INTRODUCTION TO ACTIVE SERVER PAGES

By Milan Kothari

Introduction to ASP

An ASP file can contain text, HTML tags and scripts. Scripts in an ASP file are executed on the server

---

What you should already know

- Before you continue you should have some basic understanding of the following:
- WWW, HTML and the basics of building Web pages
- A scripting language like JavaScript or VBScript

What is ASP?

- ASP stands for Active Server Pages
- ASP is a program that runs inside IIS
- IIS stands for Internet Information Services
- IIS comes as a free component with Windows 2000
- IIS is also a part of the Windows NT 4.0 Option Pack
- The Option Pack can be downloaded from Microsoft
- PWS is a smaller - but fully functional - version of IIS
- PWS can be found on your Windows 95/98 CD

ASP Compatibility

- ASP is a Microsoft Technology
- To run IIS you must have Windows NT 4.0 or later
- To run PWS you must have Windows 95 or later
- ChiliASP is a technology that runs ASP without Windows OS
- InstantASP is another technology that runs ASP without Windows

What is an ASP File?

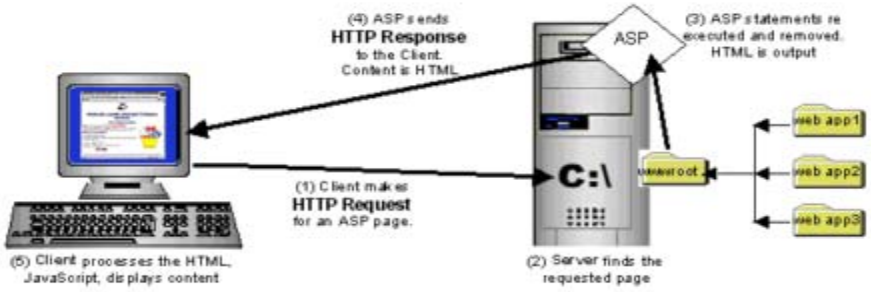
- An ASP file is just the same as an HTML file
- An ASP file can contain text, HTML, XML, and scripts
- Scripts in an ASP file are executed on the server
- An ASP file has the file extension ".asp"

How Does ASP Differ from HTML?

- When a browser requests an HTML file, the server returns the file
- When a browser requests an ASP file, IIS passes the request to the ASP engine. The ASP engine reads the ASP file, line by line, and executes the scripts in the file. Finally, the ASP file is returned to the browser as plain HTML

What can ASP do for you?

- Dynamically edit, change or add any content of a Web page
- Respond to user queries or data submitted from HTML forms
- Access any data or databases and return the results to a browser
- Customize a Web page to make it more useful for individual users
- The advantages of using ASP instead of CGI and Perl, are those of simplicity and speed
- Provides security since your ASP code can not be viewed from the browser
- Since ASP files are returned as plain HTML, they can be viewed in any browser
- Clever ASP programming can minimize the network traffic



The Process of Serving An Active Server Page

- The browser sends a request to the Server for an Active Server Page
- The web server receives the request and recognizes that the request is for an ASP file because the requested file has the extension of .asp
- The Web Server retrieves the proper Active Server Page file from disk or memory.
- The Web Server sends the file to a special program named the ASP.dll
- The ASP.dll processes the Active Server Page from top to bottom. Any internal commands encountered are executed. The result(output) of this process is standard HTML file.
- The (Standard) HTML file is sent back to the browser.

Run ASP on Your PC

You can run ASP on your own PC without an external server. To do that, you must install Microsoft's Personal Web Server (PWS) or Internet Information Server (IIS) on your PC.

How to Run ASP on your own PC

You can run ASP on your own PC without an external server. To do that, you must install Microsoft's Personal Web Server (PWS) or Internet Information Server (IIS) on your PC.

If you are serious about using ASP, you should have at least Windows 98, Second Edition.

If you are really serious about using ASP, you should go for Windows 2000.

How to install PWS and run ASP on Windows 95

Personal Web Server (PWS) is not shipped with Windows 95 !!

To run ASP on Windows 95, you will have to download "Windows NT 4.0 Option Pack" from Microsoft.

How to install PWS and run ASP on Windows NT

Personal Web Server (PWS) is not shipped with Windows NT !!

To run ASP on Windows NT, you will have to download "Windows NT 4.0 Option Pack" from Microsoft.

How to install PWS and run ASP on Windows 98

1. Open the **Add-ons** folder on your Windows98 CD, find the **PWS** folder and run the **setup.exe** file.
2. An **Inetpub folder** will be created on your harddrive. Open it and find the **wwwroot** folder.
3. **Create a new folder**, like "MyWeb", under wwwroot.
4. **Use a text editor** to write some ASP code, save the file as "test1.asp" in the "MyWeb" folder.
5. Make sure your Web server is running - The installation program has added a new icon on your task bar (this is the PWS symbol). Click on the icon and press the Start button in the window that appears.
6. **Open your browser** and type in "http://localhost/MyWeb/test1.asp", to view your first ASP page.

How to install PWS and run ASP on Windows ME

Personal Web Server (PWS) is not included with Windows Me !!

How to install IIS and run ASP on Windows 2000

1. From your **Start Button**, go to **Settings**, and **Control Panel**
2. In the Control Panel window select **Add/Remove Programs**
3. In the Add/Remove window select **Add/Remove Windows Components**
4. In the Wizard window check **Internet Information Services**, **click OK**
5. An **Inetpub folder** will be created on your harddrive
6. Open the Inetpub folder, and find a folder named **wwwroot**

7. **Create a new folder**, like "MyWeb", under wwwroot.
8. **Use a text editor** to write some ASP code, save the file as "test1.asp" in the "MyWeb" folder
9. Make sure your Web server is running - The installation program has added a new icon on your task bar (this is the IIS symbol). Click on the icon and press the Start button in the window that appears.
10. **Open your browser** and type in "http://localhost/MyWeb/test1.asp", to view your first ASP page

---

How to install IIS and run ASP on Windows XP Professional

Note: You cannot run ASP on Windows XP Home Edition.

1. Insert the Windows XP Professional CD-Rom into your CD-Rom Drive
2. From your **Start Button**, go to **Settings**, and **Control Panel**
3. In the Control Panel window select **Add/Remove Programs**
4. In the Add/Remove window select **Add/Remove Windows Components**
5. In the Wizard window check **Internet Information Services**, **click OK**
6. An **Inetpub folder** will be created on your harddrive
7. Open the Inetpub folder, and find a folder named **wwwroot**
8. **Create a new folder**, like "MyWeb", under wwwroot.
9. **Use a text editor** to write some ASP code, save the file as "test1.asp" in the "MyWeb" folder
10. Make sure your Web server is running - The installation program has added a new icon on your task bar (this is the IIS symbol). Click on the icon and press the Start button in the window that appears.
11. **Open your browser** and type in "http://localhost/MyWeb/test1.asp", to view your first ASP page

ASP Syntax

You cannot view the ASP source code by selecting "View source" in a browser, you will only see the output from the ASP file, which is plain HTML. This is because the scripts are executed on the server before the result is sent back to the browser.

In our ASP tutorial, every example displays the hidden ASP source code. This will make it easier for you to understand how it works.

The Basic Syntax Rule

An ASP file normally contains HTML tags, just like an HTML file. However, an ASP file can also contain **server scripts**, surrounded by the delimiters **<%** and **%>**. Server scripts are **executed on the server,** and can contain any expressions, statements, procedures, or operators valid for the scripting language you prefer to use.

The Response Object

The **Write** method of the ASP **Response Object** is used to send content to the browser. For example, the following statement sends the text "Hello World" to the browser:

```
<%
response.write ("Hello World!")
%>
```

VBScript

You may use different scripting languages in ASP files. However, the default scripting language is VBScript:

```
<html>
<body>
<%
response.write("Hello World!")
%>
</body>
</html>
```

The example above writes "Hello World!" into the body of the document.

JavaScript

To set JavaScript as the default scripting language for a particular page you must insert a language specification at the top of the page:

```
<%@ language="javascript"%>
<html>
<body>
<%
Response.Write("Hello World!")
%>
</body>
</html>
```

Note: Unlike VBScript - JavaScript is case sensitive. You will have to write your ASP code with uppercase letters and lowercase letters when the language requires it.

Other Scripting Languages

ASP is shipped with VBScript and JScript (Microsoft's implementation of JavaScript). If you want to script in another language, like PERL, REXX, or Python, you will have to install script engines for them.

Important: Because the scripts are executed on the server, the browser that displays the ASP file does not need to support scripting at all!

ASP Variables

A variable is used to store information.

If the variable is declared outside a procedure it can be changed by any script in the ASP file. If the variable is declared inside a procedure, it is created and destroyed every time the procedure is executed.

Examples

For Examples : Refer -> CD

Lifetime of Variables

A variable declared outside a procedure can be accessed and changed by any script in the ASP file.

A variable declared inside a procedure is created and destroyed every time the procedure is executed. No scripts outside the procedure can access or change the variable.

To declare variables accessible to more than one ASP file, declare them as session variables or application variables.

Session Variables

Session variables are used to store information about ONE single user, and are available to all pages in one application. Typically information stored in session variables are name, id, and preferences.

Application Variables

Application variables are also available to all pages in one application. Application variables are used to store information about ALL users in a specific application.

ASP Procedures

In ASP you can call a JavaScript procedure from a VBScript and vice versa.

Examples

For Examples : Refer -> CD

Procedures

The ASP source code can contain procedures and functions:

```
<html>
<head>
<%
sub vbproc(num1,num2)
response.write(num1*num2)
end sub
%>
</head>
<body>
<p>Result: <%call vbproc(3,4)%></p>
</body>
</html>
```

Insert the <%@ language="*language*" %> line above the <html> tag to write procedures or functions in another scripting language than default:

```
<%@ language="javascript" %>
<html>
<head>
<%
function jsproc(num1,num2)
{
Response.Write(num1*num2)
}
%>
</head>
<body>
<p>Result: <%jsproc(3,4)%></p>
</body>
</html>
```

Differences Between VBScript and JavaScript

When calling a VBScript or a JavaScript procedure from an ASP file written in VBScript, you can use the "call" keyword followed by the procedure name. If a procedure requires parameters, the parameter list must be enclosed in parentheses when using the "call" keyword. If you omit the "call" keyword, the parameter list must not be enclosed in parentheses. If the procedure has no parameters, the parentheses are optional.

When calling a JavaScript or a VBScript procedure from an ASP file written in JavaScript, always use parentheses after the procedure name.

ASP Forms and User Input

The Request.QueryString and Request.Form commands may be used to retrieve information from forms, like user input.

Examples

For Examples -> Refer CD

User Input

The Request object may be used to retrieve user information from forms:

```
<form method="get" action="simpleform.asp">
First Name: <input type="text" name="fname">
<br />
Last Name: <input type="text" name="lname">
<br /><br />
<input type="submit" value="Submit">
</form>
```

User input can be retrieved in two ways: With Request.QueryString or Request.Form.

Request.QueryString

The Request.QueryString command is used to collect values in a form with method="get". Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

If a user typed "Bill" and "Gates" in the form example above, the URL sent to the server would look like this:

```
http://www.atmiya.com/simpleform.asp?fname=Bill&lname=Gates
```

Assume that the ASP file "simpleform.asp" contains the following script:

```
<body>
Welcome
<%
```

```
response.write(request.querystring("fname"))
response.write(" " & request.querystring("lname"))
%>
</body>
```

The browser will display the following in the body of the document:

```
Welcome Bill Gates
```

Request.Form

The Request.Form command is used to collect values in a form with method="post". Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

If a user typed "Bill" and "Gates" in the form example above, the URL sent to the server would look like this:

```
http://www.atmiya.com/simpleform.asp
```

Assume that the ASP file "simpleform.asp" contains the following script:

```
<body>
Welcome
<%
response.write(request.form("fname"))
response.write(" " & request.form("lname"))
%>
</body>
```

The browser will display the following in the body of the document:

```
Welcome Bill Gates
```

Form Validation

User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and you reduce the server load.

You should consider using server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

ASP Cookies

A cookie is often used to identify a user.

Examples

For Example -> Refer CD

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests for a page with a browser, it will send the cookie too. With ASP, you can both create and retrieve cookie values.

How to Create a Cookie

The "Response.Cookies" command is used to create cookies.

Note: The Response.Cookies command must appear BEFORE the <html> tag.

In the example below, we will create a cookie named "firstname" and assign the value "Alex" to it:

```
<%
Response.Cookies("firstname")="Alex"
%>
```

It is also possible to assign properties to a cookie, like setting a date when the cookie should expire:

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("firstname").Expires=#May 10,2002#
%>
```

How to Retrieve a Cookie Value

The "Request.Cookies" command is used to retrieve a cookie value.

In the example below, we retrieve the value of the cookie named "firstname" and display it on a page:

```
<%
fname=Request.Cookies("firstname")
response.write("Firstname=" & fname)
%>
```

Output:

Firstname=Alex

A Cookie with Keys

If a cookie contains a collection of multiple values, we say that the cookie has Keys.

In the example below, we will create a cookie collection named "user". The "user" cookie has Keys that contains information about a user:

```
<%
Response.Cookies("user")("firstname")="John"
Response.Cookies("user")("lastname")="Smith"
Response.Cookies("user")("country")="Norway"
Response.Cookies("user")("age")="25"
%>
```

Read all Cookies

Look at the following code:

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("user")("firstname")="John"
Response.Cookies("user")("lastname")="Smith"
Response.Cookies("user")("country")="Norway"
Response.Cookies("user")("age")="25"
%>
```

Assume that your server has sent all the cookies above to a user.

Now we want to read all the cookies sent to a user. The example below shows how to do it (note that the code below checks if a cookie has Keys with the HasKeys property):

```
<html>
<body>
<%
dim x,y
for each x in Request.Cookies
  response.write("<p>")
  if Request.Cookies(x).HasKeys then
```

```
    for each y in Request.Cookies(x)
      response.write(x & ":" & y & "=" & Request.Cookies(x)(y))
      response.write("<br />")
    next
  else
    Response.Write(x & "=" & Request.Cookies(x) & "<br />")
  end if
  response.write "</p>"
next
%>
</body>
</html>
```

Output:

firstname=Alex

user:firstname=John
user:lastname=Smith
user: country=Norway
user: age=25

What if a Browser Does NOT Support Cookies?

If your application deals with browsers that do not support cookies, you will have to use other methods to pass information from one page to another in your application. There are two ways of doing this:

1. Add parameters to a URL

You can add parameters to a URL:

```
<a href="welcome.asp?fname=John&lname=Smith">
Go to Welcome Page</a>
```

And retrieve the values in the "welcome.asp" file like this:

```
<%
fname=Request.querystring("fname")
lname=Request.querystring("lname")
response.write("<p>Hello " & fname & " " & lname & "!</p>")
response.write("<p>Welcome to my Web site!</p>")
%>
```

2. Use a form

You can use a form. The form passes the user input to "welcome.asp" when the user clicks on the Submit button:

```
<form method="post" action="welcome.asp">
First Name:  <input type="text" name="fname" value="">
Last Name: <input type="text" name="lname" value="">
<input type="submit" value="Submit">
</form>
```

Retrieve the values in the "welcome.asp" file like this:

```
<%
fname=Request.form("fname")
lname=Request.form("lname")
response.write("<p>Hello " & fname & " " & lname & "!</p>")
response.write("<p>Welcome to my Web site!</p>")
%>
```

ASP Session Object

The Session object is used to store information about, or change settings for a user session. Variables stored in the Session object hold information about one single user, and are available to all pages in one application.

Examples

For Examples -> Refer CD

Session Object

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

ASP solves this problem by creating a unique cookie for each user. The cookie is sent to the client and it contains information that identifies the user. This interface is called the Session object.

The Session object is used to store information about, or change settings for a user session. Variables stored in the Session object hold information about one single user, and are available to all pages in one application. Common information stored in session variables are name, id, and preferences. The server creates a new Session object for each new user, and destroys the Session object when the session expires.

The Session object's collections, properties, methods, and events are described below:

Collections

Collection	Description
Contents	Contains all the items appended to the session through a script command
StaticObjects	Contains all the objects appended to the session with the HTML <object> tag

Properties

Property	Description
CodePage	Specifies the character set that will be used when displaying dynamic content
LCID	Sets or returns an integer that specifies a location or region. Contents like date, time, and currency will be displayed according to that location or region
SessionID	Returns a unique id for each user. The unique id is generated by the server
Timeout	Sets or returns the timeout period (in minutes) for the Session object in this application

Methods

Method	Description
Abandon	Destroys a user session
Contents.Remove	Deletes an item from the Contents collection
Contents.RemoveAll()	Deletes all items from the Contents collection

Events

Event	Description
Session_OnEnd	Occurs when a session ends
Session_OnStart	Occurs when a session starts

ASP Application Object

A group of ASP files that work together to perform some purpose is called an application. The Application object in ASP is used to tie these files together.

Application Object

An application on the Web may be a group of ASP files. The ASP files work together to perform some purpose. The Application object in ASP is used to tie these files together.

The Application object is used to store and access variables from any page, just like the Session object. The difference is that ALL users share one Application object, while with Sessions there is one Session object for EACH user.

The Application object should hold information that will be used by many pages in the application (like database connection information). This means that you can access the information from any page. It also means that you can change the information in one place and the changes will automatically be reflected on all pages.

Store and Retrieve Application Variables

Application variables can be accessed and changed by any page in the application.

You can create Application variables in "Global.asa" like this:

```
<script language="vbscript" runat="server">

Sub Application_OnStart
application("vartime")=""
application("users")=1
End Sub

</script>
```

In the example above we have created two Application variables: "vartime" and "users".

You can access the value of an Application variable like this:

```
There are
<%
Response.Write(Application("users"))
%>
active connections.
```

Loop Through the Contents Collection

The Contents collection contains all application variables. You can loop through the Contents collection, to see what's stored in it:

```
<%
dim i
For Each i in Application.Contents
  Response.Write(i & "<br />")
Next
%>
```

If you do not know the number of items in the Contents collection, you can use the Count property:

```
<%
dim i
dim j
j=Application.Contents.Count
For i=1 to j
  Response.Write(Application.Contents(i) & "<br />")
Next
%>
```

Loop Through the StaticObjects Collection

You can loop through the StaticObjects collection, to see the values of all objects stored in the Application object:

```
<%
dim i
For Each i in Application.StaticObjects
  Response.Write(i & "<br />")
Next
%>
```

Lock and Unlock

You can lock an application with the "Lock" method. When an application is locked, the users cannot change the Application variables (other than the one currently accessing it). You can unlock an application with the "Unlock" method. This method removes the lock from the Application variable:

```
<%
Application.Lock
  'do some application object operations
Application.Unlock
%>
```

ASP Including Files

The #include directive is used to create functions, headers, footers, or elements that will be reused on multiple pages.

The #include Directive

You can insert the content of one ASP file into another ASP file before the server executes it, with the #include directive. The #include directive is used to create functions, headers, footers, or elements that will be reused on multiple pages.

How to Use the #include Directive

Here is a file called "mypage.asp":

```
<html>
<body>
<h3>Words of Wisdom:</h3>
<p><!--#include file="wisdom.inc"--></p>
<h3>The time is:</h3>
<p><!--#include file="time.inc"--></p>
</body>
</html>
```

Here is the "wisdom.inc" file:

```
"One should never increase, beyond what is necessary,
the number of entities required to explain anything."
```

Here is the "time.inc" file:

```
<%
Response.Write(Time)
%>
```

If you look at the source code in a browser, it will look something like this:

```
<html>
<body>
<h3>Words of Wisdom:</h3>
<p>"One should never increase, beyond what is necessary,
the number of entities required to explain anything."</p>
<h3>The time is:</h3>
<p>11:33:42 AM</p>
</body>
</html>
```

Syntax for Including Files

To include a file in an ASP page, place the #include directive inside comment tags:

```
<!--#include virtual="somefilename"-->
```

```
or
<!--#include file ="somefilename"-->
```

The Virtual Keyword

Use the virtual keyword to indicate a path beginning with a virtual directory.

If a file named "header.inc" resides in a virtual directory named /html, the following line would insert the contents of "header.inc":

```
<!-- #include virtual ="/html/header.inc" -->
```

The File Keyword

Use the file keyword to indicate a relative path. A relative path begins with the directory that contains the including file.

If you have a file in the html directory, and the file "header.inc" resides in html\headers, the following line would insert "header.inc" in your file:

```
<!-- #include file ="headers\header.inc" -->
```

Note that the path to the included file (headers\header.inc) is relative to the including file. If the file containing this #include statement is not in the html directory, the statement will not work.

You can also use the file keyword with the syntax (..\) to include a file from a higher-level directory.

Tips and Notes

In the sections above we have used the file extension ".inc" for included files. Notice that if a user tries to browse an INC file directly, its content will be displayed. If your included file contains confidential information or information you do not want any users to see, it is better to use an ASP extension. The source code in an ASP file will not be visible after the interpretation. An included file can also include other files, and one ASP file can include the same file more than once.

Important: Included files are processed and inserted before the scripts are executed.

The following script will not work because ASP executes the #include directive before it assigns a value to the variable:

```
<%
fname="header.inc"
%>
<!--#include file="<%=fname%>"-->
```

You cannot open or close a script delimiter in an INC file. This script will not work:

```
<%
For i = 1 To n
  <!--#include file="count.inc"-->
Next
%>
```

But this script will work:

```
<% For i = 1 to n %>
<!--#include file="count.inc" -->
<% Next %>
```

ASP The Global.asa file

The Global.asa file is an optional file that can contain declarations of objects, variables, and methods that can be accessed by every page in an ASP application.

The Global.asa file

The Global.asa file is an optional file that can contain declarations of objects, variables, and methods that can be accessed by every page in an ASP application. All valid browser scripts (JavaScript, VBScript, JScript, PerlScript, etc.) can be used within Global.asa.

The Global.asa file can contain only the following:

- Application events
- Session events
- <object> declarations
- TypeLibrary declarations

Note: The Global.asa file must be stored in the root directory of the ASP application, and each application can only have one Global.asa file.

Events in Global.asa

In Global.asa you can tell the application and session objects what to do when the application/session starts and what to do when the application/session ends. The code for this is placed in event handlers. The Global.asa file can contain four types of events:

Application_OnStart - This event occurs when the FIRST user calls the first page from an ASP application. This event occurs after the Web server is restarted or after the Global.asa file is edited. The "Session_OnStart" event occurs immediately after this event.

Session_OnStart - This event occurs EVERY time a NEW user requests his or hers first page in the ASP application.

Session_OnEnd - This event occurs EVERY time a user ends a session. A user ends a session after a page has not been requested by the user for a specified time (by default this is 20 minutes).

Application_OnEnd - This event occurs after the LAST user has ended the session. Typically, this event occurs when a Web server stops. This procedure is used to clean up settings after the Application stops, like delete records or write information to text files.

A Global.asa file could look something like this:

```
<script language="vbscript" runat="server">
sub Application_OnStart
''''some code
end sub
sub Application_OnEnd
''''some code
end sub
sub Session_OnStart
''''some code
end sub
sub Session_OnEnd
''''some code
end sub
</script>
```

Note: We cannot use the ASP script delimiters (<% and %>) to insert scripts in the Global.asa file, we will have to put the subroutines inside the HTML <script> tag.

<object> Declarations

It is possible to create objects with session or application scope in Global.asa by using the <object> tag.

Note: The <object> tag should be outside the <script> tag!

Syntax

```
<object runat="server" scope="scope" id="id"
{progid="progID"|classid="classID"}>
....
</object>
```

Parameter	Description
scope	Sets the scope of the object (either Session or Application)
id	Specifies a unique id for the object

ProgID	An id associated with a class id. The format for ProgID is [Vendor.]Component[.Version] Either ProgID or ClassID must be specified.
ClassID	Specifies a unique id for a COM class object. Either ProgID or ClassID must be specified.

Examples

The first example creates an object of session scope named "MyAd" by using the ProgID parameter:

```
<object runat="server" scope="session" id="MyAd"
progid="MSWC.AdRotator">
</object>
```

The second example creates an object of application scope named "MyConnection" by using the ClassID parameter:

```
<object runat="server" scope="application" id="MyConnection"
classid="Clsid:8AD3067A-B3FC-11CF-A560-00A0C9081C21">
</object>
```

The objects declared in the Global.asa file can be used by any script in the application:

GLOBAL.ASA:

```
<object runat="server" scope="session" id="MyAd"
progid="MSWC.AdRotator">
</object>
```

You could reference the object "MyAd" from any page in the ASP application:

SOME .ASP FILE:

```
<%=MyAd.GetAdvertisement("/banners/adrot.txt")%>
```

TypeLibrary Declarations

A TypeLibrary is a container for the contents of a DLL file corresponding to a COM object. By including a call to the TypeLibrary in the Global.asa file, the constants of the COM object can be accessed, and errors can be better reported by the ASP code. If your Web application relies on COM objects that have declared data types in type libraries, you can declare the type libraries in Global.asa.

Syntax

```
<!--METADATA TYPE="TypeLib"
file="filename"
uuid="typelibraryuuid"
version="versionnumber"
lcid="localeid"
-->
```

Parameter	Description
file	Specifies an absolute path to a type library. Either the file parameter or the uuid parameter is required
uuid	Specifies a unique identifier for the type library. Either the file parameter or the uuid parameter is required
version	Optional. Used for selecting version. If the requested version is not found, then the most recent version is used
localeid	Optional. The locale identifier to be used for the type library

Error Values

The server can return one of the following error messages:

Error Code	Description
ASP 0222	Invalid type library specification
ASP 0223	Type library not found
ASP 0224	Type library cannot be loaded
ASP 0225	Type library cannot be wrapped

Note: METADATA tags can appear anywhere in the Global.asa file (both inside and outside <script> tags). However, it is recommended that METADATA tags appear near the top of the Global.asa file.

Restrictions

Restrictions on what you can include in the Global.asa file:

- You can not display text that is written in the Global.asa file. This file can't display information
- You can not use the #include directive in Global.asa
- You can only use Server and Application objects in the Application_OnStart and Application_OnEnd subroutines. In the Session_OnEnd subroutine, you can use Server, Application, and Session objects. In the Session_OnStart subroutine you can use any built-in object

How to use the Subroutines

Global.asa is often used to initialize variables.

The example below shows how to detect the exact time a visitor first arrives on a Web site. The time is stored in a Session variable named "started", and the value of the "start" variable can be accessed from any ASP page in the application:

```
<script language="vbscript" runat="server">
sub Session_OnStart
Session("started")=now()
end sub
</script>
```

Global.asa can also be used to control page access.

The example below shows how to redirect every new visitor to another page, in this case to a page called "newpage.asp":

```
<script language="vbscript" runat="server">
sub Session_OnStart
Response.Redirect("newpage.asp")
end sub
</script>
```

And you can include functions in the Global.asa file.

In the example below the Application_OnStart subroutine occurs when the Web server starts. Then the Application_OnStart subroutine calls another subroutine named "getcustomers". The "getcustomers" subroutine opens a database and retrieves a record set from the "customers" table. The record set is assigned to an array, where it can be accessed from any ASP page without querying the database:

```
<script language="vbscript" runat="server">
sub Application_OnStart
getcustomers
end sub
sub getcustomers
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=conn.execute("select name from customers")
Application("customers")=rs.GetRows
rs.Close
conn.Close
end sub
</script>
```

Global.asa Example

In this example we will create a Global.asa file that counts the number of current visitors.

- The Application_OnStart sets the Application variable "visitors" to 0 when the server starts
- The Session_OnStart subroutine adds one to the variable "visitors" every time a new visitor arrives
- The Session_OnEnd subroutine subtracts one from "visitors" each time this subroutine is triggered

The Global.asa file:

```
<script language="vbscript" runat="server">
Sub Application_OnStart
Application("visitors")=0
End Sub
Sub Session_OnStart
Application.Lock
Application("visitors")=Application("visitors")+1
Application.UnLock
End Sub
Sub Session_OnEnd
Application.Lock
Application("visitors")=Application("visitors")-1
Application.UnLock
End Sub
</script>
```

To display the number of current visitors in an ASP file:

```
<html>
<head>
</head>
<body>
<p>
There are <%response.write(Application("visitors"))%>
online now!
</p>
</body>
</html>
```

ASP Response Object

The ASP Response object is used to send output to the user from the server.

Examples

For Example -> Refer CD

Response Object

The ASP Response object is used to send output to the user from the server. Its collections, properties, and methods are described below:

Collections

Collection	Description
Cookies	Sets a cookie value. If the cookie does not exist, it will be created, and take the value that is specified

Properties

Property	Description
Buffer	Specifies whether to buffer the page output or not
CacheControl	Sets whether a proxy server can cache the output generated by ASP or not
Charset	Appends the name of a character-set to the content-type header in the Response object
ContentType	Sets the HTTP content type for the Response object
Expires	Sets how long (in minutes) a page will be cached on a browser before it expires
ExpiresAbsolute	Sets a date and time when a page cached on a browser will expire
IsClientConnected	Indicates if the client has disconnected from the server
Pics	Appends a value to the PICS label response header
Status	Specifies the value of the status line returned by the server

Methods

Method	Description
AddHeader	Adds a new HTTP header and a value to the HTTP response
AppendToLog	Adds a string to the end of the server log entry
BinaryWrite	Writes data directly to the output without any character conversion
Clear	Clears any buffered HTML output
End	Stops processing a script, and returns the current result
Flush	Sends buffered HTML output immediately
Redirect	Redirects the user to a different URL
Write	Writes a specified string to the output

ASP Request Object

The ASP Request object is used to get information from the user.

QueryString Collection Examples
For Example -> Refer CD

Form Collection Examples
For Example -> Refer CD

Other Examples
For Example -> Refer CD

Request Object

When a browser asks for a page from a server, it is called a request. The ASP Request object is used to get information from the user. Its collections, properties, and methods are described below:

Collections

Collection	Description
ClientCertificate	Contains all the field values stored in the client certificate
Cookies	Contains all the cookie values sent in a HTTP request
Form	Contains all the form (input) values from a form that uses the post method
QueryString	Contains all the variable values in a HTTP query string
ServerVariables	Contains all the server variable values

Properties

Property	Description
TotalBytes	Returns the total number of bytes the client sent in the body of the request

Methods

Method	Description
BinaryRead	Retrieves the data sent to the server from the client as part of a post request and stores it in a safe array

ASP Session Object

The Session object is used to store information about, or change settings for a user session. Variables stored in the Session object hold information about one single user, and are available to all pages in one application.

Examples

For Example -> Refer CD

Session Object

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

ASP solves this problem by creating a unique cookie for each user. The cookie is sent to the client and it contains information that identifies the user. This interface is called the Session object.

The Session object is used to store information about, or change settings for a user session. Variables stored in the Session object hold information about one single user, and are available to all pages in one application. Common information stored in session variables are name, id, and preferences. The server creates a new Session object for each new user, and destroys the Session object when the session expires.

The Session object's collections, properties, methods, and events are described below:

Collections

Collection	Description
Contents	Contains all the items appended to the session through a script command
StaticObjects	Contains all the objects appended to the session with the HTML <object> tag

Properties

Property	Description
CodePage	Specifies the character set that will be used when displaying dynamic content
LCID	Sets or returns an integer that specifies a location or region. Contents like date, time, and currency will be displayed according to that location or region
SessionID	Returns a unique id for each user. The unique id is generated

	by the server
Timeout	Sets or returns the timeout period (in minutes) for the Session object in this application

Methods

Method	Description
Abandon	Destroys a user session
Contents.Remove	Deletes an item from the Contents collection
Contents.RemoveAll()	Deletes all items from the Contents collection

Events

Event	Description
Session_OnEnd	Occurs when a session ends
Session_OnStart	Occurs when a session starts

ASP Server Object

The ASP Server object is used to access properties and methods on the server.

Examples

For Example -> Refer CD

Server Object

The ASP Server object is used to access properties and methods on the server. Its properties and methods are described below:

Properties

Property	Description
ScriptTimeout	Sets or returns the maximum number of seconds a script can run before it is terminated

Methods

Method	Description
CreateObject	Creates an instance of an object
Execute	Executes an ASP file from inside another ASP file
GetLastError()	Returns an ASPError object that describes the error condition that

	occurred
HTMLEncode	Applies HTML encoding to a specified string
MapPath	Maps a specified path to a physical path
Transfer	Sends (transfers) all the information created in one ASP file to a second ASP file
URLEncode	Applies URL encoding rules to a specified string

ASPError Object (ASP 3.0)

The ASPError object is used to display detailed information of any error that occurs in scripts in an ASP page.

The ASPError Object

The ASPError object is implemented in ASP 3.0 and it is only available in IIS5.

The ASP Error object is used to display detailed information of any error that occurs in scripts in an ASP page. The ASPError object is created when Server.GetLastError is called, so the error information can only be accessed by using the Server.GetLastError method.

Example

```
<html>
<body>

<%
'The following line creates an error
dim i for i=1 to 1 next

'Call the GetLastError() method to trap the error
dim objerr
set objerr=Server.GetLastError()

'The variable objerr now contains the ASPError object
response.write("ASP Code=" & objerr.ASPCode)
response.write("<br />")
response.write("Number=" & objerr.Number)
response.write("<br />")
response.write("Source=" & objerr.Source)
response.write("<br />")
response.write("Filename=" & objerr.File)
response.write("<br />")
response.write("LineNumber=" & objerr.Line)
%>
</body>
</html>
```

The ASPError object's properties are described below (all properties are read-only):

Note: The properties below can only be accessed through the Server.GetLastError() method.

Properties

Property	Description
ASPCode	Returns an error code generated by IIS
ASPDescription	Returns a detailed description of the error (if the error is ASP-related)
Category	Returns the source of the error (was the error generated by ASP? By a scripting language? By an object?)
Column	Returns the column position within the file that generated the error
Description	Returns a short description of the error
File	Returns the name of the ASP file that generated the error
Line	Returns the line number where the error was detected
Number	Returns the standard COM error code for the error
Source	Returns the actual source code of the line where the error occurred

ASP FileSystemObject Object

The FileSystemObject object is used to access the file system on the server.

Examples

For Example -> Refer CD

The FileSystemObject Object

The FileSystemObject object is used to access the file system on the server. This object can manipulate files, folders, and directory paths. It is also possible to retrieve file system information with this object.

The following code creates a text file (c:\test.txt) and then write some text to the file:

```
<%
dim fs,fname
set fs=Server.CreateObject("Scripting.FileSystemObject")
set fname=fs.CreateTextFile("c:\test.txt",true)
fname.WriteLine("Hello World!")
```

```
fname.Close
set fname=nothing
set fs=nothing
%>
```

The FileSystemObject object's properties and methods are described below:

Properties

Property	Description
Drives	Returns a collection of all Drive objects on the computer

Methods

Method	Description
BuildPath	Appends a name to an existing path
CopyFile	Copies one or more files from one location to another
CopyFolder	Copies one or more folders from one location to another
CreateFolder	Creates a new folder
CreateTextFile	Creates a text file and returns a TextStream object that can be used to read from, or write to the file
DeleteFile	Deletes one or more specified files
DeleteFolder	Deletes one or more specified folders
DriveExists	Checks if a specified drive exists
FileExists	Checks if a specified file exists
FolderExists	Checks if a specified folder exists
GetAbsolutePathName	Returns the complete path from the root of the drive for the specified path
GetBaseName	Returns the base name of a specified file or folder
GetDrive	Returns a Drive object corresponding to the drive in a specified path
GetDriveName	Returns the drive name of a specified path
GetExtensionName	Returns the file extension name for the last component in a specified path
GetFile	Returns a File object for a specified path
GetFileName	Returns the file name or folder name for the last component in a specified path
GetFolder	Returns a Folder object for a specified path
GetParentFolderName	Returns the name of the parent folder of the last component in a specified path
GetSpecialFolder	Returns the path to some of Windows' special folders
GetTempName	Returns a randomly generated temporary file or folder

MoveFile	Moves one or more files from one location to another
MoveFolder	Moves one or more folders from one location to another
OpenTextFile	Opens a file and returns a TextStream object that can be used to access the file

ASP TextStream Object

The TextStream object is used to access the contents of a text file.

Examples

For Example -> Refer CD

The TextStream Object

The TextStream object is used to access the contents of text files.

The following code creates a text file (c:\test.txt) and then writes some text to the file (the variable f is an instance of the TextStream object):

```
<%
dim fs, f
set fs=Server.CreateObject("Scripting.FileSystemObject")
set f=fs.CreateTextFile("c:\test.txt",true)
f.WriteLine("Hello World!")
f.Close
set f=nothing
set fs=nothing
%>
```

To create an instance of the TextStream object you can use the CreateTextFile or OpenTextFile methods of the FileSystemObject object, or you can use the OpenAsTextStream method of the File object.

The TextStream object's properties and methods are described below:

Properties

Property	Description
AtEndOfLine	Returns true if the file pointer is positioned immediately before the end-of-line marker in a TextStream file, and false if not
AtEndOfStream	Returns true if the file pointer is at the end of a TextStream file, and false if not

Column	Returns the column number of the current character position in an input stream
Line	Returns the current line number in a TextStream file

Methods

Method	Description
Close	Closes an open TextStream file
Read	Reads a specified number of characters from a TextStream file and returns the result
ReadAll	Reads an entire TextStream file and returns the result
ReadLine	Reads one line from a TextStream file and returns the result
Skip	Skips a specified number of characters when reading a TextStream file
SkipLine	Skips the next line when reading a TextStream file
Write	Writes a specified text to a TextStream file
WriteLine	Writes a specified text and a new-line character to a TextStream file
WriteBlankLines	Writes a specified number of new-line character to a TextStream file

ASP Drive Object

The Drive object is used to return information about a local disk drive or a network share.

Examples

For Example -> Refer CD

The Drive Object

The Drive object is used to return information about a local disk drive or a network share. The Drive object can return information about a drive's type of file system, free space, serial number, volume name, and more.

Note: You cannot return information about a drive's content with the Drive object. For this purpose you will have to use the Folder object.

To work with the properties of the Drive object, you will have to create an instance of the Drive object through the FileSystemObject object. First; create a FileSystemObject object

and then instantiate the Drive object through the GetDrive method or the Drives property of the FileSystemObject object.

The following example uses the GetDrive method of the FileSystemObject object to instantiate the Drive object and the TotalSize property to return the total size in bytes of the specified drive (c:):

```
<%
Dim fs,d
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set d=fs.GetDrive("c:")
Response.Write("Drive " & d & ":")
Response.Write("Total size in bytes: " & d.TotalSize)
set d=nothing
set fs=nothing
%>
```

Output:

Drive c: Total size in bytes: 4293563392

The Drive object's properties are described below:

Properties

Property	Description
AvailableSpace	Returns the amount of available space to a user on a specified drive or network share
DriveLetter	Returns one uppercase letter that identifies the local drive or a network share
DriveType	Returns the type of a specified drive
FileSystem	Returns the file system in use for a specified drive
FreeSpace	Returns the amount of free space to a user on a specified drive or network share
IsReady	Returns true if the specified drive is ready and false if not
Path	Returns an uppercase letter followed by a colon that indicates the path name for a specified drive
RootFolder	Returns a Folder object that represents the root folder of a specified drive
SerialNumber	Returns the serial number of a specified drive
ShareName	Returns the network share name for a specified drive
TotalSize	Returns the total size of a specified drive or network share
VolumeName	Sets or returns the volume name of a specified drive

ASP File Object

The File object is used to return information about a specified file.

Examples

For Example -> Refer CD

The File Object

The File object is used to return information about a specified file.

To work with the properties and methods of the File object, you will have to create an instance of the File object through the FileSystemObject object. First; create a FileSystemObject object and then instantiate the File object through the GetFile method of the FileSystemObject object or through the Files property of the Folder object.

The following code uses the GetFile method of the FileSystemObject object to instantiate the File object and the DateCreated property to return the date when the specified file was created:

```
<%
Dim fs,f
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set f=fs.GetFile("c:\test.txt")
Response.Write("File created: " & f.DateCreated)
set f=nothing
set fs=nothing
%>
```

Output:

```
File created: 9/19/2001 10:01:19 AM
```

The File object's properties and methods are described below:

Properties

Property	Description
Attributes	Sets or returns the attributes of a specified file
DateCreated	Returns the date and time when a specified file was created
DateLastAccessed	Returns the date and time when a specified file was last accessed
DateLastModified	Returns the date and time when a specified file was last modified
Drive	Returns the drive letter of the drive where a specified file or folder

	resides
Name	Sets or returns the name of a specified file
ParentFolder	Returns the folder object for the parent of the specified file
Path	Returns the path for a specified file
ShortName	Returns the short name of a specified file (the 8.3 naming convention)
ShortPath	Returns the short path of a specified file (the 8.3 naming convention)
Size	Returns the size, in bytes, of a specified file
Type	Returns the type of a specified file

Methods

Method	Description
Copy	Copies a specified file from one location to another
Delete	Deletes a specified file
Move	Moves a specified file from one location to another
OpenAsTextStream	Opens a specified file and returns a TextStream object to access the file

ASP Folder Object

The Folder Object is used to return information about a specified folder.

The Folder Object

The Folder object is used to return information about a specified folder.

To work with the properties and methods of the Folder object, you will have to create an instance of the Folder object through the FileSystemObject object. First; create a FileSystemObject object and then instantiate the Folder object through the GetFolder method of the FileSystemObject object.

The following code uses the GetFolder method of the FileSystemObject object to instantiate the Folder object and the DateCreated property to return the date when the specified folder was created:

```
<%
Dim fs,fo
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set fo=fs.GetFolder("c:\test")
```

```
Response.Write("Folder created: " & fo.DateCreated)
set fo=nothing
set fs=nothing
%>
```

Output:

```
Folder created: 10/22/2001 10:01:19 AM
```

The Folder object's collections, properties, and methods are described below:

Collections

Property	Description
Files	Returns a collection of all the files in a specified folder
SubFolders	Returns a collection of all subfolders in a specified folder

Properties

Property	Description
Attributes	Sets or returns the attributes of a specified folder
DateCreated	Returns the date and time when a specified folder was created
DateLastAccessed	Returns the date and time when a specified folder was last accessed
DateLastModified	Returns the date and time when a specified folder was last modified
Drive	Returns the drive letter of the drive where the specified folder resides
IsRootFolder	Returns true if a folder is the root folder and false if not
Name	Sets or returns the name of a specified folder
ParentFolder	Returns the parent folder of a specified folder
Path	Returns the path for a specified folder
ShortName	Returns the short name of a specified folder (the 8.3 naming convention)
ShortPath	Returns the short path of a specified folder (the 8.3 naming convention)
Size	Returns the size of a specified folder
Type	Returns the type of a specified folder

Methods

Method	Description
Copy	Copies a specified folder from one location to another
Delete	Deletes a specified folder
Move	Moves a specified folder from one location to another

CreateTextFile	Creates a new text file in the specified folder and returns a TextStream object to access the file

ASP Components

ASP AdRotator Component

Examples

For Example -> Refer CD

The ASP AdRotator component creates an AdRotator object that displays a different image each time a user enters or refreshes a page. A text file includes information about the images.

Syntax

```
<%
set adrotator=server.createobject("MSWC.AdRotator")
adrotator.GetAdvertisement("textfile.txt")
%>
```

Example

Assume we have a file called "banners.asp". It looks like this:

```
<html>
<body>
<%
set adrotator=Server.CreateObject("MSWC.AdRotator")
response.write(adrotator.GetAdvertisement("ads.txt"))
%>
</body>
</html>
```

The file "ads.txt" looks like this:

```
*
atmiya.gif
http://www.atmiya.com/
Visit Atmiya
80
```

```
microsoft.gif
http://www.microsoft.com/
Visit Microsoft
20
```

The lines below the asterisk in the file "ads.txt" specifies the images to be displayed, the hyperlink addresses, the alternate text (for the images), and the display rates in percent of the hits. We see that the Atmiya image will be displayed for 80 % of the hits and the Microsoft image will be displayed for 20 % of the hits in the text file above.

Note: To get the links to work when a user clicks on them, we will have to modify the file "ads.txt" a bit:

```
REDIRECT                                                    banners.asp
*
atmiya.gif
http://www.atmiya.com/
Visit                                                          Atmiya
80
microsoft.gif
http://www.microsoft.com/
Visit                                                       Microsoft
20
```

The redirection page (banners.asp) will now receive a querystring with a variable named URL containing the URL to redirect to.

Note: To specify the height, width, and border of the image, you can insert the following lines under REDIRECT:

```
REDIRECT                                                    banners.asp
WIDTH                                                              468
HEIGHT                                                              60
BORDER                                                               0
*
atmiya.gif
...
...
```

The last thing to do is to add some lines of code to the "banners.asp" file:

```
<%
url=Request.QueryString("url")
If                  url<>""              then              Response.Redirect(url)
%>

<html>
<body>
<%
```

```
set adrotator=Server.CreateObject("MSWC.AdRotator")
response.write(adrotator.GetAdvertisement("textfile.txt"))
%>
</body>
</html>
```

That's all!!

Properties

Property	Description	Example
Border	Specifies the size of the borders around the advertisement	<% set adrot=Server.CreateObject("MSWC.AdRotator") adrot.Border="2" Response.Write(adrot.GetAdvertisement("ads.txt")) %>
Clickable	Specifies whether the advertisement is a hyperlink	<% set adrot=Server.CreateObject("MSWC.AdRotator") adrot.Clickable=false Response.Write(adrot.GetAdvertisement("ads.txt")) %>
TargetFrame	Name of the frame to display the advertisement	<% set adrot=Server.CreateObject("MSWC.AdRotator") adrot.TargetFrame="target='_blank'" Response.Write(adrot.GetAdvertisement("ads.txt")) %>

Methods

Method	Description	Example
GetAdvertisement	Returns HTML that displays the advertisement in the page	<% set adrot=Server.CreateObject("MSWC.AdRotator") Response.Write(adrot.GetAdvertisement("ads.txt")) %>

ASP Browser Capabilities Component

Examples

For Example -> Refer CD

ASP Browser Capabilities Component

The ASP Browser Capabilities component creates a BrowserType object that determines the type, capabilities and version number of each browser that visits your site.

When a browser connects to a server, an HTTP User Agent Header is also sent to the server. This header contains information about the browser (like browser type and version number). The BrowserType object then compares the information in the header with information in a file on the server called "Browscap.ini".

If there is a match between the browser type and version number sent in the header and the information in the "Browsercap.ini" file, you can use the BrowserType object to list the properties of the matching browser. If there is no match for the browser type and version number in the Browscap.ini file, it will set every property to "UNKNOWN".

Syntax

```
<%
Set MyBrow=Server.CreateObject("MSWC.BrowserType")
%>
```

The example below creates a BrowserType object in an ASP file, and displays a table showing some of the capabilities of the current browser:

```
<html>
<body>
<%
Set MyBrow=Server.CreateObject("MSWC.BrowserType")
%>
<table border="1" width="100%">
<tr>
<th>Client OS</th>
<th><%=MyBrow.platform%></th>
</tr><tr>
<td >Web Browser</td>
<td ><%=MyBrow.browser%></td>
</tr><tr>
<td>Browser version</td>
<td><%=MyBrow.version%></td>
</tr><tr>
<td>Frame support?</td>
<td><%=MyBrow.frames%></td>
</tr><tr>
<td>Table support?</td>
<td><%=MyBrow.tables%></td>
</tr><tr>
<td>Sound support?</td>
<td><%=MyBrow.backgroundsounds%></td>
</tr><tr>
<td>Cookies support?</td>
<td><%=MyBrow.cookies%></td>
</tr><tr>
<td>VBScript support?</td>
```

```
<td><%=MyBrow.vbscript%></td>
</tr><tr>
<td>JavaScript support?</td>
<td><%=MyBrow.javascript%></td>
</tr>
</table>
</body>
</html>
```

Output:

Client OS	WinNT
Web Browser	IE
Browser version	5.0
Frame support?	True
Table support?	True
Sound support?	True
Cookies support?	True
VBScript support?	True
JavaScript support?	True

The Browscap.ini File

The "Browsercap.ini" file is used to declare properties and to set default values for browsers.

This section is not a tutorial on how to maintain "Browsercap.ini" files, it only shows you the basics; so you get an idea what a "Browsercap.ini" file is all about.

The "Browsercap.ini" file can contain the following:

```
[;comments]
[HTTPUserAgentHeader]
[parent=browserDefinition]
[property1=value1]
[propertyN=valueN]
[Default Browser Capability Settings]
[defaultProperty1=defaultValue1]
[defaultPropertyN=defaultValueN]
```

Parameter	Description
comments	Optional. Any line that starts with a semicolon are ignored by the BrowserType object
HTTPUserAgentHeader	Optional. Specifies the HTTP User Agent header to associate with the browser-property value statements specified in propertyN. Wildcard

	characters are allowed
browserDefinition	Optional. Specifies the HTTP User Agent header-string of a browser to use as the parent browser. The current browser's definition will inherit all of the property values declared in the parent browser's definition
propertyN	Optional. Specifies the browser properties. The following table lists some possible properties: • ActiveXControls - Support ActiveX® controls? • Backgroundsounds - Support background sounds? • Cdf - Support Channel Definition Format for Webcasting? • Tables - Support tables? • Cookies - Support cookies? • Frames - Support frames? • Javaapplets - Support Java applets? • Javascript - Supports JScript? • Vbscript - Supports VBScript? • Browser - Specifies the name of the browser • Beta - Is the browser beta software? • Platform - Specifies the platform that the browser runs on • Version - Specifies the version number of the browser
valueN	Optional. Specifies the value of propertyN. Can be a string, an integer (prefix with #), or a Boolean value
defaultPropertyN	Optional. Specifies the name of the browser property to which to assign a default value if none of the defined HTTPUserAgentHeader values match the HTTP User Agent header sent by the browser
defaultValueN	Optional. Specifies the value of defaultPropertyN. Can be a string, an integer (prefix with #), or a Boolean value

A "Browsercap.ini" file might look something like this:

```
;IE 5.0
[IE 5.0]
browser=IE
Version=5.0
majorver=#5
minorver=#0
frames=TRUE
tables=TRUE
cookies=TRUE
backgroundsounds=TRUE
vbscript=TRUE
javascript=TRUE
javaapplets=TRUE
ActiveXControls=TRUE
beta=False
;DEFAULT BROWSER
[*]
browser=Default
```

```
frames=FALSE
tables=TRUE
cookies=FALSE
backgroundsounds=FALSE
vbscript=FALSE
javascript=FALSE
```

ASP Quick Reference

ASP Quick Reference from Atmiya. Print it, and fold it in your pocket.

Basic Syntax

ASP scripts are surrounded by <% and %>. To write some output to a browser:

```
<html>
<body>
<% response.write("Hello World!") %>
</body>
</html>
```

The default language in ASP is VBScript. To use another scripting language, insert a language specification at the top of the ASP page:

```
<%@ language="javascript" %>
<html>
<body>

<%
....
%>
```

Forms and User Input

Request.QueryString is used to collect values in a form with method="get". Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

Request.Form is used to collect values in a form with method="post". Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

ASP Cookies

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests for a page with a browser, it will send the cookie too.

The Response.Cookies command is used to create cookies:

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("firstname").Expires="May 10,2002"
%>
```

Note: The Response.Cookies command must appear BEFORE the <html> tag!

The "Request.Cookies" command is used to retrieve a cookie value:

```
<%
fname=Request.Cookies("firstname")
response.write("Firstname=" & fname)
%>
```

Including Files

You can insert the content of one ASP file into another ASP file before the server executes it, with the #include directive. The #include directive is used to create functions, headers, footers, or elements that will be reused on multiple pages

Syntax:

```
<!--#include virtual="somefile.inc"-->
or
<!--#include file ="somefile.inc"-->
```

Use the virtual keyword to indicate a path beginning with a virtual directory. If a file named "header.inc" resides in a virtual directory named /html, the following line would insert the contents of "header.inc":

```
<!-- #include virtual ="/html/header.inc" -->
```

Use the file keyword to indicate a relative path. A relative path begins with the directory that contains the including file. If you have a file in the html directory, and the file "header.inc" resides in html\headers, the following line would insert "header.inc" in your file:

```
<!-- #include file ="headers\header.inc" -->
```

Use the file keyword with the syntax (..\) to include a file from a higher-level directory.

Global.asa

The Global.asa file is an optional file that can contain declarations of objects, variables, and methods that can be accessed by every page in an ASP application.

Note: The Global.asa file must be stored in the root directory of the ASP application, and each application can only have one Global.asa file.

The Global.asa file can contain only the following:

- Application events
- Session events
- <object> declarations
- TypeLibrary declarations

Application and Session Events

In Global.asa you can tell the application and session objects what to do when the application/session starts and what to do when the application/session ends. The code for this is placed in event handlers. **Note:** We do not use <% and %>, to insert scripts in the Global.asa file, we have to put the subroutines inside the HTML <script> tag:

```
<script language="vbscript" runat="server">
sub Application_OnStart
  ' some code
end sub
sub Application_OnEnd
  ' some code
end sub
sub Session_OnStart
  ' some code
end sub
sub Session_OnEnd
  ' some code
end sub
</script>
```

<object> Declarations

It is also possible to create objects with session or application scope in Global.asa by using the <object> tag. **Note:** The <object> tag should be outside the <script> tag!

Syntax:

```
<object runat="server" scope="scope" id="id"
{progid="progID"|classid="classID"}>
.....
</object>
```

TypeLibrary Declarations

A TypeLibrary is a container for the contents of a DLL file corresponding to a COM object. By including a call to the TypeLibrary in the Global.asa file, the constants of the COM object can be accessed, and errors can be better reported by the ASP code. If your Web application relies on COM objects that have declared data types in type libraries, you can declare the type libraries in Global.asa.

Syntax:

```
 <!--METADATA TYPE="TypeLib"
file="filename"
uuid="typelibraryuuid"
version="versionnumber"
lcid="localeid"
-->
```

The Session Object

The Session object is used to store information about, or change settings for a user session. Variables stored in the Session object hold information about one single user, and are available to all pages in one application.

Collections

- Contents - Holds every item added to the session with script commands
- StaticObjects - Holds every object added to the session with the <object> tag, and a given session
- Contents.Remove(*item/index*) - Deletes an item from the Contents collection
- Contents.RemoveAll() - Deletes every item from the Contents collection

Properties

- CodePage - Sets the code page that will be used to display dynamic content
- LCID - Sets the locale identifier that will be used to display dynamic content
- SessionID - Returns the session id
- Timeout - Sets the timeout for the session

Method

- Abandon - Kills every object in a session object

Application Object

A group of ASP files that work together to perform some purpose is called an application. The Application object in ASP is used to tie these files together. All users share one Application object. The Application object should hold information that will be used by many pages in the application (like database connection information).

Collections

- Contents - Holds every item added to the application with script commands
- StaticObjects - Holds every object added to the application with the <object> tag
- Contents.Remove - Deletes an item from a collection
- Contents.RemoveAll - Deletes every item from a collection

Methods

- Lock - Prevents a user from changing the application object properties
- Unlock - Allows a user to change the application object properties

The Response Object

The Response Object is used to send output to the user from the server.

Collection

- Cookies(name) - Sets a cookie value. If the cookie does not exist, it will be created, and take the value that is specified

Properties

- Buffer - Whether to buffer the output or not. When the output is buffered, the server will hold back the response until all of the server scripts have been processed, or until the script calls the Flush or End method. If this property is set, it should be before the <html> tag in the ASP file
- CacheControl - Sets whether proxy servers can cache the output or not. When set to Public, the output can be cached by a proxy server
- Charset(*charset_name)* - Sets the name of the character set (like "ISO8859-1") to the content type header
- ContentType - Sets the HTTP content type (like "text/html", "image/gif", "image/jpeg", "text/plain"). Default is "text/html"
- Expires - Sets how long a page will be cached on a browser before it expires
- ExpiresAbsolute - Sets a date and time when a page cached on a browser will expire
- IsClientConnected - Checks if the client is still connected to the server
- Pics(*pics_label*) - Adds a value to the pics label response header
- Status - Specifies the value of the status line

Methods

- AddHeader(*name, value*) - Adds an HTML header with a specified value
- AppendToLog *string* - Adds a string to the end of the server log entry
- BinaryWrite(*data_to_write*) - Writes the given information without any character-set conversion
- Clear - Clears the buffered output. Use this method to handle errors. If Response.Buffer is not set to true, this method will cause a run-time error
- End - Stops processing the script, and return the current result
- Flush - Sends buffered output immediately. If Response.Buffer is not set to true, this method will cause a run-time error
- Redirect(*url*) - Redirects the user to another url
- Write(*data_to_write*) - Writes a text to the user

Request Object

When a browser asks for a page from a server, it is called a request. The Request Object is used to get information from the user.

Collection

- ClientCertificate - Holds field values stored in the client certificate
- Cookies(*name*) - Holds cookie values
- Form(*element_name*) - Holds form (input) values. The form must use the post method
- QueryString(*variable_name*) - Holds variable values in the query string
- ServerVariables(*server_variable*) - Holds server variable values

Property

- TotalBytes - Holds the total number of bytes the client is sending in the body of the request

Method

- BinaryRead - Fetches the data that is sent to the server from the client as part of a post request

Server Object

The Server Object is used to access properties and methods on the server.

Property

- ScriptTimeout - Sets how long a script can run before it is terminated

Method

- CreateObject(*type_of_object*) - Creates an instance of an object
- Execute(*path*) - Executes an ASP file from inside another ASP file. After executing the called ASP file, the procedural control is returned to the the original ASP file
- GetLastError() - Returns an ASPError object that will describe the error that occurred
- HTMLEncode(*string*) - Applies HTML encoding to a string
- MapPath(*path*) - Maps a relative or virtual path to a physical path
- Transfer(*path*) - Sends all of the state information to another ASP file for processing. After the transfer, procedural control is not returned to the original ASP file
- URLEncode(*string*) - Applies URL encoding rules to a string

Introduction to ADO

What is ADO?

- ADO stands for **A**ctiveX **D**ata **O**bjects
- With ADO you can access and manipulate data in a database
- You can move data from a server to a client, manipulate the data, and return updates to the server, with ADO's Remote Data Service (RDS)
- ADO and RDS are automatically installed with Microsoft IIS

One of the major complaints from developers using Microsoft products is the confusing array of choices for getting to databases. Early on, you could only use the Open Database Connectivity (ODBC) Application Programming Interface (API), which was not particularly easy to implement, especially in the VB environment. Visual Basic started including a variety of objects to connect to databases, including Data Access Objects (DAO), designed for the Microsoft Access database, and Remote Data Objects (RDO), designed for enterprise databases like Oracle and SQL Server.

While both technologies worked fine, you couldn't take your understanding of one technology to work with the other. The object models were different enough to require you to relearn what you had learned in one model. There were also the ongoing confusions of when to use one instead of the other. In addition, some of the older methods were still available (such as talking directly to the ODBC API), as were some variations like ODBCDirect.

Something had to change to simplify the whole landscape, and that change was the introduction of Active Data Objects (ADO). ADO is a key component of Microsoft's Universal Data Access strategy. The idea is to give programmers a single set of objects that can be used in any Microsoft environment and for any platform. If you are using VB with SQL Server, you can use ADO. If you are using VB with Oracle, you can use ADO. If you're using Windows CE or Active Server Pages, you can use ADO. See the point?

OLE DB

OLE DB is a technology designed by Microsoft to make it easier to access all types of data through a single set of interfaces. Most programmers know how to access some type of database, such as Access, Oracle, SQL Server, or dBase. However, each one of these databases has a slightly different query language. Some are similar to standard SQL, but dBase has a different language structure than SQL.

Besides traditional databases, there are other sources of data that may interest your users. If you're working on a Web server, you may want to access data that Microsoft Index Server has produced from your Web site. Index Server is designed to make searchable indexes of your Web data. You may also want to access information in plain text files or in other known document types, such as Microsoft Word or Adobe Acrobat. You may also have a need some data mining using Online Analytical Processing (OLAP) tools.

All of these sources of data are now accessible through OLE DB. OLE DB uses a driver called a provider. A provider knows how a particular type of data is arranged, regardless of the type of data. The provider translates the given request into a request it can process against

its particular type of data. The programmer only has to worry about submitting a request that resembles standard SQL language, and the provider takes care of the rest.

OLE DB emphasizes the break between the components involved in an application. The application submits a request to the provider, which then translates the request to the data source so that the data can be sent to the application. If the data source driver changes, it won't necessarily affect the application's functionality. Just as with object-oriented applications, encapsulation is helpful here to protect your applications from the whims of your product vendors, who tend to change their interfaces just as you've gotten used to one version.

ODBC

Open Database Connectivity (ODBC) is a specification for a database API. The API is an independent standard supported by a variety of product vendors, including Oracle, Informix, Sybase, and Microsoft. Drivers for these databases are provided by both the vendors and third-party companies, such as Intersolv.

While OLE DB is able to talk directly to several different types of databases, some databases do not yet have OLE DB providers available. In these cases, you can use the ODBC driver for the database in combination with the OLE DB provider for ODBC. Using this method creates an extra layer of interface between your code and the database; that is, ADO talks to OLE DB, which talks to ODBC, which talks to the database. This method would be recommended if you are planning to upgrade the application or the database at some point. Because more and more product vendors are releasing OLE DB providers, new applications should be built using ADO and OLE DB, even if that means using the extra layer of ODBC for the present. This method will require the least code to change to OLE DB/ADO when your database releases an OLE DB provider.

Remote Data Service

Part of Microsoft's strategy is to make data available everywhere on every platform. Besides traditional applications, the Web is gaining ground as a popular way to publish corporate data. The use of this method will only gain ground as the Internet grows and the use of corporate extranets becomes more popular. A feature introduced in Internet Explorer 3.0 and higher is the use of the Remote Data Service (RDS). RDS enables applications to access OLE DB providers running on remote machines or in separate processes on the same machine.

This feature makes it easier to create dynamic Web pages. Instead of bringing down all the possible data a page could use, you get data as you need it. A good example is a tree-based interface. You load the tree with the top-level data when the page loads. As the user clicks on a node, you request the data for the node. Microsoft makes use of this strategy on their MSDN Online site, and it makes the page operation quicker than it would be otherwise. The only downside to this technology is the requirement that only Microsoft Internet Explorer will work with RDS. If you can't guarantee that all your users are using IE, you will have to look at doing more server-side database operations or use another language such as Java to make your components work properly.

Active Data Objects

Regardless of the driver combinations you are using, you will be using Active Data Objects to manipulate the data. ADO provides a number of objects used to traverse all types of data. If you're familiar with DAO or RDO, using ADO should be a fairly easy transition. There are a few differences, as indicated in the following section of this chapter. ADO defines seven objects:

- Connection
- Command
- Recordset
- Parameter
- Field
- Property
- Error

In addition, there are four collections used in ADO:

- Fields
- Parameters
- Properties
- Errors

The rest of the chapter helps you understand how these objects are used to manipulate your data sources.

ADO Database Connection

Before a database can be accessed from a web page, a database connection has to be established.

Create a DSN-less Database Connection

The easiest way to connect to a database is to use a DSN-less connection. A DSN-less connection can be used against any Microsoft Access database on your web site.

If you have a database called "northwind.mdb" located in a web directory like "c:/webdata/", you can connect to the database with the following ASP code:

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
%>
```

Note, from the example above, that you have to specify the Microsoft Access database driver (Provider) and the physical path to the database on your computer.

Create an ODBC Database Connection

If you have an ODBC database called "northwind" you can connect to the database with the following ASP code:

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Open "northwind"
%>
```

With an ODBC connection, you can connect to any database, on any computer in your network, as long as an ODBC connection is available.

An ODBC Connection to a MS Access Database

Here is how to create a connection to a MS Access Database:

1. Open the **ODBC** icon in your Control Panel.
2. Choose the **System DSN** tab.
3. Click on **Add** in the System DSN tab.
4. **Select** the Microsoft Access Driver. Click **Finish.**
5. In the next screen, click **Select** to locate the database.
6. Give the database a **D**ata **S**ource **N**ame (DSN).
7. Click **OK**.

Note that this configuration has to be done on the computer where your web site is located. If you are running Personal Web Server (PWS) or Internet Information Server (IIS) on your own computer, the instructions above will work, but if your web site is located on a remote server, you have to have physical access to that server, or ask your web host to do this for you.

The ADO Connection Object

A Connection object represents a single session with a data source. In ADO, you can have multiple Connection objects with each one pointing to a different data source. This can be helpful if you are accessing multiple data sources and combining the results on a Web page of some sort. In the case of a client/server database system, it may be equivalent to an actual network connection to the server. Depending on the functionality supported by the OLE DB provider, some collections, methods, or properties of a Connection object may not be available. This section describes the Connection object and how it is used to access data sources.

Making a Connection

To get started with any other ADO object, you first have to have a Connection object. A Connection object is given information about how to connect to the data source. The code shown here is one example of how to connect to the BIBLIO database included with Visual Basic.

```
Dim dcnDB   ' As ADODB.Connection

Set dcnDB = Server.CreateObject("ADODB.Connection")
dcnDB.ConnectionString = _
   & "Provider=Microsoft.Jet.OLEDB.3.51;" _
   & "Data Source=C:\Visual Studio\VB98\Biblio.mdb"
dcnDB.Open
```

You can substitute any Access 97 database following the Data Source parameter in the third line of code. For Access 2000, change "3.51" to "4.0" and use the Data Source parameter in the same way. For Access 2000 databases, you should always use the 4.0 provider instead of the 3.51 because of some documented bugs in using the wrong version of the provider. It may appear to work, but be careful.

As mentioned in preceding chapters, the Dim statement will help document your code by specifying what the variable is designed to hold. In this case, we are using the prefix dcn to specify a variable used to hold a Connection object. The next line uses the Server.CreateObject method to instantiate a Connection object. However, the data connection is still not open. The third line specifies how to connect to the database. The Provider parameter specifies the OLE DB Provider to use; in this case, we are using the Jet 3.51 Provider, which corresponds to Access 97. We also have to specify the pathname to the database in the Data Source parameter. Finally, the Open method activates the connection to the database.

The ConnectionString property will change depending on the data source. The code shown here can be used to connect to a SQL Server (any version) database. Notice the extra parameters in the ConnectionString property.

```
Dim dcnDB   ' As ADODB.Connection

Set dcnDB = Server.CreateObject("ADODB.Connection")
dcnDB.ConnectionString = _
   & "Provider=SQLOLEDB;" _
   & "Data Source=db.server.com" _
   & "Initial Catalog=PUBS;" _
   & "User ID=myuser;Password=mypassword;"
dcnDB.Open
```

To begin, the Provider property is different for SQL Server. We then need to specify both the database server and the database name we want to use. The Data Source can be either a LAN server name, an Internet-style address as shown in the example, or a numerical IP address, such as 252.100.100.0. The Initial Catalog is the database on the server that we need to access. Finally, SQL Server requires a user ID and password, so both of these parameters are provided in the ConnectionString property.

You can also create a Data Source Name (DSN) on your server if you don't want to use this method of creating connections. This method is called the "DSN-Less" method and is my preference since I often can't get access to the Control Panel of the machines I'm using. If you are using a DSN, simply specify the DSN as your connection string like so:

```
Dim dcnDB   ' As ADODB.Connection

Set dcnDB = Server.CreateObject("ADODB.Connection")
dcnDB.ConnectionString = "DSN=MyDatabaseDSN"
dcnDB.Open
```

While this method does have the advantage of being shorter code, any changes to the connection string involve a trip to the server's Control Panel in one way or another.

Closing a Connection

When you are at the end of your ASP page, you should always close any open data connections. This practice enables the data source server to release the system resources associated with that connection. Too many connections left open will, over time, cause the system to run out of resources eventually and grind to a halt. Closing a connection is easy. Simply use the Close method:

```
dcnDB.Close
```

Before closing the connection, you should also make sure that any other objects using the connection have been closed. Otherwise, you could possibly use an object that was no longer valid, which would cause an error. In addition, you should clear your object references with a statement as follows:

```
Set dcnDB = Nothing
```

Even though Microsoft says that it will clean up these objects when the page is done, it's a good idea to clean things up yourself. Garbage collection, which is the process of reclaiming available memory, has gotten a major overhaul in ASP.NET, which is the next version of Active Server Pages.

ADO Recordset

To be able to read database data, the data must first be loaded into a recordset.

Create an ADO Table Recordset

After an ADO Database Connection has been created, as demonstrated in the previous chapter, it is possible to create an ADO Recordset.

Suppose we have a database named "Northwind", we can get access to the "Customers" table inside the database with the following lines:

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=Server.CreateObject("ADODB.recordset")
rs.Open "Customers", conn
%>
```

Create an ADO SQL Recordset

We can also get access to the data in the "Customers" table using SQL:

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=Server.CreateObject("ADODB.recordset")
rs.Open "Select * from Customers", conn
%>
```

Extract Data from the Recordset

After a recordset is opened, we can extract data from recordset.

Suppose we have a database named "Northwind", we can get access to the "Customers" table inside the database with the following lines:

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=Server.CreateObject("ADODB.recordset")
rs.Open "Select * from Customers", conn
for each x in rs.fields
   response.write(x.name)
   response.write(" = ")
   response.write(x.value)
next
%>
```

ADO Display

The most common way to display data from a recordset, is to display the data in an HTML table.

Examples

For Example -> Refer CD

Display the Field Names and Field Values

We have a database named "Northwind" and we want to display the data from the "Customers" table (remember to save the file with an .asp extension):

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs = Server.CreateObject("ADODB.recordset")
rs.Open "Select * from Customers", conn
do until rs.EOF
  for each x in rs.Fields
    Response.Write(x.name)
    Response.Write(" = ")
    Response.Write(x.value & "<br />")
  next
  Response.Write("<br />")
  rs.MoveNext
loop
rs.close
conn.close
%>
</body>
</html>
```

Display the Field Names and Field Values in an HTML Table

We can also display the data from the "Customers" table inside an HTML table with the following lines (remember to save the file with an .asp extension):

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs = Server.CreateObject("ADODB.recordset")
rs.Open "Select Companyname, Contactname from
```

```
Customers", conn
%>
<table border="1" width="100%">
<%do until rs.EOF%>
   <tr>
   <%for each x in rs.Fields%>
      <td><%Response.Write(x.value)%></td>
   <%next
   rs.MoveNext%>
   </tr>
<%loop
rs.close
conn.close
%>
</table>
</body>
</html>
```

Add Headers to the HTML Table

We want to add headers to the HTML table to make it more readable (remember to save the file with an .asp extension):

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers"
rs.Open sql, conn
%>
<table border="1" width="100%">
  <tr>
  <%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
  next%>
  </tr>
  <%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
      <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
  <%loop
  rs.close
  conn.close
  %>
</table>
</body>
</html>
```

ADO Queries

We may use SQL to create queries to specify only a selected set of records and fields to view.

Examples

For Example -> Refer CD

Display Selected Data

We want to display only the records from the "Customers" table that have a "Companyname" that starts with an A (remember to save the file with an .asp extension):

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers
WHERE CompanyName LIKE 'A%'"
rs.Open sql, conn
%>
<table border="1" width="100%">
  <tr>
  <%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
  next%>
  </tr>
  <%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
      <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
  <%loop
  rs.close
  conn.close%>
</table>
</body>
</html>
```

ADO Sort

We may use SQL to specify how to sort the data in the record set.

Examples

For Example -> Refer CD

Sort the Data

We want to display the "Companyname" and "Contactname" fields from the "Customers" table, ordered by "Companyname" (remember to save the file with an .asp extension):

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM
Customers ORDER BY CompanyName"
rs.Open sql, conn
%>

<table border="1" width="100%">
  <tr>
  <%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
  next%>
  </tr>
  <%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
      <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
  <%loop
  rs.close
  conn.close%>
</table>
</body>
</html>
```

ADO Add Records

We may use the SQL INSERT INTO command to add a record to a table in a database.

Add a Record to a Table in a Database

We want to add a new record to the Customers table in the Northwind database. We first create a form that contains the fields we want to collect data from:

```
<html>
<body>
<form method="post" action="demo_add.asp">
<table>
<tr>
<td>CustomerID:</td>
<td><input name="custid"></td>
</tr><tr>
<td>Company Name:</td>
<td><input name="compname"></td>
</tr><tr>
<td>Contact Name:</td>
<td><input name="contname"></td>
</tr><tr>
<td>Address:</td>
<td><input name="address"></td>
</tr><tr>
<td>City:</td>
<td><input name="city"></td>
</tr><tr>
<td>Postal Code:</td>
<td><input name="postcode"></td>
</tr><tr>
<td>Country:</td>
<td><input name="country"></td>
</tr>
</table>
<br /><br />
<input type="submit" value="Add New">
<input type="reset" value="Cancel">
</form>
</body>
</html>
```

When the user presses the submit button the form is sent to a file called "demo_add.asp". The "demo_add.asp" file contains the code that will add a new record to the Customers table:

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
sql="INSERT INTO customers (customerID,companyname,"
sql=sql & "contactname,address,city,postalcode,country)"
sql=sql & " VALUES "
```

```
sql=sql & "('" & Request.Form("custid") & "',"
sql=sql & "'" & Request.Form("compname") & "',"
sql=sql & "'" & Request.Form("contname") & "',"
sql=sql & "'" & Request.Form("address") & "',"
sql=sql & "'" & Request.Form("city") & "',"
sql=sql & "'" & Request.Form("postcode") & "',"
sql=sql & "'" & Request.Form("country") & "')"
on error resume next
conn.Execute sql,recaffected
if err<>0 then
  Response.Write("No update permissions!")
else
  Response.Write("<h3>" & recaffected & " record added</h3>")
end if
conn.close
%>

</body>
</html>
```

Important

If you use the SQL INSERT command be aware of the following:

- If the table contains a primary key, make sure to append a unique, non-Null value to the primary key field (if not, the provider may not append the record, or an error occurs)
- If the table contains an AutoNumber field, do not include this field in the SQL INSERT command (the value of this field will be taken care of automatically by the provider)

What about Fields With no Data?

In a MS Access database, you can enter zero-length strings ("") in Text, Hyperlink, and Memo fields IF you set the AllowZeroLength property to Yes.

Note: Not all databases support zero-length strings and may cause an error when a record with blank fields is added. It is important to check what data types your database supports.

ADO Update Records

We may use the SQL UPDATE command to update a record in a table in a database.

Update a Record in a Table

We want to update a record in the Customers table in the Northwind database. We first create a table that lists all records in the Customers table:

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=Server.CreateObject("ADODB.Recordset")
rs.open "SELECT * FROM customers,conn
%>
<h2>List Database</h2>
<table border="1" width="100%">
<tr>
<%
for each x in rs.Fields
  response.write("<th>" & ucase(x.name) & "</th>")
next
%>
</tr>
<% do until rs.EOF %>
<tr>
<form method="post" action="demo_update.asp">
<%
for each x in rs.Fields
  if x.name="customerID" then%>
    <td>
    <input type="submit" name="customerID" value="<%=x.value%>">
    </td>
  <%else%>
    <td><%Response.Write(x.value)%></td>
  <%end if
next
%>
</form>
<%rs.MoveNext%>
</tr>
<%
loop
conn.close
%>
</table>
</body>
</html>
```

If the user clicks on the button in the "customerID" column he or she will be taken to a new file called "demo_update.asp". The "demo_update.asp" file contains the source code on how to create input fields based on the fields from one record in the database table. It also contains a "Update record" button that will save your changes:

```
<html>
```

```
<body>
<h2>Update Record</h2>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
cid=Request.Form("customerID")
if Request.form("companyname")="" then
  set rs=Server.CreateObject("ADODB.Recordset")
  rs.open "SELECT * FROM customers WHERE customerID=" & cid,conn
  %>
  <form method="post" action="demo_update.asp">
  <table>
  <%for each x in rs.Fields
  <tr>
  <td><%=x.name%></td>
  <td><input name="<%=x.name%>" value="<%=x.value%>"></td>
  next
  %>
  </tr>
  </table>
  <br /><br />
  <input type="submit" value="Update record">
  </form>
<%
else
  sql="UPDATE customers SET "
  sql=sql & "customersID='" & cid & "',"
  sql=sql & "companyname='" & Request.Form("companyname") & "',"
  sql=sql & "contactname='" & Request.Form("contactname") & "',"
  sql=sql & "address='" & Request.Form("address") & "',"
  sql=sql & "city='" & Request.Form("city") & "',"
  sql=sql & "postalcode='" & Request.Form("postalcode") & "',"
  sql=sql & "country='" & Request.Form("country") & "',"
  sql=sql & " WHERE customersID=" & cid
  on error resume next
  conn.Execute sql
  if err<>0 then
    response.write("No update permissions!")
  else
    response.write("Record " & cid & " was updated!")
  end if
end if
conn.close
%>
</body>
</html>
```

ADO Delete Records

We may use the SQL DELETE command to delete a record in a table in a database.

Delete a Record in a Table

We want to delete a record in the Customers table in the Northwind database. We first create a table that lists all records in the Customers table:

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=Server.CreateObject("ADODB.Recordset")
rs.open "SELECT * FROM customers,conn
%>
<h2>List Database</h2>
<table border="1" width="100%">
<tr>
<%
for each x in rs.Fields
  response.write("<th>" & ucase(x.name) & "</th>")
next
%>
</tr>
<% do until rs.EOF %>
<tr>
<form method="post" action="demo_delete.asp">
<%
for each x in rs.Fields
  if x.name="customerID" then%>
    <td>
    <input type="submit" name="customerID" value="<%=x.value%>">
    </td>
  <%else%>
    <td><%Response.Write(x.value)%></td>
  <%end if
next
%>
</form>
<%rs.MoveNext%>
</tr>
<%
loop
conn.close
%>
</table>
</body>
</html>
```

If the user clicks on the button in the "customerID" column he or she will be taken to a new file called "demo_delete.asp". The "demo_delete.asp" file contains the source code on how to create input fields based on the fields from one record in the database table. It also contains a "Delete record" button that will delete the current record:

```
<html>
<body>
<h2>Delete Record</h2>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
cid=Request.Form("customerID")
if Request.form("companyname")="" then
  set rs=Server.CreateObject("ADODB.Recordset")
  rs.open "SELECT * FROM customers WHERE customerID=" & cid,conn
  %>
  <form method="post" action="demo_update.asp">
  <table>
  <%for each x in rs.Fields
  <tr>
  <td><%=x.name%></td>
  <td><input name="<%=x.name%>" value="<%=x.value%>"></td>
  next
  %>
  </tr>
  </table>
  <br /><br />
  <input type="submit" value="Delete record">
  </form>
<%
else
  sql="DELETE FROM customers"
  sql=sql & " WHERE customersID=" & cid
  on error resume next
  conn.Execute sql
  if err<>0 then
    response.write("No update permissions!")
  else
    response.write("Record " & cid & " was deleted!")
  end if
end if
conn.close
%>
</body>
</html>
```

ADO Command Object

Command Object

The ADO Command object is used to execute a single query against a database. The query can perform actions like creating, adding, retrieving, deleting or updating records.

If the query is used to retrieve data, the data will be returned as a RecordSet object. This means that the retrieved data can be manipulated by properties, collections, methods, and events of the Recordset object.

The major feature of the Command object is the ability to use stored queries and procedures with parameters.

ProgID

```
set objCommand=Server.CreateObject("ADODB.command")
```

Properties

Property	Description
ActiveConnection	Sets or returns a definition for a connection if the connection is closed, or the current Connection object if the connection is open
CommandText	Sets or returns a provider command
CommandTimeout	Sets or returns the number of seconds to wait while attempting to execute a command
CommandType	Sets or returns the type of a Command object
Name	Sets or returns the name of a Command object
Prepared	Sets or returns a Boolean value that, if set to True, indicates that the command should save a prepared version of the query before the first execution
State	Returns a value that describes if the Command object is open, closed, connecting, executing or retrieving data

Methods

Method	Description
Cancel	Cancels an execution of a method
CreateParameter	Creates a new Parameter object
Execute	Executes the query, SQL statement or procedure in the CommandText property

Collections

Collection	Description
Parameters	Contains all the Parameter objects of a Command Object
Properties	Contains all the Property objects of a Command Object

ADO Connection Object

Connection Object

The ADO Connection Object is used to create an open connection to a data source. Through this connection, you can access and manipulate a database.

If you want to access a database multiple times, you should establish a connection using the Connection object. You can also make a connection to a database by passing a connection string via a Command or Recordset object. However, this type of connection is only good for one specific, single query.

ProgID

```
set objConnection=Server.CreateObject("ADODB.connection")
```

Properties

Property	Description
Attributes	Sets or returns the attributes of a Connection object
CommandTimeout	Sets or returns the number of seconds to wait while attempting to execute a command
ConnectionString	Sets or returns the details used to create a connection to a data source
ConnectionTimeout	Sets or returns the number of seconds to wait for a connection to open
CursorLocation	Sets or returns the location of the cursor service
DefaultDatabase	Sets or returns the default database name
IsolationLevel	Sets or returns the isolation level
Mode	Sets or returns the provider access permission
Provider	Sets or returns the provider name
State	Returns a value describing if the connection is open or closed
Version	Returns the ADO version number

Methods

Method	Description
BeginTrans	Begins a new transaction
Cancel	Cancels an execution
Close	Closes a connection
CommitTrans	Saves any changes and ends the current transaction
Execute	Executes a query, statement, procedure or provider specific text
Open	Opens a connection
OpenSchema	Returns schema information from the provider about the data

	source
RollbackTrans	Cancels any changes in the current transaction and ends the transaction

Events

Note: You cannot handle events using VBScript or JScript (only Visual Basic, Visual C++, and Visual J++ languages can handle events).

Event	Description
BeginTransComplete	Triggered after the BeginTrans operation
CommitTransComplete	Triggered after the CommitTrans operation
ConnectComplete	Triggered after a connection starts
Disconnect	Triggered after a connection ends
ExecuteComplete	Triggered after a command has finished executing
InfoMessage	Triggered if a warning occurs during a ConnectionEvent operation
RollbackTransComplete	Triggered after the RollbackTrans operation
WillConnect	Triggered before a connection starts
WillExecute	Triggered before a command is executed

Collections

Collection	Description
Errors	Contains all the Error objects of the Connection object
Properties	Contains all the Property objects of the Connection object

ADO Error Object

Error Object

The ADO Error object contains details about data access errors that have been generated during a single operation.

ADO generates one Error object for each error. Each Error object contains details of the specific error, and are stored in the Errors collection. To access the errors, you must refer to a specific connection.

To loop through the Errors collection:

```
<%
```

```
for each objErr in objConn
  response.write("<p>")
  response.write("Description: ")
  response.write(objErr.Description & "<br />")
  response.write("Help context: ")
  response.write(objErr.HelpContext & "<br />")
  response.write("Help file: ")
  response.write(objErr.HelpFile & "<br />")
  response.write("Native error: ")
  response.write(objErr.NativeError & "<br />")
  response.write("Error number: ")
  response.write(objErr.Number & "<br />")
  response.write("Error source: ")
  response.write(objErr.Source & "<br />")
  response.write("SQL state: ")
  response.write(objErr.SQLState & "<br />")
  response.write("</p>")
next
%>
```

Syntax

```
objErr.property
```

Properties

Property	Description
Description	Returns an error description
HelpContext	Returns the context ID of a topic in the Microsoft Windows help system
HelpFile	Returns the full path of the help file in the Microsoft Windows help system
NativeError	Returns an error code from the provider or the data source
Number	Returns a unique number that identifies the error
Source	Returns the name of the object or application that generated the error
SQLState	Returns a 5-character SQL error code

ADO Field Object

Field Object

The ADO Field object contains information about a column in a Recordset object. There is one Field object for each column in the Recordset.

ProgID

```
set objField=Server.CreateObject("ADODB.field")
```

Properties

Property	Description
ActualSize	Returns the actual length of a field's value
Attributes	Sets or returns the attributes of a Field object
DefinedSize	Returns the defined size of a field
Name	Sets or returns the name of a Field object
NumericScale	Sets or returns the number of decimal places allowed for numeric values in a Field object
OriginalValue	Returns the original value of a field
Precision	Sets or returns the maximum number of digits allowed when representing numeric values in a Field object
Status	Returns the status of a Field object
Type	Sets or returns the type of a Field object
UnderlyingValue	Returns the current value of a field
Value	Sets or returns the value of a Field object

Methods

Method	Description
AppendChunk	Appends long binary or character data to a Field object
GetChunk	Returns all or a part of the contents of a large text or binary data Field object

Collections

Collection	Description
Properties	Contains all the Property objects for a Field object

ADO Parameter Object

Parameter Object

The ADO Parameter object provides information about a single parameter used in a stored procedure or query.

A Parameter object is added to the Parameters Collection when it is created. The Parameters Collection is associated with a specific Command object, which uses the Collection to pass parameters in and out of stored procedures and queries.

Parameters can be used to create Parameterized Commands. These commands are (after they have been defined and stored) using parameters to alter some details of the command before it is executed. For example, an SQL SELECT statement could use a parameter to define the criteria of a WHERE clause.

There are four types of parameters: input parameters, output parameters, input/output parameters and return parameters.

Syntax

```
objectname.property
objectname.method
```

Properties

Property	Description
Attributes	Sets or returns the attributes of a Parameter object
Direction	Sets or returns how a parameter is passed to or from a procedure
Name	Sets or returns the name of a Parameter object
NumericScale	Sets or returns the number of digits stored to the right side of the decimal point for a numeric value of a Parameter object
Precision	Sets or returns the maximum number of digits allowed when representing numeric values in a Parameter
Size	Sets or returns the maximum size in bytes or characters of a value in a Parameter object
Type	Sets or returns the type of a Parameter object
Value	Sets or returns the value of a Parameter object

Methods

Method	Description
AppendChunk	Appends long binary or character data to a Parameter object
Delete	Deletes an object from the Parameters Collection

ADO Property Object

Property Object

The ADO Property object represents a dynamic characteristic of an ADO object that is defined by the provider.

Each provider that talks with ADO has different ways of interacting with ADO. Therefore, ADO needs to store information about the provider in some way. The solution is that the provider gives specific information (dynamic properties) to ADO. ADO stores each provider property in a Property object that is again stored in the Properties Collection. The Collection

is assigned to either a Command object, Connection object, Field object, or a Recordset object.

ProgID

```
set objProperty=Server.CreateObject("ADODB.property")
```

Properties

Property	Description
Attributes	Returns the attributes of a Property object
Name	Sets or returns a the name of a Property object
Type	Returns the type of a Property object
Value	Sets or returns the value of a Property object

ADO Record Object

Record Object (ADO version 2.5)

The ADO Record object is used to hold a row in a Recordset, a directory, or a file from a file system.

Only structured databases could be accessed by ADO in versions prior 2.5. In a structured database, each table has the exact same number of columns in each row, and each column is composed of the same data type.

The Record object allows access to data-sets where the number of columns and/or the data type can be different from row to row.

Syntax

```
objectname.property
objectname.method
```

Properties

Property	Description
ActiveConnection	Sets or returns which Connection object a Record object belongs to
Mode	Sets or returns the permission for modifying data in a Record object
ParentURL	Returns the absolute URL of the parent Record
RecordType	Returns the type of a Record object
Source	Sets or returns the src parameter of the Open method of a

	Record object
State	Returns the status of a Record object

Methods

Method	Description
Cancel	Cancels an execution of a CopyRecord, DeleteRecord, MoveRecord, or Open call
Close	Closes a Record object
CopyRecord	Copies a file or directory to another location
DeleteRecord	Deletes a file or directory
GetChildren	Returns a Recordset object where each row represents the files in the directory
MoveRecord	Moves a file or a directory to another location
Open	Opens an existing Record object or creates a new file or directory

Collections

Collection	Description
Properties	A collection of provider-specific properties
Fields	Contains all the Field objects in the Record object

The Fields Collection's Properties

Property	Description
Count	Returns the number of items in the fields collection. Starts at zero. Example: countfields = rec.Fields.Count
Item(named_item/number)	Returns a specified item in the fields collection. Example: itemfields = rec.Fields.Item(1) or itemfields = rec.Fields.Item("Name")

ADO Recordset Object

Examples

For Example -> Refer CD

Recordset Object

The ADO Recordset object is used to hold a set of records from a database table. A Recordset object consist of records and columns (fields).

In ADO, this object is the most important and the most used object to manipulate data from a database.

ProgID

```
set objRecordset=Server.CreateObject("ADODB.recordset")
```

When you first open a Recordset, the current record pointer will point to the first record and the BOF and EOF properties are False. If there are no records, the BOF and EOF property are True.

Recordset objects can support two types of updating:

- **Immediate updating** - all changes are written immediately to the database once you call the Update method.
- **Batch updating** - the provider cache multiple changes and then send them to the database with the UpdateBatch method.

In ADO there are 4 different cursor types defined:

- **Dynamic cursor** - Allows you to see additions, changes, and deletions by other users.
- **Keyset cursor -** Like a dynamic cursor, except that you cannot see additions by other users, and it prevents access to records that other users have deleted. Data changes by other users will still be visible.
- **Static cursor** - Provides a static copy of a recordset for you to use to find data or generate reports. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side Recordset object.
- **Forward-only cursor** - Allows you to only scroll forward through the Recordset. Additions, changes, or deletions by other users will not be visible.

The cursor type can be set by the CursorType property or by the CursorType parameter in the Open method.

Note: Not all providers support all methods or properties of the Recordset object.

Properties

Property	Description
AbsolutePage	Sets or returns a value that specifies the page number in the Recordset object
AbsolutePosition	Sets or returns a value that specifies the ordinal position of

	the current record in the Recordset object
ActiveCommand	Returns the Command object associated with the Recordset
ActiveConnection	Sets or returns a definition for a connection if the connection is closed, or the current Connection object if the connection is open
BOF	Returns true if the current record position is before the first record, otherwise false
Bookmark	Sets or returns a bookmark. The bookmark saves the position of the current record
CacheSize	Sets or returns the number of records that can be cached
CursorLocation	Sets or returns the location of the cursor service
CursorType	Sets or returns the cursor type of a Recordset object
DataMember	Sets or returns the name of the data member that will be retrieved from the object referenced by the DataSource property
DataSource	Specifies an object containing data to be represented as a Recordset object
EditMode	Returns the editing status of the current record
EOF	Returns true if the current record position is after the last record, otherwise false
Filter	Sets or returns a filter for the data in a Recordset object
Index	Sets or returns the name of the current index for a Recordset object
LockType	Sets or returns a value that specifies the type of locking when editing a record in a Recordset
MarshalOptions	Sets or returns a value that specifies which records are to be returned back to the server
MaxRecords	Sets or returns the maximum number of records to return to a Recordset object from a query
PageCount	Returns the number of pages with data in a Recordset object
PageSize	Sets or returns the maximum number of records allowed on a single page of a Recordset object
RecordCount	Returns the number of records in a Recordset object
Sort	Sets or returns the field names in the Recordset to sort on
Source	Sets a string value or a Command object reference, or returns a String value that indicates the data source of the Recordset object
State	Returns a value that describes if the Recordset object is open, closed, connecting, executing or retrieving data
Status	Returns the status of the current record with regard to

	batch updates or other bulk operations
StayInSync	Sets or returns whether the reference to the child records will change when the parent record position changes

Methods

Method	Description
AddNew	Creates a new record
Cancel	Cancels an execution
CancelBatch	Cancels a batch update
CancelUpdate	Cancels changes made to a record of a Recordset object
Clone	Creates a duplicate of an existing Recordset
Close	Closes a Recordset
CompareBookmarks	Compares two bookmarks
Delete	Deletes a record or a group of records
Find	Searches for a record in a Recordset that satisfies a specified criteria
GetRows	Copies multiple records from a Recordset object into a two-dimensional array
GetString	Returns a Recordset as a string
Move	Moves the record pointer in a Recordset object
MoveFirst	Moves the record pointer to the first record
MoveLast	Moves the record pointer to the last record
MoveNext	Moves the record pointer to the next record
MovePrevious	Moves the record pointer to the previous record
NextRecordset	Clears the current Recordset object and returns the next Recordset object by looping through a series of commands
Open	Opens a database element that gives you access to records in a table, the results of a query, or to a saved Recordset
Requery	Updates the data in a Recordset by re-executing the query that made the original Recordset
Resync	Refreshes the data in the current Recordset from the original database
Save	Saves a Recordset object to a file or a Stream object
Seek	Searches the index of a Recordset to find a record that matches the specified values
Supports	Returns a boolean value that defines whether or not a Recordset object supports a specific type of functionality

Update	Saves all changes made to a single record in a Recordset object
UpdateBatch	Saves all changes in a Recordset to the database. Used when working in batch update mode

Events

Note: You cannot handle events using VBScript or JScript (only Visual Basic, Visual C++, and Visual J++ languages can handle events).

Event	Description
EndOfRecordset	Triggered when you try to move to a record after the last record
FetchComplete	Triggered after all records in an asynchronous operation have been fetched
FetchProgress	Triggered periodically in an asynchronous operation, to state how many more records that have been fetched
FieldChangeComplete	Triggered after the value of a Field object change
MoveComplete	Triggered after the current position in the Recordset has changed
RecordChangeComplete	Triggered after a record has changed
RecordsetChangeComplete	Triggered after the Recordset has changed
WillChangeField	Triggered before the value of a Field object change
WillChangeRecord	Triggered before a record change
WillChangeRecordset	Triggered before a Recordset change
WillMove	Triggered before the current position in the Recordset changes

Collections

Collection	Description
Fields	Indicates the number of Field objects in the Recordset object
Properties	Contains all the Property objects in the Recordset object

The Fields Collection's Properties

Property	Description
Count	Returns the number of items in the fields collection. Starts at zero. Example: countfields = rs.Fields.Count

Item(named_item/number)	Returns a specified item in the fields collection. Example: itemfields = rs.Fields.Item(1) or itemfields = rs.Fields.Item("Name")

The Properties Collection's Properties

Property	Description
Count	Returns the number of items in the properties collection. Starts at zero. Example: countprop = rs.Properties.Count
Item(named_item/number)	Returns a specified item in the properties collection. Example: itemprop = rs.Properties.Item(1) or itemprop = rs.Properties.Item("Name")

ADO Stream Object

Stream Object (ADO version 2.5)

The ADO Stream Object is used to read, write, and manage a stream of binary data or text.

A Stream object can be obtained in three ways:

- From a URL pointing to a document, a folder, or a Record object
- By instantiating a Stream object to store data for your application
- By opening the default Stream object associated with a Record object

Syntax

```
objectname.property
objectname.method
```

Properties

Property	Description
CharSet	Sets or returns a value that specifies into which

	character set the contents are to be translated. This property is only used with text Stream objects (type is adTypeText)
EOS	Returns whether the current position is at the end of the stream or not
LineSeparator	Sets or returns the line separator character used in a text Stream object
Mode	Sets or returns the available permissions for modifying data
Position	Sets or returns the current position (in bytes) from the beginning of a Stream object
Size	Returns the size of an open Stream object
State	Returns a value describing if the Stream object is open or closed
Type	Sets or returns the type of data in a Stream object

Methods

Method	Description
Cancel	Cancels an execution of an Open call on a Stream object
Close	Closes a Stream object
CopyTo	Copies a specified number of characters/bytes from one Stream object into another Stream object
Flush	Sends the contents of the Stream buffer to the associated underlying object
LoadFromFile	Loads the contents of a file into a Stream object
Open	Opens a Stream object
Read	Reads the entire stream or a specified number of bytes from a binary Stream object
ReadText	Reads the entire stream, a line, or a specified number of characters from a text Stream object
SaveToFile	Saves the binary contents of a Stream object to a file
SetEOS	Sets the current position to be the end of the stream (EOS)
SkipLine	Skips a line when reading a text Stream
Write	Writes binary data to a binary Stream object
WriteText	Writes character data to a text Stream object