Le Active Server Pages (ASP) sono inserite nei file con estensione .asp.
Un file .asp è un file testuale che contiene le seguenti proprietà:

- Testo
- Tag HTML
- Comandi di Script. Un comando di script istruisce il computer a fare qualcosa, come assegnare un valore ad una variabile.

E' alquanto facile creare un file .asp: Basta rinominare un qualsiasi file HTML, sostituendo i file con le estensione esistenti .htm o .html con l'estensione .asp. Per creare uno script .asp disponibile agli utenti del Web, bisogna salvare il file in una directory appartenente al Web server ed essere sicuri di avere il permesso d esecuzione degli script. Quando il file verrà chiamato dal browser il processo ASP restituirà codice HTML
Uno Script è composto da una serie di comandi di Script.

L'esecuzione di uno script invia una serie di comandi ad un motore di scripting (scripting engine), che lo interpreta. Gli Script sono scritti in linguaggi aventi precise regole; tra queste: per poter usare un linguaggio di scripting, il server deve disporre di uno scripting engine che conosca il linguaggio. ASP contiene gli scripting engines per i linguaggi di scripting VBScript and JScript. Il primo linguaggio di scripting che ASP assume, se non altrimenti stabilito, è VBScript.
ASP in realtà non è un linguaggio di scripting ma offre un environment (ambiente) che processa gli scripts che vengono incorporati nelle pagine HTML.

Delimitatori

I Tag HTML sono differenziati dal testo dai delimitatori. Un delimitatore è un carattere o una sequenza di caratteri che marcano l'inizio e la fine di una unità.
Nel caso dell'HTML, questi delimitatori sono il simbolo minore (<) e maggiore (>).

In modo del tutto simile, i comandi e le espressioni di output degli script ASP sono differenziati sia dal testo che dai Tag HTML dai delimitatori. ASP usa i delimitatori ***<% e %>*** per includere i comandi di script.
Ad esempio, il comando ***<% nMioNumero = 5 %>*** assegna il valore numerico 5 alla variabile nMioNumero.

ASP usa i delimitatori ***<%= e %>*** per racchiudere le espressioni di output.
Ad esempio, l'espressione di output ***<%= nMioNumero %>*** invia il valore 5 (valore corrente della variable) al browser.
E' possibile includere nei delimitatori ASP qualsiasi espressione valida per il linguaggio di scripting primario.
Ad esempio, le seguenti linee producono la fine del testo con l'orario corrente del server:

Questa pagina è stata aggiornata dal browser alle ore <%= Now %>.

In questo caso, il Web server restituisce il valore della funzione di VBScript Now al browser con il testo precedente.
Una istruzione, in VBScript e negli altri linguaggi di scripting, è una unità sintatticamente completa che esprime un insieme di azioni, dichiarazioni, o definizioni. L'istruzione condizionale If...Then...Else che mostriamo, sono un classico esempio.

```
<%
If Time >=#12:00:00 AM# And Time < #12:00:00 PM# Then
cSaluto = "Buon Girno!"
Else
cSaluto = "Buona Sera!"
End If
%>
```

Queste istruzioni memorizzano il valore "Buon Giorno!" oppure il valore "Buona Sera!" nella variabile cSaluto. Non inviano nessun valore al browser client.

Le seguenti linee inviano il valore al browser client:

```
<FONT COLOR="GREEN">
<%= cSaluto %>
</FONT>
```

E' possibile includere del testo HTML tra le sezioni delle istruzioni.

Per esempio, il seguente script, che miscela HTML con l'istruzione **If...Then...Else**, produce lo stesso risultato dello script della sezione precedente:

```
<FONT COLOR="GREEN">
<% If Time > = #12:00:00 AM# And Time < #12:00:00 PM# Then %>
Buon Giorno!
<% Else %>
Buona Sera!
<% End If %>
</FONT>
```

Se la condizione è vera, ciò se è Mezzanotte o comunque prima di Mezzogiorno, allora il Web Server invia l'HTML che soddisfa la condizione (Buon Giorno) al browser; altrimenti, invia l'HTML che soddisfa la condizione Else (Buona Sera) al browser.
Le istruzioni, espressioni, comandi, e le procedure che vengono usate con i delimitatori di script devono essere valide per il linguaggio di script primario.
Il linguaggio primario d scripting di ASP per default è settato a VBScript.
Naturalmente, con ASP è possibile usare altri linguaggi di scripting; basta usare i tag di script HTML <SCRIPT> e </SCRIPT>, insieme con gli attributi LANGUAGE e RUNAT, per racchiudere le procedure scritte in altri linguaggi supportati dallo scripting engine.

Ad esempio, il seguente file .asp processa la procedura JScript chiamata MiaFunzione.

```
<HTML>
<BODY>
<% Call MiaFunzione %>
</BODY>
</HTML>

<SCRIPT RUNAT=SERVER LANGUAGE=JSCRIPT>
function MiaFunczione ()
{
Response.Write("è stata chiamata la funzione MiaFunzione")
}
</SCRIPT>
```

Non includere nel tag <SCRIPT> alcuna espressione di output o comando di script che non siano parte della procedura completa.
Server-side includes è un meccanismo che viene usato per inserire informazioni in un file prima che venga processato. ASP implementa solo la direttiva al pre-processore #INCLUDE per utilizzare questo meccanismo.
E' possibile usare questa direttiva per inserire il contenuto di un altro file in un file .asp prima che processi il file .asp. Si usa la seguente sintassi:

```
<!--#INCLUDE VIRTUAL|FILE="nome_file"-->
```

Dove si deve scrivere o VIRTUAL o FILE, in quanto Parole Chiave (Keywords) che indicano il tipo di percorso che si sta usando per includere il file, il nome e il percorso che si vuole includere.

L'inclusione dei file non richiede una speciale estensione per il nome del file; però, Microsoft raccommanda di dare ai file da includere l'estensione ***.inc*** per distinguerli dagli altri tipi di file.

Uso della Parola Chiave Virtual

La parola chiave Virtual è usata per indicare un percorso che inizia con una directory virtuale.

```
<!--#INCLUDE VIRTUAL="/miadirectory/fileincluso.inc"-->
```

Uso della Parola Chiave File

La parola chiave File è usata per indicare un percorso relativo. Un percorso relativo inizia con la directory che include il file.

Per esempio, se si ha un file nella directory MiaDirectory, ed il file FileMio.inc è nella directory MiaDirectory\FileInclusi, la seguente linea inserisce il file FileMio.inc nel file:

<!--#INCLUDE FILE="FileInclusi/FileMio.inc"-->

Notare che il percorso per includere il file, FileInclusi/FileMio.inc, è relativo al file da includere; se lo script contiene questa istruzione di inclusione l'istruzione non lavorerà.

E' anche possibile usare il parametro FILE con la sintassi ../../ per includere un file da una directory di livello superiore, se la chiave del Registry EnableParentPaths è settata ad 1.
Un file incluso può includere altri file. Un file .asp può includere lo stesso file più di una volta, l'importante è che l'istuzione <INCLUDE> non causi un ciclo infinito.
Ad esempio, se il file Primo.asp include il file Second.inc, Secondo.inc non deve a sua volta includere Primo.asp. **Non esiste la possibilità di includere sè stesso**.
ASP genera un messaggio di errore e termina di processare la richiesta del file .asp.

ASP include i file prima dell'esecuzione dei comandi di script.
Comunque, non è possibile usare un comando di script per creare il nome di un file incluso.
Ad esempio, il seguente script non includerà il file Intestazione1.inc poichè ASP tenterà di eseguire la direttiva #Include prima di assegnare il nome del file alla variabile.

<!-- This script will fail -->
<% name=(intestazione1 & ".inc") %>
<!--#include file="<%= name %>"-->

I comandi e le procedure di Script devono essere contenute interamente nei delimitatori di script <% e %>, nei tag HTML <SCRIPT> e </SCRIPT>, o nei tag HTML <OBJECT> e </OBJECT>.
Non è possibile aprire un delimitatore di script in un file .asp incluso, quindi chiudere il delimitatore in un file incluso; lo script o i comandi di script devono essere in unità complete.
In pratica, non è possibile inserire un delimitatore dentro un altro se il primo non è stato chiuso.
Ad esempio, le seguenti linee non funzioneranno:

<!-- Questo script non funziona -->
<%
For i = 1 To n
comandi nel file principale
<!--#include file="Intestazione1.inc " -->
Next
%>

Il seguente script funziona:

<%
For i = 1 to n
comandi nel file principale
%>
<!--#include file="Intestazione1.inc " -->
<% Next %>
Naturalmente ASP è usato principalmente per processare Script server-side, è possibile estenderlo e arricchirlo generando script client-side che verranno processati dal browser client.
Questo risultato viene raggiunto combinando gli script client-side, che sono inclusi nei commenti HTML, con gli script server-side che sono inclusi nei delimitatori:

```
<SCRIPT LANGUAGE="VBScript">
<!--
client script
<% server script %>
client script
<% server script %>
client script
...
-->
</SCRIPT>
```

Con questa funzinalità negli script, è possibile creare applicazioni dinamiche e funzionali.
Ad esempio, il seguente script usa un database per dare un particolare client script come risultato delle richieste degli utenti.

Nel seguente script, ASP richiede dati dal database (in questo caso, i dati sono pertinenti ai nomi e agli indirizzi di una rubrica) e generano una subroutine per ogni linea di dati.
Queste subroutine controllano quando un utente clicca i links mostrati nella paigna nel browser client.

Questo script non funziona da solo. Qui vengono soltanto illustrate le funzionalità di ASP se usate insieme ad un database, uno script server-side, uno script client-side.

```
<!-- Questo Script &egrave; incompleto -->
<%
Set rsRubrica = Server.CreateObject("ADODB.Recordset")
rsRubrica.Open "SELECT Nome.*, Indirizzo.* FROM Indirizzi INNER JOIN Nomi ON
Albums.ArtistID = Indirizzi.IndirizzoID", Session("Conn"), 1, 2

Do WhilersRubrica.EOF = False
%>
<SCRIPT LANGUAGE="VBScript">
<!--Sub Enhanced_OnLoad()
Enhanced.DrawBuffer = 500000
End Sub
Sub SparaNome<%=rsRubrica("IndirizzoID")%>_MouseEnter()
NomeCognome.Caption = "<%= rsRubrica ("NomeCognome")%>"
ANomeIndirizzo.Caption = "<%= rsRubrica("NomeIndirizzo")%>"
Divider.Visible = true
End Sub
Sub SparaNome1<%= rsRubrica("IndirizzoID")%>_Click()
Window.Location.HRef = "dettagli.asp?Indirizzoid=<%= rsRubrica("IndirizzoID")%>"
End Sub-->
</SCRIPT>
<%
rsAlbums.MoveNext
Loop
%>
```

Il linguaggio di Scripting è un passo intermedio tra l'HTML e un linguaggio di programmazione come Java, C++, e Visual Basic. HTML è generalmente usato per formattare e linkare la pagina. Il linguaggio di programmazione è invece tipicamente usato per dare una serie di istruzioni complete ai computer.

I Linguaggi di Scripting sono una via di mezzo tra i due, comunque un linguaggio di scripting ha funzioni più vicine ad un linguaggio di programmazione, piuttosto che ai semplici documenti HTML. La differenza principale tra un linguaggio di scripting e un linguaggio di programmazione è che la sintassi e le regole del linguaggio di scripting sono meno rigide e complicate di quelle del linguaggio di programmazione.

I motori di Scripting sono degli oggetti COM (Component Object Model) che processano gli script. Active Server Pages fornisce un host environment per il motore di scripting e distribuisce gli script con i file .asp a questo motore, affinchè vengano processati.
Per ogni linguaggio di Scripting usato insieme agli script ASP, il relativo motore di scripting deve essere installato sul Server Web.
Ad esempio, VBScript è il linguaggio di default di Active Server Pages, così il motore VBScript risiede come oggetto COM accessibile da Active Server Pages in modo che esso possa processare gli script VBScript.
Naturalmente, Active Server Pages può forire un ambiente di Scripting per altri linguaggi di Scripting, incluso JScript, REXX, e Perl.

Active Server Pages rende possibile agli sviluppatori Web di scrivere procedure complete mediante l'uso di molti linguaggi di scripting senza avere il problema della compatibilita' dei browser. Infatti, diversi linguaggi di scripting possono essere usati in un singolo file .asp. Ciò può essere fatto mediante l'identificazione del lnguaggio di scripting in un tag HTML all'inizio delle procedure di script.

In aggiunta, poichè gli scripts sono letti e processati sul lato server, il browser client che richiede i file .asp non necessita del supporto per lo scripting.
VBScript è il linguaggio di scripting usato come primario.
Se si usa il linguaggio di scripting primario, che usa i delimitatori <% e %>, è possibile porre qualsiasi comando di VBScript nei delimitatori di scripting. Le Active Server Pages processeranno i comandi all'interno dei delimitatori come VBScript.
Active Server Pages offre la possibilità di settare qualsiasi linguaggio di scripting come linguaggio di scripting primario. E' possibile settare il linguaggio di Scripting Primario pagina per pagina, oppure per tutte le pagine sul Server Web.

Per cambiare il linguaggio di scripting primario per tutte le pagine in tutte le applicazioni, si deve cambiare il valore della chiave DefaultScriptLanguage nel registry .

La procedura per il settaggio di scripting primario dipende principalmente dal linguaggio che supportano la scelta dalla sintassi Object.Method (oggetto.metodo).
Per i linguaggi che supportano l'uso della sintassi Object.Method ed usano le parentesi per racchiudere i parametri, come VBScript e JScript, è possibile cambiare il linguaggio di scripting primario per una singola pagina, mediante l'aggiunta di una linea di comando all'inizio del file .asp. La sintassi per questo comando è:

<%@ LANGUAGE = LinguaggioDiScripting %>

dove LinguaggioDiScripting è il linguaggio di scripting primario che si vuole settare per una particolare pagina.

Vanno seguite queste linee guida quando si setta il linguaggio di scripting primario per una pagina:

- Settare il linguaggio di scripting primario nella prima linea del file.
- Non settare il linguaggio primario di scripting della pagina in un file incluso.
- Mettere uno spazo tra @ e LANGUAGE.
- Non inserire altri elementi oltre a @ e LANGUAGE = LinguaggioDiScrpting tra i delimitatori di script (<% e %>).

Se non vengono seguite queste linee guida per il settaggio del linguaggio di scripting primario per una pagina, verrà generato un errore.
Per usare un linguaggio che non supporta la sintassi Object.Method come linguaggio di scripting primario, si deve innanzitutto creare la Chiave nel Registry LanguageEngines con la corrispondente SottoChiave LanguageName ed il valore:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
  \W3SVC
    \ASP
      \LanguageEngines
          \LanguageName
              Value: Write REG_SZ: Response.WriteEquiv |
              Value: WriteBlock REG_SZ: Response.WriteBlockEquiv |
```

dove LanguageName è il nome del linguaggio scelto, Response.WriteEquiv è l'equivalente del linguaggio di Response.Write, e Response.WriteBlockEquiv è l'equivalente del linguaggio di Response.WriteBlock. Il simbolo Pipe (|) è un inserimento usato da Active Server Pages per inviare espressioni e blocchi HTML blocks che sono normalmente processati con i metodi Response.Write e Response.WriteBlock. Questo potrebbe essere fatto automaticamente quando viene installato il linguaggio di scriping addizionale.

Nota
Alcuni linguaggi di scripting sono sensibili agli spazi bianchi e ai caratteri di newline; potrebbe non essere possibile usare questi linguggio come primari mediante il cambiamento delle chiavi al registry come precedentemente descritto.
Un'alternativa per usare questi come linguaggi primari di scripting è scrivere manualmente i blocchi HTML al browser, in modo da usare Active Server Pages per manipolare automaticamente le direttive dello script ***<% ... %>*** e l'HTML. Un'altra opzione è scrivere funzioni in quel linguaggio tra i tag di scripting (***<SCRIPT> ... </SCRIPT>***) e chiamare le funzioni da un altro linguaggio.
Una caratteristica molto attraente di Active Server Pages è la capacità di incorporare diverse procedure di linguaggi di scripting in un sngolo file .asp. Con questa funzionalità, è possibile usare linguaggi di scripting che hanno caratteristiche particolari per la risoluzione di un problema specifico.

Creazione di Procedure

Una procedura è un gruppo di comandi script che compiono task specifici. E' possibile definire una propria procedura e chiamarla ripetutamente nei propri scripts. Le definizioni di Procedure possono apparire nei tag <SCRIPT> e </SCRIPT> e devono seguire le regole per le dicharazioni dei linguaggi di scripting. E' possibile anche definire una procedura tra i delimitatori di script (<% e %>) nello stesso modo del linguaggio di scripting come primario.

Le definizioni di procedure si possono porre negli stessi file .asp che chiamano le procedure, o è possibile immettere le procedure comunemente usate in un file .asp condiviso ed usare l'istruzione server-side include (in pratica, <!--#INCLUDE FILE= ...) per includerlo in altri file .asp che chiamano le procedure. Alternativamente, esiste la possibilità di compattare le procedure nei componenti server ActiveX.

Richiamo delle Procedure

Per chiamare una procedura, bisogna includere il nome della procedura in un comando.

Per VBScript, è anche possibile usare la parola chiave (keyword) Call per chiamare una procedura. Naturalmente, se la procedura chiamata richiede argomenti, la lista degi argomenti deve essere racchiusa nelle parentesi. Se si omette la keyword Call, si devono anche omettere le parentesi attorno alla lista degli argomenti. Se si usa la sintassi Call per chiamare un qualsiasi oggetto built-in (che vedremo in seguito) o una funzione user-defined (definita dall'utente), il valore di ritorno della funzione è inutile. Se si sta chiamando una procedura JScript da una VBScript, si devono usare le parentesi dopo il nome della procedura; se la procedura non ha argomenti, si usano le parentesi vuote.

Il seguente esempio illustra come creare e chiamare procedure mediante l'uso di due differenti linguaggi di scripting (VBScript e JScript).

```
<HTML>
<BODY>
<TABLE>
<% Call Echo %>
</TABLE>
<% Call StampaData %>
</BODY>
</HTML>

<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Echo
Response.Write _
"<TR><TD>Nome</TD><TD>Valore</TD></TR>"
Set Params = Request.QueryString
For Each p in Params
Response.Write "<TR><TD>" & p & "</TD><TD>" & _
Params(p) & "</TD></TR>"
Next
End Sub
</SCRIPT>

<SCRIPT LANGUAGE=JScript RUNAT=Server>
function StampaData()
{
var x
x = new Date()
Response.Write(x.toString())
}
</SCRIPT>
```

Nota Per passare un array intero ad una procedura in VBScript, usare il nome dell'array seguito da parentesi vuote; in JScript, usare le parentesi quadre vuote.

Uso di VBScript e JScript

Quando si usa VBScript su un server con ASP, due caratteristiche di VBScript sono disablitate. Perchè gli scripts Active Server Pages sono eseguiti sul server, le istruzioni VBScript che presentano elementi con interfacce utente, InputBox e MsgBox, non sono supportati.
Le funzioni VBScript CreateObject e GetObject non sono supportate. L'uso di queste istruzioni causerà errori.

Commenti HTML

Poichè tutti gli script ASP sono processati sul lato server, non è necessario includere commenti ai tag HTML per nascondere gli script dai browser che non supportano lo scripting, come viene spesso fatto con gli script lato client. Tutti i comandi ASP sono processati prima che i contenuti siano inviati al browser.

Commenti VBScript Comments

Il REM del BASIC e lo stile dei commenti apostrofati sono supportati in VBScript.
Diversamente dai commenti HTML, questi vengono rimossi quando lo script è processato e i dati non vengono inviati al client.

```
<%
REM Questa linea e le seguenti due sono commenti
'La funzione PrintTable stampa tutti
'gli elementi in un array.
Call PrintTable(myarray())
%>
```

Importante

Non è possibile includere un commento in un espressione di output.
Ad esempio, la prima linea seguente funzionerà, ma la seconda no, poichè inizia con <%=.

<% i = i +1 'questa incrementa i. lo script funziona. %>

<%= miavar 'questo stampa a variabile miavar. questo script non funziona. %>

Commenti JScript

I caratteri di commento // sono supportati in JScript. Questi caratteri devono essere usati in ogni linea di commento.

```
<% Call StampaData %>
<SCRIPT LANGUAGE=JScript RUNAT=Server>
function StampaData()
{
var x
x = new Date()
Response.Write(x.getDate())
}
// Questa è una definizione per la procedura StampaData.
// Questa procedura invia la data corrente al browser sul lato client.
</SCRIPT>
```

Spesso si vogliono ottenere informazioni su di un utente quali, ad esempio, il tipo di browser che l'utente sta utilizzando. Si può anche voler ricevere infrmazioni da un utente, ad esempio, quando l'utente immette informazioni in un form. L'oggetto built-in ASP Request ricava queste informazioni facilmente.

L'oggetto Request dà accesso a qualsiasi informazioni che viene passata con una richiesta HTTP (HTTP request). Questa include:

- Un set standard di informazioni incluse in un set di variabili server.
- Un set di parametri passati con il metodo POST.
- Un set di Parametri richiesti (query) attaccate al metodo GET.
- Cookie che sono passati da un browser. I Cookie pemettono ad un set di informazioni di essere associate ad un utente.

- Certificati Client.

L'oggetto Request ha cinque collezioni (cllections) associate:

- QueryString
- Form
- Cookies
- ServerVariables
- ClientCertificate

Si può usare la seguente sintassi generale per accedere alle informazioni nell'oggetto Request:

Request.NomeCollection(variabile)

Dove *NomeCollection* può essere QueryString, Form, Cookies, ServerVariables, o ClientCertificate, e *variabile* è il nome dela varibile nella collezione a cui si vuole accedere.

Si può usare la seguente sintassi generale per accedere alle varibili nel'oggetto Request senza includere il nome della collection:

Request(*nomevariabile*)

Le collections sono ricercate in questo ordine: QueryString, Form, Cookies, ServerVariables, ClientCertificate. Viene ritornata la prma varibile in cui viene riscontrata *nomevariabile*.

Se una pagina HTML ha più di una varibile con lo stesso nome, siate sicuri di includere il nome della collezione Request ed il nome della variabile.

Un *form* HTML è il mezzo usato più frequentemente per ottenere informazioni da un utente Web. Una casella di testo di un form, un pulsante di opzione, e una check box, mostrata in una pagina HTML da un browser, forniscono agli utente un facile modo per inviare le informazioni. Quando un utente clicca sul pulsante Submit, il browser invia le informazioni raccolte al Server Web.

Si può usare un file .asp per raccogliere o processare i valori di un form HTML in tre modi:

- Un file .htm statico può contenere un form che posta i suoi valori ad un file.asp.
- Un file .asp può creare un form che posta le informazioni ad un altro file .asp.
- Un file .asp può creare un form che posta le informazioni a se stesso, che è, lo stesso file .asp che contiene il form.

I primi due metodi operano allo stesso modo, come form che interagiscono con altri programmi gateway, eccetto che, con ASP, si possono includere comandi che leggono e rispondono alle scelte degli utenti.

Creare un file .asp che contiene la definizione di un form che posta le informazoni a sè stesso è un un pò più complicato ma molto più potente del lavorare con i forms.

E possibile usare la varibile server QUERY_STRING per processare le informazioni QUERY_STRING da una user request (richiesta dell'utente). Se il form method è POST,la collection QueryString contiene tutte le informazioni passate come parametri dopo il punto interrogativo nell'URL. Se il form method è GET, la collection QueryString contiene tutte le informazioni passate nel form.

Ad esempio, quando un utente invia la seguente richiesta URL, la collection Request.QueryString conterrebbe due valori : `nome` ed `età`.

<A HREF=" size="2">fileasp.asp?nome=Giuseppe+Guerrasio&età=19">

Il seguente script usa l'oggetto Request per accedere a questi valori.

Benevenuto, <%= Request.QueryString("nome") %>.
La tua età è <%= Request.QueryString("età") %>.

In questo caso, il seguente testo dovrebbe essere inviato all'utente:

Benvenuto, Giuseppe Guerrasio. La tuà età è 30.

La collection QueryString manipola anche automaticamente il caso di varibili multiple con lo stesso nome. Quando compare una strnga di richiesta del tipo `nome=Giuseppe&nome=Massimiliano&nome=ASP`, ad esempio, ASP crea una nuova collezione chiamata `nome` che a turno contenga i tre valori: Giuseppe, Massimiliano, e ASP. Ognuno di questi valori è indicizzato da un Intero(Integer), con i seguenti risultanti:

Referenza	**Valore**
Request.QueryString("nome")(1)	Giuseppe
Request.QueryString("nome")(2)	Massimiliano
Request.QueryString("nome")(3)	ASP

Una collection creata in questo modo supporta la proprietà Count. La proprietà Count descrive quante items (parti) contiene una collection. In questo esempio, il valore di Request.QueryString("nome") è 3 , poichè tre valori separati sono memorizzati nella collection nome.

Se si vuole usare il metodo Response.QueryString per ottenere accesso alla varibile nome, l'output diventerebbe una stringa delimitata da virgola. Nell'esempio, il valore di Request.QueryString("nome") sarebbe

Giuseppe, Massimiliano, ASP"

La collection Form contiene tutti i valori che un utente inserisce in un form inviato tramite il metodo POST. Ad esempio, un utente riempie ed invia il seguente form:

```
<form action="invio.asp" method="post">
<p>Nome: <input name="nome" size=48>
<p>Sito Preferitor: <select name="sito">
<option>HTML.it <option>gif-animate.it <option>font.it <option>emerchant.it
</select>
<p><input type=submit>
</form>
```

Il seguente script è la pagina del risultato (invio.asp):

```
Benvenuto, <%= Request.Form("nome") %>.
Il tuo sito preferito è <%= Request.Form("sito") %>.
```

La collection Form permette l'utilizzo di parametri multipli con lo stesso nome allo stesso modo della collection QueryString.

La collection ServerVariables fornisce informazioni dalle intestazioni HTTP (HTTP headers) che sono passate insieme ad una richiesta utente come Varibili Enviroment del Server Web. E' possibile utilizzare queste informazioni per personalizzare le risposte agli utenti. Questo script accede alla variabile server SERVER_PORT definita dallo standard Common Gateway (CGI):

Questa Richiesta HTTP è stata ricevuta sulla
porta TCP/IP <%= Request("SERVER_PORT") %>.

Il seguente script, che fornisce il contenuto basato sul linguaggio utente, accede alla variabile di intestazione HTTP HTTP_ ACCEPT_LANGUAGE:

```
<% language = Request.ServerVariables("HTTP_ACCEPT_LANGUAGE")
If language = "en" Then %>
  <!--#INCLUDE FILE="myapp/Englishpage.asp"-->
<% Else %>
  <!--#INCLUDE FILE="myapp/Otherlang.asp"-->
<% End If %>

<HTML>
<BODY>

<!-- Questo è GetEmail.asp -->

<%
If IsEmpty(Request("Email")) Then
  Msg = "Sei pregato di inserire il tuo indirizzo email."
ElseIf InStr(Request("Email"), "@") = 0 Then
  Msg = "Sei pregato di inserire il tuo indirizzo email" & _
  " nel form username@location."
Else
  Msg = "Questo Script processerà " & _
  "l'indirizzo Email valido."
End If
%>

<FORM METHOD="POST" ACTION="getmail.asp">
<PRE>
Email: <INPUT TYPE="TEXT" NAME="Email" SIZE=30
VALUE="<%= Request("Email") %>">
<%= Msg %> <P>
<INPUT TYPE="SUBMIT" VALUE="Submit">
</PRE>
</FORM>
</BODY>
</HTML>
```

Un cookie è un token che il browser client invia al Web server, o che il Web server invia al browser client. I Cookie permettono ad un set di informazioni di essere associate ad un utente. Gli script ASP possono sia prendere che settare i valori di un cookie mediante l'uso della collection Cookies.

Per ricevere un valore da un cookie è necessario usare la collection Request.Cookies. Ad esempio, se la richiesta HTTP del client setta animale=tigre, la seguente istruzione riceverà il valore tigre:

<%= Request.Cookies("animale") %>

Se una richiesta HTTP invia valori multipli per lo stesso cookie, ASP crea un cookie indicizzato. Ad ogni vaore è assegnata una chiave; si può ricevere un particolare valore di una chiave cookie mediante l'uso della sintassi Request.Cookies("nome")("chiave"). ad esempio, se un client invia le seguenti richieste HTTP:

animale=elefante&elefante=Africano

Il seguente comando script restituisce il valore Africano:

<%= Request.Cookies("animale")("elefante") %>

E' possibile usare l'oggetto built-in ASP Response per controllare le informazioni da inviare all'utente mediante l'uso di:

1. metodo Response.Write per inviare informazioni direttamente al browser.
2. metodo Response.Redirect per dirigere un utente ad un URL diverso da quello richiesto.
3. metodo Response.ContentType per controllare il tipo di contenuti da inviare.
4. metodo Response.Cookies per settare i valori dei cookie.
5. metodo Response.Buffer per effettuare il buffering delle informazioni.

Il metodo Write è il metodo dell'oggetto Response usato più frequentemente. E' possibile usare il metodo Write per inviare informazioni ad un utente dall'interno dei delimitatori ASP.

Response.Write variant

Dove variant può essere qualsiasi tipo di dato supportato dal linguaggio di scripting primario di default.

Ad esempio, le seguenti istruzioni inviano un saluto all'utente:

```
<%
If utente_nuovo Then
  Response.Write "<H3 ALIGN=CENTER>Ciao Nuovo Utente</H3>"
Else
  Response.Write "<H3 ALIGN=CENTER>Bentornato Vecchio Utente</H3>"
End If
%>
```

Il metodo Response.Write è utilizzato principalmente se si vuole inviare contenuti all'utente dall'interno di una procedura.

Non si deve usare Response.Write per inviare contenuti all'utente. I contenuti che non sono nei delimitatori di script vengono inviati direttamente al browser, che li formatta e li visualizza. Ad esempio, il seguente script produce esattamente lo stesso output dello script precedente:

```
<H3 ALIGN=CENTER>
<% If utente_nuovo Then %>
Ciao Nuovo Utente.
<% Else %>
Bentornato Vecchio Utente.
<% End If %>
</H3>
```

Allo stesso modo di come si inviano dei cntenuti agli utenti, è possibile redirigere il browser ad un altro URL con il metodo Redirect.

Response.Redirect *URL*

Ad esempio, se si vuole essere sicuri che l'ingresso alla pagina ASP avvenga da una pagina particolare, si può controllare se l'utente proviene da questa pagina; se non è così, si può inviare l'utente ad un altra pagina.

```
<%
If Not Session("Proveniente_dalla_Home_Page") Then
Response.Redirect "homepage.asp"
End If
%>
```

Se si usa Response.Redirect da un file .asp dopo che i contenuti sono già stati inviati all'utente, verrà generato un messaggio di errore.

E' possibile usare la proprietà ContentType del'oggetto Response per settare il tipo stringa di contenuti HTTP (HTTP content type) per i contenuti da inviare all'utente. La sintassi generale è

Response.ContentType = ContentType

dove *ContentType* è una stringa che descrive il tipo di contenuti. Per una lista completa del tipo di contenuti supportati, controllare la documentazione del Browser Web o le correnti specifiche del protocollo HTTP.

ad esempio, se si vuole inviare il sorgente (che è, il file .asp da cui si genera una pagina ASP) al browser, settare il ContentType a text/plain:

<% Response.ContentType = "text/plain" %>

Il browser mostrerà la pagina come testo, senza interpretare la pagina come HTML.
Un cookie è un token che un browser client invia al server Web o il server Web invia al browser client. I Cookies permettono il settaggio di informazioni da associare all'utente. Gli script ASP possono sia leggere che settare i valori dei cookies mediante l'uso della collezione Cookies.

Per settare il valore di un cookie, usare Response.Cookies. se il cookie non esiste già, Response.Cookies ne creerà uno nuovo:

<% Response.Cookies("animale")="elefante" %>

Ugualmente, per settare il valore di una chiave di un cookie:

<% Response.Cookies("animale")("elefante")="Africano" %>

Se un cookie che esiste già ha il valore relativo alla chiave ma Response.Cookies non specifica un nome di chiave, il valore della chiave esistente sarà cancellato. Ugualmente, se esiste un cookie che non ha un valore per la chiave, ma Response.Cookies specifica un nome di chiave ed un valore, il valore esistente del cookie sarà cancellato e il nuovo valore della chiave verrà creato.
Il settaggio di default per il bufferng di tutte le pagine ASP è disabilitato. Naturalmente, è possibile settare la proprietà Buffer del'oggetto Response su True (vero) per processare tutti gli script su di una pagina prima di inviare qualsiasi cosa all'utente:

<% Response.Buffer = True %>

E' possibile usare il buffering per determinare in quale punto della processazione della pagina non si vuole inviare il contenuto precedente. Si può, anche, redirigere l'utente ad un'altra pagina con il metodo Redirect dell'oggetto Response, o ripulire il buffer con il metodo Clear dell'oggetto Response e inviare contenuti differenti all'utente. Il seguente esempio usa tutti questi metodi:

```
<% Response.Buffer = True %>
<html>
<body>
.
.
.
<%
If Request("FName") = "" Then
  Response.Clear
  Response.Redirect "/aspsamp/samples/test.html"
  Response.End
Else
  Response.Write Request("FName")
End If
%>
</body>
</html>
```

Quando si chiama il metodo Buffer in uno script e non si chiama il metodo Flush nello stesso script, il server manterrà ancora vive (Keep-Alive) le richieste fatte dal client. Il beneficio delo scrivere script in questo modo è che le performance del server sono migliori poichè il server non dovrà creare una nuova connessione per ogni richiesta del client (assumendo che il server, client, ed i proxi supportino tutti le richieste keep-alive).

D'altronde, un potenziale contropareggio per questo approccio è che il buffering preventivo per ognuna di queste risposte prima che siano mostrate all'utente, finchè il server abbia finito di processare gli script per i file .asp correnti.

Il Buffering è disabilitato per default per tutte le pagine ASP in tutte le applicazioni mediante il settaggio del BufferingOn del registry a 0.
I Programmatori di Script spesso cercano su una base regolare ciò che gli necessita per effettuare determinati compiti nei loro script.
Ad esempio, si deve avere un numero di script , tutti alla fine effettuano compiti differenti, ma tutti necessitano di informazioni da parte degli utenti.
Gli Oggetti (Objects) ci preservano dalla fatica di "Reinventare la ruota" ogni volta che è necessario effettuare un compito comune. Un oggetto è una combinazione di programmazione e dati che possono essere messi insieme in una unità.

Per usare la maggior parte degi oggetti, si deve prima creare un'istanza dell'oggetto. Active Server Pages (ASP) include cinque oggetti che non richiedono l'intestazione.
La seguente tabella riassume questi oggetti built-in, per quali compiti vengono usati, e un breve esempio.

Oggetto	**Compito**	**Esempio**
Oggetto Request	Ricevere informazioni da un utente.	Lezione 6.1
Oggetto Response	Inviare informazioni ad un utente.	Lezione 6.2
Oggetto Server	Controllare l'enviroment di esecuzione ASP.	Lezione 6.3
Oggetto Session	Memorizzare informazioni sula Sessione di un utente.	Lezione 6.4
Oggetto Application	Condividere informazioni tra i diversi utenti di una applicazione.	Lezione 6.5

La sintassi mediate la quale si ottiene l'accesso ad un oggetto dipende dal linguaggio di scripting che si sta usando. Poichè il linguaggio primario di scripting di ASP per Default è VBScript, gli esempi che appariranno in questo corso usano la sintassi VBScript, tranne quando non specificato diversamente. Se si vuole utilizzare un altro linguaggio, fare riferimento alla documentazione del linguaggio per la sintassi appropriata su come lavorare con gli oggetti.

Gli oggetti Request e Response contengono *collezioni*. Una collezione è un set pezzi di informazioni collegate in cui è possibile accedere allo stesso modo. Si può anche ottenere l'accesso alle informazioni in una collezione mediante l'uso delle istruzioni For...Each.

Si ottiene l'accesso agli oggetti da uno script mediante l'uso dei *metodi* e delle *proprietà*.

Un metodo è una procedura che agisce su di un oggetto. La sintassi da generare è

Oggetto.Metodo *parametri*

Dove *parametri* può essere un variante, dato, stringa o URL, a seconda del metodo;

Una *proprietà* è un nome di attributo di un oggetto. Le Proprietà definiscono le caratteristiche di un oggetto, come la grandezza, il colore, e la posizione sullo schermo; o lo stato di un oggetto, come abilitato o disabilitato. La sintassi generale è

Oggetto.Proprietà *parametri*

Dove *parametri* può essere un valore, una stringa, o un flag, a secondo dela proprietà.

Oggi iniziamo un percorso per scoprire il funzionamento e lo sviluppo di Active Server Page.

Queste pagine sono tutorial, completi di esempi, per aiutare tutti coloro che non hanno dimestichezza con

l'ASP ma vogliono imparare a sviluppare la prime applicazioni.

Per aiutare a sviluppare degli esempi vi consiglio di creare sul vostro disco una directory chiamata **tutorial_asp1**, così da avere un riferimento comune.
Le pagine ASP (Active Server Pages), a differenza di quelle HTML che sono elaborate dal client, interpretano la richiesta del client e restituiscono al Browser pagine in formato HTML.

Come primo esempio potete scrivere la frase "**HTML.it**".
Per farlo, nella directory di esempio create una pagina chiamata **esempio1.html**. Utilizzando il seguente codice:

```
<TABLE><TD>HTML.IT</TD></TABLE>
```

Richiamando la pagina con il suo percorso fisico (DISCO:\tutorial_asp1\esempio1.html), verrà visualizzata la frase "HTML.IT".

Tentiamo ora di far scrivere la stessa frase sfruttando l'ASP.
Con il linguaggio ASP si deve comandare al server di scrivere la frase richiedendo la generazione di un codice HTML che possa visualizzarla. Utilizzate quindi il codice sotto indicato salvando poi il file **esempio1.asp**.

```
<TABLE><TD>
<% response.write("HTML.it")%>
</TD></TABLE>
```

Come potete notare l'apertura della tabella è in HTML seguita poi da "<%" comando che informa il server che segue uno script.

Il primo comando che incontrate è **RESPONSE.WRITE** cioè "**SCRIVI**". Ogni volta che vorremo far scrivere qualcosa all'interno del codice ASP dovremo utilizzare **RESPONSE.WRITE**, e dipendentemente da quello che è contenuto nelle parentesi, aggiungere i doppi apici. Infatti la frase HTML.IT non è una variabile, ma semplice testo che quindi deve essere racchiuso tra gli apici doppi. Se pubblichiamo lo script salvandolo nella pagina indicata, e tentiamo di visualizzare la pagina come in precedenza, ci verrà chiesto **se vogliamo salvare il file**. Questo avviene a causa della mancata elaborazione del file ASP. I file ASP infatti non possono essere richiamati come normali pagine HTML (vedi esempio1.html) , ma devono essere elaborati dal server e quindi pubblicati all'interno di applicazioni WEB. E' necessario quindi installare un server web sul nostro PC, così che si possa raggiungere il file esempio1.asp, con il percorso http://mioweb/esempio1.asp
Si dovrà quindi installare un server WEB sul nostro PC per effettuare le varie prove. Possiamo installare Personal Web Server per Windows 95/98 contenuto nel CD di Windows 95/98, o scaricabile cliccando qui.
Per Windows NT Server, invece, vi consigliamo i tutorial di HTML.it.

L'installazione di PWS su Windows 98/95 è abbastanza semplice in quanto sufficientemente guidata.
L'installazione base ci chiederà dove risiedono i file che dovranno essere la nostra Home Page. Per default ci proporrà il percorso **c:\inetpub\wwwroot**.
Terminata l'installazione possiamo aprire il browser e digitare l'indirizzo **http://nomemacchina** oppure **http://127.0.0.1** , entrambi gli indirizzi puntano alla root principale del Vs. sito e cioè al file default.asp contenuto in **c:\inetpub\wwwroot**.

A questo punto possiamo creare il nostro percorso per raggiungere i file di esempio. **Ma come raggiungerli?**

- Cliccando sull'icona di PWS si aprirà la finestra di personalizzazione

Cliccando sulla voce "Impostazioni avanzate" ci verranno mostrate tutte le Virtual Directory del nostro sito. Cioè tutte le directory fisiche che possono essere raggiunte con il percorso http://127.0.0.1. Possiamo ora aggiungere la nostra directory di prova.
Cliccare sul pulsante "Aggiungi", si aprirà una nuova finestra

La nuova finestra chiamata "Aggiungi Directory", permette di aggiungere nuove directory al nostro sito. aggiungiamo la nostra " **Tutorial_asp**" andandola a ricercare con il pulsante "sfoglia ", fatto qesto digitiamo il suo Alias "**Tutorial**".

Dobbiamo ora stabilire se questa nuova virtual directory potrà essere letta (Lettura), e se dovrà eseguire degli script come pagine ASP.
Per entrambi i casi dobbiamo confermare i due flag che si trovano nell'insieme "Accesso", che sono Lettura e Script. Diamo OK e se tutto è stato fatto correttamente il nostro server web conterrà nel suo albero la nuova virtual directory "tutorial".

Evidenziando la directory tutorial potremo stabilire altri parametri come:

Visualizza documenti predefiniti - Caricamento di una pagina predefinita quando si punta l'indirizzo http://127.0.0.1/tutorial, non confermatelo.

Consenti esplorazione sito web - Permette di visualizzare il contenuto della directory utilizzando il browser , confermatelo.

Ora se tutto è andato bene digitando l'indirizzo http://127.0.0.1/tutorial il browser visualizzerà i due file che abbiamo creato **esempio1.html** e **esempio1.asp**.

Possiamo ora provare i file creati nella prima lezione, all'indirizzo del Vs. PC
http://127.0.0.1/tutorial/esempio1.html e http://127.0.0.1/tutorial/esempio1.asp.
Entrambi i file danno lo stesso risultato cioè "**HTML.IT**", utilizzando lo stesso codice sul browser. Infatti leggeremo per entrambi i file lo stesso codice.

Questo avviene in quanto l'ASP non fa altro che inviare al client la pagina HTML da visualizzare, elaborata dal server.

Abbiamo visto come visualizzare un testo fisso all'interno di uno script ASP, ma come posso visualizzare dati variabili?

I dati variabili necessitano di linguaggi di programmazione che nel mondo della rete possono essere: VBScript, JScript, JavaScript, Perl ed altri ancora. VBScript è sufficentemente intuitivo e soprattutto, essendo di larga diffusione, ha il vantaggio di informazioni ed esempi reperibili in Rete. Sono comunque molte le occasioni in cui è possibile sfruttare funzioni JavaScript in comunione con VBScript all'interno di file ASP.

Ma come strutturare correttamente una pagina ASP che utilizzi VBScript?

Come accennato prima, esistono molti linguaggi di programmazione utlizzabili per ASP, si deve quindi informare il server Web sulla tipologia di linguaggio che si sta utlizzando. Microsoft per default stabilisce come linguaggio il VBScript, è quindi superfluo indicarlo, ma è buona norma farlo all'interno di tutte le pagine, inserendolo come prima linea.

```
<%@ Language=VBscript %>
```

Tornando quindi all'esempio esempio1.asp lo script diverrà

```
<%@ Language=VBscript %>
<TABLE><TD>
<% response.write("HTML.it")%>
</TD></TABLE>
```

A questo punto modifichiamo lo script visualizzando la frase:

"Tutorial ASP di HTML.it del (data di oggi)"

Aprite il file esempio1.asp con Notepad, oppure con un editor HTML evoluto (Homesite, Visual InterDev, ASPEdit, FrontPage..), uno dei vantaggi che potrete apprezzare utlizzando questi editor consiste nella colorazione differenziata dello script che è utilissima nella fase di sviluppo e nell'individuazione del numero di riga.

Se utilizzate Notepad ricordate che per salvare i file con l'estensione .asp, dovrete indicare il nome del file e la sua estensione tra doppi apici. Altrimenti verrà salvato come .txt.

```
<%@ Language=VBscript %>
<% oggi=date%>

<TABLE><TD>
<% response.write"Tutorial di HTML.it del "&oggi %>
```

Nel codice creato abbiamo inserito la nostra prima variabile *oggi*, infatti questo "contenitore" viene riempito dal valore *date* che esprime la data odierna prelevandola dal server. E' importante sottolineare che le variabili dovrebbero essere sempre dichiarate al fine di rendere più stabile e sicura l'applicazione. Dichiarare una variabile vuol dire dimensionarla, e nell'esempio che abbiamo rappresentato basterà aggiungere:

```
<%@ Language=VBscript %>
<% dim oggi %>

<% oggi=date%>

<TABLE><TD>
<% response.write"Tutorial di HTML.it del "&oggi %>
</TD></TABLE>
```

Se avessimo avuto più variabili come ad esempio ora,mese,anno potevamo utilizzare la seguente sintassi:

```
<% dim oggi,ora,mese,anno %>
```

Per essere certi di non dimenticare nessuna variabile si dovrà utilizzare *Option Explicit*, il quale genera un errore se nella pagina trova variabili non dimensionate, indicando quale variabile è ancora da dimensionare.

```
<%@ Language=VBscript %>
<% Option Explicit %>

<% dim oggi %>

<% oggi=date%>

<TABLE><TD>
<% response.write"Tutorial di HTML.it del "&oggi %>
</TD></TABLE>
```

Analizziamo la parte che scrive il testo della pagina, infatti come detto in precedenza il testo fisso dovrà essere inserito tra i doppi apici:

```
response.write"Tutorial di HTML.it del "
```

La variabile *oggi* potrà essere aggiunta al testo con l'ausilio di una congiunzione *(&)* e priva di doppi apici:

```
response.write"Tutorial di HTML.it del "&oggi
```

Per ottimizzare il codice cerchiamo di eliminare tutte le aperture e chiusure degli script inutili, aggiungendo dei commenti:

```
<%@ Language=VBscript %>
<% Option Explicit %>

<%

' Apro lo script

' I commenti possono essere inseriti nello script precedendoli con Apice

' Dimensiono le variabili

dim oggi
```

' Creo la variabile oggi con il valore di data odierno

oggi=date

' Chiudo lo script

%>

<TABLE><TD>
<% response.write"Tutorial di HTML.it del "&oggi %>
</TD></TABLE>

Abbiamo quindi creato la nostra prima applicazione ASP, visualizzando la data di oggi. Provate ora ad aggiungere anche il mese e l'anno in corso.
Per farlo sappiate che il mese e l'anno potranno essere definiti con:

month(now) year(now)

Ricordate che le variabili vanno sempre dichiarate, e che le variabili non devono essere incluse tra doppi apici. La sintassi per valorizzare una variabile è:

nome della variabile = Valore

Es. (mese = month(now))