

```
Response.Write "You chose 1"  
Case "2":  
    Response.Write "You chose 2"  
Case "3":  
    Response.Write "You chose 3"  
Case "4":  
    Response.Write "You chose 4"  
End Select  
%>
```

### Check Your Progress

1. Define Script.
2. What is Binary Write method and Expires Property?

---

## 1.8 LET US SUM UP

ASP was "born" in November 1996 when Microsoft announced its design of an Active Platform. The Active Platform reflects Microsoft's ideas about how a desktop computer and a server computer should communicate. It consists of two parts: the Active Desktop and the Active Server. The **Active Desktop** refers to the client side, or the user's side, where HTML files are displayed on a web browser. The **Active Server** refers to the server-side component. In a *client-server model*, two computers work together to perform a task. A client computer requests some needed information from a server computer. The server returns this information and the client acts on it. To execute ASP pages on the computer, we need to be running a Web server. If we don't have a Web server installed that can handle ASP pages, when we request an ASP page through a browser, a dialog is received whether we want to save ASP file to disk.

Script means a small relatively limited interpreted language. When the server invokes the ASP scripting engine, it parses the file and inserts any include files. ASP process helps us perform several operations that are difficult or impossible with straight HTML. If the variable is declared outside a procedure it can be changed by any script in the ASP file.

---

## 1.9 KEYWORDS

**Active Server Pages:** ASP was "born" in November 1996 when Microsoft announced its design of an Active Platform.

**Client-Server Model:** In a *client-server model*, two computers work together to perform a task.

**Script:** It means a small relatively limited interpreted language.

---

## 1.10 QUESTIONS FOR DISCUSSION

1. What are Active Server Pages?
2. What is a Client Server Model?
3. What is client-side scripting technologies?

4. How do you run ASP pages?
5. How do you set up a personal web server?
6. What does Response/Write do?
7. What is that your ASP Script is returning to the Browser?
8. What are the ASP processes?
9. Write a VBScript statement using either VarType or TypeName that is equivalent to the IsDate function.
10. Write the statement that calculates the length of the hypotenuse of a right triangle. Assume the lengths of the other two sides are stored in sngSide1 and sngSide2.
11. Write the explicit declaration for variables that will store a user's name, age, e-mail, and birth date. Choose names (variables) according to the guidelines discussed.

### Check Your Progress: Modal Answers

1. Script means a small relatively limited interpreted language. We need to know what interpreted code is. Computers don't understand code as we write it. Instead, other programs translate the code we write into machine instructions.
2. Binary Write method is used to write non-textual data to the browser. We can use this method to send image, audio, or other binary to the browser.

Expires Property sets the time interval, in minutes, before the Page expires. Until the specified period elapses the browser may redisplay the Page from cache after the specified period. The browser must return to the server to.

## 1.11 SUGGESTED READINGS

Jeffrey C. Jackson, *Web Technologies*, Prentice Hall, 2007

Godbole, *Web Technologies*, Tata McGraw-Hill, 2003

Ramesh Bangia, *Internet and Web Design*, firewall media

Gopalan, Gopalan/akilandswari, *Web Technology: A Developer S Perspective*, PHI Learning Pvt. Ltd.

Ramesh Bangia, *Web Technologies*, firewall media

---

## LESSON

# 2

## UNDERSTANDING OBJECTS

---

### CONTENTS

- 2.0 Aims and Objectives
- 2.1 Introduction
- 2.2 What is an Object?
- 2.3 Building Blocks of Objects (Properties, Methods, Instances of Objects)
  - 2.3.1 Instances and Classes
  - 2.3.2 Properties
  - 2.3.3 Methods
  - 2.3.4 Events
  - 2.3.5 Synchronous vs. Asynchronous
  - 2.3.6 Encapsulation
- 2.4 Built-in ASP Objects
  - 2.4.1 Application Object
  - 2.4.2 ASP Request Object
  - 2.4.3 ASP Response Object
  - 2.4.4 ASP Session Object
  - 2.4.5 ASP Server Object
  - 2.4.6 ASP Error Object (ASP 3.0)
- 2.5 The Global.asa File
  - 2.5.1 Firing Sequence of The Global.asa File
  - 2.5.2 Changing the Global.asa Contents
  - 2.5.3 Events in Global.asa
  - 2.5.4 <object> Declarations
  - 2.5.5 TypeLibrary Declarations
  - 2.5.6 Restrictions
  - 2.5.7 How to use the Subroutines
- 2.6 Let us Sum up
- 2.7 Keywords
- 2.8 Questions for Discussion
- 2.9 Suggestion Readings

---

## 2.0 AIMS AND OBJECTIVES

---

After studying this lesson, you will be able to:

- Understand what is an object
- An introduction to properties, methods, and events
- How can we change the characteristics of an object?
- Discuss ASP application object
- Discuss request object
- Discuss response object
- Discuss session object
- Discuss ASP server object
- Discuss ASP error object
- Understand global.asa file

---

## 2.1 INTRODUCTION

---

In our study of objects, we will find that an object is a software representation of a real-world item, or object. Software objects feature properties (that describe the object), methods (that allow the object to do things for you), and events (code that is executed automatically when a certain situation - an event - occurs).

Once we have looked at what an object is, we'll then be able to look at how objects are useful to us in developing ASP applications. Developing web applications requires us to deal with both the client-side and server-side programming, and therefore we'll take a look at how objects can be used on both sides of the wire.

---

## 2.2 WHAT IS AN OBJECT?

---

In the real world, we already know what objects are. They're the things we look at, pick up, and use every day - things like our chairs, our telephones, and our computers. All these are solid, tangible entities.

However, if we want to describe a telephone to somebody in abstract terms, we can do this by talking about it in terms of its essential characteristics - what properties it has, what it can do, and how we can interact with it. All telephones have these characteristics, and we can use them to establish exactly how one telephone differs from the next.

So, for our telephone's physical properties, we could say that it has a microphone, a speaker, and a dialing pad. We could also say that it lets us do things, such as call someone and talk to them. Our telephone will also tell us when certain events are happening: for example, if a friend is trying to call you, your telephone will ring to let you know. This ringing will prompt you to take some action, like picking up the handset and talking to your friend. As an abstract object, our telephone has:

- Certain properties that define and describe it

- A set of things or methods that it lets us do
- The ability to prompt action when events occur

We can use these three attributes to describe physical objects and abstract concepts. In a few minutes we will describe how these real-world ideas are replicated in software objects, but for now let's go a little deeper into our real-world telephone. By learning about what objects are, we can then look at how to use them in a way known as object-based programming. In the object-based way of programming, the application is broken down into a set of objects. In doing this, you can build the application in a two stage process. Firstly, you create the objects you will need in your application and then you set up the relationships and interactions between objects. Later in this chapter, we will see how the objects of Active Server Pages relate and interact with each other and allow us to build our applications.

*Example:*



Here is our telephone. To look at this as an object, let's put down some information about it. We will be classifying the information into three categories:

- Things that describe the telephone
- Things that we can do with the phone
- Things that the telephone tells us and that we can react to

Let's look at each of these aspects in turn:

Describe the telephone	The telephone is gray The telephone is made of plastic The handset weighs 6.5 ounces The telephone has 12 keys The telephone number is (714) 555-1523 The telephone is connected to the exchange
What can we do with it?	We can place an outgoing call We can answer an incoming call We can hang up the current call We can enter our calling card number We can disconnect it from the exchange
What can it tell us that we can react to?	Someone is trying to call us The person we are calling is busy Another person is calling while we are talking

*How It Works*

The three categories that we have created in the left-hand column can be applied to any object. In fact, the best way to describe an object is to break down its characteristics into these three categories, and

put information about your object into these categories. Any information that you have about a particular object can be included in one of these categories.

If you have another telephone that features all these characteristics, except that its color is blue, then we can describe your telephone by changing that one part of the description above. Moreover, this technique works for any type of object, both real world and software.

---

## 2.3 BUILDING BLOCKS OF OBJECTS (PROPERTIES, METHODS, INSTANCES OF OBJECTS)

---

### *Object Terms*

So far, we have used verbose English terms to describe our three categories. In the world of objects, we need terms that concisely describe each of these three categories. These terms are properties, methods and events. In addition to these terms, we need to look at the term instance as it relates to objects. In this section, we'll look more carefully at what each of these means in abstract terms.

### 2.3.1 Instances and Classes

When we are talking about a unique object, we can use the term instance to say that we are talking about a particular telephone object - your telephone for example - that has a specific set of properties and values. When we want to talk about another telephone, we use a different instance of the telephone object. In this way, both you and I can have instances of the telephone object. For example, my telephone (my instance of a telephone object) is gray and comes with special answer-phone facilities, your telephone (your instance of a telephone object) may be red, blue, green etc. These instances represent completely different physical objects. However, since they are both instances of the same object description or template, they share the same types of characteristics such as methods, properties (although the values can be different), and events. When a specific instance of an object is created from the template for the object, the object is said to have been instantiated. What actually happens is that a copy is made of all of the properties and events from the object description, but the methods (frequently a big chunk of code) remain in the original place and this section of code is used by all of the different instantiations of that one object.

So we've mentioned object descriptions or templates, but it's time to give them their proper name in ASP; classes. We mentioned that each object can have different instances. For instance, my telephone is small and white and has 12 buttons. Your telephone will probably be different to that, but they're both recognizable as telephones and they both provide the same function. They both conform to a set of rules for what a telephone does - they connect to the local telephone line, they both have a numeric keypad and somewhere to speak into. A class in ASP is like a set of design rules that an object must conform to. It would be no good if my telephone didn't have a handset, or a numeric keypad, even if it did plug into the telephone socket on the wall. In a class there should be a minimum set of functions that your object must be able to perform.



### 2.3.2 Properties

When talking about those characteristics that describe an object, we are talking about the properties of the object. Each property of the object describes a particular aspect of the object. The property is actually described as a name/value pair. This means that for every named property, there is a single

unique value that describes that property for this instance of the object. If we go back to our telephone example, we can create a table that lists each of the property names and the value of each property.

Property Name	Property Value
Color	Grey
Material	Plastic
Weight	6.5 ounces
NumberOfKeys	12
TelephoneNumber	(714) 555-1523
Connected	Yes

We now have a set of properties that describe this instance. The properties of an object are used to represent a set of values associated with the object. A new instance of the object may have different property values, but it has the same property names.

		
Color	Grey	Blue
Material	Plastic	Thermoplastic
Weight	6.5 ounces	22 ounces
NumberOfKeys	12	12
TelephoneNumber	(714) 555-123	(615) 555-8329
Connected	Yes	Yes

Even with different property values, these two telephones are instances of the same object template. Since we know that all telephone objects have a 'Color' property, we can determine the color of each of the phones by examining its 'Color' property value. We can use properties in two ways. We can read from them or we can also write to them. So if we wanted, we could have changed the cover of our telephone to a different color, if we required.

Now that we have a way of describing the telephone object, let's take a look at what we can do with it.

### 2.3.3 Methods

Another characteristic of objects is that they can perform functions for us. For example the Place Call method would perform several functions for you. It will connect you to the local exchange, the exchange will route your call, and then when it reaches the destination, and it will make the destination phone ring. These built-in actions occur whenever you pick up the handset and dial a number. This is a capability that has been built in to the machine.

However not all objects have functions like this. A chair object allows you to sit in it, so you could say that it is functioning to support your body. Objects that perform tasks that are more 'functional' are said to have methods. The tasks that an object can perform are called methods.

A method is defined as an action that an object can take. The code in a method is executed when the method is called. This calling is done by a command you write in the script of your ASP page. Once

we have created an instance of an object, we can tell it to perform a certain task calling one of its methods.

Let's illustrate this using the telephone example. Our telephone object can carry out five methods. Each of these methods will cause the telephone object to perform an action. Here is a list of functions that the methods of the telephone object can perform:

Method Name	Description
PlaceCall	Place an outgoing call
Answer	Answer an incoming call
HangUp	Hang up the current call
SendCardNumber	Enter or send our calling card number
Disconnect	Disconnect the phone from the exchange

These methods are used when we want our telephone object to perform a certain function; all we need to do is tell it to execute the corresponding method.

Methods are actually blocks of code that are written by the designer of the object (Microsoft, for example). The reason methods exist is because lots of programmers want to do the same job, so it is worth it for the gurus at Microsoft to write the code to do that job, test it, optimize it, and get it in great shape, and then bundle it up in the object. We, as programmers, can then use that code pre-made. Instead of re-inventing the wheel we spend our time on the unique parts of our project.

#### *Parameters of Methods*

You may have noticed that some of the methods can be executed directly, while others look like they will need additional information. To contrast these two ideas, consider the following examples:

- Suppose that our telephone receives an incoming call (in the next section, we'll see how we can tell that this is happening). All we need to do to answer the call is to use the 'Answer' method of our telephone object.
- Suppose that we want to place a call. Simply calling the Place Call method isn't enough in this case: we need to supply more information (for example, the telephone number!) in order to complete the action.

Let's look more closely at the second of these examples. The telephone object has a Telephone Number property, and this is used to identify our telephone's own telephone number (i.e. the number that other people use to call us). So, the Telephone Number property of our phone isn't going to help us to make outgoing telephone calls.

So, how do we tell the phone which number we want to call? It's possible, I guess, for the telephone object to have another property, called Outgoing Telephone Number, that would identify the desired number; but that would be too cumbersome, because every time we wanted to make a call we would have to:

- Set the Outgoing Telephone Number property value to the desired phone number
- Execute the 'Call' method of the telephone object to place the call



As you know, telephones just don't work that way. It would be much more elegant (and intuitive) to have some way of passing the outgoing phone number to the 'Call' method, so that we can place an outgoing call in a single step. This is done by passing a parameter to the 'Call' method. With this in place, we can place an outgoing call by simply executing the 'Call' method and telling it which number we want to call, like this:

Execute the 'Call' method of the telephone object, passing the outgoing telephone number as a parameter. Parameters here are just the same as the arguments (parameters) we passed to functions and subroutines.

If we look again at the methods of the telephone object, we can identify those methods that require parameters, and what the values of those parameters mean to the object:

Method Name	Parameters
PlaceCall	Outgoing telephone number
Answer	No Parameters
HangUp	No Parameters
SendCardNumber	Calling card number, PIN
Disconnect	No Parameters

You can see that a method can have none, one, or more than one parameter. The Send Card Number method actually requires two parameters. You are required to pass in both the calling card number and the Personal Identification Number (PIN) in order for the method to execute properly. Information passed as parameters of the method for execution by the method, will only be executed if all parameters have been supplied.

### *Return Values*

In addition to passing parameters to a method, the method can also return information to us. The value returned by a method is (rather conveniently) called a return value. If a method has a return value, then it will pass information back to us. This information could have a number of purposes. For example, the return value might be an indication of whether or not the method completed successfully. Alternatively, the method could also pass back the results of some processing that it did for us. It can even return another object as a result.

As the user of an object, we can decide whether we want to do anything with the return value. If the information is pertinent to what we are doing, then we can capture the return value and do something with it later. If we do not care what the return value is, we can just ignore it and continue with our work.

Just as the methods of the telephone object can have parameters, we can identify those methods that pass return values (and these can be passed as parameters to other methods), and what those values mean.

Method Name	Return Value
PlaceCall	True (if call completed successfully) False (if call failed)
Answer	No Return Value
HangUp	True (if telephone was hung up successfully) False (if not)
SendCardNumber	True (if card was accepted) False (if card was not accepted)
Disconnect	No Return Value

### 2.3.4 Events

We have now looked at two of the three characteristics of an object. The properties and methods of an object are ways that the user of the object can communicate with the object. Now, what if the object needs to communicate with the program that created it?

As an example, consider what happens when our telephone receives an incoming call. The fact is that it needs some way of telling us to answer the call. How will the telephone communicate this information to us?

Again, it's possible for the telephone object to have a property (called Incoming Call, perhaps) that was set to 'True' whenever an incoming call was present. However, there are two disadvantages to this. First, it would require the user of the telephone object to check this property on a regular basis. Second, the user would require a great deal of knowledge of the inner workings of the object, which isn't ideal.

What is needed is a way for the object to tell the user that something has happened. The mechanism for this is called an event. An object generates an event whenever something of interest happens. In our telephone example, when the telephone receives an incoming call it tells us so in the form of an event - we'll call it the Incoming Call event. (On most telephones, this particular event takes the form of the telephone ringing.)

The telephone object would generate an Incoming Call event every time an incoming call is received. In a physical phone, the ringing sound is the phone notifying you of the Incoming Call event. When the user receives this event, it can execute the 'Answer' method (pick up the handset), and begin the call. This frees the user from having to check regularly for incoming calls: the event is designed to notify the user just at the appropriate moment.

Just like methods, events can have parameters. These parameters can hold specific information about the event. For example, if our telephone supports CallerID - a feature that reveals the identity of the incoming caller - then the Incoming Call event could include a parameter that contains the telephone number of the incoming caller.

Here is a list of the events that our telephone object will generate, along with their associated parameters:

Event Name	Parameters
IncomingCall	Incoming CallerID information
LineBusy	No Parameters

There are a couple of useful pieces of terminology that are often used in this context. When an object generates an event, the object can be said to fire the event. When the object has fired the event, we say that the user must handle the event.

### 2.3.5 Synchronous vs. Asynchronous

One of the advantages of working with objects and events is that it awakens us to the concept of asynchronous programming. First off, let's look at the definitions of synchronous and asynchronous.

These terms refer to how two separate actions are related to each other. If the first action must be completed before the second one begins, then these two actions are said to be synchronous. If the second action can begin at any time, no matter what the status of the first action, then these two actions are said to be asynchronous.

We've already discussed what it would be like if we our objects didn't support events. For example, to detect an incoming call, you would need to constantly check the value of some property to see whether an incoming call was waiting. While you're performing these frequent, regular checks, you would be unable to perform other tasks dependent on the outcome of that task. This is an example of synchronous activity. For example in the real world you might be waiting for a telephone call from a friend to let you know what time you should meet them. You can't go out until you've arranged a specific time. It's the same in programming. Your program could be waiting on a Wait For an Incoming Call method in a While ... Wend loop, and it could be stuck there until the call was detected, refusing to return control to the main body of your program.

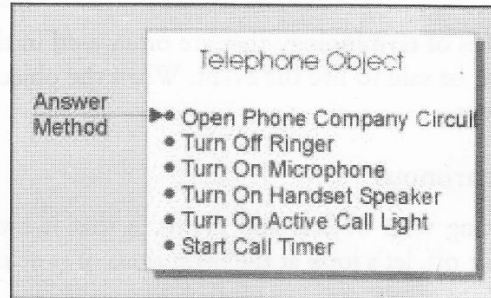
With events, we can have asynchronous activity. By having an event handler that is called when the object fires the 'Incoming Call' event, we can perform other tasks (for example, making an outgoing phone call) without having to devote any effort to monitoring the incoming call status. Our event handler code will be dormant until such a time as it detects the 'Incoming Call' event, and then sets about dealing with the incoming call.

This is not to say that all synchronous is bad and all asynchronous is good. You will see many instances in the real world where it makes sense to use a synchronous activity to perform a certain type of processing. Likewise, we will also see instances where an asynchronous activity is not an optimal way of dealing with an event.

### 2.3.6 Encapsulation

One great thing about objects is that you don't have to understand what's going on underneath the shell, to know how to operate them. With our telephone we don't need to know how our voice is projected from the phone, down the wires to the nearest exchange, and how from there it gets to our intended destination. This is all hidden from us. It's the same in ASP - you don't need to know how the object was programmed, for example, in C++ or VB (objects can be created in many languages), to be able to interact with it. The concept of a user of an object not being concerned with the inner workings of the object is known as encapsulation. For example, when you use a telephone to answer

an incoming call, all you need to do is pick up the handset. You don't need to know how the transistors are connected to each other inside the telephone. This is the equivalent of executing the Answer method. You do not need to know what's going on underneath - that's all encapsulated within the Answer method. This is an example of encapsulation.



One advantage of encapsulating the workings of an object within a method is that the implementation of the method can be changed without having to adjust the client. For example, suppose the phone company decides to change the way that an incoming call is answered. Without encapsulation, all of the users of the telephone object would have to be adjusted to support this new way of answering the phone. Instead, by encapsulating these new steps within the Answer method, the actions of the client never need to be changed: with either system, all the client needs to do is execute the 'Answer' method. Not only does encapsulation make the telephone user's life easier; it allows the developer of the telephone object to change the implementation at any time.

---

## 2.4 BUILT-IN ASP OBJECTS

---

Now that we have a basic understanding of what an object is, we can move on to looking at how programming concepts have changed from traditional methods by using objects. When working with objects in software development, we will create objects that have properties, events and methods. We can use these three attributes to describe physical objects and abstract concepts. Either way, the programmatic object will allow us to interact with it through its properties, events and methods.

### 2.4.1 Application Object

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

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

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

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

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

### Collections

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

### Methods

Method	Description
Contents.Remove	Deletes an item from the Contents collection
Contents.RemoveAll()	Deletes all items from the Contents collection
Lock	Prevents other users from modifying the variables in the Application object
Unlock	Enables other users to modify the variables in the Application object (after it has been locked using the Lock method)

### Events

Event	Description
Application_OnEnd	Occurs when all user sessions are over, and the application ends
Application_OnStart	Occurs before the first new session is created (when the Application object is first referenced)

### Store and Retrieve Application Variables

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

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

```
<script language="vbscript" runat="server">
Sub Application_OnStart
application("vartime")=""
application("users")=1
End Sub
</script>
```

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

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

There are

```
<%
Response.Write(Application("users"))
%>
```

active connections.

***Loop Through the Contents Collection***

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

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

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

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

***Loop Through the StaticObjects Collection***

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

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

***Lock and Unlock***

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

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

***Pitfalls of Application Variables***

Because only one instance of the Application object exists for the entire Web application, application variables can be used more liberally than session variables. However, the Application object does take

up memory on the Web server, so only items that need to be stored in application scope should be entered into the Application object. Two common pitfalls should be avoided when working with application variables:

- *Pitfall 1:* Do not put objects into the Application object unless vitally needed.
- *Pitfall 2:* Only create application variables that are necessary. Why create unneeded application variables when they'll only waste your Web server's memory?

A common pitfall among developers is wanting to place objects in the Application. One object that is particularly alluring to put into the Application is the ADO Connection object, which is used to connect to a database. We'll discuss this object in detail during Week 3. It may seem like a good idea to create a single database connection object and have all users communicate to a database through that object. However, as with most other objects, it is always best to wait to create the object until you need it. In fact, the ADO Connection object will degrade your server's performance if put into the Application object.

Like the Session object, the Application object is easy to use, and the temptation to create a plethora of application variables is high indeed. Many developers use the Application object to store many Web site-wide constants - for example, a navigational footer common to all Web pages, or perhaps the Webmaster's email address. Although it's better to place these items in the Application object than in the Session object, they belong best in a static text file on the Web server. This text file can then be included into any ASP page that needs to display the navigation footer or the Webmaster's email address. We will discuss how to include files on Day 13.

Sometimes, however, the use of application variables is preferred to include files. If the data you need to store changes often, such as the last post to a message board, then the information should be stored in the Application object. Include files should only be used if the data is static and not susceptible to frequent change.

Although you can afford to be less prudent when creating application variables than you can be when creating session variables, you should still strive to use the minimum needed amount of such variables. Because the Application object persists in the Web server's memory, the fewer the application variables stored within it, the less drain on the Web server's performance. Therefore, the Application should remain free of objects and contain only needed application variables.

## 2.4.2 ASP Request Object

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

### *Collections*

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

**Properties**

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

**Methods**

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

Send query information when a user clicks on a link using QueryString

```
<html>
<body>
<a href="demo_simplequerystring.asp?color=green">Example</a>
<%
Response.Write (Request.QueryString)
%>
</body>
</html>
```

**O/P:**

**Example:** color=green

How to use information from forms

```
<html>
<body>
<form action="demo_simpleform.asp" method="post">
Your name: <input type="text" name="fname" size="20" />
<input type="submit" value="Submit" />
</form>
<%
dim fname
fname=Request.Form("fname")
If fname<>" " Then
    Response.Write("Hello " & fname & "!<br />")
    Response.Write("How are you today?")
End If
%>
</body>
</html>
```



**O/P:**

Your Name:

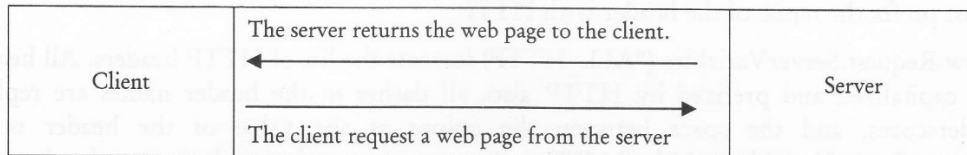
**Results:**

Hello Vicki!

How are you today?

*Accessing the HTTP Headers (Useful HTTP Headers, Reading the HTTP Headers with Request, Server Variables)*

In a client-server model, when client is a web browser communicating with the server, requesting a web page.



When the client requests a web page from the server it not only sends the URL of the web page requested but also some additional information. This extra information consists of useful facts about the client for Eg: what browser is being used, what operating system the client is running, what URL the user just came from. Each piece of additional information is referred as a request header.

A request header is a single line of text that browser sends to the web server when requesting to view any web page.

When the server sends back the requested web page to the client, it also sends a set of headers, known as response headers. Response headers are additional bits of information about the web page being sent to the client. Both the request headers and the response headers are referred to, more generally, as HTTP headers.

A HTTP header is a single piece of information, sent either from the client to the server, when requesting a page, request.

*Standard HTTP Headers*

HTTP Header name	Description
HTTP_ACCEPT	A list of MIME type the client will Accept
HTTP_ACCEPT_LANGAGUE	what type of language the browser expects
HTTP_CONNECTION	the type of connection established between the client and web server
HTTP_HOST	the host name of the computer
HTTP_USER_AGENT	the browser type and version and operating system, information system of the client
HTTP_REFERER	the full URL of the web page containing the hyperlink use to reach the currently executing ASP page
HTTP_COOKIE	the cookies sent from the browser

*Reading HTTP Headers with Request.ServerVariables*

Using ASP, one can read the headers that the browser sends to the web server using the request object. Specifically, the use of the server variables is collections of the request objects. To display HTTP headers simply issue the following statement:-

```
<%=Request.ServerVariables("all_raw")%>
```

This displays the exact list of header sent by the browser to the web server to display a formulated list of headers, use the following commands:

```
<%=request.server variable ("HTTP_headername") %>
```

*Note:*

1. Must prefix the name of the header with HTTP
2. How Request.ServerVariables ("ALL\_HTTP") formats the list of HTTP headers. All header names are capitalized and prefixed by HTTP\_ also, all dashes in the header names are replaced with underscores, and the space between the colons at the value of the header is removed. Request.ServerVariables ("ALL\_RAW") performs no formatting to the request headers.
3. Reference header is present if the page has reached through a hyperlink on a different web page.

*Accessing Environmental Variables*

It involves Useful Environment Variables, Reading the Environment Variables, Using Request, ServerVariables.

The HTTP headers are useful for obtaining information about the current visitor but tell nothing about the web server or the asp page that is being requested by the client.

Environmental variables are bits of information that the web server makes available to any program that requests them. Environmental variables contain information such as the name of the web server, the URL of the currently processing ASP page, or the name of the name of the web server software being used.

*Commonly used Environment Variables*

Environment variables	Description
URL	the URL of the ASP page from after http\\www.your.webserver.com/up to the query string.
Path_info	the same as the URL environment variable
Path_translated	the full, physical path of the currently executing ASP page.
Appl_physical_path	the physical address of the web's root directory
Query_string	the query string (equivalent to request every string)
Server_name	the web server's computer name
Server_software	the name of the web software

*Reading Environmental Variables using Request.ServerVariables*

The environmental variables are accessed much like the HTTP headers. The Request.ServerVariables collection is used in the following format:

### Request.ServerVariables (Environment Variable Name)

List all server variables. ASP contains ASP code that lists all the items in the Request.ServerVariables collection. The server variable collection contains both environment variables and HTTP headers < both will be displayed when listing the contents of the collection.

Many environment variables do not contain a value. For Eg, the environment variables prefixed with CERT all contain empty strings as their values. This is because these variables are used only when the client and server use certificates. When a browser and web server communicate over a secure channel, certificates are used to ensure the identity of the client to the server.

All environment variables do not have values all the times. When a web server is not using SSL, the certificate environment variables are empty strings, much like the referrer HTTP header contains an empty string when the page is not visited via a hyperlink. There are some that are never empty such as URL, PATH\_INFO, and PATH\_TRANSLATED.

If you want to display the URL of the currently running ASP page, you would use the URL environment variable. The URL environment variable does not show the http: web server name, simply the full virtual path and filename. Another environment variable, Server\_Name, contains the actual host name of the web server. The SERVER\_NAME environment variable

### 2.4.3 ASP Response Object

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

ASP CODE	OUTPUT
<pre>&lt;html&gt; &lt;body&gt; &lt;% response, write ("Hello World!") %&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<p>Hello World!</p>

Its collections, properties, and methods are described below:

#### Collections

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

#### Properties

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

**Methods**

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

**Examples:****Format text with HTML tags in ASP**

This example demonstrates how to combine text and HTML tags with ASP.

ASP CODE	OUTPUT
<pre>&lt;html&gt; &lt;body&gt; &lt;% response.write("&lt;h2&gt;You can use HTML tags to format the text!&lt;/h2&gt;") %&gt;  &lt;% response.write("&lt;p style='color:#0000ff'&gt;This text is styled with the style attribute!&lt;/p&gt;") %&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<p>You can use HTML tags to format the text!</p> <p>This text is styled with the style attribute!</p>

**Redirect the user to a different URL**

This example demonstrates how to redirect the user to a different URL.

ASP CODE	OUTPUT
<pre>&lt;% if Request.Form("select")&lt;&gt;" then Response.Redirect(Request.Form("select")) end if %&gt;  &lt;html&gt; &lt;body&gt;  &lt;form action="demo_redirect.asp" method="post"&gt; &lt;input type="radio" name="select" value="demo_server.asp"&gt; Server Example&lt;br&gt;</pre>	<p>Top of Form</p> <p><input type="radio"/> Server Example</p> <p><input type="radio"/> Text Example</p> <p><input type="submit" value="Go!"/></p> <p>Bottom of Form</p> <p>Top of Form</p> <p>Bottom of Form</p>

Contd...

```
<input type="radio" name="select"
value="demo_text.asp">
Text Example<br><br>
<input type="submit" value="Go!">
</form>
</body>
</html>
```

**Show a Random Link**

This example demonstrates a link, each time you load the page, it will display one of two links: ICFAI.com! OR Refsnesdata.no! There is a 50% chance for each of them.

ASP CODE	OUTPUT
<pre>&lt;html&gt; &lt;body&gt;  &lt;% randomize() r=rnd() if r&gt;0.5 then response.write("&lt;a href='http://www.ICFAI.com'&gt;ICFAI.com!&lt;/a&gt;") else response.write("&lt;a href='http://www.refsnesdata.no'&gt;Refsnesdata.no!&lt;/a&gt;") end if %&gt; &lt;p&gt; This example demonstrates a link, each time you load the page, it will display one of two links: ICFAI.com! OR Refsnesdata.no! There is a 50% chance for each of them. &lt;/p&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>Refsnesdata.no! This example demonstrates a link, each time you load the page, it will display one of two links: ICFAI.com! OR Refsnesdata.no! There is a 50% chance for each of them. Bottom of Form Top of Form Bottom of Form</pre>

**Controlling the Buffer**

This example demonstrates how you can control the buffer.

ASP CODE	OUTPUT
<pre>&lt;% Response.Buffer=true %&gt; &lt;html&gt; &lt;body&gt; &lt;p&gt; This text will be sent to your browser when my response buffer is flushed. &lt;/p&gt; &lt;% Response.Flush %&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>This text will be sent to your browser when my response buffer is flushed. Bottom of Form Top of Form Bottom of Form</pre>

***Clear the Buffer***

This example demonstrates how you can clear the buffer.

ASP CODE	OUTPUT
<pre>&lt;% Response.Buffer=true %&gt; &lt;html&gt; &lt;body&gt; &lt;p&gt;This is some text I want to send to the user.&lt;/p&gt; &lt;p&gt;No, I changed my mind. I want to clear the text.&lt;/p&gt; &lt;% Response.Clear %&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<p>Bottom of Form Top of Form Bottom of Form</p>

***End a Script in the Middle of Processing and Return the Result***

This example demonstrates how to end a script in the middle of processing.

ASP CODE	OUTPUT
<pre>&lt;html&gt; &lt;body&gt; &lt;p&gt;I am writing some text. This text will never be&lt;br&gt; &lt;% Response.End %&gt; finished! It's too late to write more!&lt;/p&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<p>I am writing some text. This text will never be Bottom of Form Top of Form Bottom of Form</p>

***Set how many minutes a Page will be cached in a Browser before it expires***

This example demonstrates how to specify how many minutes a page will be cached in a browser before it expires.

ASP CODE	OUTPUT
<pre>&lt;%Response.Expires=-1%&gt; &lt;html&gt; &lt;body&gt; &lt;p&gt;This page will be refreshed with each access!&lt;/p&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<p>This page will be refreshed with each access!</p>

***Set a Date/time when a Page cached in a Browser will expire***

This example demonstrates how to specify a date/time a page cached in a browser will expire.

ASP CODE	OUTPUT
<pre>&lt;% Response.ExpiresAbsolute=#May 05,2001 05:30:30# %&gt; &lt;html&gt; &lt;body&gt; &lt;p&gt;This page will expire on May 05, 2001 05:30:30!&lt;/p&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>This page will expire on May 05, 2001 05:30:30!</pre>

### *Check if the user is still connected to the Server*

This example demonstrates how to check if a user is disconnected from the server.

ASP CODE	OUTPUT
<pre>&lt;html&gt; &lt;body&gt; &lt;% If Response.IsClientConnected=true then Response. Write ("The user is still connected!") else Response. Write ("The user is not connected!") end if %&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>The user is still connected!</pre>

### *Set the Type of Content*

This example demonstrates how to specify the type of content.

ASP CODE	OUTPUT
<pre>&lt;% Response.ContentType="text/html" %&gt; &lt;html&gt; &lt;body&gt; &lt;p&gt;This is some text&lt;/p&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>This is some text</pre>

### Set the Name of the Character Set

This example demonstrates how to specify the name of the character set.

ASP CODE	OUTPUT
<pre>&lt;% Response.Charset="ISO8859-1" %&gt; &lt;html&gt; &lt;body&gt; &lt;p&gt;This is some text&lt;/p&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<p>This is some text</p>

### 2.4.4 ASP Session Object

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

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

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

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

#### When does a Session Start?

A session starts when:

- A new user requests an ASP file, and the Global.asa file includes a Session\_OnStart procedure
- A value is stored in a Session variable
- A user requests an ASP file, and the Global.asa file uses the <object> tag to instantiate an object with session scope.

#### When does a Session End?

A session ends if a user has not requested or refreshed a page in the application for a specified period. By default, this is 20 minutes.



If you want to set a timeout interval that is shorter or longer than the default, you can set the **Timeout** property.

The example below sets a timeout interval of 5 minutes:

```
<%
Session.Timeout=5
%>
```

To end a session immediately, you may use the **Abandon** method:

```
<%
Session.Abandon
%>
```

**Note:** The main problem with sessions is WHEN they should end. We do not know if the user's last request was the final one or not. So we do not know how long we should keep the session "alive". Waiting too long for an idle session uses up resources on the server, but if the session is deleted too soon the user has to start all over again because the server has deleted all the information. Finding the right timeout interval can be difficult!

**Tip:** If you are using session variables, store **SMALL** amounts of data in them.

#### *Store and Retrieve Session Variables*

The most important thing about the Session object is that you can store variables in it.

The example below will set the Session variable *username* to "Donald Duck" and the Session variable *age* to "50":

```
<%
Session("username")="Donald Duck"
Session("age")=50
%>
```

When the value is stored in a session variable it can be reached from ANY page in the ASP application:

```
Welcome <%Response.Write(Session("username"))%>
```

The line above returns: "Welcome Donald Duck".

You can also store user preferences in the Session object, and then access that preference to choose what page to return to the user.

The example below specifies a text-only version of the page if the user has a low screen resolution:

```
<%If Session("screenres")="low" Then%>
  This is the text version of the page
<%Else%>
  This is the multimedia version of the page
<%End If%>
```

#### *Remove Session Variables*

The Contents collection contains all session variables.

It is possible to remove a session variable with the Remove method.

The example below removes the session variable "sale" if the value of the session variable "age" is lower than 18:

```
<%
If Session.Contents("age")<18 then
    Session.Contents.Remove("sale")
End If
%>
```

To remove all variables in a session, use the RemoveAll method:

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

### *Loop Through the Contents Collection*

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

```
<%
Session("username")="Donald Duck"
Session("age")=50
dim i
For Each i in Session.Contents
    Response.Write(i & "<br />")
Next
%>
```

### **Result:**

```
Username
Age
```

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

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

**Result:**

Session variables: 2  
 Donald Duck  
 50

**Loop Through the StaticObjects Collection**

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

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

**Pitfalls of Session Variables**

- Session variables and cookies are synonymous. So if a user has set his browser not to accept any cookies, your Session variables won't work for that particular web surfer.
- An instance of each session variable is created when a user visits the page, and these variables persist for 20 minutes AFTER the user leaves the page! (*Actually, these variables persist until they "timeout". This timeout length is set by the web server administrator. I have seen sites that the variables will collapse in as little as 3 minutes, and others that persist for 10, and still others that persist for the default 20 minutes.*) So, if you put any large objects in the Session (such as ADO recordsets, connections, etc.), you are asking for serious trouble! As the number of visitors increase, your server will experience dramatic performance woes by placing large objects in the Session.
- Since Session variables can be created on the fly, used whenever, and do not require the developer to dispose of them explicitly, the overuse of Session variables can lead to very unreadable and unmaintainable code.
- Session variables take you one step closer to VB programming in the sense that you can grab one without initializing the variable, use it whenever you want to, and not have to worry about releasing it when you've finished using it. And **WHO** wants to go there? Not me.

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

**Collections**

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

**Properties**

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

**Methods**

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

**Events**

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

**Examples:****Set and return the LCID**

This example demonstrates the "LCID" property. This 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.

ASP CODE	OUTPUT
<pre>&lt;html&gt; &lt;body&gt; &lt;% response.write("&lt;p&gt;") response.write("The default LCID for this page is: " &amp; Session.LCID &amp; "&lt;br /&gt;") response.write("The Date format for the above LCID is: " &amp; date() &amp; "&lt;br /&gt;") response.write("The Currency format for the above LCID is: " &amp; Format Currency(350)) response.write("&lt;/p&gt;") Session.LCID=1036 response.write("&lt;p&gt;") response.write("The LCID is now changed to: " &amp; Session.LCID &amp; "&lt;br /&gt;") response.write("The Date format for the above LCID is: " &amp; date() &amp; "&lt;br /&gt;")</pre>	<pre>The default LCID for this page is: 2048 The Date format for the above LCID is: 8/3/2004 The Currency format for the above LCID is: \$350.00 The LCID is now changed to: 1036 The Date format for the above LCID is: 03/08/2004 The Currency format for the above LCID is: 350,00 F The LCID is now changed to: 3079 The Date format for the above LCID is: 03.08.2004</pre>

Contd...

<pre> response.write("The Currency format for the above LCID is: " &amp; Format Currency(350)) response.write("&lt;/p&gt;") Session.LCID = 3079 response.write("&lt;p&gt;") response.write("The LCID is now changed to: " &amp; Session.LCID &amp; "&lt;br /&gt;") response.write("The Date format for the above LCID is: " &amp; date() &amp; "&lt;br /&gt;") response.write("The Currency format for the above LCID is: " &amp; FormatCurrency(350)) response.write("&lt;/p&gt;") Session.LCID = 2057 response.write("&lt;p&gt;") response.write("The LCID is now changed to: " &amp; Session.LCID &amp; "&lt;br /&gt;") response.write("The Date format for the above LCID is: " &amp; date() &amp; "&lt;br /&gt;") response.write("The Currency format for the above LCID is: " &amp; Format Currency(350)) response.write("&lt;/p&gt;") %&gt; &lt;/body&gt; &lt;/html&gt;                 </pre>	<p>The Currency format for the above LCID is: ö\$ 350,00</p> <p>The LCID is now changed to: 2057</p> <p>The Date format for the above LCID is: 03/08/2004</p> <p>The Currency format for the above LCID is: £350.00</p>
---	---

**Return the SessionID**

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

ASP CODE	OUTPUT
<pre> &lt;html&gt; &lt;body&gt; &lt;% Response.Write (Session.SessionID) %&gt; &lt;/body&gt; &lt;/html&gt;                 </pre>	<p>944414367</p>

**A Session's Timeout**

This example demonstrates the "Timeout" property. This example sets and returns the timeout (in minutes) for the session.

ASP CODE	OUTPUT
<pre> &lt;html&gt; &lt;body&gt; &lt;% response.write("&lt;p&gt;") response.write("Default Timeout is: " &amp; Session. Timeout &amp; "minutes.") response.write ("&lt;/p&gt;") Session. Timeout=30 response.write ("&lt;p&gt;") response.write("Timeout is now: " &amp; Session. Timeout &amp;"minutes.") response.write("&lt;/p&gt;") %&gt; &lt;/body&gt; &lt;/html&gt;                 </pre>	<p>Default Timeout is: 20 minutes.</p> <p>Timeout is now: 30 minutes.</p>

### 2.4.5 ASP Server Object

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

#### Properties

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

#### Methods

Method	Description
Create Object	Creates an instance of an object
Execute	Executes an ASP file from inside another ASP file
GetLastError()	Returns an ASPError object that describes the error condition that occurred
HTMLEncode	Applies HTML encoding to a specified string
Map Path	Maps a specified path to a physical path
Transfer	Sends (transfers) all the information created in one ASP file to a second ASP file
URLEncode	Applies URL encoding rules to a specified string

#### Examples:

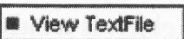
##### *When a file was last modified?*

Checks when this file was last modified.

ASP CODE	OUTPUT
<pre>&lt;html&gt; &lt;body&gt; &lt;% Set fs = Server.CreateObject ("Scripting.FileSystemObject") Set rs = s.GetFile (Server.MapPath("demo_lastmodified.asp")) modified = rs.DateLastModified %&gt; This file was last modified on: &lt;%response.write(modified) Set rs = Nothing Set fs = Nothing %&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>This file was last modified on: 18/05/2003 14:48:10</pre>

##### *Open a text file for reading*

This example opens the file "Textfile.txt" for reading.

ASP CODE	OUTPUT
<pre> &lt;html&gt; &lt;body&gt; &lt;% Set FS = Server.CreateObject("Scripting.FileSystemObject") Set RS = FS.OpenTextFile(Server.MapPath("text") &amp; "\TextFile.txt",1) While not rs.AtEndOfStream     Response. Write RS.ReadLine     Response. Write("&lt;br /&gt;") Wend %&gt; &lt;p&gt; &lt;a href="text/textfile.txt"&gt;&lt;img border="0" src="/images/btn_view_text.gif"&gt;&lt;/a&gt; &lt;/p&gt; &lt;/body&gt; &lt;/html&gt; </pre>	<pre> Hello World line 1 Hello World line 2 Hello World line 3 </pre> <div data-bbox="1040 352 1222 390" style="border: 1px solid black; padding: 2px; display: inline-block;">  </div>

### Homemade bit counter

This example reads a number from a file, adds 1 to the number, and writes the number back to the file.

ASP CODE	OUTPUT
<pre> &lt;% Set FS=Server.CreateObject("Scripting.FileSystemObject") Set RS=FS.OpenTextFile(Server.MapPath("counter.txt"), 1, False) fcount=RS.ReadLine RS.Close fcount=fcount+1 'This code is disabled due to the write access security on our server: 'Set RS=FS.OpenTextFile(Server.MapPath("counter.txt"), 2, False) 'RS.Write fcount 'RS.Close Set RS=Nothing Set FS=Nothing %&gt; &lt;html&gt; &lt;body&gt; &lt;p&gt; This page has been visited &lt;%=fcount%&gt; times. &lt;/p&gt; &lt;/body&gt; &lt;/html&gt; </pre>	<pre> This page has been visited 12345 times. </pre>

### 2.4.6 ASP Error Object (ASP 3.0)

The ASP Error object is used to display detailed information of any error that occurs in scripts in an ASP page. The ASP Error object is implemented in ASP 3.0 and it is only available in IIS5.

The ASP Error object is created when Server.GetLastError is called, so the error information can only be accessed by using the Server.GetLastError method.

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

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

#### Properties

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

## 2.5 THE GLOBAL.ASA FILE

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

The Global.asa file can contain only the following:

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

A global.asa file is the file that is related directly to the application and session objects. It serves the following purposes:



- It defines how the objects events are handled.
- It allows you to create component object instances with application or session scope.

The global.asa file must appear in the root directory of an ASP application. The global.asa file includes two containers:

- The object container <OBJECT>
- Script container <SCRIPT>

Code is included between <SCRIPT> </SCRIPT> tags, and object declaration is done within <OBJECT> </OBJECT> tags.

We have shown below the attributes for <OBJECT> tag and <SCRIPT> tag respectively:

- Attributes for <OBJECT> tag:

```
<OBJECT RUNNAT=Server Scope ID=Identifier
    {PROGID="progID"|CLASSID="classID"}>
```

- Attributes for <SCRIPT> tag:

```
<SCRIPT LANGUAGE=scriptlanguage RUNNAT=server>
```

The application and session object includes two events:

- OnStart
- OnEnd

When using VBScript, both the events are considered as a SUB and has a form where Object\_Event is Application\_OnStart, Application\_OnEnd, Session\_OnStart, or Session\_OnEnd.

```
<SCRIPT LANGUAGE=ScriptLanguage RUNNAT=server>
```

```
    SUB Object_Event
        'here is your event code'
    END SUB
    'Events SUBs in VBScript
```

```
</SCRIPT>
```

### 2.5.1 Firing Sequence of The global.asa file

On the creation of an application object, the server searches for the global.asa file. If the file is found, the script for the Application\_OnStart event is processed.

After this event, the sub-routine included in the Session\_OnStart is executed and finally the .asp file including html page is executed.

On the calling of Session.Abandon method, Session\_OnEnd event is triggered.

Code for this event is processed before the session object is destroyed.

The Application\_OnEnd event is triggered, when the system shuts down. Code for the Application\_OnEnd event is then executed before the Application object is destroyed.

### 2.5.2 Changing the Global.asa Contents

The recompilation of the server is done, if the global.asa file is changed. The server must destroy the current Application and session objects and restart, to recompile the file.

First, all the active requests are processed by the server. No more requests are processed by the server, until the Application\_OnEnd event has been processed.

When the active requests are processed, the following happens:

- The active sessions are abandoned. This triggers the Session\_OnEnd event for each session.
- The application is abandoned. This triggers the Application\_OnEnd event.
- Another requests restart the application object and create new session objects. On this, Application\_OnStart and Session\_OnStart events are triggered.

### 2.5.3 Events in Global.asa

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

1. **Application\_OnStart:** This event occurs when the FIRST user calls the first page from an ASP application. This event occurs after the Web server is restarted or after the Global.asa file is edited. The "Session\_OnStart" event occurs immediately after this event.
2. **Session\_OnStart:** This event occurs EVERY time a NEW user requests his or her first page in the ASP application.
3. **Session\_OnEnd:** This event occurs EVERY time a user ends a session. A user ends a session after a page has not been requested by the user for a specified time (by default this is 20 minutes).
4. **Application\_OnEnd:** This event occurs after the LAST user has ended the session. Typically, this event occurs when a Web server stops. This procedure is used to clean up settings after the Application stops, like delete records or write information to text files.

A Global.asa file could look something like this:

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

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

### 2.5.4 <object> Declarations

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

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

#### Syntax

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

Parameter	Description
scope	Sets the scope of the object (either Session or Application)
Id	Specifies a unique id for the object
ProgID	An id associated with a class id. The format for ProgID is [Vendor.]Component[.Version] Either ProgID or ClassID must be specified.
ClassID	Specifies a unique id for a COM class object. Either ProgID or ClassID must be specified.

#### Examples:

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

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

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

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

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

GLOBAL.ASA:

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

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

SOME .ASP FILE:

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