

Active Server Pages Reference

Brian Lalonde

Contents

Syntax Basics	1
Server Directives	3
global.asa file	5
Application object	7
Application.Contents collection.....	7
Application.Lock method.....	7
Application.StaticObjects collection.....	7
Application.Unlock method.....	8
Session object	9
Session.Abandon method.....	9
Session.CodePage property.....	9
Session.Contents collection.....	9
Session.LCID property.....	10
Session.SessionID property.....	10
Session.StaticObjects collection.....	10
Session.Timeout property.....	10
Server object	13
Server.CreateObject method.....	13
Server.Execute method.....	13
Server.GetLastError method.....	13
Server.HTMLEncode method.....	14
Server.MapPath method.....	14
Server.ScriptTimeout property.....	14
Server.Transfer method.....	14
Server.URLEncode method.....	15
Request object	17
Request.BinaryRead method.....	17
Request.ClientCertificates collection.....	18
Request.Cookies collection.....	18
Request.Form collection.....	18
Request.QueryString collection.....	18
Request.ServerVariables collection.....	18
Request.TotalBytes property.....	19
Response object	21
Response.AddHeader method.....	21
Response.AppendToLog method.....	21
Response.BinaryWrite method.....	21
Response.Buffer property.....	21
Response.CacheControl property.....	22
Response.Clear method.....	22
Response.ContentType property.....	22
Response.Charset property.....	22
Response.Cookies collection.....	23
Response.End method.....	23
Response.Expires property.....	23
Response.ExpiresAbsolute property.....	23
Response.Flush method.....	23
Response.IsClientConnected property.....	24
Response.PICS property.....	24
Response.Redirect method.....	24
Response.Status property.....	24
Response.Write method.....	24
IStringList collection	27
Internationalization	29

Syntax Basics

Special ASP blocks, and some rudimentary SSI support.

Details

Active Server Pages are standard HTML files with embedded script.

Two forms of the *Server Side Include* (SSI) directives are supported, and are processed before any ASP blocks:

- `<!--#include virtual="/absolute_virtual_path"-->`
- `<!--#include file="relative_physical_path"-->`

These directives can be used anywhere a comment would be legal in the HTML. Conventionally, included files are given the `.inc` filename extension.

Warning

When using Python as an ASP scripting language, whitespace is *very* significant.

```
<% Response.Write(Application('started')) %> <!-- ERROR -->
<%Response.Write(Application('started'))%> <!-- OK -->
```

ASP Block Types

Server Directive Block

`<%@ server_directives %>`Sets various ASP options ([]). This block can only occur at the top of the ASP file, and only once.

Tip

To ensure that your pages will still work, unaltered, after moving to a new hosting provider (especially one unwilling or unable to change the IIS settings), or after a reinstall/upgrade of the server software (which could replace all of your custom IIS settings with the defaults), include a `Language` directive for any non-VBScript page, and an `EnableSessionState` directive whenever the `Session` object is used (session support can be very resource-intensive, and should be disabled unless needed).

Write Blocks

`<%= expression %>`Converted to `Response.Write(expression)`. *This block cannot contain line breaks.*

Note

Adjacent write blocks will 'eat' any whitespace between them, so `<%= firstname %> <%= lastname %>` will result in `BrianLalonde` being sent to the browser. To prevent this, use the space (or nonbreaking space) character entity `<%= firstname %> <%= lastname %>`, or include the space character in a single write block `<%= firstname+" "+lastname %>`.

Procedural Blocks

`<% procedural_script %>`Executes the procedural script, but doesn't directly insert anything into the resulting HTML. Translates literally into `Response.WriteBlock(procedural_script)`.

Server Directives

Set the script language, code page, or locale; or enable session or transaction.

Details

Language

Establishes the page's scripting language. VBScript is the default when IIS is installed, but the default language is configurable for each web application. JScript is an alternative language, available on all IIS servers. PerlScript [<http://www.activestate.com/products/activeperl/>], Python [<http://www.activestate.com/products/activepython/>], and various other languages can also be installed and used, but can be somewhat slower than the lightweight (and less complete) VBScript or JScript.

EnableSessionState

True or False. Sessions identify a user connection. True is the default when IIS is installed, but the session state can be set independently for each web application. The IIS server uses connection info, cookies, etc. to discern one user/connection from another. This is required to use the [Session](#) object.



Tip

Sessions incur some overhead, and should be disabled for servers or individual web applications in the server settings unless required.

CodePage

See [].

LCID

See [].

Transaction

One of Required, Requires_New, Supported, or Not_Supported. Determines whether a transaction will be initiated. With transactions enabled, theObjectContext object becomes available:

SetAbort() method

Aborts the page's transaction.

SetComplete() method

The page declares that its part in the transaction is finished.

OnTransactionAbort event

Called when the transaction fails.

OnTransactionCommit event

Called when the transaction completes.

global.asa file

Create application- or session-level objects or values.

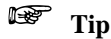
ASA File Elements

Type Library Declarations

Allows the use of an object library's enumerated constants throughout the application. For example, by including the ADO typelib, you can use `conn.OpenSchema(ADODB.adSchemaTables)` instead of `conn.OpenSchema(20)`.

```
<!-- METADATA TYPE="TypeLib" UUID="TypeLibID", FILE="TypeLibFile" VERSION="major.minor"
LCID="LocaleID" -->
```

Although the version and locale can be requested, there will be no error if the server cannot find the variant specified.



You can add a descriptive name to the typelib declaration before the METADATA keyword, to make it more readable.

Some Common TypeLib Declarations

```
<!-- ADODB 2.5 METADATA TYPE="TypeLib" UUID="{00000205-0000-0010-8000-00AA006D2EA4}" -->
<!-- ADODB 2.6 METADATA TYPE="TypeLib" UUID="{00000206-0000-0010-8000-00AA006D2EA4}" -->
<!-- ADODB 2.7 METADATA TYPE="TypeLib" UUID="{EF53050B-882E-4776-B643-EDA472E8E3F2}" -->
<!-- ADODB 2.8 METADATA TYPE="TypeLib" UUID="{2A75196C-D9EB-4129-B803-931327F72D5C}" -->
<!-- ADOMD METADATA TYPE="TypeLib" UUID="{22813728-8BD3-11D0-B4EF-00A0C9138CA4}" -->
<!-- ADOX METADATA TYPE="TypeLib" UUID="{00000600-0000-0010-8000-00AA006D2EA4}" -->
<!-- CDONTS METADATA TYPE="TypeLib" UUID="{0E064ADD-9D99-11D0-ABE5-00AA0064D470}" -->
<!-- CDSYS METADATA TYPE="TypeLib" UUID="{CD000000-8B95-11D1-82DB-00C04FB1625D}" -->
<!-- Scripting FSO METADATA TYPE="TypeLib" UUID="{420B2830-E718-11CF-893D-00A0C9054228}" -->
<!-- MSWC.IISLog METADATA TYPE="TypeLib" UUID="{B758F2F9-A3D6-11D1-8B9C-080009DCC2FA}" -->
<!-- MSXML2 METADATA TYPE="TypeLib" UUID="{F5078F18-C551-11D3-89B9-0000F81FE221}" -->
<!-- WinHttp5 METADATA TYPE="TypeLib" UUID="{C92A03CF-B92B-404F-9AC5-58664A592E4C}" -->
<!-- WinHttp5.1 METADATA TYPE="TypeLib" UUID="{662901FC-6951-4854-9EB2-D9A2570F2B2E}" -->
```

Object Declarations

Creates a COM object for the application or user session that is available to the entire application. This allows you to set up a database connection, for example, and re-use it on each Active Server Page in the application, rather than re-connecting for each individual page.

```
<object runat="Server" scope="scope" id="varname" progid="ProgID",
classid="ClassID"></object>
```

The *scope* for an object must be either `Application` (a single object for the entire application) or `Session` (a copy of the object for each user session). The object will be available in the global namespace throughout the application, using the name *varname*, and in the `StaticObjects` collection of the `Application` or `Session` object (depending on the object's scope).



Creating and initializing an object once per application or user is significantly faster than doing it repeatedly per page. For example, creating an `ADODB.Connection` for the application and connecting it to the database in the `Application_OnStart` event, rather than on each page, saves the user the time of creating and connecting (and disconnecting, and releasing) the database connection on each page that uses the object. Multiple application objects: database connections, commands, XSLT templates, etc. taken together can make a site feel noticeably faster.



Creating a single object for multiple pages also creates a single point of failure. For example, if a global database connection is unexpectedly disconnected, all the pages that use the object will be unable to access the database, unless they can detect the problem and reconnect.

Also, be aware that COM objects used concurrently must be thread-safe (Able to run for multiple pages at once.) [<http://en.wikipedia.org/wiki/Thread-safe>].

Event Handler Script

Application and session event handlers are defined in one or more script sections, each of which can use a different language. An event handler is a subroutine with one of the following names:

Application_onStart

Called the first time a page of the application is processed.

Application_onEnd

Called when the web service is shut down.

Session_onStart

Called the first time a user accesses an application that has sessions enabled. Used to initialize user-specific settings.

Session_onEnd

Called when the user session expires.

OnTransactionAbort

Called when the transaction fails, in an application that has enabled transactions.

OnTransactionCommit

Called when the transaction completes, in an application that has enabled transactions.

```
<!-- ADO DB 2.5 METADATA TYPE="TypeLib" UUID="{00000205-0000-0010-8000-00AA006D2EA4}" -->
<object runat="Server" scope="Application" id="conn" progid="ADODB.Connection"></object>
<script language="VBScript" runat="Server">
Sub Application_onStart
    conn.Open "ConnectionString"
    Application("MaxLogonTries")= 3
End Sub
</script>
```

Example 3.1—Sample global.asa

Application object

Stores values and objects, fires events, and provides locking, at the application level.

Application.Contents collection

Associative array that can be used to store simple values, keyed by name.

Usage

```
Application.Contents(name)= value ' set an application value
Application(name)= value ' same thing, since Contents is the default member of Application
Application(name) ' use the value
Application.Contents.Remove name ' remove the value
Application.Contents.RemoveAll ' remove all application values
```

Details

The `Contents` collection is an associative array that maps key strings to values, available to the entire application. Altering the `Application.Contents` collection usually occurs in the `Application_onStart` event in [], but can also be performed in an ASP page (though the application should be [] first).



The `Contents` collection can only store certain primitive values, like dates and times, numbers, strings, and some arrays. The collection cannot store more complex objects, such as COM objects (see []), JScript associative arrays and objects, PerlScript hashes and objects, etc.

Application.Contents Members

Remove (name) method (ASP 3.0)

Deletes the specified item from the collection.

RemoveAll () method (ASP 3.0)

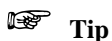
Deletes all items from the collection.

Application.Lock method

Obtains exclusive write access to the `Application` object.

Usage

```
Application.Lock
' ... modify application object ...
Application.Unlock
```



To avoid concurrency issues (such as a race condition [[http:// en. wikipedia. org/ wiki/ Race_ condition](http://en.wikipedia.org/wiki/Race_condition)]), always lock the application before altering the `Application.Contents` collection (except in the `Application_onStart` event handler in []).



All other ASP blocks in the entire application will wait while the application is locked.

Application.StaticObjects collection

Collection of application objects declared in the `global.asa` file.

Usage

```
Application.StaticObjects(name).member ' access the named object
name.member ' same thing, since static objects are imported into the global namespace
```

Details

Application-scope COM objects created via the `object` tag in the [] file are stored in the `StaticObjects` collection, in addition to being available to the global namespace.

Application.Unlock method

Releases exclusive write access to the `Application` object.

Usage

```
Application.Lock  
' ... modify application object ...  
Application.Unlock
```

Details

See [].

Session object

Stores persistent information for each user session.

Warning

Using the `Session` object requires that sessions be enabled for the application. Otherwise, an error will occur.

Warning

User sessions require the client to support and accept cookies in order to function properly. It is a good idea to configure your server to support P3P [<http://www.w3.org/P3P/>], in order to support newer browsers with more paranoid privacy settings.

Session.Abandon method

Closes the session, freeing up all associated resources.

Usage

`Session.Abandon`

Details

Tip

Always abandon the session once you know the user has finished with the application. Providing a link to a simple logoff page is one way to accomplish this.

```
<%  
Session.Abandon  
Response.Redirect "/" ' send user to the site's home page  
>
```

Example 5.1.1—Logoff.asp

Session.CodePage property

Sets the code page (Windows character set) for a session.

Usage

`Session.CodePage= codepage`

Details

See [].

Session.Contents collection

Collection of variables stored in the `Session` object.

Usage

```
Session.Contents(name)= value ' set an session value  
Session(name)= value ' same thing, since Contents is the default member of Session  
Session(name) ' use the value  
Session.Contents.Remove name ' remove the value  
Session.Contents.RemoveAll ' remove all session values
```

Details

The `Contents` collection is an associative array that maps key strings to values, available to the entire application. This is similar to the `Application.Contents` collection, but do not require locking to change, since only a single client has access to a session.

 **Note**

The `Contents` collection can only store certain primitive values, like dates and times, numbers, strings, and some arrays. The collection cannot store more complex objects, such as COM objects (see []), JScript associative arrays and objects, PerlScript hashes and objects, etc.

Session.Contents Members

Remove(*name*) method (ASP 3.0)

Deletes the specified item from the collection.

RemoveAll() method (ASP 3.0)

Deletes all items from the collection.

Session.LCID property

Sets the locale ID for a session.

Usage

```
Session.LCID= localeid
```

Details

See [].

Session.SessionID property

Returns this session identifier.

Usage

```
sid= Session.SessionID
```

Details

The session identifier is the cookie value used to distinguish one session from another.

 **Note**

This property is really only useful for debugging, to see if a session ID changes unexpectedly.

Session.StaticObjects collection

Collection of session objects declared in the `global.asa` file.

Usage

```
Session.StaticObjects(name).member ' access the named object  
name.member ' same thing, since static objects are imported into the global namespace
```

Details

Session-scope COM objects created via the `object` tag in the [] file are stored in the `StaticObjects` collection, in addition to being available to the global namespace.

Session.Timeout property

Number of minutes of inactivity before this session times out.

Usage

```
Session.Timeout= mins
```

Details

The number of minutes before a session expires can be set in the IIS administration console; when IIS is installed, it is 20. This property allows you to set a different timeout for just the current session.



Tip

This can be a useful way of keeping a session open during an extremely slow process.

Server object

A collection of methods provided by the webserver.

Server.CreateObject method

Creates an instance of a COM server component.

Usage

```
Set obj= Server.CreateObject(progid)
```

Details

This method creates and returns a COM object specified by *progid*.

** Important**

Using `Server.CreateObject` rather than a language construct (like `CreateObject(progid)` in VBScript, `new ActiveXObject(progid)` in JScript, or `$Win32::OLE->new(progid)` in PerlScript) allows IIS to manage the object, by handling concurrency, termination, etc.

Server.Execute method


Runs another ASP file as if it were a subroutine in the current file.

Usage

```
Server.Execute aspfile
```

Details

This method executes the specified ASP file, as if it were part of the calling file: all application, session, server, request, and response properties and methods from the calling page are available to the called page.

 **Tip**

Server-side includes are always imported into an ASP, but using `Server.Execute` allows you to use only the modules you need.

Server.GetLastError method

Returns the last error as an `ASPErrors` object, for use in custom error pages.

Usage

```
Set err= Server.GetLastError
```

ASPErrors Properties

- `ASPCode`
- `Number`
- `Source`
- `Category`
- `File`
- `Line`
- `Column`
- `Description`
- `ASPDescription`

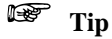
Server.HTMLEncode method

Escapes special HTML characters.

Usage

```
<%= Server.Encode(text) %>
```

Details



Use this method any time you are writing database fields or form values, to ensure that any special characters (< > & " ') are converted to HTML-safe (and XML-safe) character entities (< > & " ').

Server.MapPath method

Returns a full physical path, given a virtual or relative path.

Usage

```
path= Server.MapPath(virtualpath)
```

Details



Useful for producing a full path to a database file, XML config file, etc. with a relative (or absolute) virtual path.

Server.ScriptTimeout property

Sets the number of seconds a script is allowed to run.

Usage

```
Server.ScriptTimeout= seconds
```

Details



The timeout is actually the larger of this value and that in the Registry.

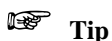
Server.Transfer method

Begins executing another ASP file, with all the variables and objects from the current file.

Usage

```
Server.Transfer aspfile
```

Details



Using `Server.Transfer` is more efficient than `Response.Redirect` (which requires another client request and server response).



This method is available only as of IIS 5.0 (Windows 2000).

Server.URLEncode method

Escapes special URL characters.

Usage

```
<%= Server.URLEncode(text) %>
```

Details



Tip

Use this method any time you are writing database fields or form values, to ensure that any special characters (like ? # etc.) are URL-escaped.

Request object

The client request.

Details

The `Request` object does not have a traditional default member. Instead, when accessing the object as a collection, it returns the first match found after searching its five member collections for the key, in this order:

1. []
2. []
3. []
4. []
5. []

Each of these collections contains a collection of [] items.



Tip

Using the `Request` collection allows a more concise and flexible access method for form variables; by using `Request("id")` rather than `Request.Form("id")`, you can pass the `id` variable via a POST form, or a simple link (from a search results page, for example) interchangeably.



Warning

It is best to avoid using the `Request` collection to access the `Request.ServerVariables` collection for two reasons: first, it is the last collection searched, which means the search will take longer; and second, an item from another collection with the same name will supercede the value, allowing exploits and causing bugs. In the following example, a malicious user could easily pass a query string of `?LOGON_USER=Administrator`.

```
If Request("LOGON_USER") = "" Then ' WRONG! Any Request collection could provide this value!  
    Response.Status= "401 Unauthorized"  
    Response.End  
End If
```

VBScript

Example 7.1—The Wrong Way To Check For Authentication

```
For i= 1 To Request("field").Count ' IStringList collections begin at 1  
    ' use value  
Next
```

VBScript

Example 7.2—Collecting Multiple Values From a Field

Request.BinaryRead method

Returns the requested number of binary bytes from those sent by the client.

Usage

```
data= Request.BinaryRead(bytes)
```

Details

This method is used when the client sends binary data to the server, for example, as a part of a file upload form input.



Note

The data returned by this method is pretty difficult to work with using VBScript or JScript; it needs to be broken apart, decoded, etc. Using a third-party component, like the `ASPFileForm` [<http://webcoder.info/downloads/aspfileform.html>], to read the fields from a file upload is advisable.

Request.ClientCertificates collection

Properties of the client certificate, when one is sent by the browser.

Usage

```
value= Request.ClientCertificates(name)
```

Details

When a client provides a certificate for authentication purposes, its fields are provided by this collection.

Request.Cookies collection

Potentially two-dimensional read-only collection of persistent client values.

Usage

```
value= Request.Cookies(name) ' get a one-dimensional persistent client value  
twodim= Request.Cookies(name).HasKeys ' check to see if the cookie is two-dimensional  
value= Request.Cookies(name)(key) ' get a two-dimensional persistent client value
```

Details

Any cookies passed to the client from the server are provided in this collection.

This collection contains a list of [] items.

Request.Form collection

Data passed to the server via a POST form.

Usage

```
value= Request.Form(name)
```

Details

All POST form values are gathered into this collection.

This collection contains a list of [] items.

```
For Each fld in Request.Form  
    msg= msg & fld & ": " & Request.Form(fld) & vbCrLf  
Next
```

VBScript

Example 7.4.1—Collecting Form Fields

Request.QueryString collection

Data passed to the server as a part of the URL, as with a GET form.

Usage

```
value= Request.QueryString(name)
```

Details

All GET form values, and any query string values, are available in this collection.

This collection contains a list of [] items.

Request.ServerVariables collection

A collection of standard CGI values.

Usage

```
value= Request.ServerVariables(name)
```

Details

The standard CGI variables are available in this collection.

Some Useful Server Variables

ALL_HTTP

A string containing all of the request headers. Check this too see what other variables are available on your server.

HTTP_VIA

The name of any proxy software between the client and the server.

HTTP_FORWARDED_FOR

The real address of a client using a proxy server.

LOGON_USER

The Windows username the client is using to access a non-anonymous address.

PATH_INFO

Extra (fake) path info, after the filename: /page.asp/more.



Note

PATH_INFO works incorrectly unless the Metabase key AllowPathInfoForScriptMapping is modified.

REMOTE_ADDR

The address of the client system.

SCRIPT_NAME

The virtual path of the currently running page.

URL

The address of the currently running page.

This collection contains a list of [] items.

Request.TotalBytes property

The size of the client request, used with **BinaryRead**.

Usage

```
size= Request.TotalBytes
```

Details

This property specifies the size, in bytes, of the request body sent by the client. It is primarily used to provide the number of bytes **Request.BinaryRead** should read.

Response object

The server response.

Response.AddHeader method

Adds a custom HTTP header.

Usage

```
Response.AddHeader name, value
```

Details

Used to send a custom HTTP header to the client, such as a P3P compact policy.



Note

This method must precede any data sent to the client, so put it at the top of your page. (Buffered pages may be able to add it later.)

Response.AppendToLog method

Appends text to the server log.

Usage

```
Response.AppendToLog text
```

Details



Note

This method can be called multiple times.

Response.BinaryWrite method

Sends binary data to the client.

Usage

```
Response.BinaryWrite data
```

Details

This method sends binary data, like an image, to the client.

Response.Buffer property

Enables a buffered response.

Usage

```
Response.Buffer= True ' enable buffering  
Response.Buffer= False ' disable buffering
```

Details

Buffering is enabled by default, starting with IIS5 (Windows 2000).



Note

This method must precede any data sent to the client, so put it at the top of your page.

Response.CacheControl property

Specifies an HTTP cache command.

Usage

```
Response.CacheControl= cachecmd
```

Details

This property allows you suggest whether a page should be cached or not.

Some Cache Control Values

public

The page may be cached.

private

The page should not be kept in a shared cached.

no-cache

The cache should always check for a newer version.

no-store

Do not cache at all.

Response.Clear method

Clears buffered response content.

Usage

```
Response.Clear
```

Details

Any buffered data is cleared, and never sent to the client.

Response.ContentType property

Specifies an HTTP MIME type.

Usage

```
Response.ContentType= mimetype
```

Details

Some content types a page can generate:

- text/xml
 - text/xsl
 - text/javascript
 - image/png
 - image/jpeg
 - image/gif
 - application/xhtml+xml
-

Response.Charset property

Adds the character set modifier to the HTTP MIME type.

Usage

```
Response.Charset= character-set
```

Details

See [].

Response.Cookies collection

Potentially two-dimensional write-only collection of persistent client values.

Usage

```
Response.Cookies(name)= value ' set a one-dimensional persistent client value
Response.Cookies(name)(key)= value ' set a two-dimensional persistent client value
Response.Cookies(name).Expires= value
Response.Cookies(name).Domain= value
Response.Cookies(name).Path= value
Response.Cookies(name).Secure= value
```

Details

Used to store information at the client, such as user preferences.

Response.End method

Flushes buffered content and ends script.

Usage

```
Response.End
```

Details



Warning

When using PerlScript, this method will not end the script.

Response.Expires property

Specifies a relative HTTP content expiration.

Usage

```
Response.Expires= mins
```

Details

Sets the HTTP expiration of the page, in minutes.

Response.ExpiresAbsolute property

Specifies an absolute HTTP content expiration.

Usage

```
Response.ExpiresAbsolute= datetime
```

Details

Sets a specific HTTP expiration for the page.

Response.Flush method

Flushes buffered content.

Usage

```
Response.Flush
```

Details

Sends all buffered data to the client.

Response.IsClientConnected property

True if the client is still waiting for a response, or false if not.

Usage

```
ccon= Response.IsClientConnected
```

Details

If a user has clicked the Stop button on their toolbar, or otherwise cancelled the page, this provides a way of halting processing, rather than continuing.

Response.PICS property

Specifies the RSAC PICS rating for the generated page.

Usage

```
Response.PICS= label
```

Details

This property is used to set the *Platform for Internet Content Selection* label, which can be obtained from the Internet Content Rating Association [<http://www.icra.org/>]. This information is used by some content filters to block objectionable material.

Response.Redirect method

Instructs the browser to request the given URL.

Usage

```
Response.Redirect url
```

Details

Sends a HTTP 302 redirect to the browser, causing the browser to request the given URL.



Tip

Using [] is more efficient than `Response.Redirect` (which requires another client request and server response).

Response.Status property

Sets the HTTP response status.

Usage

```
Response.Status= status
```

Details

This property sets the HTTP status code [<http://webcoder.info/reference/httpstatus.html>], which reports the success of the page to the client. If not specified, the status is 200 (500 if an error occurs).

401 `Unauthorized` is useful for forcing a logon when [] is empty.

Response.Write method

Sends text to the client.

Usage

```
Response.Write text
```

Details

This method sends text to the client.

IStringList collection

A list of values associated with a [Request](#) collection variable name.

Usage

```
id= Request("id") ' usually there is just one value
For i= 1 To Request("ids").count ' sometimes there can be several
    lookupid Request("ids").item(i) ' access the i-th value
    lookupid Request("ids")(i) ' alternate access method
Next
```

Members

Count property

Returns the number of items in the list.

Item (*index*) method

Returns the string from the requested position.

Details

[Request](#) collections contain the values of variables passed to the ASP page, usually via forms. These variables can contain multiple values, using an HTML select-multiple list, or just by giving multiple fields the same name. All values in these collections are returned as an [IStringList](#), an array of strings values. Using the default [IStringList](#) member returns the values separated by a comma and a space, or just the value if there is only a single list element. For example, for a query string of `?x=5&x=7&x=11`, `Request("x")` would return `5, 7, 11`, and `Request("x")(2)` would return `7`.

Internationalization

Provides formatting and alphabets for various cultures.

Locale

The locale determines the formatting for dates (field order and separator), times, currency (currency symbol and default displayed precision), and numbers (decimal separator and grouping separators). The first value found in this list is used as the ASP locale:

1. []
2. []
3. a nonzero `AspLCID` metabase value for the application
4. the server locale



Note

The VBScript locale exists independantly of the ASP locale. Since VBScript itself frequently does the formatting in ASP, be sure to call `SetLocale` when changing locale.

For a listing of locale identifiers, see the Locale ID Chart [<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vsmcLCID.asp>] .

CodePage and Charset

The character set is the method used to represent text electronically. You may notify the client which character set is in use by setting [], which appends `; charset=character-set` to the HTTP `Content-Type` header.

The code page, a numeric character set ID, controls how text is encoded as it is sent to the client. The first value found in this list is used as the code page:

1. []
2. []
3. the `AspCodePage` metabase value for the application
4. the server code page

Some Common Character Sets

Character Set	Name	Code Page
ASCII	us-ascii	20127
Western European (Windows)	windows-1252	1252
Unicode, 8-bit encoding	utf-8	65001
Unicode, 16-bit encoding	utf-16	1200

For a complete listing of character sets and code pages, see the Character Set chart [<http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/reference/charsets/charset4.asp>] .

Index

A

Active Server Pages 1

S

Server Side Include 1

T

thread-safety 5

R

race condition 7

P

Platform for Internet Content Selection 24