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

Cookies

The Cookies collection is used to set or get cookie values. If the cookie does not exist, it will be created, and take the value that is specified.

Syntax

```
Response.Cookies(name)[(key)|.attribute]=value

variablename=Request.Cookies(name)[(key)|.attribute]
```

Parameter	Description
Name	Required. The name of the cookie
Value	Required for the Response.Cookies command. The value of the cookie
Attribute	Optional. Specifies information about the cookie. Can be one of the following parameters: • Domain - Write-only. The cookie is sent only to requests to this domain • Expires - Write-only. The date when the cookie expires. If no date is specified, the cookie will expire when the session ends • HasKeys - Read-only. Specifies whether the cookie has keys (This is the only attribute that can be used with the Request.Cookies command) • Path - Write-only. If set, the cookie is sent only to requests to this path. If not set, the application path is used • Secure - Write-only. Indicates if the cookie is secure
Key	Optional. Specifies the key to where the value is assigned

Examples

The "Response.Cookies" command is used to create a cookie or to set a cookie value:

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

In the code above, we have created a cookie named "firstname" and assigned the value "Alex" to it.

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

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

Now the cookie named "firstname" has the value of "Alex", and it will expire from the user's computer at May 10, 2002.
The "Request.Cookies" command is used to get a cookie value.
In the example below, we retrieve the value of the cookie "firstname" and display it on a page:

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

Output:

```
Firstname=Alex
```

A cookie can also contain 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"
%>
```

The code below reads all the cookies your server has sent to a user. Note that the code 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
```

Properties

1. Buffer

The Buffer property specifies whether to buffer the output or not. When the output is buffered, the server will hold back the response to the browser until all of the server scripts have been processed, or until the script calls the Flush or End method.

Note: If this property is set, it should be before the <html> tag in the .asp file

Syntax

response.Buffer[=flag]

Parameter	**Description**
Flag	A boolean value that specifies whether to buffer the page output or not. False indicates no buffering. The server will send the output as it is processed. False is default for IIS version 4.0 (and earlier). Default for IIS version 5.0 (and later) is true. True indicates buffering. The server will not send output until all of the scripts on the page have been processed, or until the Flush or End method has been called.

Examples

Example 1

In this example, there will be no output sent to the browser before the loop is finished. If buffer was set to False, then it would write a line to the browser every time it went through the loop.

```
<%response.Buffer=true%>
<html>
<body>
<%
for i=1 to 100
  response.write(i & "<br />")
next
%>
</body>
</html>
```

Example 2

```
<%response.Buffer=true%>
<html>
<body>
<p>I write some text, but I will control when
the text will be sent to the browser.</p>
<p>The text is not sent yet. I hold it back!</p>
<p>OK, let it go!</p>
<%response.Flush%>
</body>
</html>
```

Example 3

```
<%response.Buffer=true%>
<html>
<body>
<p>This is some text I want to send to the user.</p>
<p>No, I changed my mind. I want to clear the text.</p>
<%response.Clear%>
</body>
</html>
```

2. ContentType

The ContentType property sets the HTTP content type for the response object.

Syntax

```
response.ContentType[=contenttype]
```

Parameter	**Description**
Contenttype	A string describing the content type. For a full list of content types, see your browser documentation or the HTTP specification.

Examples

If an ASP page has no ContentType property set, the default content-type header would be:

content-type:text/html

Some other common ContentType values:

```
<%response.ContentType="text/HTML"%>
<%response.ContentType="image/GIF"%>
<%response.ContentType="image/JPEG"%>
<%response.ContentType="text/plain"%>
```

This example will open an Excel spreadsheet in a browser (if the user has Excel installed):

```
<%response.ContentType="application/vnd.ms-excel"%>
<html>
<body>
<table>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</table>
</body>
</html>
```

3. Expires

The Expires property sets how long (in minutes) a page will be cached on a browser before it expires. If a user returns to the same page before it expires, the cached version is displayed.

Syntax

```
response.Expires[=number]
```

Parameter	**Description**
Number	The time in minutes before the page expires

Examples

Example 1

The following code indicates that the page will never be cached:

```
<%response.Expires=-1%>
```

Example 2

The following code indicates that the page will expire after 1440 minutes (24 hours):

```
<%response.Expires=1440%>
```

4. <u>ExpiresAbsolute</u>

he ExpiresAbsolute property sets a date and time when a cached page on a browser will expire. If a user returns to the same page before this date/time, the cached version is displayed.

Syntax
response.ExpiresAbsolute[=[date][time]]

Parameter	**Description**
Date	Specifies the date on which the page will expire. If this parameter is not specified, the page will expire at the specified time on the day that the script is run.
Time	Specifies the time at which the page will expire. If this parameter is not specified, the page will expire at midnight of the specified day.

Examples

The following code indicates that the page will expire at 4:00 PM on October 11, 2012:

```
<%response.ExpiresAbsolute=#October 11,2012 16:00:00#%>
```

5. <u>Status</u>

The Status property specifies the value of the status line returned by the server.

Tip: Use this property to modify the status line returned by the server.

Syntax
response.Status=statusdescription

Parameter	Description
statusdescription	A three-digit number and a description of that code, like 404 Not Found. **Note:** Status values are defined in the HTTP specification.

Examples

```
<%
ip=request.ServerVariables("REMOTE_ADDR")
if ip<>"194.248.333.500" then
  response.Status="401 Unauthorized"
  response.Write(response.Status)
  response.End
end if
%>
```

Methods

1.Clear

The Clear method clears any buffered HTML output.

Note: This method does not clear the response headers, only the response body.

Note: If response.Buffer is false, this method will cause a run-time error.

Syntax

```
response.Clear
```

Examples

```
<%
response.Buffer=true
%>
<html>
<body>
<p>This is some text I want to send to the user.</p>
<p>No, I changed my mind. I want to clear the text.</p>
<%
response.Clear
%>
</body>
</html>
```

Output:

```
(nothing)
```

2.End

The End method stops processing a script, and returns the current result.

Note: This method will flush the buffer if Response.Buffer has been set to true. If you do not want to return any output to the user, you should call Response.Clear first.

Syntax

```
Response.End
```

Examples

```
<html>
<body>
<p>I am writing some text. This text will never be
<%
Response.End
%>
finished! It's too late to write more!</p>
</body>
</html>
```

Output:

```
I am writing some text. This text will never be
```

3. <u>Flush</u>

The Flush method sends buffered HTML output immediately.

Note: If response.Buffer is false, this method will cause a run-time error.

Syntax

```
Response.Flush
```

Example

```
<%
Response.Buffer=true
%>
<html>
<body>
<p>I write some text, but I will control when the
text will be sent to the browser.</p>
<p>The text is not sent yet. I hold it back!</p>
<p>OK, let it go!</p>
<%
Response.Flush
%>
</body>
</html>
```

Output:

```
I write some text, but I will control when the
text will be sent to the browser.
```

The text is not sent yet. I hold it back!

OK, let it go!

4. Redirect

The Redirect method redirects the user to a different URL.

Syntax
Response.Redirect URL

Parameter	**Description**
URL	Required. The URL that the user (browser) is redirected to

Examples
```
<%
Response.Redirect "http://cloudfactory.com"
%>
```

5. Write

The Write method writes a specified string to the output.

Syntax

Response.Write variant/variable

Parameter	**Description**
Variant	Required. The data to write

Examples

Example 1
```
<%
Response.Write "Hello World"
%>
```

Output:

```
Hello World
```

Example 2
```
<%
name="John"
Response.Write(name)
%>
```

Output:

```
John
```

Example 3
```
<%
Response.Write("Hello<br />World")
%>
```

Output:

```
Hello
World
```

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 a visitor. Its collections, properties, and methods are described below:

Collections

1.**Cookies**(Refer to Response Object)

2.**Form**

The Form collection is used to retrieve the values of form elements from a form that uses the POST method.

Syntax

Request.Form(element)[(index)|.Count]

Parameter	**Description**
Element	Required. The name of the form element from which the collection is to retrieve values
Index	Optional. Specifies one of multiple values for a parameter. From 1 to Request.Form(parameter).Count.

Examples

Example 1

You can loop through all the values in a form request. If a user filled out a form by specifying two values - Blue and Green - for the color element, you could retrieve those values like this:

```
<% for i=1 to Request.Form("color").Count
  Response.Write(Request.Form("color")(i) & "<br />")
next
%>
```

Output:

```
Blue
Green
```

Example 2

Consider the following form:

```
<form action="submit.asp" method="post">
<p>First name: <input name="firstname"></p>
<p>Last name: <input name="lastname"></p>
<p>Your favorite color:
<select name="color">
<option>Blue</option>
<option>Green</option>
<option>Red</option>
<option>Yellow</option>
<option>Pink</option>
</select>
</p>
<p><input type="submit"></p>
</form>
```

The following request might be sent:

```
firstname=John&lastname=Dove&color=Red
```

Now we can use the information from the form in a script:

```
Hi, <%=Request.Form("firstname")%>.
Your favorite color is <%=Request.Form("color")%>.
```

Output:

```
Hi, John. Your favorite color is Red.
```

If you do not specify any element to display, like this:

Form data is: <%=Request.Form%>

the output would look like this:

Form data is: firstname=John&lastname=Dove&color=Red

3.<u>QueryString:</u>

The QueryString collection is used to retrieve the variable values in the HTTP query string.

The HTTP query string is specified by the values following the question mark (?), like this:

<a href= "test.asp?txt=this is a query string test">Link with a query string</a>

The line above generates a variable named txt with the value "this is a query string test".

Query strings are also generated by form submission, or by a user typing a query into the address bar of the browser.

Note: If you want to send large amounts of data (beyond 100 kb) the Request.QueryString cannot be used.

Syntax

Request.QueryString(variable)[(index)|.Count]

Parameter	**Description**
Variable	Required. The name of the variable in the HTTP query string to retrieve
Index	Optional. Specifies one of multiple values for a variable. From 1 to Request.QueryString(variable).Count

Examples

Example 1

To loop through all the n variable values in a Query String:

The following request is sent:

http://www.w3schools.com/test/names.asp?n=John&n=Susan

and names.asp contains the following script:

```
<%
for i=1 to Request.QueryString("n").Count
  Response.Write(Request.QueryString("n")(i) & "<br />")
next
%>
```

The file names.asp would display the following:

```
John
Susan
```

Example 2

The following string might be sent:

http://cloudfactory.com/names.asp?name=John&age=30

this results in the following QUERY_STRING value:

name=John&age=30

Now we can use the information in a script:

```
Hi, <%=Request.QueryString("name")%>.
Your age is <%= Request.QueryString("age")%>.
```

Output:

```
Hi, John. Your age is 30.
```

If you do not specify any variable values to display, like this:

```
Query string is: <%=Request.QueryString%>
```

the output would look like this:

```
Query string is: name=John&age=30
```

4. <u>ServerVariables:</u>

The ServerVariables collection is used to retrieve the server variable values.

Syntax

Request.ServerVariables (server_variable)

Parameter	**Description**

server_variable	Required. The name of the server variable to retrieve

Server Variables

Variable	**Description**
ALL_HTTP	Returns all HTTP headers sent by the client. Always prefixed with HTTP_ and capitalized
ALL_RAW	Returns all headers in raw form
APPL_MD_PATH	Returns the meta base path for the application for the ISAPI DLL
APPL_PHYSICAL_PATH	Returns the physical path corresponding to the meta base path
AUTH_PASSWORD	Returns the value entered in the client's authentication dialog
AUTH_TYPE	The authentication method that the server uses to validate users
AUTH_USER	Returns the raw authenticated user name
CERT_COOKIE	Returns the unique ID for client certificate as a string
CERT_FLAGS	bit0 is set to 1 if the client certificate is present and bit1 is set to 1 if the cCertification authority of the client certificate is not valid
CERT_ISSUER	Returns the issuer field of the client certificate
CERT_KEYSIZE	Returns the number of bits in Secure Sockets Layer connection key size
CERT_SECRETKEYSIZE	Returns the number of bits in server certificate private key
CERT_SERIALNUMBER	Returns the serial number field of the client certificate
CERT_SERVER_ISSUER	Returns the issuer field of the server certificate
CERT_SERVER_SUBJECT	Returns the subject field of the server certificate
CERT_SUBJECT	Returns the subject field of the client certificate
CONTENT_LENGTH	Returns the length of the content as sent by the client
CONTENT_TYPE	Returns the data type of the content
GATEWAY_INTERFACE	Returns the revision of the CGI specification used by the server

HTTP_<*HeaderName*>	Returns the value stored in the header *HeaderName*
HTTP_ACCEPT	Returns the value of the Accept header
HTTP_ACCEPT_LANGUAGE	Returns a string describing the language to use for displaying content
HTTP_COOKIE	Returns the cookie string included with the request
HTTP_REFERER	Returns a string containing the URL of the page that referred the request to the current page using an <a> tag. If the page is redirected, HTTP_REFERER is empty
HTTP_USER_AGENT	Returns a string describing the browser that sent the request
HTTPS	Returns ON if the request came in through secure channel or OFF if the request came in through a non-secure channel
HTTPS_KEYSIZE	Returns the number of bits in Secure Sockets Layer connection key size
HTTPS_SECRETKEYSIZE	Returns the number of bits in server certificate private key
HTTPS_SERVER_ISSUER	Returns the issuer field of the server certificate
HTTPS_SERVER_SUBJECT	Returns the subject field of the server certificate
INSTANCE_ID	The ID for the IIS instance in text format
INSTANCE_META_PATH	The meta base path for the instance of IIS that responds to the request
LOCAL_ADDR	Returns the server address on which the request came in
LOGON_USER	Returns the Windows account that the user is logged into
PATH_INFO	Returns extra path information as given by the client
PATH_TRANSLATED	A translated version of PATH_INFO that takes the path and performs any necessary virtual-to-physical mapping
QUERY_STRING	Returns the query information stored in the string following the question mark (?) in the HTTP request
REMOTE_ADDR	Returns the IP address of the remote host making the request
REMOTE_HOST	Returns the name of the host making the request

REMOTE_USER	Returns an unmapped user-name string sent in by the user
REQUEST_METHOD	Returns the method used to make the request
SCRIPT_NAME	Returns a virtual path to the script being executed
SERVER_NAME	Returns the server's host name, DNS alias, or IP address as it would appear in self-referencing URLs
SERVER_PORT	Returns the port number to which the request was sent
SERVER_PORT_SECURE	Returns a string that contains 0 or 1. If the request is being handled on the secure port, it will be 1. Otherwise, it will be 0
SERVER_PROTOCOL	Returns the name and revision of the request information protocol
SERVER_SOFTWARE	Returns the name and version of the server software that answers the request and runs the gateway
URL	Returns the base portion of the URL

Examples

You can loop through all of the server variables like this:

```
<%
for each x in Request.ServerVariables
  response.write(x & "<br />")
next
%>
```

The following example demonstrates how to find out the visitor's browser type, IP address, and more:

```
<html>
<body>
<p>
<b>You are browsing this site with:</b>
<%Response.Write(Request.ServerVariables("http_user_agent"))%>
</p>
<p>
<b>Your IP address is:</b>
<%Response.Write(Request.ServerVariables("remote_addr"))%>
</p>
<p>
<b>The DNS lookup of the IP address is:</b>
<%Response.Write(Request.ServerVariables("remote_host"))%>
</p>
<p>
```

```
<b>The method used to call the page:</b>
<%Response.Write(Request.ServerVariables("request_method"))%>
</p>
<p>
<b>The server's domain name:</b>
<%Response.Write(Request.ServerVariables("server_name"))%>
</p>
<p>
<b>The server's port:</b>
<%Response.Write(Request.ServerVariables("server_port"))%>
</p>
<p>
<b>The server's software:</b>
<%Response.Write(Request.ServerVariables("server_software"))%>
</p>
</body>
</html>
```

Properties

TotalBytes

The TotalBytes property is a read-only property that returns the total number of bytes the client sent in the body of the request.

Syntax
varbytes=Request.Totalbytes

Example

The following code sets the variable a equal to the total number of bytes sent in the body of the request:

```
<%
dim a
a=Request.TotalBytes
%>
```

Server Object

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

Properties
ScriptTimeOut

The ScriptTimeout property sets or returns the maximum number of seconds a script can run before it is terminated.

Syntax

Server.ScriptTimeout[=NumSeconds]

Parameter	**Description**
NumSeconds	The maximum number of seconds a script can run before the server terminates it. Default is 90 seconds

Examples

Example 1

Set the script timeout:

```
<%
Server.ScriptTimeout=200
%>
```

Example 2

Retrieve the current value of the ScriptTimeout property:

```
<%
response.write(Server.ScriptTimeout)
%>
```

Methods

1.CreateObject

The CreateObject method creates an instance of an object.

Note: Objects created with this method have page scope. They are destroyed when the server are finished processing the current ASP page. To create an object with session or application scope, you can either use the <object> tag in the Global.asa file, or store the object in a session or application variable.

Syntax
Server.CreateObject(progID)

Part	**Description**
progID	Required. The type of object to create

Example 1

This example creates an instance of the server component MSWC.AdRotator:

```
<%
Set adrot=Server.CreateObject("MSWC.AdRotator")
%>
```

Example 2

An object stored in a session variable is destroyed when the session ends. However, you can also destroy the object by setting the variable to Nothing or to a new value:

```
<%
Session("ad")=Nothing
%>
```

or

```
<%
Session("ad")="a new value"
%>
```

Example 3

You cannot create an instance of an object with the same name as a built-in object:

```
<%
Set Application=Server.CreateObject("Application")
%>
```

2. Execute

The Execute method executes an ASP file from inside another ASP file. After executing the called .asp file, the control is returned to the original .asp file.

Syntax
Server.Execute(path)

Parameter	**Description**
Path	Required. The location of the ASP file to execute

Example
File1.asp:

```
<%
response.write("I am in File 1!<br />")
```

```
Server.Execute("file2.asp")
response.write("I am back in File 1!")
%>
```

File2.asp:

```
<%
response.write("I am in File 2!<br />")
%>
```

Output:

```
I am in File 1!
I am in File 2!
I am back in File 1!
```

Also look at the Server.Transfer method to see the difference between the Server.Execute and Server.Transfer methods.

3.<u>GetLastError</u>

The GetLastError method returns an ASPError object that describes the error condition that occurred.

By default, a Web site uses the file \iishelp\common\500-100.asp for processing ASP errors. You can either use this file, or create your own. If you want to change the ASP file for processing the 500;100 custom errors you can use the IIS snap-in.

Note: A 500;100 custom error will be generated if IIS encounters an error while processing either an ASP file or the application's Global.asa file.

Note: This method is available only before the ASP file has sent any content to the browser.

Syntax
Server.GetLastError()

Examples

Example 1

In the example an error will occur when IIS tries to include the file, because the include statement is missing the file parameter:

```
<!--#include f="header.inc" -->
<%
response.write("sometext")
%>
```

Example 2

In this example an error will occur when compiling the script, because the "next" keyword is missing:

```
<%
dim i
for i=1 to 10
  ........
nxt
%>
```

Example 3

In this example an error will occur because the script attempts to divide by 0:

```
<%
dim i,tot,j
i=0
tot=0
j=0

for i=1 to 10
  tot=tot+1
next

tot=tot/j
%>
```

4. MapPath

The MapPath method maps a specified path to a physical path.

Note: This method cannot be used in Session.OnEnd and Application.OnEnd.

Syntax

Server.MapPath(path)

Parameter	**Description**
Path	Required. A relative or virtual path to map to a physical path. If this parameter starts with / or \, it returns a path as if this parameter is a full virtual path. If this parameter doesn't start with / or \, it returns a path relative to the directory of the .asp file being processed

Examples

Example 1

For the example below, the file "test.asp" is located in C:\Inetpub\Wwwroot\Script.

The file "test.asp" (located in C:\Inetpub\Wwwroot\Script) contains the following code:

```
<%
response.write(Server.MapPath("test.asp") & "<br />")
response.write(Server.MapPath("script/test.asp") & "<br />")
response.write(Server.MapPath("/script/test.asp") & "<br />")
response.write(Server.MapPath("\script") & "<br />")
response.write(Server.MapPath("/") & "<br />")
response.write(Server.MapPath("\") & "<br />")
%>
```

Output:

```
c:\inetpub\wwwroot\script\test.asp
c:\inetpub\wwwroot\script\script\test.asp
c:\inetpub\wwwroot\script\test.asp
c:\inetpub\wwwroot\script
c:\inetpub\wwwroot
c:\inetpub\wwwroot
```

Example 2

How to use a relative path to return the relative physical path to the page that is being viewed in the browser:

```
<%
response.write(Server.MapPath("../"))
%>
```

or

```
<%
response.write(Server.MapPath("..\"))
%>
```

5. Transfer

The Transfer method sends (transfers) all the state information (all application/session variables and all items in the request collections) created in one ASP file to a second ASP file.

When the second ASP page completes its tasks, it will NOT return to the first ASP page (like the Execute method).

Note: The Transfer method is an efficient alternate for the Response.Redirect. A redirect forces the Web server to handle an extra request while the Server.Transfer method transfers execution to a different ASP page on the server, and avoids the extra round trip.

Syntax
Server.Transfer(path)

Parameter	**Description**
Path	Required. The location of the ASP file to which control should be transferred

Example
File1.asp:

```
<%
response.write("Line 1 in File 1<br />")
Server.Transfer("file2.asp")
response.write("Line 2 in File 1<br />")
%>
```

File2.asp:

```
<%
response.write("Line 1 in File 2<br />")
response.write("Line 2 in File 2<br />")
%>
```

Output:

```
Line 1 in File 1
Line 1 in File 2
Line 2 in File 2
```

<u>Application Object</u>

An application on the Web may consists of several ASP files that work together to perform some purpose. The Application object 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 (with Sessions there is ONE Session object for EACH user).

The Application object holds information that will be used by many pages in the application (like database connection information). The information can be accessed from any page. The information can also be changed in one place, and the changes will automatically be reflected on all pages.

The Application object is initialized by IIS when the first .asp page from within the given virtual directory is requested. It remains in the server's memory until either the web service is stopped or the application is explicitly unloaded from the web server

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

Collections

1.Contents

The Contents collection contains all the items appended to the application/session through a script command.

Tip: To remove items from the Contents collection, use the Remove and RemoveAll methods.

Syntax

Application.Contents(Key)

Parameter	**Description**
Key	Required. The name of the item to retrieve

Example 1

Notice that both name and objtest would be appended to the Contents collection:

```
<%
Application("name")="Harry"
Set Application("objtest")=Server.CreateObject("ADODB.Connection")
%>
```

Example 2

To loop through the Contents collection:

```
<%
for each x in Application.Contents
  Response.Write(x & "=" & Application.Contents(x) & "<br />")
next
%>
```

or:

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

Example 3

```
<%
Application("date")="2001/05/05"
Application("author")="Michael"

for each x in Application.Contents
  Response.Write(x & "=" & Application.Contents(x) & "<br />")
```

```
next
%>
```

Output:

```
date=2001/05/05
author= Michael
```

2.<u>StaticObjects</u>

The StaticObjects collection contains all the objects appended to the application/session with the HTML <object> tag.

Syntax

```
Application.StaticObjects(Key)
```

Parameter	Description
key	Required. The name of the item to retrieve

Example 1

To loop through the StaticObjects collection:

```
<%
for each x in Application.StaticObjects
  Response.Write(x & "<br />")
next
%>
```

Example 2

In Global.asa:

```
<object runat="server" scope="application"
id="MsgBoard" progid="msgboard.MsgBoard">
</object>

<object runat="server" scope="application"
id="AdRot" progid="MSWC.AdRotator">
</object>
```

In an ASP file:

```
<%
for each x in Application.StaticObjects
  Response.Write(x & "<br />")
next
%>
```

Output:

```
MsgBoard
AdRot
```

Methods
1. Contents.Remove

The Contents.Remove method deletes an item from the Contents collection.

Syntax

```
Application.Contents.Remove(name|index)
```

Parameter	Description
name	The name of the item to remove
index	The index of the item to remove

Example 1

```
<%
Application("test1")=("First test")
Application("test2")=("Second test")
Application("test3")=("Third test")

Application.Contents.Remove("test2")

for each x in Application.Contents
  Response.Write(x & "=" & Application.Contents(x) & "<br />")
next
%>
```

Output:

```
test1=First test
test3=Third test
```

Example 2

```
<%
Application("test1")=("First test")
Application("test2")=("Second test")
Application("test3")=("Third test")

Application.Contents.Remove(2)
```

```
for each x in Application.Contents
  Response.Write(x & "=" & Application.Contents(x) & "<br />")
next
%>
```

Output:

```
test1=First test
test3=Third test
```

2.**Contents.RemoveAll**

The Contents.RemoveAll method deletes all items from the Contents collection.

Syntax

```
Application.Contents.RemoveAll()
```

```
<%
Application.Contents.RemoveAll()
%>
```

3. **Lock and UnLock**

Lock Method

The Lock method prevents other users from modifying the variables in the Application object (used to ensure that only one client at a time can modify the Application variables).

Unlock Method

The Unlock method enables other users to modify the variables stored in the Application object (after it has been locked using the Lock method).

Syntax

```
Application.Lock

Application.Unlock
```

Example

The example below uses the Lock method to prevent more than one user from accessing the variable visits at a time, and the Unlock method to unlock the locked object so that the next client can increment the variable visits:

```
<%
Application.Lock
Application("visits")=Application("visits")+1
Application.Unlock
%>
```

This page has been visited
<%=Application("visits")%> times!

<u>Events</u>
Application_OnEnd

Application_OnStart

Application_OnStart Event

The Application_OnStart event occurs before the first new session is created (when the Application object is first referenced).

This event is placed in the Global.asa file.

Note: Referencing to a Session, Request, or Response objects in the Application_OnStart event script will cause an error.

Application_OnEnd Event

The Application_OnEnd event occurs when the application ends (when the web server stops).

This event is placed in the Global.asa file.

Note: The MapPath method cannot be used in the Application_OnEnd code.

Syntax

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

Sub Application_OnStart
. . .
End Sub

Sub Application_OnEnd
. . .
End Sub

</script>
```

Examples

Global.asa:

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

Sub Application_OnEnd()
Application("totvisitors")=Application("visitors")
```

```
End Sub

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>
```

<u>**The Global.asa file**</u>

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
- the #include directive

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.

Session Object

When you are working with an application on your computer, 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 open the application and when you close it. However, 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 user's computer and it contains information that identifies the user. This interface is called the Session object.

The Session object stores information about, or change settings for a user session.

Variables stored in a 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

1. Contents

The Contents collection contains all the items appended to the session through a script command.

Tip: To remove items from the Contents collection, use the Remove and RemoveAll methods.

Syntax
Session.Contents(Key)

Example 1

Notice that both name and objtest would be appended to the Contents collection:

```
<%
Session("name")="Hege"
Set Session("objtest")=Server.CreateObject("ADODB.Connection")
%>
```

Example 2

To loop through the Contents collection:

```
<%
for each x in Session.Contents
  Response.Write(x & "=" & Session.Contents(x) & "<br />")
```

```
next
%>
```

or:

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

Example 3

```
<%
Session("name")="Hege"
Session("date")="2001/05/05"

for each x in Session.Contents
  Response.Write(x & "=" & Session.Contents(x) & "<br />")
next
%>
```

Output:

```
name=Hege
date=2001/05/05
```

2. <u>StaticObjects</u>

The StaticObjects collection contains all the objects appended to the session with the HTML <object> tag.

Syntax
Session.StaticObjects(Key)

Parameter	**Description**
Key	Required. The name of the item to retrieve

Example 1

To loop through the StaticObjects collection:

```
<%
for each x in Session.StaticObjects
  Response.Write(x & "<br />")
next
%>
```

Example 2

In Global.asa:

```
<object runat="server" scope="session"
id="MsgBoard" progid="msgboard.MsgBoard">
</object>

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

In an ASP file:

```
<%
for each x in Session.StaticObjects
  Response.Write(x & "<br />")
next
%>
```

Output:

```
MsgBoard
AdRot
```

Properties

1.CodePage

The CodePage property specifies the character set that will be used when displaying dynamic content.

Example of some code pages:

- 1252 - American English and most European languages
- 932 - Japanese Kanji

Syntax

```
Session.CodePage(=Codepage)
```

Parameter	**Description**
Codepage	Defines a code page (character set) for the system running the script engine

Examples

```
<%
Response.Write(Session.CodePage)
%>
```

Output:

```
1252
```

2. LCID

The LCID property 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.

Syntax

```
Session.LCID(=LCID)
```

Parameter	Description
LCID	A locale identifier

Examples

```
<%
response.write("<p>")
response.write("Default LCID is: " & Session.LCID & "<br />")
response.write("Date format is: " & date() & "<br />")
response.write("Currency format is: " & FormatCurrency(350))
response.write("</p>")

Session.LCID=1036

response.write("<p>")
response.write("LCID is now: " & Session.LCID & "<br />")
response.write("Date format is: " & date() & "<br />")
response.write("Currency format is: " & FormatCurrency(350))
response.write("</p>")

Session.LCID=3079

response.write("<p>")
response.write("LCID is now: " & Session.LCID & "<br />")
response.write("Date format is: " & date() & "<br />")
response.write("Currency format is: " & FormatCurrency(350))
response.write("</p>")

Session.LCID=2057

response.write("<p>")
response.write("LCID is now: " & Session.LCID & "<br />")
response.write("Date format is: " & date() & "<br />")
response.write("Currency format is: " & FormatCurrency(350))
response.write("</p>")
%>
```

Output:

```
Default LCID is: 2048
```

```
Date format is: 12/11/2001
Currency format is: $350.00

LCID is now: 1036
Date format is: 11/12/2001
Currency format is: 350,00 F

LCID is now: 3079
Date format is: 11.12.2001
Currency format is: öS 350,00

LCID is now: 2057
Date format is: 11/12/2001
Currency format is: £350.00
```

3.<u>SessionID</u>

The SessionID property returns a unique id for each user. The unique id is generated by the server.

Syntax
Session.SessionID

Examples
```
<%
Response.Write(Session.SessionID)
%>
```

Output:

```
772766038
```

4. <u>TimeOut</u>

The Timeout property sets or returns the timeout period for the Session object for this application, in minutes. If the user does not refresh or request a page within the timeout period, the session will end.

Syntax
Session.Timeout[=nMinutes]

Parameter	**Description**
nMinutes	The number of minutes a session can remain idle before the server terminates it. Default is 20 minutes

Examples
```
<%
response.write("<p>")
response.write("Default Timeout is: " & Session.Timeout)
response.write("</p>")

Session.Timeout=30

response.write("<p>")
response.write("Timeout is now: " & Session.Timeout)
response.write("</p>")
%>
```

Output:

```
Default Timeout is: 20

Timeout is now: 30
```

Methods

1.Abandon

The Abandon method destroys a user session.

Note: When this method is called, the current Session object is not deleted until all of the script on the current page have been processed. This means that it is possible to access session variables on the same page as the call to Abandon, but not from another Web page.

Syntax

```
Session.Abandon
```

Examples

File1.asp:

```
<%
Session("name")="Hege"
Session.Abandon
Response.Write(Session("name"))
%>
```

Output:

```
Hege
```

File2.asp:

```
<%
Response.Write(Session("name"))
%>
```

Output:

(none)

2. Contents.Remove

The Contents.Remove method deletes an item from the Contents collection.

Syntax
Session.Contents.Remove(name|index)

Parameter	**Description**
Name	The name of the item to remove
Index	The index of the item to remove

Example 1
```
<%
Session("test1")=("First test")
Session("test2")=("Second test")
Session("test3")=("Third test")

Session.Contents.Remove("test2")

for each x in Session.Contents
  Response.Write(x & "=" & Session.Contents(x) & "<br />")
next
%>
```

Output:

```
test1=First test
test3=Third test
```

Example 2
```
<%
Session("test1")=("First test")
Session("test2")=("Second test")
Session("test3")=("Third test")

Session.Contents.Remove(2)

for each x in Session.Contents
  Response.Write(x & "=" & Session.Contents(x) & "<br />")
next
%>
```

Output:

```
test1=First test
test3=Third test
```

3.<u>Contents.RemoveAll</u>

The Contents.RemoveAll method deletes all items from the Contents collection.

Syntax

```
Session.Contents.RemoveAll()
```

Example:

```
<%
Session.Contents.RemoveAll()
%>
```

Events

Session_OnStart Event

The Session_OnStart event occurs when the server creates a session.

This event is placed in the Global.asa file.

Session_OnEnd Event

The Session_OnEnd event occurs when the session ends (abandoned or times out).

This event is placed in the Global.asa file.

(see Application Object events for the example)