

The Request Object

The **Request** object retrieves the values that the client browser passed to the server during an HTTP request.

The ServerVariables Collection

The `Request.ServerVariables` collection contains standard information automatically sent by the browser for every request.

The Request.Cookies Collection

The **Cookies** collection enables you to retrieve the values of the cookies sent in an HTTP request.

The QueryString

The **QueryString** collection retrieves the values of the variables in the HTTP query string. The HTTP query string is specified by the values following the question mark (?). Several different processes can generate a query string

The Request.Form collection

When a browser submits form data using the post method, the ASP engine parses the raw HTTP form data and stores it in the `Request.Form` collection. The form collection, like other ASP collections, stores data by key. If a key has more than one value, the values are stored in indexed form.

Post vs.Get Method

You normally post form data using the POST method, but there's another method called GET that you can use to send data to the server. When you use the GET method, the browser appends form data to the URL. That means you can't retrieve it on the server using the `Request.Form` collection-you must use the `Request.QueryString` collection instead .

Forcing ASP Pages to Refresh – Response Object

ASP pages are, by nature, dynamic. In contrast, HTML pages are static. A great deal of work in the Web arena has gone into improving the response time for HTML pages. One of the best way to improve time is to move the page source closer to the requesting browser. To help do that, most browsers cache pages on your hard drive. The first time you load a page it may take a long time, because the browser must retrieve the page content from a remote server, but after the first request the browser caches the page. Subsequent requests for the same page load from the cache, so the page displays very quickly.

Similarly, many proxy servers also cache pages. When a browser makes a request through a proxy server, the proxy checks to see if it has a copy of the page. If so, it returns the page to the browser, avoiding a long round trip to the remote server.

Page caching works beautifully for static HTML pages, but doesn't work well at all for ASP pages. ASP pages often change because the back-end content has changed or the content changes in response to user input. If the browser displays a cached page, the page may be out of date, or may not respond properly to user input. The ASP server-side code does not run because the proxy intercepts the request before it reaches your server. The `Response` object has several properties that can help solve this problem.

Response.Expires Property

These relatively simple properties add response headers to the page. You use them to force the browser to request a new copy of your page rather than loading it from the browser's cache.

The `Response.Expires` property sets the length of time, in minutes, until the page contents are no longer valid. After the specified number of minutes, the page expires. Until the page expires, the browser can display a cached copy of the page and does not have to re-request the page from the server to display the page. For eg: `Response.expires=10` tell the browser that the content of the page is valid for up to 10 minutes.

Response Objects methods And properties**Buffer**

A temporary holding area in memory where information can be stored

The **Buffer** property indicates whether to buffer page output. When page output is buffered, the server does not send a response to the client until all of the server scripts on the current page have been processed, or until the **Flush** or **End** method has been called. The **Buffer** property cannot be set after the server has sent output to the client. For this reason, the call to **Response.Buffer** should be the first line of the .asp file.

Syntax:

Response.Buffer [= *flag*]

Parameters**Flag :**

Specifies whether or not to buffer page output. It can be one of the following values. FALSE:No buffering. This is the default value. The server sends output to the client as it is processed.

```
<%Response.Buffer=False%>
```

TRUE:The server does not send output to the client until all of the ASP scripts on the current page have been processed, or until the **Flush** or **End** method has been called.

```
<%Response.Buffer=True%>
```

If **Response.Buffer** has been set to TRUE, calling **Response.End** will flush the buffer. If you do not want output returned to the user, you should call

```
<%
Response.Clear
Response.End
%>
```

Flush

The **Flush** method sends buffered output immediately. This method will cause a run-time error if **Response.Buffer** has not been set to TRUE.

Syntax

Response.Flush

If the **Flush** method is called on an ASP page, the server does not honor Keep-Alive requests for that page.

End

The **End** method causes the Web server to stop processing the script and return the current result. The remaining contents of the file are not processed.

Syntax

Response.End

Clear

The **Clear** method erases any buffered HTML output. However, the **Clear** method only erases the response body; it does not erase response headers. You can use this method to handle error cases. Note that this method will cause a run-time error if

Response.Buffer has not been set to TRUE.

Syntax

Response.Clear

The Server Object

The server object allows you to determine information about the server and manipulate it through the server object properties and methods. The ASP Server object gives you high-level access to the server itself. Using its properties and methods, you **can create objects**, execute code in other ASP files, translate virtual paths to physical paths, and perform server-side redirects.

You saw how to use the Response.Redirect method to force the browser to request a different file from the server. In previous Asp versions, this was the only way to transfer execution from one file to another on the server.

The problem with the Response.Redirect method is that the redirect message must travel from the server to the browser, then to the server again. That's highly inefficient.

The Server.Transfer method performs a server-side redirect. That means that the redirect occurs entirely on the server. The browser requests a file, the server redirects the request to another file, and the browser receives the results. Because the browser is no longer involved in the redirect, the Server.Transfer method also maintains the contents of form and Querystring data, so they are available in the redirected script. Using the Server.Transfer method is much more efficient. You should preferentially use Server.Transfer over Response.Redirect with model ASP scripts.

How to time out a script:

When you run an ASP page on a server, your scripting can contain errors or the network may be slow. This can cause valuable server resources – such as memory, processing time, etc. to be occupied. To avoid this, servers always timeout a page after a certain time (usually 90 sec). This is defined in the settings of the Internet Information Server (IIS) where you set up a web-site.

This means that for 90 sec the page can continue running the script, however, if the script does not finish within that time, then an error occurs and the page needs to be refreshed

However, all scripts cant be short and sometimes you need to control this timeout period from your script, simply because its doing a huge job, while other pages can stay within the default setting. You can control the servers timeout settings using the ASP engine through the *Server.ScriptTimeout* property with the following syntax:

`Server.ScriptTimeout = nnn (second)`

This way the ASP page will override the web settings and run the page for longer or shorter times, as you want. Add the following line to the script below the line `<% option explicit %>`. `<% Server.ScriptTimeout=10 %>`

The Server.Execute Method

Sometimes you may need to execute a script from within another script. You can do so by using the Server.Execute method. To execute a file, simply pass the filename to the Server.Execute method as `Server.Execute "some File.asp"`

Using server objects and components is a very powerful functionality of Web Applications.

When you create Web applications, you usually cannot control what browser the user has or what controls are registered on the user's computer. By installing controls on the server and using them in server script, you make the features of those controls available to any user, no matter what browser is in use.

The following table lists common server objects and components.

Object	Description
Request	Intrinsic IIS object that provides access to any information passed into the script through an HTTP request, such as form information, search strings, browser information, and information stored in cookies. For more information and examples of using the Request object, see Gathering Information Using Forms , Sharing Dynamic Information , and Creating Portable Script .
Response	Intrinsic IIS object that sends information to the user by writing information into a Web page stream or to the user's browser. For more information and examples of using the Response object, see Navigating Conditionally and Sharing Dynamic Information .
Session, Application	Intrinsic IIS objects that allow you to set and get values that persist between pages in your Web application. For more details, see Sharing Dynamic Information .
Server	Intrinsic IIS object that allows you to create instances of objects that are registered on the server, including bundled components and objects that you create.
AdRotator, BrowserCapability, TextStream, and NextLink	Components that are bundled with IIS, and allow you to display a changing set of images, store and get information about specific browsers, read and write to text files, and create an ordered path through pages.
ActiveX Data Objects (ADO)	Bundled components that allow you to connect to and query databases. For information and examples, see Database Tasks , Database Samples , and

When you are writing server script, you cannot directly manipulate client objects such as the browser window,

an HTML form, Java applets, or DHTML objects,

because these objects do not reside on the server.

If you are working with the server's intrinsic controls such as the Request or Response objects, you can simply

reference the objects in your script,

However, for all other objects, like the components you saw in the table that comes with IIS you create the object. You can do so by using the CreateObject method of the Server object

The CreateObject method of the Server Object creates an instance of a server component. A component is An application or development tool that can use objects supplied by another application, or which exposes its own objects for use by another application. A component is usually a dll or an executable file. A component contains class definitions, objects and their methods and properties which a web application can use.

Syntax

Server.CreateObject(progID)

Parameters

progID

Specifies the type of object to create

The *progID* argument is usually the fully qualified class name of the object being created;.

Session object

State:

One of the most powerful functionality of ASP applications which makes it very powerful on the internet is the ability to remember where a person is in a web application and what that person is doing. Remembering that information is called state maintenance because you are keeping track (maintaining) the persons current state with your application.

What happens in an HTTP request

In the HTTP protocol All interaction between a browser and a Web server takes place in discrete request and response pairs.

Nothing in the HTTP protocol allows a server to keep track of the users making the requests. In an HTTP request the server cannot keep track of which browser is requesting a page. After a server has finished responding to a request, the server can't continue to identify the browser that made it. From the perspective of a Web server, each new request is made by a new individual. For this reason, the HTTP protocol is called a stateless protocol.

The HTTP protocol can't be used to retain the state of a user. This is a very serious limitation because it means that you can't identify a user over multiple Web pages.

Sessions were introduced to fix this problem.

A session is something that starts the moment a user requests a page from the server

Session is the interval a single user spends interacting with your web application. A session begins when a browser instance requests a page from the server.

How does a web server keep track or identify a user session?

ASP Sessions use cookies to identify and keep track of user sessions. cookie is an item of information that the web server instructs the browser to store. the browser then sends the cookie back to the web server with each subsequent Request to the same server.

When a user first requests a page from the server, the server creates a single cookie in the user's browser to track the session.

Cookie is a small file that the server stores on the users computer to maintain user information.

HTTP transactions are stateless; therefore the server gives the browser a unique token in response to its first request so the browser can identify itself to the server for subsequent requests.

It gives each browser a Session ID cookie, and keeps the matching SessionID value.

The SessionID cookie created for each user is named ASPSESSIONID. The only purpose of this cookie is to provide a unique identifier for each user. ASP uses the ASPSESSIONID cookie to map individual browsers to data associated with an individual session.

Any information you want to store about that browser gets stored with the server's copy of the ASPSESSIONID. The next time the browser requests a page, the server can find the information associated with the server's copy of the ASPSESSIONID.

If the browser accepts the cookie, it will return the cookie on all subsequent requests to that site. The ASP engine keeps a list of the cookies. When the browser sends the ASPSESSIONID cookie, the ASP engine matches the cookie value with a server-side list.

Why Is the SessionID Cookie Important?

That's why the SessionID cookie is important- with it, the server can recognize the browser and can store and retrieve associated values. Without it, the browser is unknown, cannot be recognized as a repeat visitor, and therefore the server cannot retrieve associated values for that browser.

Storing Information on the User's Computer

A convenient way to maintain information about a user is to use a cookie. When a session is first started, the server sends a cookie to the client browser. Each time the client browser requests a page from that server, it sends the cookie back to the server, which can then read the cookie and identify the client browser.

You can use cookies to store your own application information, such as user preferences. Cookies are available until the date specified in the cookies' Expires attribute.

To store information in a cookie

- To put information into the cookie, use the Response.Cookies collection. To get information out of a cookie, use the Request.Cookies collection. For example:
- `<% Response.Cookies ("FavoriteColor")="Red" %>`
- `<% txtFavorite = Request.Cookies("FavoriteColor")%>`

Cookies can store multiple values. Each value in the cookie is assigned a key by which you identify it.

To store a value in a cookie, you use the Response object, specifying the name of the cookie to update, and the value. If the cookie does not already exist, the Response object creates it.

```
<% Response.Cookies("Preferences")("FavoriteColor")="Red" %>
```

How does the server know when a session ends? In other words, how does the server know whether a user has left your Web site for another one or has turned off his browser..

The server assumes that if someone hasn't requested or refreshed a page for a period of more than 20 minutes, that person has left, and times out that user's session. This strategy allows the server to recover resources that it has been using to track the user's session.

A web application can control the maximum amount of time for which a user is allowed to be inactive before a session times out.

Now that ASP extends the Web server to differentiate between the same page request by one browser and another browser, you'll find that you quickly want the Web server to remember what a person has already requested.

For example, if you create a page that asks users to enter their name, those users have a right to expect that you will remember that name in future pages, at minimum for the entire session. You can store the name in a Session object variable. Now you can retrieve the name on any subsequent page request and feel confident that you won't accidentally display the wrong name to the wrong user.

Session Object

You use the session object to store values associated with a single-browser. You can use the **Session** object to store information needed for a particular user-session

The session object has the following properties and Methods

SessionID : Returns the session identifier of the user.

Abandon : Destroys a Session object and releases its resources.

Timeout : The length of time, in minutes, of the Timeout period.

The Web server automatically creates a **Session** object when a Web page from the application is requested by a user who does not already have a session. The server destroys the **Session** object when the session expires or is abandoned.

Variables stored in the **Session** object are not discarded when the user jumps between pages in the application; instead, these variables persist for the entire user-session.

SessionID: The **SessionID** property returns the session identification for this user. Each session has a unique identifier that is generated by the server when the session is created. The session ID is returned as a **LONG** data type.

Syntax

Session.SessionID

Abandon : The **Abandon** method destroys all the objects stored in a **Session** object and releases their resources. If you do not call the **Abandon** method explicitly, the server destroys these objects when the session times out.

Syntax:

Session.Abandon

Remarks:

When the **Abandon** method is called, the current **Session** object is queued for deletion, but is not actually deleted until all of the script commands on the current page have been processed. This means that you can access variables stored in the **Session** object on the same page as the call to **Abandon**, but not in any subsequent Web pages.

Timeout: The **Timeout** property specifies the timeout period assigned to the **Session** object for this application, in minutes. If the user does not refresh or request a page within the timeout period, the session ends.

Syntax:

Session.Timeout [= *nMinutes*]

Parameters

nMinutes

Specifies the number of minutes that a session can remain idle before the server terminates it automatically. The default is 20 minutes.

The Application Object

- When you are writing applications, you may need to make information pertaining to an application available to all users of the application. For example, you might want to create variables that enable developers to customize an application for their site by setting a different greeting or changing the title font on all ASP pages. These changes will be seen by any user who visits the site.
- You can make data available to all users of an application from all pages of an application in several ways. The most frequently used is to give a variable or an object instance application scope by storing it in the ASP **Application** object.
- An ASP-based application is a collection of ASP pages and ActiveX components
- You can use the **Application** object to share information among all users of a given application. An ASP-based application is defined as all the .asp files in a virtual directory and its subdirectories

The Application object can store information like the Session object. When you store data in the Session object, the data is available only to that session. In contrast, data stored in the Application object is available to any page in the application; in other words, the information is global to the application.

Any session can access values in the Application object, but you need to ensure that only one session has permission to change the values at any given time. To do that, you lock the Application object before adding or changing a value, and unlock it when you have finished your modifications, for example:

```
Application.Lock
```

```
Application("myKey")=myValue
```

```
Application.Unlock
```

If you forget to lock the Application object, multiple scripts may try to change the Application object values all at the same time, causing errors. If you forget to unlock the Application object, it unlocks automatically when the ASP page completes.

Methods of the Application object

Syntax

Application.method

Methods:[../../../../AcadWeb/Student_Resources/IT/ASP_Session-Object_Tutorials/mk:@MSITStore:C:/Program Files/Microsoft Visual](#)

[Studio/MSDN98/98VS/1033/ASP.chm::/devdoc/good/iishelp/iis/htm/asp/introbj_84h7.htm](#)

Lock: The **Lock** method prevents other clients from modifying **Application** object properties.

Unlock: The **Unlock** method allows other clients to modify **Application** object properties.

- If you do not call the **Unlock** method explicitly, the server unlocks the locked **Application** object when the .asp file ends or times out.

The main difference is between the Application object and other ASP collection is that there's only one Application object for all users of your application. All the other collections are user-specific--- the Asp engine creates them for a single transaction or session

The File System Object Model

In ASP applications , most file access happens automatically.

When you reference a file, the server opens the file reads its contents , or executes it ; but Even in Web applications , you sometimes need to read information from or write information to a file

This is made possible through the File System object model

The FSO object model gives your applications the ability to create, alter, move, and delete folders,

It also enables you to gain information about folders, such as their names, the date they were created or last modified, and so forth.

The FSO object model has these objects:

Drive

Allows you to gather information about drives attached to the system, such as how much room is available, what its share name is, and so forth. Note that a "drive" isn't necessarily a hard disk. It can be a CD-ROM drive, a RAM disk, and so forth. Also, drives aren't required to be physically attached to the system; they can be also be logically connected through a LAN.

Folder

Allows you to create, delete, or move folders, plus query the system as to their names, paths, and so on.

Files

Allows you to create, delete, or move files, plus query the system as to their names, paths, and so on.

FileSystemObject

The main object of the group, full of methods that allow you to create, delete, gain information about, and generally manipulate drives, folders, and files. Many of the methods associated with this object duplicate those in the other objects.

TextStream

Enables you to read and write text files.

Programming in the FSO object model involves three main tasks:

- Use the CreateObject method, to create a FileSystemObject object.
- Using the appropriate method on the newly-created object.
- Accessing the object's properties.

Creating a FileSystemObject Object

The first step is to create a FileSystemObject object to work with. You can do this in two ways:

Dimension a variable :Dim fso

Use the CreateObject method to create a FileSystemObject object:

Set fso = CreateObject("Scripting.FileSystemObject")

- ▶ The next step is to use the appropriate method of the FileSystemObject object. For example, if you want to *create* a new object, you can use either CreateFolder or CreateTextFile

CreateTextFile Method:-

```
<%@ Language=VBScript %>
<% Option Explicit %>
<HTML><HEAD></HEAD>
<BODY>
<P>
</P>
<% Dim fso, a
Set fso = Server.CreateObject("Scripting.FileSystemObject")
Set a=fso.CreateTextFile(\\MachineName\DriveName\Filename,True)
<!For eg: Set a = fso.CreateTextFile("\\Pollux\C_Pollux\first.txt", True)_>
a.WriteLine("This is a test.")
a.Close
%>
</BODY></HTML>
```

If you want to delete objects, you can use the DeleteFile and DeleteFolder methods of the FileSystemObject object, or the Delete method of the File and Folder objects.

The Text Stream object

- ▶ Now whenever we talk of accessing files on the computer's system , the most essential operations involved are
- ▶ Creating files
- ▶ Reading files
- ▶ Writing to files
- ▶ For all the above operations we have to use the TextStreamObject

Let us consider the following code again

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set a = fso.CreateTextFile("c:\testfile.txt", True)a.WriteLine("This is a test.")
a.Close
```

Here CreateObject function returns the **FileSystemObject** (fso).

The **CreateTextFile** method then Creates a specified file name and returns a **TextStream** object that can be used to read from or write to the file.

Syntax

Object . **CreateTextFile**(*filename* [, *overwrite* [, *unicode*]])

The **CreateTextFile** method has these parts:

Part	Description
<i>object</i>	Required. Always the name of a FileSystemObject or Folder object.
<i>filename</i>	Required. <u>String expression</u> that identifies the file to create.
<i>overwrite</i>	Optional. Boolean value that indicates if an existing file can be overwritten. The value is True if the file can be overwritten; False if it can't be overwritten. If omitted, existing files are not overwritten.
<i>unicode</i>	Optional. Boolean value that indicates whether the file is created as a Unicode or ASCII file. The value is True if the file is created as a Unicode file; False if it's created as an ASCII file. If omitted, an ASCII file is assumed.

- ▶ In the following code, a is the **TextStream** object returned by the **CreateTextFile** method on the **FileSystemObject**:
- ▶ **WriteLine** and **Close** are two methods of the **TextStream** Object.

TextStream Object Methods

Close Method

Read Method

ReadAll Method

ReadLine Method

Skip Method

SkipLine Method

Write Method

WriteLine Method

WriteBlankLines Method

Syntax:

object.Close

Description: Closes an open **TextStream** file.

Syntax:

object.Read(characters)

Description: Reads a specified number of characters from a **TextStream** file and returns the resulting string.

The Read method syntax has these parts:

Object: Required. Always the name of a **TextStream** object.

Characters: Required. Number of characters you want to read from the file.

Syntax:

object.ReadAll

Description: Reads an entire **TextStream** file and returns the resulting string

Syntax:

object.ReadLine

Description: Reads an entire line (up to, but not including, the newline character) from a **TextStream** file and returns the resulting string.

Syntax:

object.Skip(characters)

Description:

Skips a specified number of characters when reading a **TextStream** file.

The Skip method syntax has these parts:

Object: Required. Always the name of a **TextStream** object.

Characters: Required. Number of characters to skip when reading a file.

Syntax:

object.SkipLine

Description:

Skips the next line when reading a **TextStream** file.

Syntax:

object.SkipALL

Description:

Skips all line when reading a **TextStream** file.

Syntax:

object.Write(string)

Description:

Writes a specified string to a **TextStream** file.

The Write method syntax has these parts:

Object: Required. Always the name of a **TextStream** object.

String: Required. The text you want to write to the file.

Syntax:

object.WriteLine([string])

Description:

Writes a specified string and newline character to a **TextStream** file.

The WriteLine method syntax has these parts:

Object: Required. Always the name of a **TextStream** object.

String: Optional. The text you want to write to the file. If omitted, a newline character is written to the file.

Syntax:

object.WriteBlankLines(lines)

Description:

Writes a specified number of newline characters to a **TextStream** file.

The WriteBlankLines method syntax has these parts:

Object: Required. Always the name of a **TextStream** object.

Lines: Required. Number of newline characters you want to write to the file.

OpenTextFile Method

Opens a specified file and returns a **TextStream** object that can be used to read from or append to the file.

Syntax

object.OpenTextFile(filename[, iomode[, create[, format]])

The **OpenTextFile** method has these parts:

Filename: Required. String expression that identifies the file to open.

Iomode: Optional. Indicates input/output mode. Can be one of two constants, either **ForReading** or

ForAppending.

Create: Optional. **Boolean** value that indicates whether a new file can be created if the specified *filename* doesn't exist. The value is **True** if a new file is created; **False** if it isn't created. The default is **False**.

Format: Optional. One of three **Tristate** values used to indicate the format of the opened file. If omitted, the file is opened as ASCII.

AtEndOfStream Property

Syntax:

object.AtEndOfStream

The *object* is always the name of a **TextStream** object.

Description: Returns **True** if the file pointer is at the end of a **TextStream** file; **False** if it is not. Read-only.

Line Property

Syntax:

object.Line

The *object* is always the name of a **TextStream** object.

Description:Read-only property that returns the current line number in a **TextStream** file.

The combination of the Request object, which contains information sent by the browser; the Response object, which lets you send data back to the browser; the Application object; the Session object; and access to databases, almost completes the requirements for a system able to provide personalized pages even when the information to be displayed changes rapidly. The only remaining requirements for ASP to be equal of a local executable application are access to other components and the host computer's file system. As you probably expect by now, ASP provides those capabilities as well.

Document Object Model through DHTML

The Dynamic HTML (DHTML) Document Object Model (DOM) allows authors direct, programmable access to the individual components of their Web documents, this model, allows the browser to react to user input, execute scripts on the fly, and display the new content without downloading additional documents from a server. According to the Document object model every HTML element is programmable.

This means that every HTML element on the page can have script behind it that can be used to interact with user actions and change the page content dynamically. This model lets a document react when the user has done something on the page, such as **move the mouse over a particular element**

Press a key, or enter information into a form input. Each event can be linked to a script that tells the browser to modify the content on the fly, without having to go back to the server for a new file.

The advantages to this are that authors will be able to create interactive Web sites with fewer pages, and users will not have to wait for new pages to download from Web servers, increasing the speed of their browsing and the performance of the Internet as a whole.

The Document Object Model gives us access to every element in a document.

Every HTML element is a scriptable object in the object model, with its own set of properties, methods, and events.

Object		Description
Window	methods properties	The window object is the top level object in the DHTML DOM. It contains information about the window and the frames
document	methods properties collections	The document object represents the HTML document, and is used to access the HTML elements inside the document
navigator	properties	The navigator object contains information about the user's browser
Event	properties collections	The event object contains information about events that occurs

The Document Object

Represents the HTML document in a given browser window .

It supports the following properties and methods.

Properties

Property	Description
Anchors	Returns a collection of all anchor elements in a document
Applets	Returns a collection of all object elements included in an applet
Body	Returns the body or frameset element
Cookie	Returns the cookies for the document
Domain	Returns the domain name of the document's server
Forms	Returns a collection of all the form elements in a document
Images	Returns a collection of all the image elements in a document
Links	Returns a collection of all the anchor elements in the document with the href attribute specified
Referrer	Returns the URL of the previous page
Title	Sets or returns the title of this document
URL	Returns the URL of the document

Methods

Method	Description
close()	Closes a document that was opened with the document.open() method
getElementById(ID)	Returns the element with a specified ID
getElementsByName(name)	Returns a collection of all elements with a specified name
open()	Opens a new document
write(text)	Writes a text to a document
writeln(text)	Writes a line of text to a document

