="Text"
    NAME="txtCustomerName" VALUE="<%=
    rsCust.Fields("CustName") %>"></TD>
NL  <TD>Contact person: <INPUT
    TYPE="Text" NAME="txtContactPerson"
    VALUE="<%=
```

```
rsCust.Fields("ContactPerson")
%>"></TD>
NL </TR>
NL …
NL </TABLE>
NL </FORM>
NL …
```

- Add a second row containing textboxes with the following attribute/value pairs:

Textbox name	Field name
txtBillingAddresss	rsCust.BillingAddress
txtContactPhone	rsCust.ContactPhone

- Add a third row with a **Submit** and **Reset** buttons:

```
NL <TABLE>
NL …
NL <TR>
NL <TD><INPUT TYPE="Submit"
   NAME="cmdSubmit" VALUE="Submit"></
   TD>
NL <TD><INPUT TYPE="Reset"
   NAME="cmdReset" VALUE="Reset"></TD>
NL </TR>
NL </TABLE>
NL </FORM>
NL …
```

To test the form, use the menu buttons you created in Section 25.3.6 and enabled with scripts in Section 26.5.

- Bring up the login page in your browser and log in as "sammy"/"sam".

- Click the "Go" button for "Update Customer Profile." You should see a form similar to that shown in Figure 28.8.

28.3.7.3 Creating a form processing script

With the HTTP protocol, you do not have access to the form values until the submit button is pressed and the browser sends the web server an HTTP request. In ASP, the target of the HTTP request is generally a script-only page that executes and then transfers the user to a page with visible content.

In this section, you will create a processing script to write changes made by the user to the database and then transfer the user back to the customer profile page.

- Create a new file called **`Customer_Process.asp`**. Since the file will not contain any HTML, you do not need to add the core HTML tags.

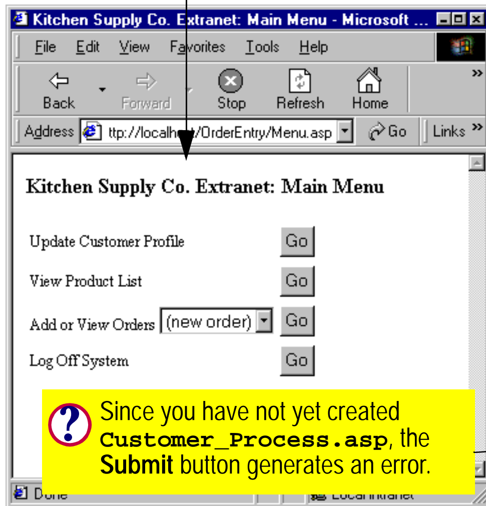
- Add the following code to transfer the values from the form to the table fields of the **`Recordset`** object:

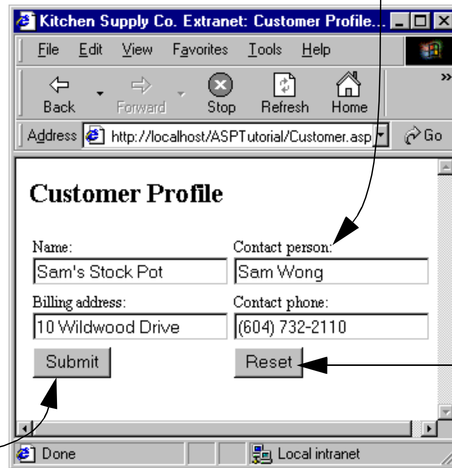
```
NL <%
NL With Session("rsCust")
```

FIGURE 28.8: Test the customer profile form by logging in successfully and clicking on the hyperlink on the menu page.

1 Login as "sammy"/"sam". You should be transferred to the main menu page.

2 Verify that the textboxes' **VALUE** attributes are correctly populated from the **Recordset** object.





3 Change values on the form and observe the effect of the **Reset** button.

Since you have not yet created **Customer_Process.asp**, the **Submit** button generates an error.

```
NL    .Fields("CustName")=
   Request.Form("txtCustomerName")
NL    .Fields("BillingAddress")=
   Request.Form("txtBillingAddress")
NL    .Fields("ContactPerson")=
   Request.Form("txtContactPerson")
NL    .Fields("ContactPhone")=
   Request.Form("txtContactPhone")
NL    .Update
NL  End With
NL  Response.Redirect "Customer.asp"
NL  %>
```

If you forget the **`Update`** method, the changes to the **`Recordset`** object will not be saved to the database.

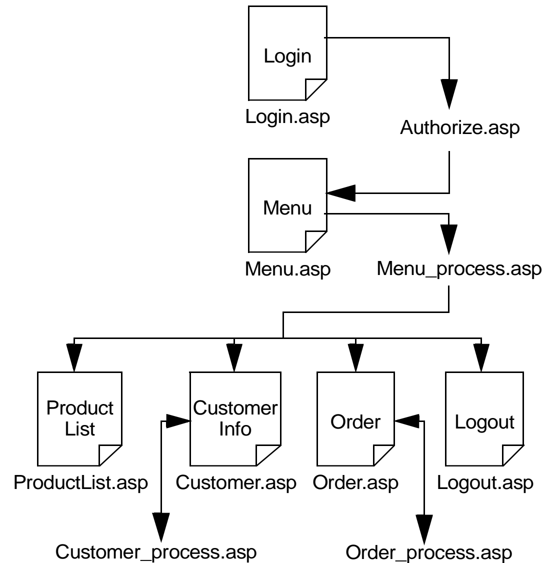
➡ Change values in the customer profile form to ensure that they are saved to the database.

28.4 Discussion

The two-file approach you used to process database updates for customers is fairly typical in ASP application development. One file (**`Customer.asp`**) contains HTML form elements and allows the user to enter new values for customer fields. The second file (**`Customer_process.asp`**) contains executable script only and never shows up in the user's browser. Instead, the code to update the database is executed and the **`Response.Redirect`** method is used to transfer the user back to the (updated) customer profile.

Your web-enabled order entry system requires a number of invisible script-only pages to process the different types of information entered by your users. The relationship of the script-only pages to the content pages that you have already created is shown in Figure 28.9.

FIGURE 28.9: The updated structure of the web-based order entry system.



28.5 Application to the project

28.5.1 Touch-ups

➡ Ensure you have implemented the login and customer profiles pages described in this lesson.

➨ If you wish, you may add more information to the customer profile form, although it is certainly not a requirement (if you know how to do four HTML textboxes, you know how to do a thousand).

➨ To save time when testing your application in subsequent lessons, set the default values for the user name and password textboxes on your login form to appropriate values. For example, set the **VALUE** attribute of **txtUserName** to "sam". In this way, you can login with one click and do not have to keep typing "sam"/"sammy".

Obviously, you should remove the **VALUE** attributes from both textboxes on your login page when you put the application into production.

28.5.2 Persistence pays

As Figure 28.9 shows, your application requires a large number of ASP pages to function properly. Many of these pages require access to the database either to:

- extract and display data (e.g., **Products.asp**),
- search the database (e.g., **Authorize.asp**), or
- update the database (**Customer_process.asp**).

Rather than create a separate ADO connection for each page, it is much simpler to create a session-level reference to the **Connection** object when it is first created.

➨ Add the following code to your **Authorize.asp** file. The code should only execute when user authentication is successful.

```
NL  Set Session("objCon") = objCon
```

28.5.3 Creating a dynamic list of products

➨ Use an ADO recordset to complete the product list page you started to populate manually in Section 24.3.5. This task requires the same skills as the employee list in Section 28.3.4.

In ADO, you can use shortcuts to create recordsets. Specifically, you can create a recordset without first creating a command object or setting the recordsets properties. The following code creates a recordset containing a list of products sorted by the **ProductID** field:

```
NL  Set rsProducts =
    Server.CreateObject
    ("ADODB.Recordset")
```

NL
```
rsProducts.Open "SELECT * FROM
Products ORDER BY ProductID",
Session("objCon"), 0, 1
```

Three arguments are applied to the recordset's **`Open`** method:

1. **Source** (**`SELECT * …`**)— source can be an ADO **`Command`** object, the name of a table, or (as in this case) an SQL statement.
2. **Connection** (**`Session("objCon")`**)— if the source is an ADO **`Command`** object (with its **`ActiveConnection`** property set), then the connection argument is optional. However, in this case, no **`Command`** object is used and thus a **`Connection`** object must be supplied.
3. **Cursor type** (**`0`**)— since the product list is read-only, the simplest type of recordset ("forward only") is used. A forward-only recordset is a read-only snap shot in which the only permitted method is **`MoveNext`** or **`MoveLast`** (**`MovePrev`** and **`MoveFirst`** violate the forward-only constraint). Use of this type of recordset greatly reduces the amount of server resources required to show the product list.
4. **Lock type** (**`1`**)— the lock type is set to read-only. Given that the cursor type is read-only already, this property does not need to be set.