

Welcome to the **DevGuru Built-in ASP Objects Quick Reference** guide. This is a valuable 71 page reference source that explains and provides comprehensive, working examples of code for all of the collections, events, methods, and properties belonging to the seven Built-in Active Server Pages objects.

The seven built-in objects are:

- [Application](#)
- [ASPErrors](#)
- [ObjectContext](#)
- [Request](#)
- [Response](#)
- [Server](#)
- [Session](#)

In the good old days (a few years ago!), most Web sites were created with HTML and simply displayed static pages. A few of the more adventurous programmers would use C or Perl to design a dynamic Web site utilizing the CGI technology. However, these techniques were plagued with security concerns and did not scale well to large sites. So, in general, dynamic sites remained relatively few in number and were time consuming and expensive to create. The introduction of the scripting languages, which could be embedded inside HTML code, opened new doors for dynamic site development. Active Server Pages (ASP), which managed to arrive just in time for the explosive growth of the World Wide Web, was a next logical step for Web-based application development. Introduced in 1996 by Microsoft, Active Server Pages proved to be an exciting, new technology that extended standard HTML by adding built-in objects, server-side scripts, access to databases, and ActiveX components. Another important development by Microsoft was to make the ASP scripting environment compliant with the Component Object Model (COM). COM created a standard communication mechanism between components. This step allowed non-vendor components, such as those offered by **DevGuru**, to share their properties, methods and events with other components in a process called OLE automation. Non-vendor components greatly extend the functionality of ASP applications.

The true power of ASP is the ease and rapidity with which developers can create and implement dynamic Web sites. Indeed, for today's modern Web commerce, a dynamic, database-driven, server-side application that interacts with the client is the norm.

ASP employs a scripting environment and [VBScript](#) is the default scripting language of choice. However, you can use other languages (such as JScript and Perl) as long as they have a scripting engine that is compatible with the ActiveX scripting standard.

Fortunately, you are not limited to just using Active Server Pages with Microsoft's Internet Information Server (IIS) and this has enhanced the popularity of ASP. For example, Chili!Soft is a proven industry leader in providing ASP engines for use with Web servers from FastTrack, Lotus, Netscape, O'Reilly, and many others. And Halcyon Software offers a brilliant Java-based implementation of the Microsoft ASP framework, allowing developers to deploy ASP applications on any platform.

Meanwhile, ASP continues to evolve. With the arrival of the millennium came the arrival of ASP version 3.0. Version 3.0 was released along with Internet Information Server (IIS) version 5.0 as part of the highly anticipated Microsoft Windows 2000. By far, the most important new feature of version 3.0 is the addition of a seventh, intrinsic object called [ASPErrors](#) which should greatly simplify error handling. Other new features include the addition of three new methods to the [Server](#) object, and two new methods to both the [Application](#) object and the [Session](#) object.

Active Server Pages has ultimately proven to be of significant value to developers and fueled a revolution in the development of Web-based applications.

The future of ASP will continue to be very interesting. Later this year (summer?), ASP-plus, which is now referred to as ASP.NET, will be formally released by Microsoft. It allows the use of the Visual Basic, C#, and Visual C++ languages with ASP.

COLLECTIONS

Contents

[Application](#)

[Session](#)

[ClientCertificate](#)

Cookies

[Request](#)

[Response](#)

[Form](#)

[QueryString](#)

[ServerVariables](#)

StaticObjects

[Application](#)

[Session](#)

EVENTS

OnEnd

[Application](#)

[Session](#)

OnStart

[Application](#)

[Session](#)

[OnTransactionAbort](#)

[OnTransactionCommit](#)

METHODS

[Abandon](#)
[AddHeader](#)
[AppendToLog](#)
[BinaryRead](#)
[BinaryWrite](#)
[Clear](#)
Contents.Remove
 [Application](#)
 [Session](#)
Contents.RemoveAll
 [Application](#)
 [Session](#)
[CreateObject](#)
[End](#)
[Execute](#)
[Flush](#)
[GetLastError](#)
[HTMLEncode](#)
[MapPath](#)
[Redirect](#)
[SetAbort](#)
[SetComplete](#)
[Transfer](#)
[URLEncode](#)
[Write](#)

OBJECTS

[Application](#)

[ASPError](#)

[ObjectContext](#)

[Request](#)

[Response](#)

[Server](#)

[Session](#)

PROPERTIES

[ASPCode](#) [File](#)
[ASPDescription](#) [IsClientConnected](#)
[Buffer](#) [LCID](#)
[CacheControl](#) [Line](#)
[Category](#) [Number](#)
[Charset](#) [Pics](#)
[CodePage](#) [ScriptTimeout](#)
[Column](#) [SessionID](#)
[ContentType](#) [Source](#)
[Description](#) [Status](#)
[Expires](#) [Timeout](#)
[ExpiresAbsolute](#)

INDEX

[Abandon](#)
[AddHeader](#)
[AppendToLog](#)
[Application](#)
[ASPCode](#)
[ASPDescription](#)
[ASPErrors](#)
[BinaryRead](#)
[BinaryWrite](#)
[Buffer](#)
[CacheControl](#)
[Category](#)
[Charset](#)
[Clear](#)
[ClientCertificate](#)
[CodePage](#)
[Column](#)
Contents
 [Application](#)
 [Session](#)
[ContentType](#)
Cookies
 [Request](#)
 [Response](#)
Contents.Remove
 [Application](#)
 [Session](#)
Contents.RemoveAll
 [Application](#)
 [Session](#)
[CreateObject](#)
[Description](#)
[End](#)
[Execute](#)
[Expires](#)
[ExpiresAbsolute](#)
[File](#)
[Flush](#)
[Form](#)
[GetLastError](#)
[HTMLEncode](#)
[IsClientConnected](#)
[LCID](#)
[Line](#)
[MapPath](#)
[Number](#)
[ObjectContext](#)
OnEnd
 [Application](#)
 [Session](#)
OnStart
 [Application](#)
 [Session](#)
[OnTransactionAbort](#)
[OnTransactionCommit](#)
[Pics](#)
[QueryString](#)
[Redirect](#)
[Request](#)
[Response](#)
[ScriptTimeout](#)
[Server](#)
[ServerVariables](#)
[Session](#)
[SessionID](#)
[SetAbort](#)
[SetComplete](#)
[Source](#)
StaticObjects
 [Application](#)
 [Session](#)
[Status](#)
[Timeout](#)
[Transfer](#)
[URLEncode](#)
[Write](#)

METHOD: Session.Abandon

Session.Abandon

The **Abandon** method terminates a user session, destroys all the objects in the current Session object, and releases its resources. However, this deletion will not occur until all of the script is processed for the current page. When the session ends, the **OnEnd** event handler is called. By default, even if you do not call **Abandon**, the **Session** object is terminated after twenty minutes of idle time.

Code:

```
-----File1.asp-----
<%
Response.Write "Your SessionID is " & Session.SessionID
Session("Application") = "DevSite"
Session.Abandon
Response.Write "The Application name is " & Session("Application")
%>
```

```
-----File2.asp-----
<%
Response.Write "Your SessionID is " & Session.SessionID
%>
```

Output:

```
-----File1.asp-----
Your SessionID is 465107831
The Application name is DevSite

-----File2.asp-----
Your SessionID is 465107831
```

METHOD: Response.AddHeader

Response.AddHeader Name, Value

The **AddHeader** method adds a new named HTTP header with a specific value to the HTTP response. Note that once a header has been added, it cannot be removed. This method must be called before any output is sent to the client unless the `Response.Buffer` is set to true.

There are two mandatory arguments.

Name

The **Name** argument is the name of the new header variable. The name cannot contain any underscores (`_`).

Value

The **Value** argument is the initial value of the new header variable.

Code:

```
<%  
Response.AddHeader "MyHeader", "ERROR"  
%>
```

METHOD: Response.AppendToLog

Response.AppendToLog(String)

The **AppendToLog** method adds (appends) a string to the end of the entry of the Web server log for this request.

There is one mandatory argument.

String

The **String** argument is the string to be appended. The string can have a maximum length of 80 characters. It cannot contain commas because the log is comma-delineated.

Code:

```
<%  
Response.AppendToLog("Error in Processing")  
%>
```

Output is appended to the end of the IIS log file:

```
10.78.176.37, - , 03/20/97, 7:55:20, W3SVC, SALES1, 10.78.176.37, Error in Processing
```

OBJECT: Application

Modified in version 3.0

The **Application** object shares, stores, and retrieves information in response to requests from users to the application.

COLLECTION PROPERTIES

[Contents](#) Object

A collection of all of the items which have been created and added to the **Application** object during client sessions through script commands, such as the **Server.CreateObject**, rather than by using the HTML <OBJECT> tag.

[StaticObjects](#) Object

A collection of all of the items which have been created and added to the **Application** object during client sessions by using the HTML <OBJECT> tag, rather than using script commands.

METHODS

[Contents.Remove](#) Method Implemented in version 3.0

The **Contents.Remove** method is used to remove a single item from a **Contents** collection.

[Contents.RemoveAll](#) Method Implemented in version 3.0

The **Contents.RemoveAll** method is used to remove all items from a **Contents** collection.

[Lock](#) Method

The **Lock** method prevents all other users from making changes in the **Application** object.

[Unlock](#) Method

The **Unlock** method allows any user to have access to any of the **Application** object properties in order to make changes.

EVENTS

[OnEnd](#) Event

The **OnEnd** event occurs when the **Application** quits. This should not occur unless all user sessions are over. The signal of this event will run a handler script in the Global.asa file, if the script exist.

[OnStart](#) Event

The **OnStart** event occurs before the start of any new session by a user (i.e., before the **Application** object is first referenced). The signal of this event will run a handler script in the Global.asa file, if the script exist.

COLLECTION PROPERTY: Application.Contents

Application.Contents(Key)

The **Contents** collection property contains a list of the items that have been created and added to the Application object. Objects can be added to the collection by using **Server.CreateObject** or by assigning scalar variables.

You can iterate through a collection using a **For Each item in ... Next** loop.

There is one mandatory argument.

Key

The **Key** argument is the name of the item to be retrieved.

Code:

```
<%  
Application("name") = "Application Maker"  
Application("publishdate") = "05/15/01"  
Application("author") = "DevGuru"  
Set Application("Obj1") = Server.CreateObject("ADODB.Connection")
```

```
For Each Item in Application.Contents  
  If IsObject(Application.Contents(Item)) Then  
    Response.Write Item & " is an object.<BR>"  
  Else  
    Response.Write Item & "=" & Application.Contents(Item) & "<BR>"  
  End If  
Next  
>%
```

Output:

```
name=Application Maker  
publishdate=05/15/01  
author=DevGuru  
OBJ1 is an object.
```

COLLECTION PROPERTY: Application.StaticObjects

Application.StaticObjects(Key)

The **StaticObjects** collection property is a collection that contains of all of the items created within the **Application** using the HTML <OBJECT> tag. You can use this collection to find the value of any property for any object.

You can iterate through a collection using a **For Each item in ... Next** loop.

There is one mandatory argument.

Key

The **Key** argument is the name of the item to be retrieved.

Code:

-----Global.asa-----

```
<OBJECT RUNAT=Server SCOPE=Application ID=MyInfo PROGID="MSWC.MyInfo">
</OBJECT>
```

```
<OBJECT RUNAT=Server SCOPE=Application ID=MyConnection
PROGID="ADODB.Connection">
</OBJECT>
```

```
<OBJECT RUNAT=Server SCOPE=Application ID=MyADRot PROGID="MSWC.ADRotator">
</OBJECT>
```

-----File.asp-----

```
<%
For Each Item In Application.StaticObjects
Response.Write Item & "<BR>"
Next
%>
```

Output:

```
MyInfo
MyConnection
MyADRot
```

METHOD: Application.Contents.Remove

Implemented in version 3.0

Application.Contents.Remove (Name|Integer)

The **Contents.Remove** method is used to delete one specified item in the **Application.Contents** collection.

You may use only one of the two possible choices for the mandatory argument.

Name or **Integer**

The **Name** argument is the name of the item to be deleted. It must be enclosed in a pair of quotes.

The **Integer** argument is the position number of the item in the collection to be deleted. The numbering sequence for a collection starts at one, not zero.

Code:

```
<%  
Application("name") = "Application Maker"  
Application("publishdate") = "05/15/01"  
Application("author") = "DevGuru"  
Set Application("Obj1") = Server.CreateObject("ADODB.Connection")
```

```
Application.Contents.Remove(1)  
Application.Contents.Remove("publishdate")
```

```
For Each Item in Application.Contents  
  If IsObject(Application.Contents(Item)) Then  
    Response.Write Item & " is an object.<BR>"  
  Else  
    Response.Write Item & "=" & Application.Contents(Item) & "<BR>"  
  End If  
Next  
>%
```

Output:

```
author=DevGuru  
Obj1 is an object.
```

METHOD: Application.Contents.RemoveAll

Implemented in version 3.0

Application.Contents.RemoveAll

The **Contents.RemoveAll** method deletes all items that are in the **Application.Contents** collection.

Remember to include the pair of empty ().

Code:

```
<%
```

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

```
%>
```

METHOD: Application.Lock

Application.Lock

The **Lock** method prevents all other users from changing any of the variables in the **Contents** collection of the **Application** object.

Application objects are designed to be shared among an unlimited number of users. Therefore, you need the ability to allow only one user at a time to make changes and you do this by locking everybody else out.

You can use the **Unlock** method to explicitly remove the **Lock** placed upon the **Application** object. Remember, to completely unlock an object, you must call **Unlock** the same number of times you have called **Lock**. Fortunately, the server will automatically unlock all locks placed on the object when the script times out or the .asp file ends.

Code:

```
<%  
Application.Lock  
%>
```

METHOD: Application.Unlock

Application.Unlock

The **Unlock** method is used to explicitly unlock the variables in the **Contents** collection of the **Application** object.

In contrast, the **Lock** method prevents all other users from changing any of the variables in the **Contents** collection of the **Application** object. This ability to lock is required since **Application** objects are designed to be shared among an unlimited number of users. Therefore, you need the ability to allow only one user at a time to make changes and you do this by locking everybody else out.

Remember, to completely unlock an object, you must call **Unlock** the same number of times you have called **Lock**. Fortunately, the server will automatically unlock all locks placed on the object when the script times out or the .asp file ends.

Code:

```
<%  
Application.Unlock  
%>
```

EVENT: Application_OnEnd

The **Application_OnEnd** event occurs when the **Application** ends. This should only happen when the web server is stopped by the operating system in a normal manner. The **Application_OnEnd** event is simply a subroutine with a reserved name that is placed within the Global.asa file. It can contain any script that you wish to run after all user sessions are finished. For example, you may wish to compute and store statistics on user sessions for future reference.

Note: The only built-in ASP objects available from within the OnEnd event handler are Server and Application.

Code:

```
-----Global.asa-----
<script Language="VBScript" RUNAT=Server>
Sub Application_OnEnd()
    Calculate_Stats()
End Sub

Sub Application_OnStart()
    Application("NumSession") = 0
    Application("NumVisited") = 0
End Sub

Sub Session_OnEnd()
    Application("NumSession") = Application("NumSession") - 1
End Sub

Sub Session_OnStart()
    Application("NumSession") = Application("NumSession") + 1
    Application("NumVisited") = Application("NumVisited") + 1
End Sub
</script>

-----File1.asp-----
Response.Write "You are " & Application("NumSession") & " of " & Application("NumVisited") &
" users."
```

EVENT: Application_OnStart

The **Application_OnStart** event occurs before the beginning of any user session. The only built-in ASP objects available from within the OnStart event handler are Server and Application. The **Application_OnStart** event is simply a subroutine with a reserved name that is placed within the Global.asa file. It contains any script that you wish to run before the first user session. For example, you could use the **Session.Timeout** property to automatically end an idle user session after 10 minutes.

Note that you must declare the scripting language used to code the event script on the first line of the Global.asa file.

Code:

```
-----Global.asa-----
<script Language="VBScript" RUNAT=Server>
Sub Application_OnEnd()
End Sub

Sub Application_OnStart()
    Application("NumSession") = 0
    Application("NumVisited") = 0
    Session.Timeout = 10
End Sub

Sub Session_OnEnd()
    Application("NumSession") = Application("NumSession") - 1
End Sub

Sub Session_OnStart()
    Application("NumSession") = Application("NumSession") + 1
    Application("NumVisited") = Application("NumVisited") + 1
End Sub
</script>

-----File1.asp-----
Response.Write "You are " & Application("NumSession") & " of " & Application("NumVisited") &
" users."
```

Output:

You are 1 of 1 users.

OBJECT: ASPError

Implemented in version 3.0

The **ASPError** object is used to obtain detailed information on an error condition. This information, provided by the nine read-only properties of **ASPError**, can only be accessed by using the [Server.GetLastError](#) method.

PROPERTIES

ASPCode Property

The **ASPCode** property returns the error code generated by IIS.

ASPDescription Property

The **ASPDescription** property returns a string that provides more detailed description of the error as compared to the short description returned by the **Description** property. (Not available for all errors.)

Category Property

The **Category** property returns a string indicating if the error was generated by IIS, an object, or a scripting language.

Column Property

The **Column** property returns a long integer defining the column position inside the ASP file where the error occurred.

Description Property

The **Description** property returns a string that is a short description of the error. If you desire a longer description, use the **ASPDescription** property.

File Property

The **File** property returns the name of the ASP (Active Server Page) file in which the error occurred.

Line Property

The **Line** property returns a long integer defining the line number inside the ASP file where the error occurred.

Number Property

The **Number** property returns a long integer defining the standard COM error code returned by a COM component.

Source Property

The **Source** property returns a string with the actual line from the source code where the error occurred (if available).

METHOD: Server.GetLastError

Implemented in version 3.0

Server.GetLastError()

The **GetLastError** method returns an instance of the **ASPErrors** object. The **ASPErrors** object exposes (gives access to) nine read-only properties that provide detailed information about an error that has occurred.

Once an ASP file has sent content to a client, you cannot use the **GetLastError** method to access information about the error.

The types of errors handled by the **ASPErrors** method are called "500;100 custom errors" and there are three major types: pre-processing, runtime, and script compiling.

By default, the error information is sent to the "\iishelp\common\500-100.asp" file. This is a new file that is part of IIS version 5.0 and does not exist in earlier versions. Here is the [source code for the 500-100.asp file](#). Since you can access this file, you can customize this code to suit your programming needs.

Also, you can create your own personal, custom ASP error file by having the Internet Information Server (IIS) 5.0 redirect the 500;100 custom errors to an error file that you create. In your ASP error file, you can run any ASP code you desire.

Here is the general concept of how to access the nine **ASPErrors** properties in an error file

Code:

```
<%  
Dim objErrorInfo  
Set objErrorInfo = Server.GetLastError  
  
Response.Write("ASPCode = " & objErrorInfo.ASPCode)  
Response.Write("ASPDescription = " & objErrorInfo.ASPDescription)  
Response.Write("Category = " & objErrorInfo.Category)  
Response.Write("Column = " & objErrorInfo.Column)  
Response.Write("Description = " & objErrorInfo.Description)  
Response.Write("File = " & objErrorInfo.File)  
Response.Write("Line = " & objErrorInfo.Line)  
Response.Write("Number = " & objErrorInfo.Number)  
Response.Write("Source = " & objErrorInfo.Source)  
%>
```

The source code for the 500-100.asp file:

It is presented to demonstrate how Microsoft programmers use the **ASPError** object to display errors. Certain parts of the code have been colorized to call your attention to specific procedures.

```
<%@ language="VBScript" %>
<%
Option Explicit

Const lngMaxFormBytes = 200

Dim objASPError, blnErrorWritten, strServername, strServerIP, strRemoteIP
Dim strMethod, lngPos, datNow, strQueryString, strURL

If Response.Buffer Then
Response.Clear
Response.Status = "500 Internal Server Error"
Response.ContentType = "text/html"
Response.Expires = 0
End If

Set objASPError = Server.GetLastError
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

<html dir=ltr>

<head>
<style>
a:link {font:8pt/11pt verdana; color:FF0000}
a:visited {font:8pt/11pt verdana; color:#4e4e4e}
</style>

<META NAME="ROBOTS" CONTENT="NOINDEX">

<title>The page cannot be displayed</title>

<META HTTP-EQUIV="Content-Type" Content="text-html; charset=Windows-1252">
</head>

<script>
function Homepage(){
<!--
// in real bits, urls get returned to our script like this:
// res://shdocvw.dll/http_404.htm#http://www.DocURL.com/bar.htm

//For testing use DocURL =
"res://shdocvw.dll/http_404.htm#https://www.microsoft.com/bar.htm"
DocURL=document.URL;
```

```
//this is where the http or https will be, as found by searching for :// but skipping the res://
protocolIndex=DocURL.indexOf("://",4);
```

```
//this finds the ending slash for the domain server
serverIndex=DocURL.indexOf("/",protocolIndex + 3);
```

```
//for the href, we need a valid URL to the domain. We search for the # symbol to find the beginning
//of the true URL, and add 1 to skip it - this is the BeginURL value. We use serverIndex as the end marker.
//urlresult=DocURL.substring(protocolIndex - 4,serverIndex);
BeginURL=DocURL.indexOf("#",1) + 1;
urlresult=DocURL.substring(BeginURL,serverIndex);
```

```
//for display, we need to skip after http://, and go to the next slash
displayresult=DocURL.substring(protocolIndex + 3 ,serverIndex);
document.write('<A HREF="' + urlresult + "'> + displayresult + "</a>");
}
//-->
</script>
```

```
<body bgcolor="FFFFFF">
```

```
<table width="410" cellpadding="3" cellspacing="5">
```

```
<tr>
<td align="left" valign="middle" width="360">
<h1 style="COLOR:000000; FONT: 13pt/15pt verdana"><!--Problem-->The page cannot be displayed</h1>
</td>
</tr>
```

```
<tr>
<td width="400" colspan="2">
<font style="COLOR:000000; FONT: 8pt/11pt verdana">There is a problem with the page you are trying to
reach and it cannot be displayed.</font></td>
</tr>
```

```
<tr>
<td width="400" colspan="2">
<font style="COLOR:000000; FONT: 8pt/11pt verdana">
```

```
<hr color="#C0C0C0" noshade>
```

```
<p>Please try the following:</p>
```

```
<ul>
<li id="instructionsText1">Click the
<a href="javascript:location.reload()">
Refresh</a> button, or try again later.<br>
</li>
```

```
<li>Open the
```

```
<script>
```

```
<!--  
if (!(window.navigator.userAgent.indexOf("MSIE") > 0) && (window.navigator.appVersion.charAt(0) == "2"))  
{  
Homepage();  
}  
//-->  
</script>
```

home page, and then look for links to the information you want.

<h2 style="font:8pt/11pt verdana; color:000000">HTTP 500.100 - Internal Server Error - ASP error
Internet Information Services</h2>

<hr color="#C0C0C0" noshade>

<p>Technical Information (for support personnel)</p>

Error Type:

<%

Response.Write objASPErrors.Category

If objASPErrors.ASPCode > "" Then Response.Write ", " & objASPErrors.ASPCode

Response.Write " (0x" & Hex(objASPErrors.Number) & ")" & "
"

Response.Write "" & objASPErrors.Description & "
"

If objASPErrors.ASPDescription > "" Then Response.Write objASPErrors.ASPDescription & "
"

blnErrorWritten = False

' Only show the Source if it is available and the request is from the same machine as IIS

If objASPErrors.Source > "" Then

strServername = LCase(Request.ServerVariables("SERVER_NAME"))

strServerIP = Request.ServerVariables("LOCAL_ADDR")

strRemoteIP = Request.ServerVariables("REMOTE_ADDR")

If (strServername = "localhost" Or strServerIP = strRemoteIP) And objASPErrors.File

<> "?" Then

Response.Write objASPErrors.File

If objASPErrors.Line > 0 Then Response.Write ", line " & objASPErrors.Line

If objASPErrors.Column > 0 Then Response.Write ", column " & objASPErrors.Column

Response.Write "
"

Response.Write "<font style=""COLOR:000000; FONT: 8pt/11pt courier
new"">"

Response.Write Server.HtmlEncode(objASPErrors.Source) & "
"

If objASPErrors.Column > 0 Then Response.Write String((objASPErrors.Column - 1), "-")
& "
"

Response.Write ""

blnErrorWritten = True

End If

End If

```

If Not blnErrorWritten And objASPErrors.File <> "?" Then
Response.Write "<b>" & objASPErrors.File
If objASPErrors.Line > 0 Then Response.Write ", line " & objASPErrors.Line
If objASPErrors.Column > 0 Then Response.Write ", column " & objASPErrors.Column
Response.Write "</b><br>"
End If
%>
</li>
<p>
<li>Browser Type:<br>
<%= Request.ServerVariables("HTTP_USER_AGENT") %>
</li>
<p>
<li>Page:<br>
<%
strMethod = Request.ServerVariables("REQUEST_METHOD")

Response.Write strMethod & " "

If strMethod = "POST" Then
Response.Write Request.TotalBytes & " bytes to "
End If

Response.Write Request.ServerVariables("SCRIPT_NAME")

lngPos = InStr(Request.QueryString, "|")

If lngPos > 1 Then
Response.Write "?" & Left(Request.QueryString, (lngPos - 1))
End If

Response.Write "</li>"

If strMethod = "POST" Then

Response.Write "<p><li>POST Data:<br>"
If Request.TotalBytes > lngMaxFormBytes Then
Response.Write Server.HtmlEncode(Left(Request.Form, lngMaxFormBytes)) & " . . ."
Else
Response.Write Server.HtmlEncode(Request.Form)
End If
Response.Write "</li>"
End If

%>
<p>
<li>Time:<br>
<%
datNow = Now()

Response.Write FormatDateTime(datNow, 1) & ", " & FormatDateTime(datNow, 3)

```

```
%>
</li>
</p>
<p>
<li>More information:<br>
<% strQueryString = "prd=iis&sbp=&pver=5.0&ID=500;100&cat=" &
Server.URLEncode(objASPError.Category) & _
"&os=&over=&hrd=&Opt1=" & Server.URLEncode(objASPError.ASPCode) & "&Opt2=" &
Server.URLEncode(objASPError.Number) & _
"&Opt3=" & Server.URLEncode(objASPError.Description)
strURL = "http://www.microsoft.com/ContentRedirect.asp?" & _
strQueryString
%>
<a href="<%= strURL %>">Microsoft Support</a>
</li>
</p>

</font></td>
</tr>

</table>
</body>
</html>
```

METHOD: Request.BinaryRead

Request.BinaryRead(Count)

The **BinaryRead** method retrieves the unparsed data that was sent to the server from the user as part of an HTTP POST (i.e. an HTML FORM request). The data read is stored in a **SafeArray** which is a variant, structure-like array that, in addition to storing the data, also stores the number of dimensions of the array and the upper bounds of those dimensions.

You can write binary data into the response stream (info sent to a client) by using the **Response.BinaryWrite(BinaryData)** method. The **BinaryData** argument is simply the output (the **SafeArray**) from **Request.BinaryRead**. For example, you could retrieve an image from a database with **Request.BinaryRead** and send it to a client using **Response.BinaryWrite**.

Using **Request.Form** after calling **BinaryRead**, and vice-versa, will cause an error.

There is one mandatory argument.

Count

Before the method is executed, **Count** contains the number of bytes to be read. After the method has returned, **Count** contains the actual number of bytes read. The number read should be less than or equal to the total byte count value returned by **Request.TotalByte**.

Code:

```
<%  
Dim ByteCount, BinRead  
ByteCount = Request.TotalBytes  
BinRead = Request.BinaryRead(ByteCount)  
...  
Response.BinaryWrite(BinRead)  
>%
```

METHOD: Response.BinaryWrite

Response.BinaryWrite(Data)

The **BinaryWrite** method writes nonstring data (binary data) to the current HTTP output without any character set conversion.

This method can use the output of the **Request.BinaryRead** method. For example, you could retrieve an image from a database with **Request.BinaryRead** and send it to a client using **Response.BinaryWrite**.

There is one mandatory argument.

Data

The **Data** argument is the data to send to the HTTP output.

Code:

```
-----File1.asp-----
<HTML>
<HEAD>
</HEAD>
<BODY>
<form action="File2.asp" method="POST">
Name:<input type="Text" name="name" maxlength="30"><BR>
Age:<input type="Text" name="age" maxlength="10"><BR>
Sex:<input type="Text" name="Sex" maxlength="10"><BR>
<input type="Submit" name="submit" value="submit"><BR>
</form>
</BODY>
</HTML>
```

```
-----File2.asp-----
<%
bytecount = Request.TotalBytes
binread = Request.BinaryRead(bytecount)
Response.BinaryWrite binread
%>
```

Output:

```
name=Jane&age=25&Sex=Female&submit=submit
```

PROPERTY: Response.Buffer

Modified in version 3.0

The **Buffer** property tells whether to buffer (temporarily store) the output page being sent to the browser. A response is not sent to the browser until all scripts are processed, or the **Flush** or **End** methods are called. The **Buffer** property cannot be set or changed after the server has sent output to the browser. Therefore, you need to set the **Buffer** property on the first line of the ASP page. If the **Buffer** is set to **True**, then the server will buffer the output. If it is set to **False**, then no buffering occurs.

In ASP version 2.0 the default value is **False** and no buffering is done.

In ASP version 3.0 the default value is **True** and the server will buffer the output. However, there is a quirk concerning the default value of **Response.Buffer** that is dependent upon how Windows 2000 (which contains IIS 5.0 and ASP 3.0) is initially installed on your computer. If you do a clean install, the default value will be **True**. If the install is an upgrade, the default value will be **False**.

In the code example, there is no output because the loop does not finish and the **Buffer** will not empty until the script has finished. If the buffer was set to **False**, then the **Response.Write** would write the number to the browser every time it went through the loop.

Code:

```
<%  
Response.Buffer = TRUE  
x=0  
Do  
    x = x+1  
    Response.Write x & "<BR>"  
Loop  
>%
```

PROPERTY: Response.CacheControl

The **CacheControl** property allows a proxy server to cache an Active Server Page. A proxy server is used to speed up the retrieval of a web page by keeping a local copy of the web page in a cache.

The **CacheControl** property can only be set equal to "**Public**" or "**Private**". The pair of double quotes are required.

If the **CacheControl** property is set equal to "**Public**", then caching can occur.

If the **CacheControl** property is set equal to "**Private**", then caching cannot occur. By default, a proxy server will not keep a cache copy.

You can use **Response.Expires** and **Response.ExpiresAbsolute** to set a time and/or date limit on the existence of the cache.

```
<% Response.CacheControl = "Public" %>
```

```
<% Response.CacheControl = "Private" %>
```

PROPERTY: Response.Charset

The **Charset** property adds the name of the character set to the content-type header contained in the response object. Unfortunately, **CharSet** will accept any string, valid or not, for the name. The default character set is ISO-LATIN-1, except on the Macintosh where the default is the Macintosh character set.

In the example, if the header is:

```
content-type: text/html
```

Then the header will become:

```
content-type: text/html; charset = MS_Kanji
```

Code:

```
<% Response.Charset("MS_Kanji") %>
```

METHOD: Response.Clear

Response.Clear

The **Clear** method clears (erases) any buffered HTML output. It does not erase the response headers, only the body. If the buffer does not exist, because **Response.Buffer** was set to **False**, a runtime error will occur.

Note that in ASP version 2.0, the default for **Response.Buffer** is set to **False**, however in ASP version 3.0, the default for **Response.Buffer** is set to **True**.

Code:

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

COLLECTION PROPERTY: Request.ClientCertificate

Request.ClientCertificate(Key [SubField])

One procedure for maintaining security in electronic transactions is to use the Secure Socket Layers (SSL) protocol. If in use by the browser, this protocol sends certificates to the server that identifies the browser. The **ClientCertificate** collection property retrieves the values of these client certification fields. An interesting feature of this protocol is the use of the https:// prefix rather than the normal http://.

You can iterate through a collection using a **For Each item in ... Next** loop.

There is one mandatory argument.

Key

The **Key** argument is a string that specifies the name of the field to retrieve. Note the use of the double quotes in the code example. A certificate contains the following seven fields:

Certificate. A string containing the binary stream of the entire certificate content.

Flags. There are two flags that provide additional client certificate information. The **ceCertPresent** flag indicates a client certificate is present. The **ceUnrecognizedIssuer** flag indicates the last certification in this chain is from an unknown user.

Issuer is a string containing the subfield values that provide information about the issuer of the certificate.

SerialNumber is a string that contains the certification serial number.

Subject is a string containing the subfield values that provide information about the subject of the certificate.

ValidFrom is the date when the certificate becomes valid.

ValidUntil is the expiration date of the certificate.

Subfield

The optional **Subfield** argument is used only with the **Issuer** or the **Subject** key fields. It is added to the key field as a suffix. For example, **IssuerL** which is the locality of the Issuer. Or, **SubjectCN** which gives the common name of the user.

The allowed values are:

C is the name of the country of origin.

CN is the common name of the user.

GN is a given name.

I is a set of initials.

L is a locality.

O is the company or organization name.

OU is the name of the organizational unit.

S is a state or province.

T is the title of the person or organization.

Code:

```
<HTML>
```

```
<HEAD>
```

```
</HEAD>
```

```
<BODY>
```

```
This certification is valid from <%= Request.ClientCertificate("ValidFrom") %> until <%=  
Request.ClientCertificate("ValidUntil") %>.
```

```
</BODY>
```

```
</HTML>
```

Output:

```
This certification is valid from 9/26/96 11:59:59 PM until 9/26/97 11:59:59 PM.
```

PROPERTY: Session.CodePage

The **CodePage** property specifies which server codepage to use when displaying dynamic Active Server Pages. A codepage is a character set used for symbol mapping. The default value of the **CodePage** varies by location and language. If you specify a new **CodePage** property, it will exist only for the duration of the session.

Code:

```
<% @CodePage = 437
//Save original codepage information
Session("CodePage") = Session.CodePage

//ID is the CodePage ID from form and
//text is the text entered in the Form
ID = Request.Form("ID")
text = Request.Form("ToTranslate")

//Look up text in specified language. The function FindTranslation
//does the work of finding the translation.
Session.CodePage = ID
translation = FindTranslation(text, ID)

//Change CodePage in order to write in English then change
//it again to write in specified language

If translation <> "" Then
    Session.CodePage = Session("CodePage")
    Response.Write "The translation of " & text & " is "
    Session.CodePage = ID
    Response.Write translation
End If
%>
```

COLLECTION PROPERTY: Session.Contents

Session.Contents(Key)

The **Contents** collection property contains a list of all of the items that has been created and added to the **Session** object through script commands, rather than by using the HTML <OBJECT> tag.

You can iterate through a collection using a **For Each item in ... Next** loop.

There is one mandatory argument.

Key

The **key** argument is the name of the item to retrieve.

Code:

```
<%  
Session("name") = "Application Maker"  
Session("publishdate") = "05/01/99"  
Session("author") = "ISSI"  
Set Session("Obj1") = Server.CreateObject("ADODB.Connection")
```

```
For Each Item in Session.Contents  
  If IsObject( Session.Contents(Item)) Then  
    Response.Write Item & " is an object.<BR>"  
  Else  
    Response.Write Item & "=" & Session.Contents(Item) & "<BR>"  
  End If  
Next  
>%
```

Output:

```
NAME=Application Maker  
PUBLISHDATE=05/01/99  
AUTHOR=ISSI  
OBJ1 is an object.
```

PROPERTY: Response.ContentType

The **ContentType** property specifies the HTTP content type, which includes a type/subtype, for the response header. The type is the general content, and the subtype is the specific content. The default is text/HTML.

This example code will produce a 2 X 2 Excel Spreadsheet in a browser, if Excel is installed on the client machine.

Code:

```
<% Response.ContentType = "application/vnd.ms-excel" %>
<HTML>
<HEAD>
</HEAD>
<BODY>
<TABLE>
<TR>
<TD>2</TD><!-- Cell : A1 --->
<TD>4</TD><!-- Cell : B1 --->
</TR>
<TR>
<TD>5</TD><!-- Cell : A2 --->
<TD>6</TD><!-- Cell : B2 --->
</TR>
</TABLE>
</BODY>
</HTML>
```

COLLECTION PROPERTY: Request.Cookies

Request.Cookies(Name)[(Key)].Attribute

The **Cookies** collection property allows you to access the values of the browser cookies sent in by requests.

There is one mandatory argument.

Name

The **Name** argument specifies the name of the cookie to be retrieved.

There are two optional arguments.

Key

The **Key** argument specifies which value from the cookie dictionary to retrieve.

Attribute

The optional **Attribute** argument is the boolean variable **HasKeys**.

True indicates that the cookie contains keys.

False indicates that the cookie does not contains keys

You can iterate through a collection using a **For Each ... Next** loop.

Code:

```
<%  
For Each objCK In Request.Cookies  
    Response.Write objCK & " = " & Request.Cookies(objCK) & "<br>"  
Next  
>%
```

Output:

PRD2 = VBScript

PRD1 = ASP

MAIN = DevSite

COLLECTION PROPERTY Response.Cookies

Response.Cookies(Name)[(Key)].Attribute=Value

The **Cookies** collection property assigns a new value to a cookie, if it exists; otherwise, the cookie is created and the value is assigned.

There are two mandatory arguments.

Name

The **Name** argument is the name of the cookie.

Value

The **Value** argument specifies the value to assign to the key or attribute.

There are two optional arguments.

Attribute

The optional **Attribute** argument can one of the five following parameters:

Domain is write-only and allows pages on a domain made up of more than one server to share cookie information.

Expires is write-only and is the date on which the cookie expires. Unless a date is specified, the cookie will expire when the session ends. If a date is specified, the cookie will be stored on the client's disk after the session ends.

HasKeys is read-only and uses **Boolean** values to specify whether the cookie contains keys.

Path is write-only and if set, the cookie is sent by this path. If this argument is not set, the default is the application path.

Secure is write-only and indicates if the cookie is secure.

Key

The optional **Key** argument specifies the **Key** to which the **Value** is assigned.

Code:

```
<%  
Response.Cookies("myProduct")("Main") = "DevSite"  
Response.Cookies("myProduct")("Prd1") = "ASP"  
Response.Cookies("myProduct")("Prd2") = "VBScript"  
Response.Write Request.Cookies("myProduct")  
%>
```

Output:

```
PRD2=VBScript&PRD1=ASP&MAIN=DevSite
```

METHOD: Session.Contents.Remove

Implemented in version 3.0

Session.Contents.Remove (Name|Integer)

The **Contents.Remove** method is used to delete one specified item in the **Session.Contents** collection.

You may use only one of the two possible choices for the mandatory argument.

Name or Integer

The **Name** argument is the name of the item to be deleted. It must be enclosed in a pair of quotes.

The **Integer** argument is the position number of the item in the collection to be deleted. The numbering sequence for a collection starts at one, not zero.

Code:

```
<%  
Session("name") = "Session Maker"  
Session("publishdate") = "05/15/01"  
Session("author") = "DevGuru"  
Set Session("Obj1") = Server.CreateObject("ADODB.Connection")
```

```
Session.Contents.Remove(1)  
Session.Contents.Remove("publishdate")
```

```
For Each Item in Session.Contents  
  If IsObject(Session.Contents(Item)) Then  
    Response.Write Item & " is an object.<BR>"  
  Else  
    Response.Write Item & "=" &Session.Contents(Item) & "<BR>"  
  End If  
Next  
>%
```

Output:

```
author=DevGuru  
Obj1 is an object.
```

METHOD: Session.Contents.RemoveAll

Implemented in version 3.0

Session.Contents.RemoveAll

The **Contents.RemoveAll** method deletes all items that are in the **Session.Contents** collection.

Remember to include the pair of empty ().

Code:

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

METHOD: Server.CreateObject

Server.CreateObject(ObjectID)

The **CreateObject** method is probably the most widely used and the most important method available through the Built-in Active Server Pages Objects. It allows you to instantiate the components of a script, or in different terms, create an instance of other objects. As a direct consequence, you can use and access any collections, events, methods, and properties associated with these objects.

There is one mandatory argument.

ObjectID

The **ObjectID** argument specifies the type of object to be created.

The following code creates a connection to a database:

Code:

```
<% Set myconn = Server.CreateObject("ADODB.Connection") %>
```

METHOD: Response.End

Response.End

The **End** method orders the web server to stop processing the script. The current results are returned and no further processing occurs. If **Response.Buffer** is set to **True**, **Response.End** will flush the buffer and then end.

Note that in ASP version 2.0, the default for **Response.Buffer** is set to **False**, however in ASP version 3.0, the default for **Response.Buffer** is set to **True**.

In the example, the second **Response.Write** will not be displayed in the output.

Code:

```
<%  
Response.Write "Hello World"  
Response.End  
Response.Write "Is this the End?"  
%>
```

Output:

Hello World

METHOD: Server.Execute

Implemented in version 3.0

Server.Execute(Path)

The **Execute** method allows you to call another ASP page from inside an ASP page. When the called ASP page completes its tasks, you are then returned to the calling ASP page. The overall effect is very similar to a function or subroutine call. Any text or output from the called ASP page will be displayed on the calling ASP page. The **Execute** method is a more useful alternative to using server-side includes.

In contrast, the **Transfer** method allows you to transfer from one ASP page to another without returning to the calling ASP page.

There is one mandatory argument.

Path

The **Path** argument is a string specifying either the absolute or relative path of the ASP page being called. The file name must be included in the path. The entire **Path** must be enclosed inside a pair of quotes. The **Path** argument cannot include a query string, however, any query string that was available to the calling ASP page will be available to the called ASP page.

Code:

```
-----CallingAsp.asp-----
```

```
<HTML>  
<BODY>
```

```
How now <%Server.Execute("CalledAsp.asp")%> cow?
```

```
</BODY>  
</HTML>
```

```
-----CalledAsp.asp-----
```

```
<%  
Response.Write "pink"  
%>
```

Output:

```
How now pink cow?
```

PROPERTY: Response.Expires

The **Expires** property specifies a length of time in minutes before a response page cached on the browser expires.

The first line of code causes the page to be refreshed (reloaded) every time it is accessed, rather than being stored in a cache.

In the second example, if the user returns to this page before the 15 minutes are up, the cached version of the page will be displayed.

```
<% Response.Expires = 0 %>
```

```
<% Response.Expires = 15 %>
```

PROPERTY: Response.ExpiresAbsolute

The **ExpiresAbsolute** property sets the precise date and time when a response page cached on the browser will expire. If the time is not given, then the page will expire exactly at midnight on the given date. If the date is not given, then the page will expire at the given time on the same day that the script is run.

The following code specifies that the page will expire at 6:00 PM on May 15th of 1999. You must use a 24 hour clock for the time and four digits for the year.

```
<% Response.ExpiresAbsolute=#May 15, 1999 18:00:00# %>
```

METHOD: Response.Flush

Response.Flush

The **Flush** method immediately sends all current buffered page content to the client if the **Response.Buffer** property is set to **True**.

If the buffer does not exist, because **Response.Buffer** is set to **False**, a runtime error will occur.

Note that in ASP version 2.0, the default for **Response.Buffer** is set to **False**, however in ASP version 3.0, the default for **Response.Buffer** is set to **True**.

Code:

```
<%  
Response.Flush  
%>
```

COLLECTION PROPERTY: Request.Form

Request.Form(Element)[(Index)].Count

The **FORM** collection property contains, and allows you to retrieve, the values of the <FORM> elements posted to a request form using the POST method.

You can iterate through a collection using a **For Each item in ... Next** loop.

There is one mandatory argument.

Element

The **Element** argument is the name of the <FORM> element which is used to retrieve a value.

There is one optional argument.

Index

The **Index** argument is used when the **Element** argument has more than one value. It is an integer, ranging from 1 to **Request.Form(Element).Count**, which is used to select values from a <FORM> element. **Count** is the total number of multiple values.

Code:

```
-----File1.asp-----
<HTML>
<HEAD>
</HEAD>
<BODY>
<form action="File2.asp" method="POST">
Name:<input type="Text" name="Name" maxlength="20"><br>
Company:<input type="Text" name="Company" maxlength="20"><br>
Position:<input type="Text" name="Position" maxlength="20"><br>
Address:<textarea name="Address" rows="3"></textarea><br>
Phone:<input type="Text" name="Phone" maxlength="20"><br>
<input type="Submit" name="Submit" value="Submit">
</form>
</BODY>
</HTML>
-----File2.asp-----
<HTML>
<HEAD>
</HEAD>
<BODY>
<% Response.Write Request.Form("Name")%> works for
<% Response.Write Request.Form("Company") %> at address
<% Response.Write Request.Form("Address") %> as a
<% Response.Write Request.Form("Position") %>.<br>
</BODY>
</HTML>
```

Output:

Jane Doe works for ISSI at address 5609 Kington Pike Knoxville, TN as a Web Designer.

METHOD: Server.HTMLEncode

Server.HTMLEncode(String)

The **HTMLEncode** method applies HTML encoding syntax to a specified string of ASCII characters. For example, this allows you to display a HTML tag on a web page and not have it treated as an actual tag.

There is one mandatory argument.

String

The **String** argument is the string to be encoded.

Code:

```
<% Response.Write Server.HTMLEncode("The tag for a table is: <Table>") %>
```

Output:
The tag for a table is: <Table>

Browser Output:
The tag for a table is: <Table>

PROPERTY: Response.IsClientConnected

Modified in version 3.0

The **IsClientConnected** property determines if the client has disconnected from the server since the last **Response.Write**. This property is particularly useful since it will prevent the server from continuing to attempt to pass information after an unexpected disconnect.

In ASP 2.0, you must first attempt to pass information to the client in order to use this property. However, in ASP 3.0 you can check the response of using this property before passing any content to the client. (Thus, a simple test will prevent the server from performing unnecessary work.)

In the example code, the output will be a list of all the book titles that are in the database. However, if the user stops the connection to the server, then the only a partial list of titles will be created.

Code:

```
<%  
set connDB=server.createobject("adodb.connection")  
connDB.Open "books", "", ""  
mySQL="select * from books"  
Set rsBookSrch = Server.CreateObject("ADODB.Recordset")  
rsBookSrch.Open mySQL, connDB, adOpenStatic, adLockPessimistic  
%>  
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
<%  
Do until (rsBookSrch.eof or Response.IsClientConnected=false)  
Response.Write rsBookSrch.Fields("Title") %> <br>  
<% rsBookSrch.MoveNext  
Loop  
rsBookSrch.Close  
connDB.Close  
%>  
</BODY>  
</HTML>
```

PROPERTY: Session.LCID

The **LCID** property uses the location identifier number to access information on display format issues, such as currency, date, and time, that are specific to a location or region. The location identifier number uniquely defines a geographic location or region. For example, the location identifier number for France is 1036.

Code:

```
<%
```

```
Response.Write "This is the LCID for this page " & Session.LCID & "<BR>"
```

```
Response.Write "This is the Time and Date format for the above LCID: " & NOW() & "<BR>"
```

```
Session.LCID = 1033
```

```
Response.Write "This is the LCID for this page now " & Session.LCID & "<BR>"
```

```
Response.Write "This is the Time and Date format for the above LCID: " & NOW() & "<BR>"
```

```
Session.LCID = 3079
```

```
%>
```

Output:

This is the LCID for this page 1033

This is the Time and Date format for the above LCID 5/12/99 10:01:33 PM

This is the LCID for this page now 3079

This is the Time and Date format for the above LCID 12.05.99 22:01:33

METHOD: Server.MapPath

Server.MapPath(Path)

The **MapPath** method maps a relative or virtual path to a physical path. This method does not check for the validity or the existence of the physical path. If the path starts with a forward or backward slash, the method returns the path as if the path is a full virtual path. If the path does not start with a slash, then the method returns the path relative to the directory of the ASP file being processed.

There is one mandatory argument.

Path

The **Path** argument is the path to be mapped.

Code:

```
<HTML>
<HEAD>
</HEAD>
<BODY>
The path of this file is <% Response.Write Server.MapPath("test.asp")
%>
The path of the file1 is <% Response.Write Server.MapPath("~/test.asp")
%>
The path of the file2 is <% Response.Write Server.MapPath("test\\test.asp")
%>
The path of the file3 is <% Response.Write Server.MapPath("")
%>
</BODY>
</HTML>
```

Output:

```
The path of this file is D:\inetpub\wwwroot\test.asp
The path of the file1 is D:\inetpub\wwwroot\test.asp
The path of the file2 is D:\inetpub\wwwroot\test\test.asp
The path of the file3 is D:\inetpub\wwwroot
```

OBJECT:ObjectContext

The **ObjectContext** object is used to control Active Server Pages transactions which are managed by the Microsoft Transaction Server (MTS). You may either complete (commit) or abort a transaction.

The **@Transaction** directive must be on the first line of the .asp file in order to use this object (see code examples). The **@Transaction** directive causes a transaction to run to completion unless an abort occurs.

METHODS

[SetAbort](#) Method

The **SetAbort** method declares that a transaction has not been completed and resources should not be updated.

[SetComplete](#) Method

The **SetComplete** method declares that the transaction has been completed and resources can be updated.

EVENTS

[OnTransactionAbort](#) Event

The **OnTransactionAbort** event occurs when a transaction has been aborted for some kind of processing error. The signal of this event will run a handler script in the same file, if it exist.

[OnTransactionCommit](#) Event

The **OnTransactionCommit** event occurs when a transaction has been successfully completed. The signal of this event will run a handler script in the same file, if it exist.

METHOD:ObjectContext.SetAbort

ObjectContext.SetAbort

The **SetAbort** method explicitly declares that a transaction has not been completed and prevents resources from being updated.

If it exists, the **OnTransactionAbort** event is processed.

Code:

-----File1.asp-----

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<form action="File2.asp" method="POST">
ProductID:<input type="Text" name="ProductID">
Product Name:<input type="Text" name="Name">
Quantity Wanted:<input type="Text" name="Order">
<input type="Submit" name="Submit" value="Submit">
</form>
</BODY>
</HTML>
```

-----File2.asp-----

```
<%
@Transaction = "Required"
set connDB=server.createobject("adodb.connection")
connDB.Open "products", "", ""
productID = Request.Form("ProductID")
mySQL="Select * from products Where productID = " & productID
Set rsProductSrch = Server.CreateObject("ADODB.Recordset")
rsProductSrch.Open mySQL, connDB, adOpenStatic, adLockPessimistic
If rsProductSrch.BOF Or rsProductSrch.EOF Then
    ObjectContext.SetAbort
Elseif rsProductSrch.Fields("Quantity") < Request.Form("Order") Then
    Response.Write "There is not enough in stock to give the present order. There are " &
rsProductSrch.Fields("Quantity") & "in stock and your order is for " & Request.Form("Order") &
".<br>"
    ObjectContext.SetAbort
Else
    rsProductSrch.Fields("Quantity") = rsProductSrch.Fields("Quantity") -
        Request.Form("Order")
    rsProductSrch.Update
    ObjectContext.SetComplete
End If
rsProductSrch.Close
connDB.Close
%>
```

```
Sub OnTransactionAborted()
Response.Write "The transaction aborted because there was not enough quantity on hand to
fill your order."
End Sub
```

Sub OnTransactionCommit()

Response.Write "The transaction was committed and your order is being sent."

End Sub

There is no quantity of the desired product.

Output:

The transaction aborted because there was not enough quantity on hand to fill your order.

There is not enough of the desired product.

Output:

There is not enough in stock to give the present order. There are 20 in stock and your order is for 25.

The transaction aborted because there was not enough quantity on hand to fill your order.

METHOD:ObjectContext.SetComplete

ObjectContext.SetComplete

The **SetComplete** method declares that the transaction of this object has been completed. In order for a transaction to be successfully completed, all individual components of the transaction must also be completed and have called **SetComplete**. By default, a script is considered to be completed if it runs to conclusion and does not abort. Therefore, in many cases, it may not be necessary to call **SetComplete** from the script.

If it exists, the **OnTransactionCommit** event is processed.

Code:

-----File1.asp-----

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<form action="File2.asp" method="POST">
ProductID:<input type="Text" name="ProductID">
Product Name:<input type="Text" name="Name">
Quantity Wanted:<input type="Text" name="Order">
<input type="Submit" name="Submit" value="Submit">
</form>
</BODY>
</HTML>
```

-----File2.asp-----

```
<%
@Transaction = "Required"
set connDB=server.createobject("adodb.connection")
connDB.Open "products", "", ""
productID = Response.Form("ProductID")
mySQL="Select * from products Where productID = " & productID
Set rsProductSrch = Server.CreateObject("ADODB.Recordset")
rsProductSrch.Open mySQL, connDB, adOpenStatic, adLockPessimistic
If rsProductSrch.BOF Or rsProductSrch.EOF Then
    ObjectContext.SetAbort
Elseif rsProductSrch.Fields("Quantity") < Response.Form("Order") Then
    Response.Write "There is not enough in stock to give the present order. There are " &
rsProductSrch.Fields("Quantity") & "."
    ObjectContext.SetAbort
Else
    rsProductSrch.Fields("Quantity") = rsProductSrch.Fields("Quantity") -
        Request.Form("Order")
    rsProductSrch.Update
    ObjectContext.SetComplete
End If
rsProductSrch.Close
connDB.Close
%>
```

```
Sub OnTransactionAborted()
Response.Write "The transaction aborted because there was not enough quantity on hand to
```

fill your order."
End Sub

Sub OnTransactionCommit()
Response.Write "The transaction was committed and your order is being sent."
End Sub

There is enough to fill the order.

Output:

The transaction was committed and your order is being sent.

EVENT:ObjectContext.OnTransactionAbort

The **OnTransactionAbort** event occurs when a transaction has been aborted for some kind of processing error. The **OnTransactionAbort** event is simply a subroutine with a reserved name that is referenced by the script. It can contain any script you may wish to run when an abort occurs.

Code:

```
<%  
@Transaction=Required Language="VBScript"  
Response.Buffer = TRUE  
Server.ScriptTimeout = 5  
%>  
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
<%  
Do  
    x=x+1  
    Response.Write x & "<BR>"  
Loop While x < 10000  
%>  
</BODY>  
</HTML>
```

```
Sub OnTransactionAbort()  
Response.Write "The transaction aborted because the script ran out of time. This is the  
Transaction Abort Handler."  
End Sub
```

Output:

The transaction aborted because the script ran out of time. This is the Transaction Abort Handler.

EVENT:ObjectContext.OnTransactionCommit

The **OnTransactionCommit** event occurs when a transaction has been completed. The **OnTransactionCommit** event is simply a subroutine with a reserved name that is referenced by the script. It can contain any script you may wish to run when the transaction is completed.

Code:

```
<% @Transaction=Required Language="VBScript" %>
<HTML>
<HEAD>
</HEAD>
<BODY>
<%
Do
    x=x+1
    Response.Write x & "<BR>"
Loop While x < 5
%>
</BODY>
</HTML>
```

```
Sub OnTransactionCommit()
```

```
Response.Write "The transaction committed. This is the Transaction Commit Handler."
```

```
End Sub
```

Output:

```
1
2
3
4
5
The transaction committed. This is the Transaction Commit Handler.
```

EVENT: Session_OnEnd

The **Session_OnEnd** event occurs when the **Session** ends. Normally, a **Session** is ended by being timed-out, either by default or by using **Session.Timeout**, or by being abandoned by using **Session.Abandon**. The **Session_OnEnd** event is simply a subroutine with a reserved name that is placed in the Global.asa file. The code in the event cannot affect the user because he has quit the site.

In the example, there is no output because the user is already gone. The first line must specify the script language.

Note, only the Application, Server, and Session built-in objects are available from within the OnEnd event handler.

Code:

```
-----Global.asa-----
```

```
<script Language="VBScript" RUNAT=Server>
```

```
Sub Application_OnEnd()
```

```
End Sub
```

```
Sub Application_OnStart()
```

```
Application("NumSession") = 0
```

```
Application("NumVisited") = 0
```

```
End Sub
```

```
Sub Session_OnEnd()
```

```
Application("NumSession") = Application("NumSession") - 1
```

```
End Sub
```

```
Sub Session_OnStart()
```

```
Application("NumSession") = Application("NumSession") + 1
```

```
Application("NumVisited") = Application("NumVisited") + 1
```

```
End Sub
```

```
</script>
```

```
-----File1.asp-----
```

```
Response.Write "You are " & Application("NumSession") & " of " & Application("NumVisited") & " users."
```

EVENT: Session_OnStart

The **Session_OnStart** event occurs just before the server creates the **Session** object. The **Session_OnStart** event is simply a subroutine with a reserved name that is placed in the Global.asa file. You can place any code you desire inside this subroutine. All of the built-in ASP objects are available from within the Session_OnStart event handler.

Code:

```
-----Global.asa-----
<script Language="VBScript" RUNAT=Server>
Sub Application_OnEnd()
End Sub

Sub Application_OnStart()
Application("NumSession") = 0
Application("NumVisited") = 0
End Sub

Sub Session_OnEnd()
Application("NumSession") = Application("NumSession") - 1
End Sub

Sub Session_OnStart()
Application("NumSession") = Application("NumSession") + 1
Application("NumVisited") = Application("NumVisited") + 1
End Sub
</script>

-----File1.asp-----
Response.Write "You are " & Application("NumSession") & " of " & Application("NumVisited") &
" users."
```

Output:

You are 1 of 3 users.

PROPERTY: Response.Pics

The **Pics** property adds the value of the PICS rating to the pics-label field in the the response header for a document or a site. This statement must be located before the <HTML> tag. The PICS label has quotes in the label, and the quotes must be replaced with, & chr(34) &. Or you can use double quotes.

The PICS rating is as follows:

violence = 2
sex = 1
language = 2
nudity = 0

Using chr(34):

Code:

```
<% Response.Pics("(PICS-1.1 <http://www.somesite.com/rating01.html> labels on " & chr(34)
& "1997.01.05R08:15-0500" & chr(34) & " until " & chr(34) & "1999.12.31T23:59-0000" &
chr(34) & " ratings(v 2 s 1 l 2 n 0))")
%>
```

Using double quotes:

```
<% Response.Pics("(PICS-1.1 labels on ""1997.01.05R08:15-0500"" until ""1999.12.31T23:59-
0000"" ratings(v 2 s1 l 2 n 0))")
%>
```

COLLECTION PROPERTY: Request.QueryString

Request.QueryString (Variable)[(Index)].Count

The **QueryString** collection property contains, and allows you to retrieve, the values of the variables in the query string which are the statements that follows a question mark. Many different processes can produce a query string such as an anchor tag, a form submission, or typing the string into the address box of the browser.

You can iterate through a collection using a **For Each item in ... Next** loop.

There is one mandatory argument.

Variable

The **Variable** specifies the name of the variable in the query string whose value is to be retrieved.

There is one optional argument.

Index

The optional **Index** argument is used when the **Variable** argument has more than one value. It is an integer, ranging from 1 to **Request.QueryString(Variable).Count**, which is used to select values from the **Variable**. **Count** is the total number of multiple values.

Code:

-----File1.asp-----

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<A HREF="File2.asp?language=ASP&type=general">Query sample</A>
</BODY>
</HTML>
```

-----File2.asp-----

```
<%
For Each item In Request.QueryString
    Response.Write(item & " = " & Request.QueryString(item) & VbCrLf)
Next
%>
```

Output:

language = ASP

type = general

METHOD: Response.Redirect

Response.Redirect(URL)

The **Redirect** method stops processing the current script and attempts to connect the client to a different URL. This is accomplished by adding an HTTP redirection header to the output stream that is being sent from the server to the client. Unfortunately, if page content has already been sent to the client and if a proxy server lies between the server and the client, an error message can be generated. Therefore it is advisable to set **Response.Buffer** to true and to call **Response.Clear** just before calling **Redirect**.

Note in ASP 2.0, the buffering default setting is false and in ASP 3.0, the buffering default setting is true.

There is one mandatory argument.

URL

The **URL** argument is the Uniform Resource Locator (URL) that the browser is redirected to.

Code:

```
-----File1.asp-----
<% Response.Buffer = true %>
<HTML>
<BODY>
<%
Response.Write "This is File1.asp and switching to File2.asp"
Response.Clear
Response.Redirect "File2.asp"
%>
</BODY>
</HTML>
```

```
-----File2.asp-----
<HTML>
<BODY>
<%
Response.Write "This is File2.asp"
%>
</BODY>
</HTML>
```

Output:

File1 is written and then the browser will load File2:

```
-----File1.asp-----
This is File1.asp and switching to File2.asp
```

```
-----File2.asp-----
This is File2.asp
```

OBJECT: Request

The **Request** object provides access to all of the information that is passed in a request from the browser to the server. This information is stored among five types of **Request** collections. A collection is similar to a data structure or array. Individual items in the collection are accessed via a unique key assigned to that item.

All variables can be accessed directly by calling **Request(variable)** without the collection name. In this case, the Web server searches the collections in the following order: **QueryString**, **Form**, **Cookies**, **ClientCertificate**, then **ServerVariables**.

PROPERTY

[TotalBytes](#) Property

The **TotalBytes** property specifies the total number of bytes sent in the body of the HTTP request.

COLLECTION PROPERTIES

[ClientCertificate](#) Collection Property

The **ClientCertificate** collection property contains the values of the client certification fields of the request.

[Cookies](#) Collection Property

The **Cookies** collection property contains the values of the cookies sent in the request.

[Form](#) Collection Property

The **Form** collection property contains the values of the <FORM> elements posted to a form using the POST method.

[QueryString](#) Collection Property

The **QueryString** collection property contains the values of the variables in the HTTP query string, which are the statements that follow a question mark.

[ServerVariables](#) Collection Property

The **ServerVariables** collection property contains the values of server environmental variables. This allows access to the HTTP headers.

METHOD

[BinaryRead \(Count\)](#) Method

The **BinaryRead** method retrieves the data that was sent to the server from the browser as part of a POST request, and returns the number of bytes read.

PROPERTY: Request.TotalBytes

Request.TotalBytes

The **TotalBytes** property is a read-only value that specifies the total number of bytes sent by the client to the server in the body of the HTTP request.

The value returned by **TotalBytes** can be used as the argument of **Request.BinaryRead(Count)**.

Code:

```
-----File1.html-----
<HTML>
<HEAD>
</HEAD>
<BODY>
<FORM ACTION="File2.asp" METHOD="POST">
Name:<INPUT TYPE="Text" NAME="name" MAXLENGTH="30"><BR>
Age: <INPUT TYPE="Text" NAME="age" MAXLENGTH="10"><BR>
Sex: <INPUT TYPE="Text" NAME="sex" MAXLENGTH="10"><BR>
<INPUT TYPE="Submit" NAME="submit" VALUE="submit"><BR>
</FORM>
</BODY>
</HTML>
```

```
-----File2.asp-----
<%
Dim ByteCount
ByteCount = Request.TotalBytes
Response.Write("ByteCount = " & ByteCount & " bytes")
%>
```

Output

It is dependent on the data entered into the form.

For example, if Name=Bill, Age=56, and Sex=male.

ByteCount = 39 bytes

COLLECTION PROPERTY: Request.ServerVariables

Request.ServerVariables (EnvironmentVariable)

The **ServerVariables** collection property contains, and allows you to retrieve, the values of server environment variables.

You can iterate through a collection using a **For Each item in ... Next** loop.

There is one mandatory argument.

EnvironmentVariable

The **EnvironmentVariable** argument can be any one of the following forty three variables:

ALL_HTTP are all HTTP headers sent by the client.

ALL_RAW retrieves all headers in the raw-form.

APPL_MD_PATH retrieves the metabase path.

APPL_PHYSICAL_PATH retrieves the physical path corresponding to the metabase path.

AUTH_PASSWORD is the value entered in the client's authentication dialog.

AUTH_TYPE is the authentication method that the server uses to validate users.

AUTH_USER is the raw authenticated user name.

CERT_COOKIE is an unique ID for client certificate which is returned as a string.

CERT_FLAGS is where bit0 is set to 1 if the client certificate is present, and bit1 is set to 1 if the certifying authority of the client certificate is invalid.

CERT_ISSUER is the issuer field of the client certificate.

CERT_KEYSIZE is the number of bits in Secure Sockets Layer connection key size.

CERT_SECRETKEYSIZE is the number of bits in the server certificate private key.

CERT_SERIALNUMBER is the serial number field of the client certificate.

CERT_SERVER_ISSUER is the issuer field of the server certificate.

CERT_SERVER_SUBJECT is the subject field of the server certificate.

CERT_SUBJECT is the subject field of the client certificate.

CONTENT_LENGTH is the length of the content header as sent by the client.

CONTENT_TYPE is the data type of the content.

GATEWAY_INTERFACE is the revision of the CGI specification used by the server.

HTTP_<HeaderName> is the value stored in the header HeaderName.

HTTP_ACCEPT returns the value of the Accept header.

HTTP_ACCEPT_ENCODING returns the value of the Accept encoding.

HTTP_ACCEPT_LANGUAGE returns a string that specifies the language to be used for displaying content.

HTTP_CONNECTION returns a string containing information about the connection.

HTTP_COOKIE returns the cookie string that was included with the request.

HTTP_HOST returns a string containing information about the host.

HTTP_REFERER returns a string containing the original URL when a redirect has occurred.

HTTP_USER_AGENT returns a string describing the browser used to send the request.

HTTP_UA_PIXELS returns a string detailing the screen resolution of the user agent.

HTTP_UA_COLOR returns a string with color information.

HTTP_UA_OS returns a string stating the operating system of the user agent.

HTTP_UA_CPU returns a string stating the processor type used by the user agent.

HTTPS returns ON if the request came in through secure channel or OFF if the request is through a non-secure channel.

HTTPS_KEYSIZE is the number of bits in Secure Sockets Layer (SSL) connection key size.

HTTPS_SECRETKEYSIZE is the number of bits in server certificate private key.

HTTPS_SERVER_ISSUER is the issuer field of the server certificate.

HTTPS_SERVER_SUBJECT is the subject field of the server certificate.

INSTANCE_ID is the ID for the Internet Information Server (IIS) instance in text format.

INSTANCE_META_PATH is the metabase path for the instance of Internet Information Server (IIS) that responds to the request.

LOCAL_ADDR returns the server address on which the request came in.

LOGON_USER is the Windows account that the user is logged into.

PATH_INFO is extra path information as given by the client.

PATH_TRANSLATED is a translated version of **PATH_INFO** that takes the path and performs any necessary virtual-to-physical mapping.

QUERY_STRING is the query information stored in the string following the question mark (?) in the HTTP request.

REMOTE_ADDR is the Internet Protocol (IP) address of the remote host making the request.

REMOTE_HOST is the name of the host making the request.

REMOTE_USER is an unmapped user-name string sent in by the user.

REQUEST_METHOD is the method used to make the request.

SCRIPT_NAME is a virtual path to the script being executed.

SERVER_NAME is the server's host name, Domain Name Server (DNS) alias, or Internet Protocol (IP) address.

SERVER_PORT is the port number to which the request was sent.

SERVER_PORT_SECURE is a string that contain a 1 if the request is being handled on the secure port, otherwise it is 0.

SERVER_PROTOCOL is the name and revision of the request information protocol.

SERVER_SOFTWARE is the name and version of the server software that answers the request and runs the gateway.

URL is the base portion of the URL.

Code:

```
<%  
IPAddress = Request.ServerVariables("REMOTE_ADDR")  
Software = Request.ServerVariables("SERVER_SOFTWARE")  
Protocol = Request.ServerVariables("SERVER_PROTOCOL")  
Response.Write "Your IP Address is " & IPAddress & " and your server is running " & Software  
& " under " & Protocol & "protocol."  
>%
```

Output:

```
Your IP Address is 127.0.0.1 and your server is running Microsoft-IIS/4.0 under HTTP/1.1  
protocol.
```

OBJECT: Response

Modified in version 3.0

The Response object sends information back to the user (browser).

PROPERTIES

Buffer Property Modified in version 3.0

The **Buffer** property tells whether the page output being sent to the browser is buffered.

CacheControl Property

The **CacheControl** property determines whether a proxy server can cache the Active Server Page.

Charset Property

The **Charset** property appends the name of the character set being used to the content-type header contained in the response object.

ContentType Property

The **ContentType** property specifies the HTTP content type/subtype for the response header.

Expires Property

The **Expires** property specifies the length of time in minutes until a cached page on the browser expires.

ExpiresAbsolute Property

The **ExpiresAbsolute** property specifies a date and time when a cached page on the browser will expire.

IsClientConnected Property Modified in version 3.0

The **IsClientConnected** property indicates whether the browser has disconnected from the server since the last Response.Write.

PICS Property

The **PICS** property specifies the value of a PICS rating label for a document or a site.

Status Property

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

COLLECTION PROPERTIES

Cookies Collection Property

The **Cookies** collection property allows you to add a cookie to a browser and add values to the cookie.

METHODS

AddHeader Name, Value Method

The **AddHeader** method adds a new named HTTP header with a specific value to the response.

AppendToLog (String) Method

The **AppendToLog** method adds a string to the end of an entry in the Web server log for this

request.

BinaryWrite (Data) Method

The **BinaryWrite** method sends specific data to the current HTTP output without any character conversions.

Clear Method

The **Clear** method clears (erases) any buffered HTML output.

End Method

The **End** method causes the web server to stop processing the script and to return the current results without processing the rest of the file.

Flush Method

The **Flush** method sends the contents of the buffer.

Redirect (URL) Method

The **Redirect** method tries to connect the browser to a different URL.

Write (variant) Method

The **Write** method sends any specified variant to the browser.

PROPERTY: Response.Status

The **Status** property specifies the value of the HTTP status line returned by the server. This is a three digit number followed by a short description.

```
Code:
<%
IPAddress = Request.ServerVariables("REMOTE_ADDR")
If IPAddress <> "208.5.64.223" Then
    Response.Status = "403 Access Forbidden"
    Response.Write Response.Status
    Response.End
End If
%>

<HTML>
<HEAD>
</HEAD>
<BODY>
You have accessed this page through the IP Address of 208.5.64.223. </BODY>
</HTML>
```

The output of the above is the following if the IP address of the client is 208.5.64.223:
You have accessed this page through the IP Address of 208.5.64.223.

Otherwise the output is as follows if the IP address is not 208.5.64.223:
403 Access Forbidden

METHOD: Response.Write

Response.Write(Variant)

The **Write** method writes any specified string to the HTTP output.

There is one mandatory argument.

Variant

The **Variant** argument is the data to be written as a string. A variant is a **VBScript** data type and includes characters, integers, and strings. The variant cannot contain the character combination of %>.

Code:

```
<%  
Response.Write "Hello World"  
Response.Write "<BR>"  
%>
```

Output:

```
Hello World<BR>
```

PROPERTY: Server.ScriptTimeout

The **ScriptTimeout** property specifies the amount of runtime in seconds for a script before it terminates. The default value is 90 seconds.

The following code causes a script to timeout if the script takes more than 150 seconds to complete.

```
<% Server.ScriptTimeout = 150 %>
```

The value of the **ScriptTimeout** property can be retrieved as well.

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

OBJECT: Server

Modified in version 3.0

The Server object provides access to the utility functions of the server.

PROPERTY

[ScriptTimeout](#) Property

The **ScriptTimeout** property is the amount of runtime in seconds for a script before it terminates.

METHODS

[CreateObject \(ObjectID\)](#) Method

The **CreateObject** method creates an instance of an object to use in an Active Server Page.

[Execute \(Path\)](#) Method Implemented in version 3.0

The **Execute** method allows you to call another ASP page from inside an ASP page. When the called ASP page completes its tasks, you are then returned to the calling ASP page.

[GetLastError](#) Method Implemented in version 3.0

The **GetLastError** method returns an **ASPErrors** object that describes any pre-processing, runtime, or script compiling errors that occurred.

[HTMLEncode \(String\)](#) Method

The **HTMLEncode** method applies HTML syntax to a specified string of ASCII characters.

[MapPath \(Path\)](#) Method

The **MapPath** method maps a relative or virtual path to a physical path.

[Transfer \(Path\)](#) Method Implemented in version 3.0

The **Transfer** method allows you to transfer all of the state information for all of the built-in objects from one ASP page to another. Unlike the **Execute** method, when the ASP page that you have transferred to is finished, you do not return the original ASP page.

[URLEncode \(String\)](#) Method

The **URLEncode** method applies URL rules to a specified string of ASCII characters.

METHOD: Server.Transfer

Implemented in version 3.0

Server.Transfer (Path)

The **Transfer** method allows you to transfer from inside one ASP page to another ASP page. All of the state information that has been created for the first (calling) ASP page will be transferred to the second (called) ASP page. This transferred information includes all objects and variables that have been given a value in an **Application** or **Session** scope, and all items in the **Request** collections. For example, the second ASP page will have the same **SessionID** as the first ASP page.

When the second (called) ASP page completes its tasks, you do not return to the first (calling) ASP page.

In contrast, the **Execute** method allows you to call another ASP page, and when the called page has completed its tasks, you return to the calling ASP page.

There is one mandatory argument.

Path

The **Path** argument is a string specifying either the absolute or relative path of the ASP page being called. The file name must be included in the path. The entire **Path** must be enclosed inside a pair of quotes.

In this example, the first ASP page (CallingAsp.asp) has no output.

Code:

```
-----CallingAsp.asp-----
<%
Application("name") = "Application Maker"
Application("publishdate") = "05/15/01"
Application("author") = "DevGuru"
Set Application("Obj1") = Server.CreateObject("ADODB.Connection")
```

```
Server.Transfer("CalledAsp.asp")
%>
```

```
-----CalledAsp.asp-----
<%;
Response.Write "Output from CalledAsp.asp"
For Each Item in Application.Contents
  If IsObject( Application.Contents(Item)) Then
    Response.Write Item & " is an object.<BR>"
  Else
    Response.Write Item & "=" & Application.Contents(Item) & "<BR>"
  End If
Next
%>
```

Output:

```
Output from CalledAsp.asp
name=Application Maker
```

publishdate=05/15/01
author=DevGuru
OBJ1 is an object.

METHOD: Server.URLEncode

Server.URLEncode(String)

The **URLEncode** method takes a string and converts it into a URL-encoded format. For example, you can use **URLEncode** to ensure that hyperlinks in your Active Server Pages are in the correct format.

There is one mandatory argument.

String

The **String** argument is the string to be encoded.

Code:

```
<% Response.Write Server.URLEncode("http://www.issi.net") %>
```

Output:

```
http%3A%2F%2Fwww%2Eissi%2Enet
```

OBJECT: Session

Modified in version 3.0

The first time a user makes a document request, a **Session** object is assigned to the user and it is used to store, share, and retrieve information concerning the user session. By default, the **Session** object is destroyed after a session has been idle for twenty minutes.

PROPERTIES

[CodePage](#) Property

The **CodePage** property specifies the server codepage that will be used to display dynamic content.

[LCID](#) Property

The **LCID** property specifies the location identifier that will be used to display dynamic content.

[SessionID](#) Property

The **SessionID** property returns the unique identifier for a user session.

[Timeout](#) Property

The **Timeout** property is the length of time in minutes before an idle session is terminated.

COLLECTION PROPERTIES

[Contents](#) Collection Property

The **Contents** collection property contains all of the items that have been created and added to the **Session** object through script commands, rather than using the HTML <OBJECT> tag.

[StaticObjects](#) Collection Property

The **StaticObjects** collection property contains all of the items created in the Global.asa file using the HTML <OBJECT> tag within the scope of the **Session** object.

METHODS

[Abandon](#) Method

The **Abandon** method terminates a user session, destroys all data and objects in the current **Session** object, and releases its resources.

[Contents.Remove](#) Method Implemented in version 3.0

The **Contents.Remove** method is used to remove a single item from a **Contents** collection.

[Contents.RemoveAll](#) Method Implemented in version 3.0

The **Contents.RemoveAll** method is used to remove all items from a **Contents** collection.

EVENTS

[OnEnd](#) Event

The **OnEnd** event occurs when the **Session** quits. The signal of this event will run a handler script in the Global.asa file, if the script exists.

[OnStart](#) Event

The **OnStart** event occurs before the start of any new **Session** by a user. The signal of this event will run a handler script in the Global.asa file, if the script exists.

PROPERTY: Session.SessionID

The **SessionID** property returns the unique session identifier for the current user. The identifier is generated by the server and is a **Long** data type.

Code:

```
<% Response.Write "Your Session ID is " & Session.SessionID %>
```

Output:

Your Session ID is 465107831

PROPERTY: Session.Timeout

The **Timeout** property specifies the number of minutes that can elapse while a session is idle before it is terminated. The default is 20 minutes. If the user does not refresh the page or request another page within the allotted time, the session is automatically terminated and the resources are released.

The following code causes the session to terminate if the session is idle for more than 35 minutes.

```
<% Session.Timeout = 35 %>
```

COLLECTION PROPERTY: Session.StaticObjects

Session.StaticObjects(Key)

The **StaticObjects** collection property contains a list of all of the items that have been created and added to the **Session** object using the HTML <OBJECT> tag.

You can iterate through a collection using a **For Each item in ... Next** loop.

There is one mandatory argument.

Key

The **Key** argument is the name of the item to retrieve.

Code:

```
-----Global.asa-----
```

```
<OBJECT RUNAT=Server SCOPE=Session ID=MyInfo PROGID="MSWC.MyInfo">
</OBJECT>
```

```
<OBJECT RUNAT=Server SCOPE=Session ID=MyConnection
PROGID="ADODB.Connection">
</OBJECT>
```

```
<OBJECT RUNAT=Server SCOPE=Session ID=MyADRot PROGID="MSWC.ADRotator">
</OBJECT>
```

```
-----File.asp-----
```

```
<%
For Each Item In Session.StaticObjects
Response.Write Item & "<BR>"
Next
%>
```

Output:

```
MyInfo
MyConnection
MyADRot
```