

Web/Database Integration with Active Server Pages

Course Notes

Nick Gould

October 1998

1	INTRODUCTION	3
1.1	COURSE SUMMARY.....	3
1.2	WHAT IS AN ACTIVE SERVER PAGE?	3
1.3	REQUIREMENTS TO RUN ACTIVE SERVER PAGES.....	4
1.4	SETTING UP YOUR SYSTEM TO RUN ACTIVE SERVER PAGES	4
1.5	WHY USE ACTIVE SERVER PAGES?.....	5
2	ACTIVE SERVER PAGE PROGRAMMING WITH VBSCRIPT.....	6
2.1	VARIABLES IN VBSCRIPT	6
2.2	OPERATORS	6
2.3	COMMENTS.....	7
2.4	PROGRAM FLOW	7
2.5	BUILT-IN FUNCTIONS.....	8
2.6	USER CREATED PROCEDURES - SUB AND FUNCTION	9
2.7	ARRAYS.....	9
2.8	INCLUDING FILES	9
2.9	TROUBLESHOOTING.....	10
3	HTML FORMS	11
3.1	HOW HTML FORMS WORK	11
3.2	SPECIFYING A HTML FORM	11
3.3	FORM CONTROLS	11
4	HOW ACTIVE SERVER PAGES HANDLE HTML FORM DATA.....	17
4.1	THE REQUEST OBJECT.....	17
4.2	SELF-CALLING ACTIVE SERVER PAGES.....	19
5	USING THE SESSION OBJECT TO MAINTAIN STATE.....	21
6	HOW ACTIVE SERVER PAGES INTERFACE WITH DATABASES.....	22
7	SQL	23
7.1	WHAT IS SQL?.....	23
7.2	SQL BASICS	23
8	CONFIGURING ODBC.....	26
8.1	CREATING AN ODBC DATA SOURCE NAME	26
9	CONNECTING TO A DATABASE USING ACTIVE SERVER PAGES	27
10	QUERYING DATABASES USING HTML FORMS.....	30
10.1	GENERATING A QUERY FROM HTML FORM RESPONSES	30
11	MODIFYING DATABASE RECORDS.....	38
11.1	ADDING RECORDS TO A DATABASE.....	38
11.2	HANDLING ERRORS	40
11.3	VALIDATING DATA	42
11.4	CHECKING THE QUERY STRING FOR SINGLE QUOTES	43
11.5	DELETING RECORDS	44
11.6	MODIFYING RECORDS - UPDATE	46
12	SUMMARY	49
13	INDEX.....	50

1 Introduction

1.1 Course summary

This course is an introduction to Microsoft's Active Server Page technology. In particular it will concentrate on how to develop Web based interfaces to databases. Although you can use a variety of scripting languages with Active Server Pages this course will concentrate on using VBScript. This course will concentrate on using MS-Access databases but the techniques used will be exactly the same if we were using other database packages.

1.2 What is an Active Server Page?

An Active Server Page is a text file which resides on a Web server. When a Web client (browser) calls an Active Server Page the Web server processes the code in the Active Server Page and returns standard HTML to the browser. The advantage Active Server Pages have over standard HTML pages is that they are dynamic; the HTML sent to the browser is generated "on-the-fly" by the server and is dependent on the previous actions of the browser user.

A browser calls an Active Server Page in the same way that it calls a standard HTML page. The only difference is in the file extension, whereas HTML pages have extension **htm** or **html**, Active Server Pages have the extension **.asp**.

An Active Server Page looks like something like this:

```
<HTML>
<BODY>
<% For i = 3 To 7 %>
<FONT SIZE="<% = i %>">Hello World!<BR>
<% Next %>
</BODY>
</HTML>
```

An Active Server Page consists of scripting language statements and standard HTML code. To distinguish between the two we put the scripting code in brackets **<% %>**

When the URL of the Active Server Page is entered in the browser, say

```
http://nt2.ec.man.ac.uk/coursesamples/example1.asp
```

the web server processes the Active Server Page code and would, in this example, return the following HTML

```
<HTML>
<BODY>
<FONT SIZE="3">Hello World!<BR>
<FONT SIZE="4">Hello World!<BR>
<FONT SIZE="5">Hello World!<BR>
<FONT SIZE="6">Hello World!<BR>
<FONT SIZE="7">Hello World!<BR>
</BODY>
</HTML>
```

The processing is invisible to the browser and the user and all they see is standard HTML.

1.2.1 Scripting Languages

The default scripting language for Active Server Page is VBScript which is a subset of Visual Basic for Applications (VBA) which itself is a subset of Visual Basic. However you are not limited to using VBScript as your scripting language. Other alternatives include Jscript and Perl.

1.3 Requirements to run Active Server Pages

Web Server

Active Server Pages runs as a component of Microsoft Internet Information Server web server (Versions 3.0 and above) on Windows NT server. It will also run with Microsoft's Personal Web Server on Windows 95. So Active Server Page technology is tied in with Microsoft web servers. However third parties are developing Active Server Page add-ins for other web servers (see www.chillisoft.com).

ODBC Drivers

If you are going to use Active Server Pages to interface with databases then you will need an ODBC driver for your particular database system. The Active Server Page installation includes drivers for Microsoft Access and Microsoft SQL Server.

Text Editor

Although it is possible to generate Active Server Pages with development tools such as FrontPage 98, we are going to concentrate on writing Active Server Page with a text editor. Third party text editors such as *TextPad* offer more functionality than the standard Windows editors like *WordPad*.

1.4 Setting up your system to run Active Server Pages

1.4.1 Installing Active Server Pages

If your Web server does not already have the Active Server Page component already installed then installing it and configuring it is a straightforward process. The Active Server Page component can be downloaded from Microsoft's website.

1.4.2 Configuring an Active Server Page Directory

Directories containing static web pages are given *read* status in the web server configuration. However, the directory that contains your Active Server Pages must have *execute* status. This can be set using the web server's administration tool.

1.4.3 Help

Active Server Pages comes with online documentation. This documentation includes full reference material, tutorials and a comprehensive sample site which is particularly useful for database integration information. A link to the documentation, in the form of an item on the taskbar, *Active Server Pages Roadmap*, is added on installation.

1.5 Why use Active Server Pages?

There are a number of alternatives for those wishing to create interactive, dynamic web pages. Alternatives include using CGI scripting and more specialist development tools such as *Cold Fusion*.

Here are some reasons for using Active Server Page technology

- Active Server Pages are browser independent.
- easy to create and use.
- offers choice of scripting languages.
- no extra software costs.

Active Server Page technology does have some drawbacks. In particular is its reliance on Microsoft platforms.

Active Server Page technology does not provide a full program development environment so activities such as debugging a script may not be straightforward. However since most Active Server Pages tend to be fairly short this is rarely a problem.

2 Active Server Page Programming with VBScript

For the purpose of this course we will be using VBScript as our scripting language, although as explained earlier there are alternatives to VBScript. This section can do no more than explain the basics of VBScript - for reference refer to the online documentation.

VBScript is a scripting language rather than a programming language, such as Visual Basic or C++. Scripting languages tend to be less rigid in their rules than programming languages. For example, in VBScript (and Perl) you need not predeclare variables before using them.

2.1 Variables in VBScript

Values are assigned to variables in the standard way. For example,

```
x = 24
```

You do not need to declare variables before using them (although many would say it is good practice to do so). There is only one data type, called *variant*. As its name implies it can store a variety of information - numeric, string, Boolean - depending on the situation. So the following code is perfectly valid:

```
x = 36  
x = "Some text"
```

This is similar to how Perl handles data types.

2.1.1 Outputting variable values

How do we return the value of a variable to the user? If we return to our original example

```
<% for i = 3 to 7 %>  
    <FONT SIZE="<% = i %>">Hello World</FONT><BR>  
<% next %>
```

We can include the value of a VBScript variable within HTML using this format:

```
<% = variable_name %>
```

Note that the spaces are not significant.

2.2 Operators

VBScript has the standard range of operators for arithmetic. i.e. +, -, * and /
It also has the use of logical operators NOT, AND, OR

2.3 Comments

We can add comments to our VBScript code using a single quote. For example

```
'this is a comment
```

2.4 Program Flow

VBScript contains the standard statements allowing the use of decisions and looping.

2.4.1 Decisions

If..Then

Example

```
<%  
If X < 0 Then  
    %>  
    <B>Negative Number</B>  
    <%  
End If  
%>
```

If...Then...Else

Example

```
If X < 0 Then  
    %>  
    <B>Negative Number</B>  
    <%  
Else  
    %>  
    <B>Positive Number</B>  
    <%  
End If  
%>
```

If...Then...ElseIf

The preceding example could be modified to take account of zero (0) i.e. neither negative or positive

```
If X < 0 Then
    %>
    <B>Negative Number</B>
    <%
ElseIf X > 0 Then
    %>
    <B>Positive Number</B>
    <%
Else
    %>
    <B>Value is 0</B>
<%
End If
%>
```

2.4.2 Loops

Do While...Loop

For example, the following code prints the value of X ten times,

```
X=10
Do While X > 0
    %>
    <B><%=X%></B><BR>
    <%
    X = X - 1
Loop
```

For...Next

```
For X = 1 To 5
    Statements
Next
```

Statements are executed 5 times.

2.5 Built-in Functions

VBScript contains a number of built-in functions. Functions take an argument and return a value. The argument is supplied in brackets. For example the *Len* function returns the number of characters in a string

```
MyWord="Potato"
X = Len(MyWord)
```

The value of X would be 6.

There are built-in functions for string manipulation, date and time, and mathematics.

2.6 User Created Procedures - Sub and Function

The Sub procedure contains a group of VBScript statements which are executed when the sub is called. The Sub may take arguments but does not return a value. A sub is defined as follows:

```
Sub sub_name  
    statement group  
End Sub
```

For example, this sub takes a string, turns its characters into uppercase and counts them,

```
Sub do_string(MyString)  
    %>  
    The original string:  <% =MyString %><BR>  
    The uppercased string:  <% =Ucase(MyString) %><BR>  
    Number of Characters:  <% =Len(MyString) %><BR>  
    <%  
End Sub
```

It is called by

```
    Call do_string("Now is the winter of our discontent")  
or  
    do_string "Now is the winter of our discontent"
```

If you use the Call statement to call a subroutine the argument list needs to be in parenthesis.

The function procedure is similar to the sub procedure except that it always returns a value.

2.7 Arrays

VBScript allows the use of arrays.

For example the following code fills an array with random integers between 1 and 10

```
DIM nos(5)  
For I = 0 to 4  
    nos(I)=INT(RND*10)  
Next
```

2.8 Including Files

You can include one or more files in an Active Server Page. This is particularly useful for headers and footers, and as a way of building up function libraries.

For example the file *header.inc* contains the text

```
<HTML>
<HEAD>
</HEAD>
<TITLE>Page Title</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
<IMG SRC="mylogo.gif ALT="logo">
```

To include *header.inc* into another ACTIVE SERVER PAGE file we would use

```
<!--#INCLUDE FILE="header.inc"-->
```

In general we use

```
<!--#INCLUDE FILE="relative_path_of_include_file"-->
```

Note that the use of the extension **.inc** is not obligatory but is standard practice.

2.9 Troubleshooting

The most common error source is that of syntax. You will get a response like this



Microsoft VBScript compilation error '800a03f6'
Expected 'End'
/coursesamples/mypage.asp, line 15

The most important bit of information is the line number. You can use this to locate the source of the syntax error. It helps to have a text editor that displays line numbers.

3 HTML Forms

In the examples so far we have been merely concerned with sending information out to the web browser, there has not been a dialogue between the user and the application. For Active Server Pages to come alive we need to use HTML forms which allow for user input.

3.1 How HTML forms work

A HTML form is populated with controls. These could include text boxes, radio buttons, and drop-down menus. When the form is filled in, and the Submit button clicked, the browser parcels up the name of each control and the value entered in each and sends this information to the Web server. This string of data is known as the *query string*. A typical query string may look like

```
surname=smith&forenames=David+John&gender=1
```

The web server passes the query string to the Active Server Page specified in the HTML form. The Active Server Page processes the query string and sends back HTML to browser via the Web server.

3.2 Specifying a HTML Form

HTML forms are specified as follows

```
<FORM METHOD="POST | GET" ACTION="path_of_Active_Server_Page">
```

Form controls

```
</FORM>
```

The METHOD parameter, which can be either GET or POST specifies how the form data is sent to the server. There are subtle differences between the GET and POST methods. In this course we will be using GET throughout. The ACTION parameter specifies the name of the Active Server Page which is going to process the data.

3.3 Form Controls

The form controls are specified as follows

1. Text box

The text box is used for a single line of text

*Syntax*¹

```
<INPUT TYPE="TEXT | PASSWORD" NAME="name_of_control" [VALUE="default_text" ]  
[SIZE="width_in_characters" ] MAXLENGTH="width_in_characters" ] >
```

¹ parameters in [] are optional

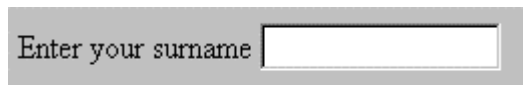
Notes

1. If the input type is PASSWORD then any characters entered appear as *'s
2. The default size is 10 characters

Example

Enter your surname `<INPUT TYPE="TEXT" NAME="surname" MAXLENGTH="30">`

Appears in the browser as

A screenshot of a web browser displaying a text input field. The label "Enter your surname" is positioned to the left of a rectangular text box. The entire element is set against a light gray background.

2. Text Window

The text window is used for larger text input and has multiple lines.

Syntax

```
<TEXT AREA NAME="name_of_control" [ROWS=rows] [COLS=columns]>
Default_text
</TEXT_AREA>
```

Example

```
Comments<BR>
<TEXTAREA NAME="comments" ROWS=3 COLS=40>Enter your comments here.
</TEXTAREA>
```

Appears in the browser as

A screenshot of a web browser displaying a text area. The label "Comments" is at the top left. Below it is a multi-line text box containing the text "Enter your comments here.". The text box has a vertical scrollbar on its right side. The entire element is set against a light gray background.

3. Check Boxes

Check boxes provide a simple yes/no choice.

Syntax

```
<INPUT TYPE="CHECKBOX" NAME="name_of_control" VALUE="Value" [CHECKED]>
```

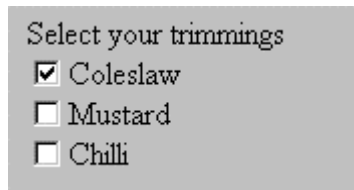
Notes

1. The CHECKED option allows you to pre-select a check box.
2. If the checkbox is ticked then *name_of_control=value* is returned to the server as part of the query string. If the control is left unchecked then nothing is returned.

Example

```
Select your trimmings<BR>
<INPUT TYPE="CHECKBOX" NAME="coleslaw" VALUE="Yes" CHECKED> Coleslaw<BR>
<INPUT TYPE="CHECKBOX" NAME="mustard" VALUE="Yes"> Mustard<BR>
<INPUT TYPE="CHECKBOX" NAME="chilli" VALUE="Yes"> Chilli<BR>
```

This appears in the browser as



Select your trimmings

☒ Coleslaw

☐ Mustard

☐ Chilli

Note that Coleslaw has been pre-selected since the CHECKED option has been added in the definition for this control.

4. Radio Buttons

Radio buttons are similar to check boxes except that only one of the radio buttons can be selected at any one time. Radio buttons are grouped by giving each one the same control name.

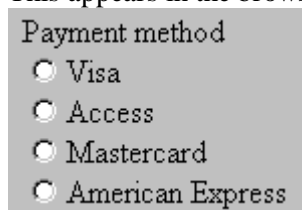
Syntax

```
<INPUT TYPE="RADIO" NAME="name_of_control" VALUE="value" [CHECKED]>
```

Example

```
Payment method<BR>
<INPUT TYPE="RADIO" NAME="pay_by" VALUE="VISA"> Visa<BR>
<INPUT TYPE="RADIO" NAME="pay_by" VALUE="ACCESS"> Access<BR>
<INPUT TYPE="RADIO" NAME="pay_by" VALUE="MASTERCARD"> Mastercard<BR>
<INPUT TYPE="RADIO" NAME="pay_by" VALUE="AMEX"> American Express
```

This appears in the browser as



Payment method

☐ Visa

☐ Access

☐ Mastercard

☐ American Express

5. Drop down Menus

Drop down menus allow the user to select one or more items from a list.

Syntax

```
<SELECT NAME="name_of_control">
<OPTION [VALUE=value1] [SELECTED]>Option 1
<OPTION [VALUE=value2] [SELECTED]>Option 2
<OPTION [VALUE=value3] [SELECTED]>Option 3
...
<OPTION [VALUE=valuen] [SELECTED]>Option n
</SELECT>
```

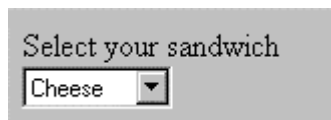
Notes

1. If The value parameter is missed out then the value returned with the control name is either *Option 1* or *Option 2* etc.
2. The optional SELECTED parameter allows you to pre-select an option.

Example

```
Select your sandwich<BR>
<SELECT NAME="sandwich">
<OPTION>Cheese
<OPTION>Beef
<OPTION>Chicken
<OPTION>Tuna
</SELECT>
```

This appears in the browser as



6. List Box

A list box functions in a similar manner to a drop down menu. The syntax is also similar.

Syntax

```
<SELECT NAME="name_of_control" [SIZE="size"] [MULTIPLE]>
<OPTION [VALUE=value1][SELECTED]>Option 1
<OPTION [VALUE=value2] [SELECTED]>Option 2
<OPTION [VALUE=value3] [SELECTED]>Option 3
...
<OPTION [VALUE=value4] [SELECTED]>Option n
</SELECT>
```

Notes

1. The SIZE parameter specifies how many options are visible at any one time. If SIZE is omitted or set to 1 then the control becomes a drop-down menu as described above.
2. The MULTIPLE option gives the user the choice of selecting more than one option by holding down the Control key whilst clicking options.

Example

```
Select your bread<BR>
<SELECT NAME="bread" SIZE="3">
<OPTION>White Sliced
<OPTION>Baguette
<OPTION>Rye
<OPTION>Ciabatta
<OPTION>Hovis
</SELECT>
```

This appears in the browser as



7. Submit and Reset Buttons

The submit button sends the form data to the server and is clicked on completion of the form. The Reset button clears any information entered and resets all controls to their default values.

Syntax

```
<INPUT TYPE="SUBMIT" [NAME="name_of_control"] [VALUE="button_text"]>
<INPUT TYPE="RESET" [VALUE="button_text"]>
```

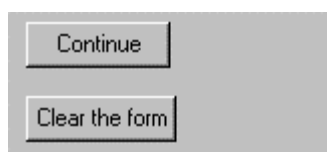
Notes

1. The option to name a submit button is useful when you wish to have multiple submit buttons. The control name and value pair can be checked by the Active Server Page to see which submit button was pressed.

Example

```
<INPUT TYPE="SUBMIT" VALUE="Continue"><P>
<INPUT TYPE="RESET" VALUE="Clear the form">
```

This appears in the browser as



8. Hidden Controls

Hidden fields do not appear on the form and their values cannot be modified by the user. They are however very useful in Web applications when we need to pass data from page to page. Since the web server has no record of your previous choices you can use hidden fields, generated by the Active Server Page, to hold values of variables.

Syntax

```
<INPUT TYPE="HIDDEN" NAME="name_of_control" VALUE="value">
```


4 How Active Server Pages handle HTML form data

4.1 The Request Object

As discussed in the previous section the web browser sends the form data in the form of a query string. The query string is parsed automatically by ASP and stored in the *Request* object.

The form control values can be accessed within VBScript using

Request("control_name")

Example 1

The following is an HTML form, *personal.html*,

```
<HTML>
<FORM METHOD="GET" ACTION="personal.asp">
<B>Surname</B><BR>
<INPUT TYPE="TEXT" NAME="surname" MAXLENGTH="30"><P>
<B>Gender</B><BR>
<INPUT TYPE="RADIO" NAME="gender" VALUE="1"> Male<BR>
<INPUT TYPE="RADIO" NAME="gender" VALUE="2"> Female<P>
<B>Age</B><BR>
<SELECT NAME="age">
<OPTION>Under 25
<OPTION>25 to 35
<OPTION>36 to 45
<OPTION>Over 46
</SELECT>
<P>
<INPUT TYPE="SUBMIT" VALUE="Continue"><P>
<INPUT TYPE="RESET" VALUE="Clear">
</FORM>
</BODY>
</HTML>
```

Which will appear as



Surname
smith

Gender
☒ Male
☐ Female

Age
Under 25 ▼

Continue

Clear

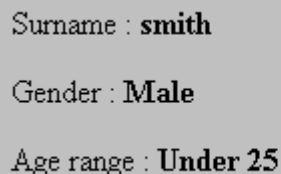
This is the Active Server Page it calls, *personal.asp*

```
<HTML>
<BODY>
Surname : <B><% =Request("surname") %></B><P>

<%If Request("gender")=1 Then%>
    Gender : <B>Male</B><P>
<%Else%>
    Gender : <B>Female</B><P>
<%End If%>

Age range : <B><% =Request("age") %></B>
</BODY>
</HTML>
```

The HTML generated by the Active Server Page will look like



Surname : **smith**

Gender : **Male**

Age range : **Under 25**

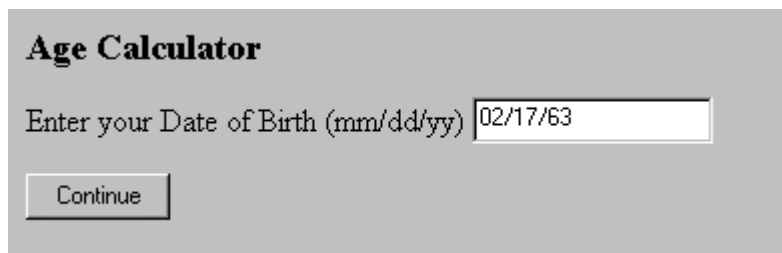
The Active Server Page simply returns the values to the user. A small piece of processing is required. Since the gender is sent as *1* for *male* and *2* for *female* the Active Server Page uses an *If...Then* statement to send the appropriate text. Note that the control name needs to be in quotations when we reference their values.

Example 2

This example involves some processing of the form data. The HTML form, *age.html*, is as follows

```
<HTML>
<TITLE>Age Form</TITLE>
<H3>Age Calculator</H3>
<FORM METHOD="POST" ACTION="age.asp">
Enter your Date of Birth (mm/dd/yy)
<INPUT TYPE="TEXT" NAME="dob"><P>
<INPUT TYPE="SUBMIT" VALUE="Continue">
</FORM>
<BODY>
</BODY>
</HTML>
```

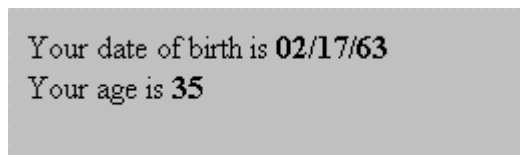
The HTML form appears as



The Active Server Page it calls, *age.asp*, is as follows

```
<HTML>
<BODY>
<%
dob=Request("dob")
days=datediff("d",dob,now())
age=int(days/365)
%>
Your date of birth is <B><%=dob %></B><BR>
Your age is <B><%=age%></B>
</BODY>
</HTML>
```

The HTML generated by the Active Server Page will appear as



In this example we are using the built-in *datediff* function to subtract the date of birth from the current date. Then we use the built-in function *int* to calculate the integer value of the number of days divided by 365 to obtain the age. Note that VBScript functions only recognises the US style of date format i.e. mm/dd/yy

4.2 Self-calling Active Server Pages

You have a choice as to how you use Active Server Pages with HTML forms. You can do as we have done in the previous two examples and have a static HTML page call an Active Server Page using the FORM tag. Alternatively you can have one ASP call another. Another option is to have an ASP call itself.

The following is an example of an self-calling ASP, *newage.asp*

```
<% If Request("dob") <> "" Then
%>
    <HTML>
    <BODY>
    <%
        dob=Request("dob")
        days=datediff("d",dob,now())
        age=int (days/365)
    %>
    Your date of birth is <B><% = dob %></B><BR>
    Your age is <B><%=age%></B>
    </BODY>
    </HTML>

<% Else %>

    <HTML>
    <TITLE>Age Form</TITLE>
    <H3>Age Calculator</H3>
    <FORM METHOD="POST" ACTION="newage.asp">
    Enter your Date of Birth (mm/dd/yy)
    <INPUT TYPE="TEXT" NAME="dob">
    <INPUT TYPE="SUBMIT" VALUE="Continue">
    </FORM>
    <BODY>
    </BODY>
    </HTML>

<% End If %>
```

The ASP takes one of two directions depending on the value of the *dob* control. If the *dob* box is not empty (i.e. <> "") then it will calculate the age and output it. If the *dob* box is empty, as it will be when the page is first executed, then the Active Server Page simply outputs the blank form which has *newage.asp* as the program it calls.

The advantage of using self-calling Active Server Page is compactness. i.e. we have one file instead of two. It is possible to have Active Server Page which call themselves or other Active Server Pages which then call the original Active Server Page.

5 Using the Session Object to Maintain State

Web pages are stateless, the user does not *log on* to the server. A page is requested via the browser, the server delivers the page and the transaction is complete. However, we may wish to keep values entered in HTML forms for use by subsequent pages. These values could represent personal preferences or a username, password combination, say.

One way of doing this is by using hidden fields in a HTML form. Values can be stored in a hidden field and passed from page to page. This method is demonstrated by the examples `hidden1.asp`, `hidden2.asp` and `hidden3.asp` (see online examples). This method although effective is somewhat cumbersome. An alternative is to use the ASP *session object*.

Say for example a HTML form has fields for `student_ID` and `password`. We can extract the entered values using the request object and store them in the session object. For example,

```
Session("student_ID")=Request("student_ID")
Session("password")=Request("password")
```

The session variables are then available to other Active Server Pages in the site. For example to reference the student ID value we use

```
student_ID=Session("Student_ID")
```

Session variables are available as long as the session is active. The session will be closed after 20 minutes of inactivity.

An example of storing personal preferences using the session object can be seen in the pages `session1.asp`, `session2.asp`, `session3.asp` (see online examples).

6 How Active Server Pages interface with databases

Active Server Pages interface with databases via ODBC (Object Database Connectivity). ODBC, which is described in more detail in a later section, is a protocol which allows applications to interface with databases. ODBC runs on a variety of platforms and is used by numerous applications and database systems and is therefore a widely used standard.

ODBC sits between the application (in our case an Active Server Page) and the database. The database is registered with ODBC using a *Data Source Name* (DSN) which is used by the application to refer to that database. Only ODBC needs to name the type of database and its location, the application merely needs to know its DSN.

We can use VBScript to pass commands to the database via ODBC. These commands use the Structured Query Language (SQL) to query the database. SQL is a standard language which can be used for querying most databases. We can use SQL to search, update and delete records from a database.

SQL is very powerful and SQL commands can get quite lengthy when used for querying complex relational databases. The following section merely covers the basics of SQL.

7 SQL

This section contains a brief description of Structured Query Language (SQL) and the syntax of some basic SQL statements required for the rest of this course.

7.1 What is SQL?

SQL is a standard database querying language which is used across a wide variety of database systems. Most database systems can be interrogated using SQL.

7.2 SQL basics

The four SQL commands we are going to concentrate on are SELECT, INSERT, UPDATE and DELETE.

7.2.1 Select

The select statement allows us to query the database tables. Say, for example, we had the following table, *students*, containing student information

reg_number	surname	forename	email	degree_code
971666	Strickland	Sarah	MSRX7SS@man.ac.uk	6540
981234	Knight	Dave	MSRXDK@man.ac.uk	6500
981235	Smith	Steve	MSRXSS2@man.ac.uk	6500
981237	Mohammed	Ali	MSRXMA2@man.ac.uk	6500
981239	Faye	Sally	MSRXSF@man.ac.uk	2550
986533	Dokic	Barry	MSRX7BD@man.ac.uk	6500
986537	Akbar	Tariq	MSRX7AT@man.ac.uk	2550

All of the fields contain text data apart from reg_number which contains integer values.

The following SQL query

```
SELECT surname,email FROM students
```

would return

surname	email
Strickland	MSRX7SS@man.ac.uk
Knight	MSRXDK@man.ac.uk
Smith	MSRXSS2@man.ac.uk
Mohammed	MSRXMA2@man.ac.uk
Faye	MSRXSF@man.ac.uk
Dokic	MSRX7BD@man.ac.uk
Akbar	MSRX7AT@man.ac.uk

We can be more selective about the data returned by using the WHERE clause. For example

```
SELECT surname FROM students WHERE degree_code='6500'
```

would return

surname
Knight
Smith
Dokic
Mohammed

Note that since *degree_code* is text data we need to supply single quotes around the degree code value.

We can further refine the query by using logical operators in the WHERE clause. For example

```
SELECT surname FROM students WHERE degree_code='6500' OR surname='Faye'
```

would return

surname
Knight
Smith
Faye
Dokic
Mohammed

When searching for textual data we do not need to specify an exact match.. For example

```
SELECT * FROM students WHERE forename LIKE 'sa*'
```

would return

reg_number	surname	forename	email	degree_code
981239	Faye	Sally	MSRXSF@man.ac.uk	2550
971666	Strickland	Sarah	MSRX7SS@man.ac.uk	6540

Note by using the * after SELECT we can return all the fields.

7.2.2 Insert

We can use the INSERT statement to add records into a table. We need to provide the field names and their associated values. For example

```
INSERT INTO students (reg_number,surname,forename,degree_code)
VALUES (979924,'Blair','Tony','6500')
```

would add an extra record to the table. Note that the textual data needs to be enclosed in single quotes. Also note we have not added any data for the *email* field.

7.2.3 Update

The UPDATE statement allows us to modify existing records in a table.

For example

```
UPDATE students SET forename="Peter" WHERE reg_number=981234
```

would change the forename of a specified student.

7.2.4 Delete

The delete statement allows us to delete records from a table. For example

```
DELETE FROM students WHERE degree_code='2550'
```

would delete two records from our table.

8 Configuring ODBC

Active Server Pages interact with databases via Object Database Connectivity (ODBC). As explained earlier, ODBC is simply a cross-platform method for allowing applications to interface with databases. The application connects to the database via ODBC and then sends the SQL query to ODBC.

To use ODBC we need to make our database known to ODBC. We do this by registering a database file in ODBC with a Data Source Name.

8.1 *Creating an ODBC Data Source Name*

1. From the Windows **Control Panel** double-click the **ODBC** icon.
2. Select the **System DSN** tab.
3. Click the **Add...** button
4. You are then prompted for a database driver. If you are using an Access database then select **Microsoft Access Driver (*.mdb)**. Click **Finish**.
5. Next to enter a **Data Source Name**. You can optionally provide a **Description**.
6. The next stage is to provide a database. Click the **Select...** button.
7. Use the file window to select a database name then **OK**.
8. Once you have provided a DSN and a database name click **OK** twice.

We can now refer to the database in an Active Server Page simply by using the DSN. We do not need to know the location of the database file or the even the type of database - all that is handled by ODBC.

9 Connecting to a database using Active Server Pages

The following four steps will allow us to query a database

1. Create instance of database component

```
Set instance_name = Server.CreateObject("ADODB.Connection")
```

We will use *instance_name* throughout when communicating with the database.

2. Open a connection to the database

```
instance_name.Open "data_source_name"
```

where *data_source_name* is the DSN as specified in ODBC

3. Create an Instance of the Recordset

```
Set recordset = Server.CreateObject("ADODB.recordset")
```

The recordset is where we are going to store the results of the query. It will contain a number of database records depending on the result of the SQL query.

4. Execute the SQL statement and store the result

```
recordset.Open query_name, instance_name, 3
```

where *query_name* is a string containing the SQL query.

The **3** is a parameter which specifies the *cursortype*. This specifies how we execute the query, an explanation of the options for this parameter are beyond the scope of this course.

As an example we are going to query an Access database of CDs. The database has a single table, *CDs*, with the following format

ID	Title	Artist	Format
1	Second Coming	Stone Roses,	CDA
2	Singles	Smiths	CDA
3	Dummy	Portishead	CDA
4	Revolver	Beatles, The	CDA
5	The Times They Are A-	Dylan, Bob	CDA
6	Love Spreads	Stone Roses,	CDS

etc.

We have registered the database with ODBC, using the Data Source Name *records*.

The following Active Server Page code, will execute a simple query on the database

```
<%  
Query="Select Title,Artist from CDs ORDER by Artist"  
Set DataConn = Server.CreateObject("ADODB.Connection")  
DataConn.Open "records"  
Set RSlist = Server.CreateObject("ADODB.recordset")  
RSlist.Open Query,DataConn,3  
%>
```

We have used *DataConn* as a name for the database component instance, and *RSlist* as the name for our results recordset.

As it stands the code executes the Query and stores it in *RSlist*, which will contain a number of records. The next stage is to actually output the data to HTML. We can do this by appending the following code

```
<HTML>  
<BODY>  
<%  
Do While Not RSlist.EOF  
    %>  
    <%=RSlist("Title")%>, <%=RSlist("Artist")%><BR>  
    %>  
    RSlist.MoveNext  
Loop  
%>  
</BODY>  
</HTML>
```

The code loops through each entry in the recordset. It does this using a Do While...Loop construction. The loop is repeated until End Of File (EOF) is reached in the recordset i.e. we have gone past the last record in the recordset. This is done by checking the state of the EOF property of the recordset. This has a value *True* when the End of File has been reached. The MoveNext function moves onto the next record in the record set. We can access the values of individual fields in the recordset by using

Recordset_name("field_name")

The output from this will loop like

```
Moon Safari, Air  
The Prime Of, Andy, Horace  
Richard D. James Album, Aphex Twin  
Donkey Rhubarb, Aphex Twin  
Come to Daddy, Aphex Twin  
Classics, Aphex Twin  
Girl/Boy ep, Aphex Twin
```

etc.

It is useful to place the output in a HTML table for easier viewing of the results. This can be done by the following code

```
<HTML>
<BODY>
<TABLE BORDER=1 CELLPADDING=0>
<TR><TD><B>Title</B></TD><TD><B>Artist</B></TD></TR>
<%
Do While Not RSlist.EOF
    %>
    <TR>
    <TD><%=RSlist("Title")%></TD>
    <TD><%=RSlist("Artist")%></TD>
    </TR>
    <%
    RSlist.Movenext
Loop
%>
</TABLE>
</BODY>
</HTML>
```

This will produce

Title	Artist
Moon Safari	Air
The Prime Of	Andy, Horace
Richard D. James Album	Aphex Twin
Donkey Rhubarb	Aphex Twin
Come to Daddy	Aphex Twin
Classics	Aphex Twin
Girl/Boy ep	Aphex Twin

etc.

It is also useful to give the number of records returned. This can be achieved by inserting the following line of code

```
<BODY>
Your search returned <B><%=RSlist.RecordCount%></B> records.<BR>
<TABLE BORDER=1>
```

This code outputs the value of the RecordCount property of the recordset. The output will look something like

Your search returned 186 records.	
Title	Artist
Moon Safari	Air
The Prime Of	Andy, Horace
Richard D. James Album	Aphex Twin
Donkey Rhubarb	Aphex Twin

10 Querying Databases using HTML forms

The Active Server Page we have developed so far for querying the database does not offer much flexibility; every record in the database is returned. We need use HTML forms to give the user more choice.

10.1 Generating a Query from HTML form responses

10.1.1 Text box and WHERE

This first example allows the user to generate a simple query based upon the value of some text entered into a form. This can be done in a number of ways - as explained earlier we can use a self-calling Active Server Page to generate both the HTML form and the output. However in this case, to keep the listings to a reasonable size, we are going to use a static HTML page to call an Active Server Page.

First the static HTML page, *artist.html*

```
<HTML>
<FORM METHOD="GET" ACTION="artist.asp">
<H3>CD Search</H3>
Enter an artist:<BR>
<INPUT TYPE="TEXT" NAME="artist">
<P><INPUT TYPE="SUBMIT" VALUE="Search">
</HTML>
```

This HTML form calls the Active Server Page, *artist.asp*. Since they both reside in the same directory we do not need to specify more than the filename of the Active Server Page in the ACTION parameter. We have created a text box, *artist*, which we will use to pass the search term to the Active Server Page.

The next stage is to construct the SQL query for this search. In SQL it will be

```
SELECT Title,Artist from CDs WHERE Artist LIKE '%smith%' ORDER BY Artist
```

This is assuming the user wishes to use *smith* as the search term. The ORDER BY clause sorts the returned database records alphabetically.

We can use VBScript string concatenation to construct the string which we will pass to the ODBC driver. The Active Server Page, *artist.asp*, will look like

```
<%
Query = "SELECT Title,Artist from CDs WHERE Artist LIKE '%"
Query = Query & Request("artist") & "%'"
Query = Query & " ORDER BY Artist"
Set DataConn = Server.CreateObject("ADODB.Connection")
DataConn.Open "records"
Set RSlist = Server.CreateObject("ADODB.recordset")
RSlist.Open Query,DataConn,3
%>
<HTML>
Your search returned <B><%=RSlist.RecordCount%></B> records.<BR>
<TABLE BORDER=1>
<TR><TD><B>Title</B></TD><TD><B>Artist</B></TD></TR>
<%
Do While Not RSlist.EOF
    %>
    <TR>
    <TD><%=RSlist("Title")%></TD>
    <TD><%=RSlist("Artist")%></TD>
    </TR>
    <%
    RSlist.Movenext
Loop
%>
</TABLE>
<A HREF="artist.html">Return to form</A>
<HR><SMALL><%=Query%></SMALL>
</HTML>
```

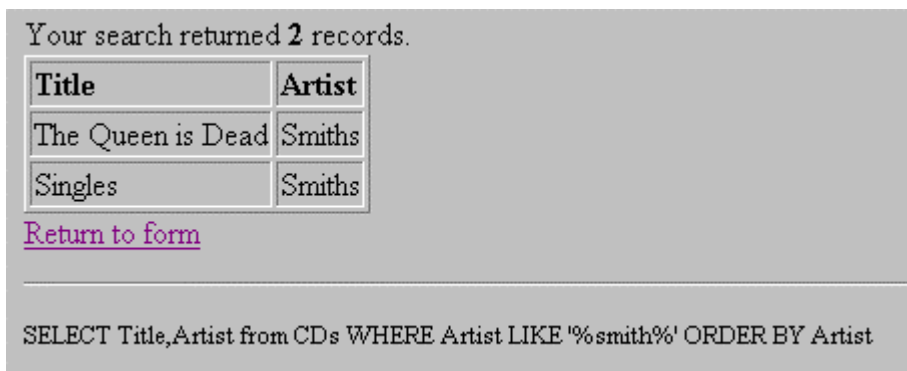
The HTML form will look like



CD Search

Enter an artist:

And the HTML generated by our Active Server Page will look like



Your search returned **2** records.

Title	Artist
The Queen is Dead	Smiths
Singles	Smiths

[Return to form](#)

SELECT Title,Artist from CDs WHERE Artist LIKE '%smith%' ORDER BY Artist

Note that since we have used the LIKE term in the SQL the search picked up *smiths* from the search term *smith*.

In the output we have included the actual SQL query which was executed. This can be useful for debugging purposes and can be removed when the application is released.

10.1.2 Providing more choice - Radio button and WHERE

As a further refinement to our search we can use radio buttons to dictate the search. For example we may want to give the user the choice of viewing albums, singles or both. The following HTML form and Active Server Page does that.

The html page, *format.html*, looks like

```
<HTML>
<FORM METHOD="GET" ACTION="format.asp">
<H3>CD Search</H3>
Enter an artist:<BR>
<INPUT TYPE="TEXT" NAME="artist">
<P>Select format<BR>
Album <INPUT TYPE="RADIO" NAME="FORMAT" VALUE="Album">
Single <INPUT TYPE="RADIO" NAME="FORMAT" VALUE="Single">
Both <INPUT TYPE="RADIO" NAME="FORMAT" VALUE="Both" CHECKED>
<P><INPUT TYPE="SUBMIT" VALUE="Search">
</HTML>
```

Here we have added three radio buttons, grouped under the name *format*, to allow the user to select a CD format.

We need to vary the SQL query depending on the user's choice of radio button. For example, if the user were to select the album format then the SQL would read

```
SELECT Title,Artist,Format from CDs WHERE Artist LIKE '%artist%' AND
format='CDA' ORDER BY Artist
```

We can do this by using an If..Then...Elseif statement to add an extra part to the WHERE clause.

The Active Server Page, *format.asp*, starts as

```
<%
format=Request("format")
Query="SELECT Title,Artist,Format from CDs WHERE Artist LIKE '%"
Query=Query & Request("artist") & "%'"
If format="Album" Then
    Query=Query & " AND format='CDA'"
ElseIf format="Single" Then
    Query=Query & " AND format='CDS'"
End If
Query = Query & " ORDER BY Artist"
```

If *both* has been selected then no addition need to be made to the SQL query string. The rest of the Active Server Page will execute the query and output the results in a table as in the previous example.

The HTML page would look like

CD Search

Enter an artist:

Select format
Album ☒ Single ☐ Both ☐

And the HTML generated by the Active Server Page would look like

Title	Artist	Format
Selected Ambient Works 85-92	Aphex Twin	CDA
Classics	Aphex Twin	CDA
Richard D. James Album	Aphex Twin	CDA
Treasure	Cocteau Twins	CDA

[Return to form](#)

```
SELECT Title,Artist,Format from CDs WHERE Artist LIKE '%twin%' AND format='CDA' ORDER BY Artist
```

The format has only been included in the output to prove that query has been successful.

10.1.3 Populating Controls with Query Results - Drop down menu and WHERE

We can use the results from an SQL query to fill a drop-down box. The following example uses a database of books. The database, with DSN of *books*, has two tables.

The main table, *books*, has the following format

ID	Title	Surname	Subject_ID	Year	Size	Price
35	A study of history	Richardson	3	1988	240x170	
377	An American quarter century	Davies	1	1995	240x170	£10.99
44	Ancient Greece and Rome	Hopwood	3	1995	240x170	£15.99
355	Approaches to popular film	Hollows	4	1995	216x138	
295	Art and Antichrist in Medieval	Muir wright	2	1995	240x170	£14.99

The table, subjects, has the format

Subject_ID	Subject
1	American Politics
2	Art History and Theory
3	Bibliographies
4	Film
5	Hispanic texts

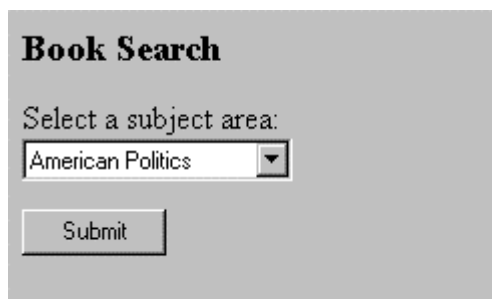
The two tables are linked on Subject_ID.

The task is to create a Web interface which will allow the user to select a subject and be presented with a list of the books on that subject. Since new subjects are being added to the database all the time then the list of subjects given to the user must be dynamic.

The first stage is to create an Active Server Page, *subject.asp* which generates a HTML form

```
<%
Query = "SELECT subject,subject_ID from subjects ORDER BY subject"
Set DataConn = Server.CreateObject("ADODB.Connection")
DataConn.Open "books"
Set RSlist = Server.CreateObject("ADODB.recordset")
RSlist.Open Query,DataConn,3
%>
<HTML>
<FORM METHOD="GET" ACTION="list.asp">
<H3>Book Search</H3>
Select a subject area:<BR>
<SELECT NAME="subject">
<%
    Do While Not RSlist.EOF
    %>
    <OPTION VALUE="<%=RSlist("subject_ID")%>"><%=RSlist("subject")%>
    <%
    RSlist.MoveNext
Loop %>
</SELECT>
<P><INPUT TYPE="SUBMIT">
</FORM>
</HTML>
```

subject.asp generates HTML which looks like



Clicking Submit sends a value, a subject area code, from the drop-down menu to *list.asp*, which performs a SELECT based upon this value. This is *list.asp*

```
<%
Query="SELECT * FROM books WHERE subject_ID=" & Request("subject")
Query = Query & " ORDER BY title"
Set DataConn = Server.CreateObject("ADODB.Connection")
DataConn.Open "books"
Set RSlist = Server.CreateObject("ADODB.recordset")
RSlist.Open Query,DataConn,3
%>
<HTML>
<TABLE BORDER=1>
<TR>
<TD><B>Title</B></TD><TD><B>Surname</B></TD>
</TR>
<%
Do While Not RSlist.EOF
    %>
    <TR>
    <TD><%=RSlist("Title")%></TD>
    <TD><%=RSlist("surname")%></TD>

    </TR>
    <%
    RSlist.Movenext
Loop
%>
</TABLE>
<A HREF="subject.asp">Return to form</A>
<HR><SMALL><%=Query%></SMALL>
</HTML>
```

The HTML generated by *list.asp* looks like

Title	Surname
An American quarter century	Davies
Congress and the Presidency	Foley
Elections USA	Davies
Governing America	Hames
Raw judicial power?	McKeever
The United States Supreme Court	McKeever

[Return to form](#)

SELECT * FROM books WHERE subject_ID=1 ORDER BY title

10.1.4 "Drill-Down"

In the preceding example the user was presented with a summary (title and surname) of each book in a particular subject area. What if they want to get the full details of a particular title? We can provide this option by using *drill-down*.

First we have to make changes to our `subject.asp` and `list.asp` programs

File	Function
<code>newssubject.asp</code>	creates a HTML form with a drop down menu of book subjects. Form calls <i>newlist.asp</i>
<code>newlist.asp</code>	returns a list of books in a particular subject area. Clicking on a book title sends the user to <code>details.asp</code>
<code>bookdetails.asp</code>	provides details on a specied book

The only change to the Active Server Page which creates the form (*subject.asp*) is to get it to call a different Active Server Page. This can be done by

```
<FORM METHOD="GET" ACTION=" newlist.asp
```

We can rename the modified file *newssubject.asp*

Also the changes to `list.asp` from the previous example is

```
<TD><A HREF="bookdetails.asp?book_ID=<%=RSlist("ID")%>">
    <%=RSlist("Title")%></A></TD>
```

(Note that this is one line)

This line, instead of merely printing out the book title, as in the previous example, turns the title into a hypertext link using the `A HREF` tag. Rather than call a static HTML page, it calls an Active Server Page (*bookdetails.asp*). However, we need to specify more than the name of the Active Server Page, we also need to pass on the ID code of the book. We can do this by creating a query string

```
<A HREF="bookdetails.asp?book_ID=value_of_book_ID">link text</A>
```

The new output will look something like

Title	Surname
An American quarter century	Davies
Congress and the Presidency	Foley
Elections USA	Davies
Governing America	Hames
Raw judicial power?	McKeever
The United States Supreme Court	McKeever
Return to form	

SELECT * FROM books WHERE subject_ID=1 ORDER BY title

The code for *bookdetails.asp* is as follows

```
<%  
Query="SELECT * FROM books WHERE ID=" & Request("book_ID")  
Set DataConn = Server.CreateObject("ADODB.Connection")  
DataConn.Open "books"  
Set RSlist = Server.CreateObject("ADODB.recordset")  
RSlist.Open Query,DataConn,3  
%>  
<HTML>  
<H3><%=RSlist("title")%></H3><P>  
Author surname: <B><%=RSlist("surname")%></B><BR>  
Year: <B><%=RSlist("Year")%></B><BR>  
Size: <B><%=RSlist("Size")%></B><BR>  
Price: <B><%=RSlist("Price")%></B>
```

The query is constructed using the book ID code in the WHERE clause. Since we passed the book ID code in a query string the variable, value pair is accessed via the Request object. Once the query has been executed we can output the details for that particular book. Note that in this case, unlike previous examples, only one record has been returned hence we do not need to cycle through a loop to output the data. The output from *bookdetails.asp*

Congress and the Presidency

Author surname: **Foley**

Year: **1996**

Size: **216x138**

Price: **£10.99**

We can use drill-down to a number of levels. We could for example allow users to click on authors name in the book details and get further details about a particular author.

Using "drill-down" we can allow users to navigate through data giving them the choice of what to look at and displaying the minimum amount of information unless more is required.

11 Modifying Database Records

As well as querying a database it is also possible to make changes to a database using Active Server Pages. The changes we can make include adding records, deleting records and modifying records.

11.1 Adding Records to a Database

If we return to the *records* database and remind ourselves of the table *CDs* and its format

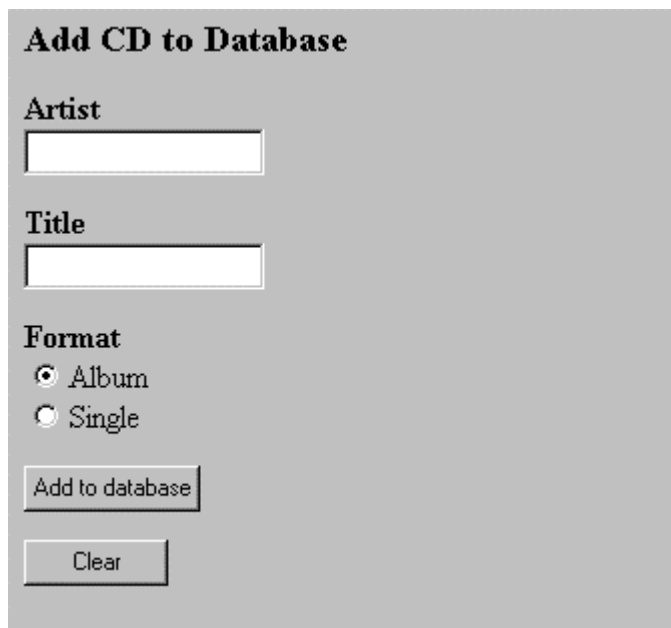
ID	Title	Artist	Format
1	Second Coming	Stone Roses,	CDA
2	Singles	Smiths	CDA
3	Dummy	Portishead	CDA
4	Revolver	Beatles, The	CDA
5	The Times They Are A-	Dylan, Bob	CDA
6	Love Spreads	Stone Roses,	CDS

We wish to create a Web form that allows us to add records to the table.

The HTML form, *addcds.html*, is straightforward

```
<HTML>
<FORM METHOD="GET" ACTION=" addcds.asp ">
<H3>Add CD to Database</H3>
<B>Artist</B><BR>
<INPUT TYPE="TEXT" NAME="artist" MAXLENGTH="50"><P>
<B>Title</B><BR>
<INPUT TYPE="TEXT" NAME="title" MAXLENGTH="50"><P>
<B>Format</B><BR>
<INPUT TYPE="RADIO" NAME="format" VALUE="CDA" CHECKED> Album<BR>
<INPUT TYPE="RADIO" NAME="format" VALUE="CDS"> Single<P>
<INPUT TYPE="SUBMIT" VALUE="Add to database"><P>
<INPUT TYPE="RESET" VALUE="Clear">
</FORM>
</BODY>
</HTML>
```

The form would look like



Add CD to Database

Artist

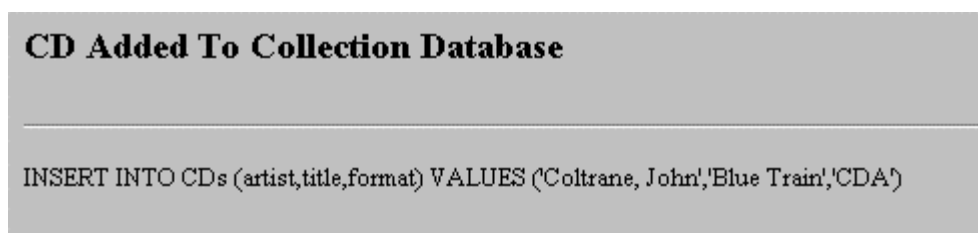
Title

Format
☒ Album
☐ Single

The Active Server Page it calls, *addcds.asp*, is as follows

```
<%  
artist=Request("artist")  
title=Request("title")  
format=Request("format")  
Query = "INSERT INTO CDs (artist,title,format) VALUES ("  
Query = Query & "'" & artist & "'," & title & "'," & format & "'" & ""  
Set DataConn = Server.CreateObject("ADODB.Connection")  
DataConn.Open "records"  
Set RSlist = Server.CreateObject("ADODB.recordset")  
RSlist.Open Query,DataConn,3  
%>  
<HTML>  
<BODY>  
<H3>CD Added To Collection Database</H3>  
<HR>  
<SMALL><%=Query%></SMALL>  
</BODY>  
</HTML>
```

On execution we would get the following HTML generated



CD Added To Collection Database

INSERT INTO CDs (artist,title,format) VALUES ('Coltrane, John','Blue Train','CDA')

Explanation of code

We use the SQL UPDATE statement as follows

```
INSERT INTO records (artist,title,format) VALUES  
( 'value_of_artist', 'Value_of_title', 'Value_of_format' )
```

Since all three fields are of data type text then we must enclose the values in single quotes.

The query is executed in the same way as we execute a SELECT query. Note however the username the Web server runs under must have write access to the database file.

11.2 Handling Errors

Errors are more likely to occur when executing an UPDATE query than in a SELECT query because there is more scope of user error. For example, if we were to omit the artist name when submitting the addcds.html form we would get the following error

```
Microsoft OLE DB Provider for ODBC Drivers error '80004005'  
  
[Microsoft][ODBC Microsoft Access 97 Driver] Field 'CDs.Artist' can't be a zero-length string.  
  
/coursesamples/addcds.asp, line 10
```

Which may make sense to the Active Server Page programmer but it is not sufficient for the average Web user.

We can check for errors by making the following modifications to our code. The new Active Server Page, *erroradd.asp*

```
<%
On Error Resume Next
this_page="erroradd.asp"
artist=Request("artist")
title=Request("title")
format=Request("format")
Query = "INSERT INTO CDs (artist,title,format) VALUES ("
Query = Query & "'" & artist & "'," & title & "'," & format & "'"
Set DataConn = Server.CreateObject("ADODB.Connection")
DataConn.Open "records"
Set RSlist = Server.CreateObject("ADODB.recordset")
RSlist.Open Query,DataConn,3
If Err <> 0 Then
    %>
    <HTML>
    <BODY>
    <H3>An Error Occured</H3>
    Make a note of the following information and report it to Dave Smith
    x3343<P>
    SQL query:<BR>
    <B><%=Query%></B><P>
    Occured on page:<BR>
    <B><%=this_page%></B><P>
    Error Message:<BR>
    <B><%=Err.Description%></B>

    </BODY>
    </HTML>

<% Else %>

    <HTML>
    <BODY>
    <H3>CD Added To Collection Database</H3>
    </BODY>
    </HTML>

<%End If%>
```

The extra code is in bold. The first statement

```
On Error Resume Next
```

tells the Active Server Page what to do when an error occurs. Rather than a fatal end to the program execution we can control events. In this case when the error occurs the Active Server Page continues with the execution of the next line (hence Resume Next). Here the error occurs when we try and execute the SQL query

```
RSlist.Open Query,DataConn,3
```

When an error occurs whilst executing this line program control resumes with the following line

```
If Err <> 0 Then
```

Which is the start of the error handling section. This section gives the user some information as to what to do when the error occurs. The output would look like

An Error Occured

Make a note of the following information and report it to Dave Smith x3343

SQL query:

INSERT INTO CDs (artist,title,format) VALUES ('','sssss','CDA')

Occured on page:

erroradd.asp

Error Message

[Microsoft][ODBC Microsoft Access 97 Driver] Field 'CDs.Artist' can't be a zero-length string.

which is of more use to the user than the previous stark error message. It is normal practice to put your error handling code in a function which could be called from several places within the Active Server Page.

11.3 Validating Data

Although it is useful to include error-handling routines in our code it makes more sense to check for likely sources of error before we get as far as the SQL query execution. We can do this by validating the data before we try to execute the SQL query.

For example we could modify our *addcds.asp* code to make sure the user has entered both an artist and a title. The following is *checkadd.asp*

```
<%
artist=Request("artist")
title=Request("title")
format=Request("format")
If artist="" or title="" Then
    %>
    <HTML>
    <BODY>
    <H3>Error - You must complete all the boxes</H3>
    </BODY>
    </HTML>
<%Else
    Execute update query and display message

<%End If%>
```

In this example we check for null values in either artist or title and print an error message, without executing the query, if either value is null. We could make this code more sophisticated by checking for null values individually and informing the user precisely which value was missing.

11.3.1 Using Client side scripting

As well as sending out HTML code an Active Server Page can send out client-side scripting such as JavaScript. This can be useful when checking for blank boxes because the checking can be done on the client rather than the server which saves time.

11.4 Checking the Query String for Single Quotes

If we return to the *erroradd.html* form and say, for example, we were to enter artist as *O'Conner,Sinead* an error would be generated

```
[Microsoft][ODBC Microsoft Access 97 Driver] Syntax error (missing operator) in query expression 'O'conner'.
```

The error is caused by the SQL query which would be

```
INSERT INTO CDs (artist,title,format) VALUES ( 'O'conner,sinead','Greatest Hits','CDA')
```

The offending section is in bold. The single quote in the artist's name has been interpreted by SQL as a text delimiter expecting it to be one of a pair. To avoid errors we need to force the SQL into treating a single quote within a string as a literal character rather than a text delimiter. We can do this by preceding the offending character with, coincidentally, another single quote.

We can use a function, *CheckString*, to check whether a string inputted by the user contains any single quotes. Here it is as used by *quote.asp*

```
<%  
FUNCTION CheckString (s)  
    pos = InStr(s, "'")  
    Do While pos > 0  
        s = Mid(s, 1, pos) & "'" & Mid(s, pos + 1)  
        pos = InStr(pos + 2, s, "'")  
    Loop  
    CheckString=s  
END FUNCTION  
artist=Request("artist")  
title=Request("title")  
format=Request("format")  
Query = "INSERT INTO CDs (artist,title,format) VALUES ("  
Query = Query & "'" & CheckString(artist) & "','" & CheckString(title) &  
"'','" & format & "'"")  
...  
Execute Query  
...
```

The function uses two built-in functions. *InStr* searches for the existence of a string (in our case a single character) within another string and returns its position in the string. The function returns zero if the string is not found. The *MidStr* function returns a specified number of characters from a string. The arguments passed to *MidStr* are string, starting position, number of characters to return.

11.5 Deleting Records

It is also possible to delete records from a database using an Active Server Page.

The following Active Server Page, *deletecd.asp*, provides the user with a form to select a CD for deletion.

```
<%
Query="SELECT ID,Title,Artist from CDs ORDER BY Artist"
Set DataConn = Server.CreateObject("ADODB.Connection")
DataConn.Open "records"
Set RSlist = Server.CreateObject("ADODB.recordset")
RSlist.Open Query,DataConn,3
%>
<HTML>
<BODY>
<FORM ACTION="deleteme.asp" METHOD="GET">

<INPUT TYPE="SUBMIT" VALUE="Delete Selected CD"><P>

<TABLE BORDER=1>
<TR><TD><B>Title</B></TD><TD><B>Artist</B></TD><TD>Delete</TD></TR>
<%
Do While Not RSlist.EOF
    %>
    <TR>
    <TD><%=RSlist("Title")%></TD>
    <TD><%=RSlist("Artist")%></TD>
    <TD>
    <INPUT TYPE=RADIO NAME="record_ID" VALUE="<%=RSlist("ID")%>">
    </TD>
    </TR>
    <%
    RSlist.Movenext
Loop
%>
</TABLE>
</FORM>
</BODY>
</HTML>
```

When executed the Active Server Page generates the following HTML output

<input type="button" value="Delete Selected CD"/>		
Title	Artist	Delete
Moon Safari	Air	<input type="radio"/>
The Prime Of	Andy, Horace	<input type="radio"/>
Richard D. James Album	Aphex Twin	<input type="radio"/>
Selected Ambient Works 85-92	Aphex Twin	<input type="radio"/>
Girl/Boy ep	Aphex Twin	<input type="radio"/>
...	...	<input type="radio"/>

Next to each entry is a radio button, called *record_ID*. The radio button takes the value of the ID number of the particular CD selected

```
<INPUT TYPE=RADIO NAME="record_ID" VALUE="<%=RSlist("ID")%>">
```

When the Submit button is selected the string **record_ID=ID_number** is passed to *deleteme.asp*

```
<%  
Query="DELETE from CDs WHERE ID=" & Request("record_ID")  
Set DataConn = Server.CreateObject("ADODB.Connection")  
DataConn.Open "records"  
Set RSlist = Server.CreateObject("ADODB.recordset")  
RSlist.Open Query,DataConn,3  
%>  
<HTML>  
<BODY>  
<H3>CD Deleted from Database</H3>  
<A HREF="deletedcd.asp">Return to list</A>  
</BODY>  
</HTML>
```

Which uses the SQL DELETE statement to delete the required record.

11.6 Modifying Records - UPDATE

We can modify a database record using the SQL UPDATE statement. The following example using a string of three Active Server Pages which allow us to make changes to the CD database.

The first Active Server Page, *listmod.asp*

```
<%
Query="SELECT * from CDs ORDER BY Artist"
Set DataConn = Server.CreateObject("ADODB.Connection")
DataConn.Open "records"
Set RSlist = Server.CreateObject("ADODB.recordset")
RSlist.Open Query,DataConn,3
%>
<HTML>
<BODY>
Click on the CD you wish to modify
<TABLE BORDER=1>
<TR><TD><B>Title</B></TD><TD><B>Artist</B></TD></TR>
<%
Do While Not RSlist.EOF
    %>
    <TR>
    <TD>
    <A HREF="changeme.asp?ID=<%=RSlist("ID")%>"><%=RSlist("Title")%></A>
    </TD>
    <TD><%=RSlist("Artist")%></TD>
    </TR>
    <%
    RSlist.Movenext
Loop
%>
</TABLE>
</BODY>
</HTML>
```

displays a list of CDs in the database and prompts the user to select a CD which requires modification. The HTML generated appears as

Title	Artist
Moon Safari	Air
The Prime Of	Andy, Horace
Girl/Boy ep	Aphex Twin
Classics	Aphex Twin
Come to Daddy	Aphex Twin

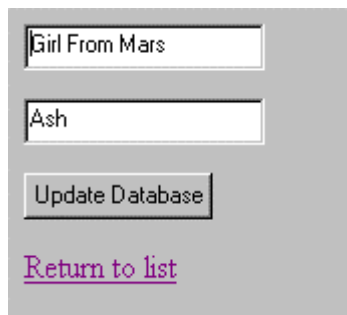
The important line is

```
<A HREF="changeme.asp?ID=<%=RSlist("ID")%>"><%=RSlist("Title")%></A>
```

Which creates a link, which when clicked, calls the page *changeme.asp*, which is as follows

```
<%
ID=Request("ID")
Query="SELECT * from CDs WHERE ID=" & ID
Set DataConn = Server.CreateObject("ADODB.Connection")
DataConn.Open "records"
Set RSlist = Server.CreateObject("ADODB.recordset")
RSlist.Open Query,DataConn,3
artist=RSlist("artist")
title=RSlist("title")
%>
<HTML>
<BODY>
<FORM METHOD="GET" ACTION="update.asp">
<INPUT TYPE="TEXT" NAME="title" VALUE="<%=title%>"><P>
<INPUT TYPE="TEXT" NAME="artist" VALUE="<%=artist%>"><P>
<INPUT TYPE="SUBMIT" VALUE="Update Database"><P>
<INPUT TYPE="HIDDEN"NAME="ID" VALUE="<%=ID%>">
<A HREF="listmod.asp">Return to list</A>
</FORM>
</BODY>
</HTML>
```

The HTML generated appears as



The user can now enter new values for title and artist in the text boxes. On clicking the submit button the new values are sent to *update.asp* which carries out the actual changes. However, as well as sending the new values we also need to send the ID of this particular record. We can do this by using a hidden field.

This is *update.asp*

```
<%
title=Request("title")
artist=Request("artist")
ID=Request("ID")
Query="UPDATE CDs SET title='" & title & "',artist='" & artist & "'"
Query=Query & " WHERE ID=" & ID
Set DataConn = Server.CreateObject("ADODB.Connection")
DataConn.Open "records"
Set RSlist = Server.CreateObject("ADODB.recordset")
RSlist.Open Query,DataConn,3
%>
<HTML>
<BODY>
<H3>Record changed</H3>
<P><A HREF="changeme.asp?ID=<%=ID%>">Check changes</A>
<HR>
<SMALL><%=Query%></SMALL>
</BODY>
</HTML>
```

The HTML it generates appears as



update.asp updates the database using the information provided by *changeme.asp*, and then displays a link back to *changeme.asp* which allows the user to review the changes.

12 Summary

This course has attempted to provide an introduction to Active Server Page technology and in particular how we can use it to generate web interfaces to databases. Given restrictions of time there are aspects such as database security and efficiency that we have not been able to look at but need consideration when developing serious Web/database applications.

13 Index

A

Active Server Page Directory	4
Arrays	9

C

Check boxes, defining	12
CheckString function	42
Client side scripting	42
Comments	7

D

Data Source Name, creating	25
database, Connecting to	26
datediff function	19
Delete, SQL	24
Do While...Loop	8
documentation	4
Drill-Down	35
Drop down menus, defining	14

E

EOF property, recordset	27
-------------------------------	----

H

Hidden controls	16
HTML forms, specification	11

I

If..Then	7
Include	9
Insert, SQL	23
InStr function	42
int function	19

L

Like, SQL	23
List box, defining	14

M

MidStr function	42
-----------------------	----

MoveNext	27
----------------	----

O

ODBC, description	21
On Error Resume Next	40
Order By, SQL	29

R

Radio buttons, defining	13
RecordCount	28
Recordset	26
Request Object	17
Roadmap	5

S

Scripting languages	4
Select, SQL	22
Self-calling Active Server Pages	19
SQL, description	22
Sub	9
Submit and Reset buttons, defining	15

T

Text box, defining	11
Text window, defining	12

U

Update, SQL	24
-------------------	----

V

Validating Data	41
Variables	6
VBScript, overview	6

W

Web servers	4
Where, SQL	23