What is ASP?

Since its introduction, the use of Microsoft's **Active Server Pages**, or ASP, has grown rapidly. Many programmers consider it *the* tool for dynamic, easily maintainable web content. The real power of ASP derives firstly from the fact that the HTML for the page is only generated when the specific page is requested by the user, and secondly from the fact that it is browser-independent, since what is sent to the browser is usually purely HTML (although it can also include client-side code), rather than relying on the browser to support a particular language or application.

ASP enables us to tailor our web pages to the specific requirements of our users and their browser type, as well as our own needs. It allows us to interact with the user, which helps to keep our site interesting and up-to-date. Although it is not the first technology to offer dynamic page creation, it is one of the fastest and most powerful. It is indicative of the impact that ASP has made that it has now got its own imitators.

This book is for primarily intended for developers who need a reference to the many ways in which ASP contributes to the running of web sites and the resources we can utilize to achieve that goal. In order to get the maximum benefit from this book you will need to have some understanding of both ASP and the Web in general. Some knowledge of a scripting language such as VBScript or JScript is assumed.

This is the book for you if you need easy access to the methods and properties which the different components and objects expose, you are looking to expand the scope and functionality of your site, or you are fairly new to ASP and need an overview of what it has to offer. We hope that you benefit from and enjoy this book.

Firstly in this chapter we explore where ASP is derived from, look at some of the essential building blocks of ASP and briefly explore what is new to ASP 3.0. We look at:

- The origins of ASP
- ASP, HTTP, HTML and IIS
- Managing state on the Web
- The role of `global.asa`

- ASP directives
- Virtual applications
- What's new in ASP version 3.0
- What's new in JScript 5.0
- What's new in VBScript 5.0

The Origins of ASP

At the same time that the huge business potential of the Web was being realised, the limitations of HTML and HTTP were also becoming very apparent to developers. The static, stateless nature of HTML pages means that, although they are great for 'on-line brochure' web sites, they do not meet the specific needs and requirements of fast-moving business, building customer loyalty and selling goods and services. Various technologies, including ASP, grew out of the need to create pages with content specific to an individual user.

A simple type of customer interaction is processing information entered by a user on an HTML form. This normally involves either a user searching for information, or alternatively the customer entering personal information that needs to be stored by a business for further processing. In either case, we probably want to communicate with a database, which cannot be done purely using HTML. The initial solution to these, and other applications that are equally problematic with HTML and HTTP, involved reading the user input and programmatically creating a response. The interface which the server exposed to connect HTML and other applications became known as the Common Gateway Interface (CGI), and can be implemented in any language (the most popular being Perl). This approach, however, requires developers to have extensive programming knowledge and is restricted by the need to compile the code. Although Perl and CGI are still valuable tools, several alternatives are now also available, including ASP. This enables sections of script to be embedded in HTML pages. ASP-embedded code can contain logic which inserts content, formats data and carries out actions depending on decisions relating to how a page is requested.

ASP, HTTP, HTML and IIS

HyperText Transfer Protocol (HTTP) is the protocol that handles requests and responses sent between a web server and browser. The HTTP Request is the format of any message sent from the client to a server. It includes the URL of the required resource and information about the client and the platform they are using. The HTTP Response can contain a resource, a redirection to another page or site, an error message, etc.

ASP provides its own `Request` and `Response` objects, which enable us to access the information stored in the HTTP Request message and Response headers respectively. Using these objects we can check for certificates, read and write cookies, and get access to browser information and forms data. We can insert data into the body of the page to be sent to the client, redirect the browser, check if the client is connected, and manage the sending of content so that the client does not wait for too long for long sections of content.

The relationship between ASP and HTML can be described as follows:

Active Server Pages is a technology that allows for the programmatic construction of HTML pages for delivery to the browser.

In other words, with ASP we can write a set of instructions that can be used to generate HTML and other content just before it is delivered. This makes it a good tool for HTML developers, because of its power and flexibility to generate fresher HTML, and ultimately produce more spectacular, interactive, personalized and up-to-date web sites.

But what actually *is* ASP? It's not a conventional programming language in the sense that Pascal and C++ are, although it does make use of existing scripting languages such as VBScript or JScript. It's also not an application in the sense that FrontPage and Word are. The best way to think of ASP is as a technology for building dynamic and interactive web pages.

An alternative way to create dynamic pages is to use client-side scripting. This must be written in a language interpreted by the client browser and hence code generally consists of sections of Javascript embedded in an HTML page. This can programmatically control the layout of the page, how the page reacts to user actions and what is shown on the page. This is all useful but has its limitations. Typical uses for client-side code are, for example, to respond to user actions like clicking their mouse on the page, passing it over certain hotspots, and also checking forms prior to sending them.

Client-side code depends on the browser supporting the scripting language, and can fall over if the language is not supported, or includes code which differs between implementations and language versions. A second limitation is that the code is accessible to the user, which makes it unsuitable for passing, for example, passwords and connection strings.

The alternative then, and this is what ASP relies on, is to include scripts which are processed by the server. These server-side scripts do not depend on the browser or the user's platform executing, as the result returned to the browser is typically in plain HTML (or text, XML etc). However, server-side script is often used in conjunction with client-side code – there's no reason why an ASP page can't contain `<SCRIPT>` sections.

IIS (previously called Internet Information Server, but renamed with IIS5 to **Internet Information Services**) was Microsoft's answer to dynamic page creation by servers. Originally IIS 1.0 consisted of a fairly standard setup with CGI support and an interface to allow more efficient execution of compiled applications written in languages like C and C++. It provided additional features to access the input and output streams. This interface is called the **Internet Server Application Programming Interface**, or **ISAPI**.

The ASP scripting engine still uses ISAPI to connect to IIS5. It runs in-process with the server. This means that it shares the same memory space with the server and can get direct access to values in that memory. This does mean that if the application fails it can cause the server to fail, but makes for a very efficient and fast process and generally gives ASP the edge over other technologies.

What does ASP Code Look Like?

When a web author writes an ASP page, it is likely to be composed of a combination of three types of syntax – some ASP, some HTML tags, and some pure text. The following table summarizes these ingredients, their purpose, and their appearance:

Type	Purpose	Interpreter	Hallmarks
Text	This is hard-coded information to be shown to the user	The viewer's browser shows the text	Simple ASCII text.
HTML tags	This consists of instructions to the browser about how to format text and display images	The viewer's browser interprets the tags to format the text	Each tag within `<  >` delimiters. Most HTML tags come in pairs (an open tag and a close tag), e.g. `<TABLE>`, `</TABLE>`.
ASP statements	This consists of instructions to the web server running ASP about how to create portions of the page to be sent out	The web server's DLL `asp.dll` performs the ASP commands	Each ASP section contained within `<%` `%>` delimiters. ASP statements support features such as variables, decision trees, cyclical repetitions etc.

The file containing these constituent parts of the ASP page is saved with an `.asp` extension.

It's not too hard to distinguish the different elements of the ASP page. Anything that falls between the `<%` and `%>` markers is **ASP script**, and will be processed on the web server by the ASP script engine.

Lets take a look at an example and at the same time demonstrate one of the keys to ASP's success – how easy it is to get started. For example, consider the few lines of code below:

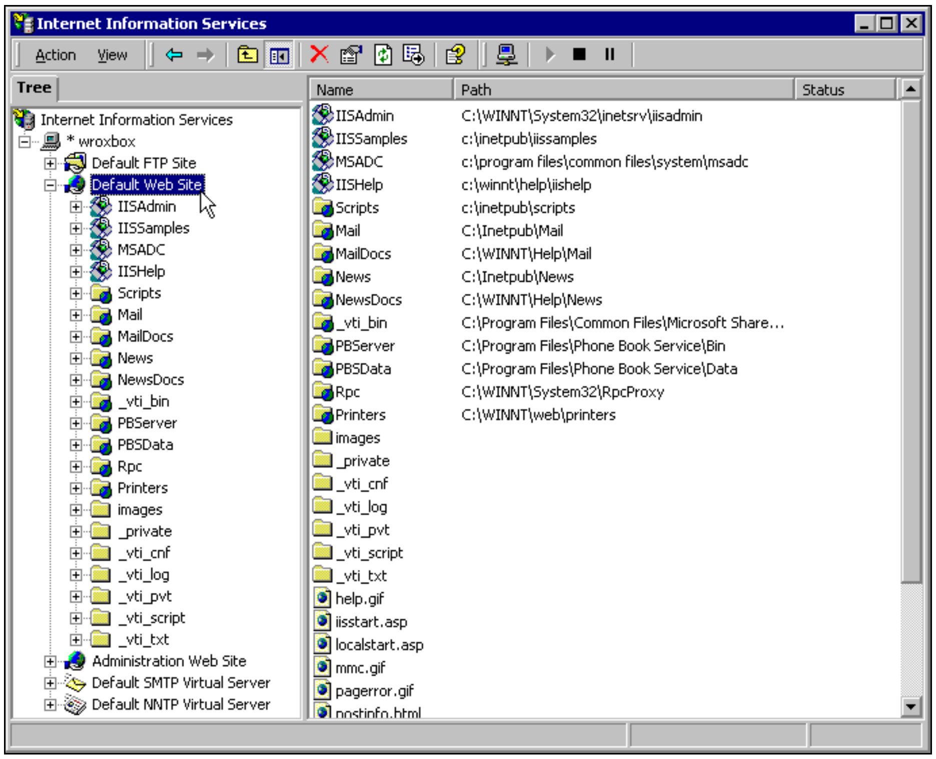
```
<HTML>
<P>This date/time is now : <%= Now() %> </P>
</HTML>
```

The content of the resulting web page depends on the HTML that is generated by the ASP code. In this particular example, the effect of the script code is to generate HTML for the time and date that the page is requested, and then to make a decision (based on the situation) on what text will be sent to the browser as part of the HTML stream.

How does ASP Work

ASP works with a single DLL called `asp.dll` (or alternatively the ASP scripting engine). This is installed by default into your `WinNT\System32\Inetsrv` directory. This DLL is responsible for taking an ASP page (indicated by the `.asp` file extension) and parsing it for any server-side script content. The script is passed to the appropriate scripting engine, to interpret for example the VBScript or JScript. The results of executing the script are combined with any text and HTML in the ASP page and the completed page is then sent back the client browser via the web server.

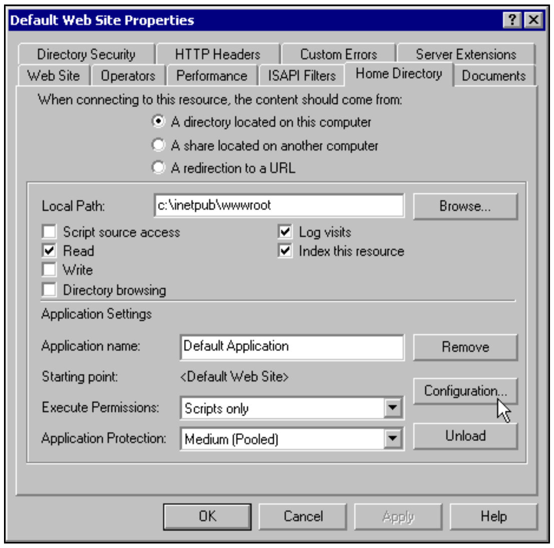
To see this, open the Internet Services Manager from the Administrative Tools section of your Start menu (for Windows 2000 Server version – or go to it via Administrative Tools in your Control Panel with Windows 2000 Professional). This runs the Microsoft Management Console (MMC) to display the entire Internet Information Services tree for IIS, which looks something like this:



```
Internet Information Services
  * wroxbox
    Default FTP Site
    Default Web Site
      IISAdmin
      IISSamples
      MSADC
      IISHelp
      Scripts
      Mail
      MailDocs
      News
      NewsDocs
      _vti_bin
      PBServer
      PBSData
      Rpc
      Printers
      images
      _private
      _vti_cnf
      _vti_log
      _vti_pvt
      _vti_script
      _vti_txt
    Administration Web Site
    Default SMTP Virtual Server
    Default NNTP Virtual Server
```

Name	Path	Status
IISAdmin	C:\WINNT\System32\inetsrv\iisadmin	
IISSamples	c:\inetpub\iissamples	
MSADC	c:\program files\common files\system\msadc	
IISHelp	c:\winnt\help\iishelp	
Scripts	c:\inetpub\scripts	
Mail	C:\Inetpub\Mail	
MailDocs	C:\WINNT\Help\Mail	
News	C:\Inetpub\News	
NewsDocs	C:\WINNT\Help\News	
_vti_bin	C:\Program Files\Common Files\Microsoft Share...	
PBServer	C:\Program Files\Phone Book Service\Bin	
PBSData	C:\Program Files\Phone Book Service\Data	
Rpc	C:\WINNT\System32\RpcProxy	
Printers	C:\WINNT\web\printers	
images		
_private		
_vti_cnf		
_vti_log		
_vti_pvt		
_vti_script		
_vti_txt		
help.gif		
iisstart.asp		
localstart.asp		
mmc.gif		
pagerror.gif		
postinfo.html		

Right-click on the Default Web Site entry and select Properties, then the Home Directory page:



Default Web Site Properties

When connecting to this resource, the content should come from:

- [✓ A directory located on this computer]
- A share located on another computer
- A redirection to a URL

Local Path: c:\inetpub\wwwroot

- ☐ Script source access
- ☑ Read
- ☐ Write
- ☐ Directory browsing
- ☑ Log visits
- ☑ Index this resource

Application Settings

Application name: Default Application

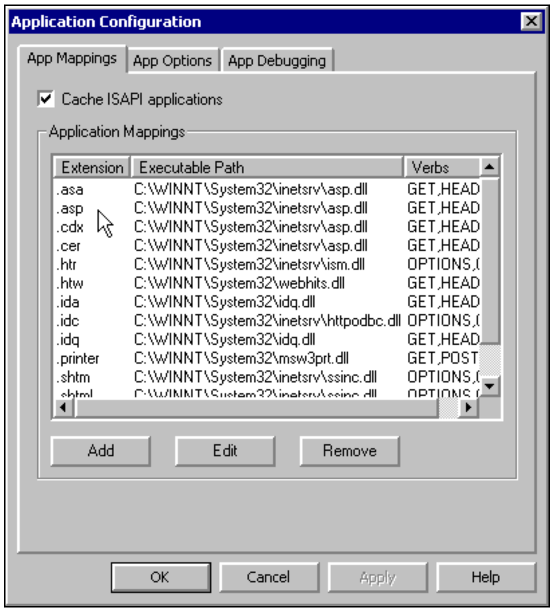
Starting point: <Default Web Site>

Execute Permissions: Scripts only

Application Protection: Medium (Pooled)

In the lower half of the page there is a name for the application and the Execute Permissions and Application Protection settings.

Then, click the Configuration button to open the Application Configuration dialog. In the App Mappings tab, you can see the way that IIS links each type of file (using the file extension) with a specific DLL:



Application Configuration

☑ Cache ISAPI applications

Application Mappings

Extension	Executable Path	Verbs
.asa	C:\WINNT\System32\inetsrv\asp.dll	GET,HEAD
.asp	C:\WINNT\System32\inetsrv\asp.dll	GET,HEAD
.cdx	C:\WINNT\System32\inetsrv\asp.dll	GET,HEAD
.cer	C:\WINNT\System32\inetsrv\asp.dll	GET,HEAD
.htr	C:\WINNT\System32\inetsrv\ism.dll	OPTIONS,(
.htw	C:\WINNT\System32\webhits.dll	GET,HEAD
.ida	C:\WINNT\System32\idq.dll	GET,HEAD
.idc	C:\WINNT\System32\inetsrv\httpodbc.dll	OPTIONS,(
.idq	C:\WINNT\System32\idq.dll	GET,HEAD
.printer	C:\WINNT\System32\msw3prt.dll	GET,POST
.shtm	C:\WINNT\System32\inetsrv\ssinc.dll	OPTIONS,(
.shtml	C:\WINNT\System32\inetsrv\ssinc.dll	OPTIONS,(

Any pages that have the `.asp` file extension are sent to the `asp.dll` for processing; you can see our `global.asa` page (which we discuss later in this chapter) is also mapped to the asp DLL, as well as several others. Pages with file extensions that are *not* mapped to a DLL, for example `.html` and `.htm` for HTML pages, and `.xml` for XML files, are simply loaded from disk and sent directly to the client.

You might like to have a look at the other file types in this page. Pages with `.ida`*,* `.idc`*, and* `.idq` *file extensions are sent to the DLL* `httpodbc.dll` *for processing. As you can guess from its name, it uses ODBC (discussed later in this book) to execute a SQL statement that returns a set of records for inclusion in the page. Likewise, the* `.shtm`*,* `.shtml` *and* `.stm` *file extensions are mapped to a DLL named* `ssinc.dll`*. These file types are traditionally used for files that require* ***server-side include*** *(**SSI**) processing.*

While you have the Application Configuration and Properties dialogs open, you might want to briefly explore it (just don't change any settings for the moment, unless you're sure that you know what you're doing, the defaults usually suffice!).

Processing an ASP File

When `asp.dll` receives an ASP page, it converts it to an output suitable for the server to send to the client. It deals with any script marked for its attention, evaluating it, and sending the result on to the server. It does this by first checking the page to see if contains any ASP code. If it does not find any, it informs IIS to send the page to the client. A new feature of Windows 2000 means that this is done with no marked performance penalty.

When ASP receives a page that does contain server-side scripting, it parses it line by line and each section of script is passed to the approproate scripting engine for compiling and execution. The result of this is inserted into any content which does not require server intervention by ASP and the whole is passed on to the client.

To make this more efficient, ASP caches the compiled code so that it does not need to be compiled again unless the source is changed. The result of this is that subsequent requests for the specific page are returned more quickly as the compilation stage is bypassed.

Including Separate Script Files

An additional feature is the ability to include separate files that contain script code. This allows us to write generic functions and both encourages and greatly simplifies code re-use. It also allows us to encapsulate processes which depend on the setup of our system and our server, and so may change.

Scripting Performance Issues

Web servers generally have plenty of spare processor cycles available (except on the busiest of sites) because the main task they have is loading pages from disk and sending them to the client. Therefore, each page request results in the processor waiting for the disk to respond. These spare cycles mean that ASP scripts can usually be executed with very little overall hit on performance. To add to this, as most page requests will be for pages where a compiled version of the script code is available, only the execution of this script needs to take place.

Of course, as the number of requests, and hence the server load, increases, the effect of having to parse and execute each ASP page takes its toll. It's wise, therefore, to squeeze as much performance as possible from the ASP interpreter. Here are some useful tips:

Avoid Mixing Scripting Languages on the Same Page

Using both scripting languages on any one page has an effect on the execution order of the code. This will mean both scripting engines will be loaded by the ASP DLL, increasing memory usage and the time taken to process the request.

Avoid Excessive Context Switching Between Script and Other Content

Having sections of ASP interspersed within other content can have a significant effect on the time it takes to process the page request. Every time a section of script ends, control is passed back to IIS (and vice versa) and this will have an impact on performance. An alternative to this is to use the `Response.Write` method (rather than using `<% = ...%>`) and this is recommended for any but small sections of code.

Build a Separate Component

For any complex processes, consider building a separate componant to install on the server. This will be far more efficient than instantiating and interpreting ASP script code. This will become even more important with the next version of ASP (currently called ASP+) in which almost everything is done using COM+.

Managing State on the Web

In a normal single-user program, such as when we build an executable application (an `.exe` file for example) using C++, Delphi etc., we take for granted the fact that we can declare a global (or `Public`) variable and then access it from anywhere in our code. All the time the application is running, the value remains valid and accessible.

The ability to hold values in memory, and relate specific values to specific users, provides **state**. You can think of it as representing the values and context of the applications' and users' internal variables throughout the life of the application.

When we create web-based **applications**, we often need be able to provide an individual *state* for each user. This might be as simple as remembering their name, or as complex as storing object references or recordsets that are different for each user. If we can't do that, we can't reasonably expect to do anything that requires more than *one* ASP page, as the variables and other references in that page are all destroyed when the page is finished executing. When the user requests the next page, we've lost all the information that they've already provided.

It is also useful to be able to store values that are global to *all* users. An obvious example is a web-style visitor counter. There's not much point in giving each user their own counter, because they usually want to see the total number of visitors, not just the number of times that they have visited. The number of visitors needs to be stored with **application-level state**, rather than **user-level state**.

With ASP we need a way of storing state information, otherwise variables and references within a page are destroyed when that page has finished execution. One of the ways of providing state between page requests and site visits is through **cookies**, which are sent along with each page request to the domain for which the cookie is valid. ASP uses a cookie to provide the concept of a user **session**, which we interact with through the ASP `Session` object.

A new and separate `Session` object is created for each individual visitor when they first access an ASP page on the server. A session identifier number is allocated to the session, and a cookie containing a specially encrypted version of the session identifier is sent to the client. The `Path` of the cookie (see the previous chapter for a description of cookie properties) is set to the path of the root of the ASP application running on our server. This will be the root of the Default Web site (i.e. "`/`") or the root directory of the ASP application containing the page they request. No `Expires` value is provided in the cookie, so it will expire when the browser is closed.

Every time that this user accesses an ASP page, ASP looks for this cookie, named `ASPSESSIONID`*xxxxxxxx*, where each *x* is an alphabetic character. If found, it can be used to connect the visitor with their current `Session`, which is held in memory on the server.

This cookie doesn't appear in the `Request.Cookies` or `Response.Cookies` collections. ASP hides it from us, but it's still there on the browser and ASP looks for it with each ASP page request.

If the client browser doesn't accept or support cookies, each new page request forces a new session to be created, thus state cannot be maintained without cookies.

As a warning you might note that if the browser does not support, or is set to reject, cookies this function will not be available. What do we do then? An alternative is to have server-side cookies. The information is stored on the server and matched to each user. We can then persistently store information on that user for a length of time determined by ourselves. This just leaves matching the user to their information and this can be done with the use of logins. Each time that a user enters the domain of our server they can be asked to log in with their name and password. We can then make our site available to them according to their specific needs and preferences.

In addition we can restrict access to resources so that we can expose varying levels of access to the general public, clients of the company or subscribers, staff or site administrators. Now we are starting to have a level of control over our site which in any other application we might take for granted.

The Role of global.asa

All ASP applications can contain a file named `global.asa`, placed in the root directory of the application and which applies to all sub-directories of this. The `global.asa` file in the root directory of the entire web site (`Inetpub\wwwroot`) defines the whole site as being part of the **Default ASP application**.

The `global.asa` file can contain code that instantiates objects and creates and sets the values of variables that will be available in either **Application-level** or **Session-level** scope. Object instances can be created using the `Server.CreateObject` method or an `<OBJECT>` element.

Creating Object Instances

If an `<OBJECT>` element is used, the `SCOPE` attribute can be set to `"Application"` or `"Session"`, and the object is then created in the appropriate context:

```
<!-- Declare ASPCounter component with application-level scope -->
<OBJECT ID="ASPCounter" RUNAT="Server" SCOPE="Application"
        PROGID="MSWC.Counters">
</OBJECT>

<!-- Declare ASPContentLink component with session-level scope -->
<OBJECT ID="ASPContentLink" RUNAT="Server" SCOPE="Session"
        PROGID="MSWC.NextLink">
</OBJECT>
...
```

The remainder of the `global.asa` file can contain ASP script that defines event handlers that run when the application or a user session starts or ends. Using VBScript this looks like:

```
...
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">

Sub Application_OnStart()
   'Code here is executed when the application starts
End Sub

Sub Application_OnEnd()
   'Code here is executed when the application ends
End Sub

Sub Session_OnStart()
   'Code here is executed when a user session starts
End Sub

Sub Session_OnEnd()
   'Code here is executed when a user session ends
End Sub

</SCRIPT>
```

Or using JScript:

```
...
<SCRIPT LANGUAGE=JScript RUNAT=Server>

function Application_OnStart() {
   // Code here is executed when the application starts
}

function Application_OnEnd() {
   // Code here is executed when the application ends
}

function Session_OnStart() {
   // Code here is executed when a user session starts
}

function Session_OnEnd() {
   // Code here is executed when a user session ends
}

</SCRIPT>
```

Within the `OnStart` event handlers, script code can be used to instantiate objects with the `Server.CreateObject` method. This code creates an instance of the Ad Rotator component (see Chapter 15):

```
Set Session("ASPAdRotator") = Server.CreateObject("MSWC.AdRotator")
```

If placed in the `Application_OnStart` event handler, the object will have application-level scope. If placed in the `Session_OnStart` event handler, the object will have session-level scope, i.e. each visitor will have a separate instance of the object. The `Server.CreateObject` method is covered in detail in Chapter 7.

Referencing Object Type Libraries

Many objects and components provide enumerated and other constants which the various methods take as their paramters. This allows us to use the constant's name in place of its value. These constants can be referenced through the `METADATA` directive. Using the `METADATA` directive we can specify a type-library which ASP will then load when the page is executed. The syntax for this is shown below:

```
<!-- METADATA TYPE="TypeLib"
              FILE="path_and_name_of_file" | UUID="type_library_uuid"
              [VERSION="major_version_number.minor_version_number"]
              LCID="locale_id" -->
```

where:

- ❑ `path_and_name_of_file` is the absolute physical path to a type library file (`.tlb`) or ActiveX DLL. If this is not provided the `type_library_uuid` must be specified.
- ❑ `type_library_uuid` is the unique identifier for the type library. Either this or the `path_and_name_of_file` parameter must be provided.
- ❑ `major_version_number.minor_version_number` (optional) defines the version of the component required. If this version is not found the most recent version is used.
- ❑ `locale_id` (optional) is the locale identifier to be used. If a type library with this locale is not found the default locale for the machine (defined during setup) will be used.

For example, this code makes the intrinsic ADO pre-defined constants available in an ASP page:

```
<!-- METADATA TYPE="TypeLib"
              FILE="c:\Program Files\Common Files\System\ado\msado15.dll"
-->
```

In order to maintain backward compatibility the file name `msado15.dll` is used for later (i.e. ADO 2.5) versions of the ADO component.

If ASP is unable to load the type library, it will return an error and halt execution of the page. The possible error values are:

Error	Description
`ASP 0222`	'Invalid type library specification'.
`ASP 0223`	'Type library not found'.
`ASP 0224`	'Type library cannot be loaded'.
`ASP 0225`	'Type library cannot be wrapped' (i.e. ASP cannot create a type library wrapper object from the type library specified).

Web Applications

We have used the term web application a number of times rather loosely, to indicate something that isn't really a web site, but isn't a 'traditional' application (an `.exe` file for example) either. We can think of a web application as a set of web pages and other resources, such as COM+ objects, that are designed to carry out some task.

> *A COM object is an instance of a COM component, which should be thought of as a compiled piece of code that can provide a service to the system, not just a single application. (COM objects are discussed further in Chapter 2, ASP, Windows 2000 and Windows DNA.)*

When IIS and ASP are installed in Windows 2000, a Default Web Site is created. This is configured as an ASP application, which involves several settings in the Properties dialog, that we looked at earlier. The `global.asa` is used to determine the way that this default application behaves.

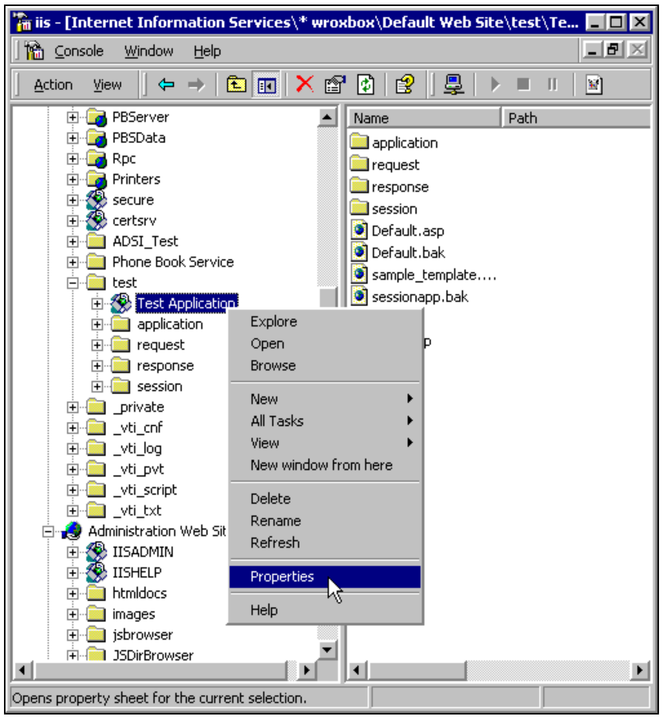
Virtual Applications

In addition to the default web application, ASP virtual applications can be created in any subdirectory of the web site. All sub-directories will then be part of this virtual application. Now, because the directory is itself within the default application for the site, this means that it will share the global space created by the default `Application` object. Any variables stored in the default application are available within the application; however, if an ASP page in the virtual application overwites a global value, the original value is maintained for the root application. This offers some protection to the server and other applications running alongside.

Creating ASP Virtual Applications

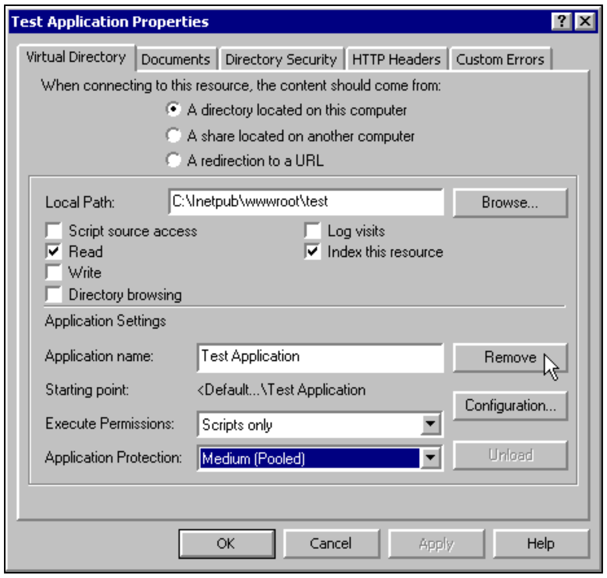
In Internet Services Manager, right-click on the directory in which you wish to create the new virtual application, select New, then Virtual Directory. This starts the New Virtual Directory Wizard, which steps through the settings required. This includes the name (or **alias**) for the new virtual application. Combined with the path of the directory selected in Internet Services Manager, this will become the URL of the application. To convert an existing directory into an application with the same name as the directory, select the directory containing the one you want to convert and use the directory name in the Virtual Directory Alias page of the wizard.

The wizard also allows you to specify the path that contains the content (pages) for the application. You can click Browse to select an existing directory. This is the directory that the new virtual application will point to. The final step allows you to select the access permissions, with the default being Read and Run Scripts. These settings can be changed later if required. The wizard then creates the new application, and marks it in Internet Services Manager with an 'open box' icon:



Opens property sheet for the current selection.

Right-click the new application and select Properties to see the settings that the wizard has chosen. The Local Path, access permissions, and Application Settings can be changed here if required. You'll also see a Remove button, which we can use to remove the virtual application:



When connecting to this resource, the content should come from:

- [✓ A directory located on this computer]
- A share located on another computer
- A redirection to a URL

Local Path: C:\Inetpub\wwwroot\test

- [ ] Script source access
- [✓] Read
- [ ] Write
- [ ] Directory browsing
- [ ] Log visits
- [✓] Index this resource

Application Settings

Application name: Test Application

Starting point: <Default...\Test Application

Execute Permissions: Scripts only

Application Protection: Medium (Pooled)

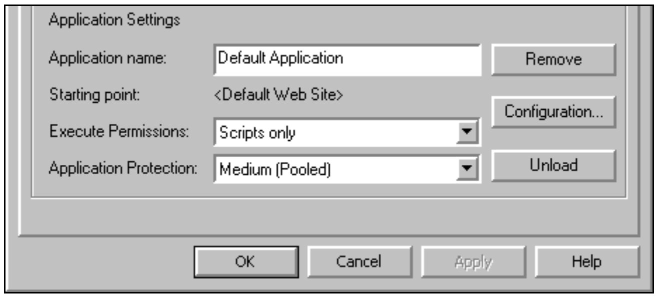
Clicking the Remove button doesn't actually remove the entry in Internet Services Manager. Instead, it converts the existing virtual application into a **virtual directory**. It will have a 'folder' icon with a 'globe' on it, indicating that this is a *redirection* to another folder on disk. It is accessed in the same way as the virtual application from which it was created (i.e. using the same URL), but does not act as an application. In other words, it doesn't support its own `Application` object but inherits the one for the default web site, or for another application within this directory's parent directories.

To delete a virtual application, select Delete from the right-click shortcut menu for the application in Internet Services Manager.

Virtual Application Configuration

Virtual applications provide control and management of objects and components that are instantiated in pages within that application's directories. The settings for a virtual application provide control over whether objects are created in the memory space of the web server, or separately in shared or individual out-of-process instances of `DLLHost.dll`.

The Properties dialog in Internet Services Manager provides these settings. At the bottom of the Home Directory page of the Properties dialog for a virtual application are two combo boxes marked Execute Permissions and Application Protection:


Application Settings
Application name: Default Application
Starting point: <Default Web Site>
Execute Permissions: Scripts only
Application Protection: Medium (Pooled)

Application Protection and Execution Settings

The Execute Permissions options are:

Execute Permission	Description
None	No scripts or executables can be run in this virtual application. In effect, this provides a quick and easy way to disable an application if required.
Scripts Only	Allows only script files, such as ASP, IDC or others to run in this virtual application. Executables cannot be run.
Scripts and Executables	Allows any script or executable to run within this virtual application.

While the Execute Permissions options control the type of execution that can take place in the virtual application, the Application Protection options affect the way that executables and components are run. The available options are:

Application Protection	**Description**
Low (IIS Process)	All application executables and components for ASP virtual applications with this setting are run in the process (i.e. the memory space) of the web server executable (`Inetinfo.exe`). Hence the web server is at risk if any one of the executables or components should fail. This provides the fastest and least resource-intensive application execution option.
Medium (Pooled)	(Default) All application executables and components from all ASP virtual applications with this setting are run in the process (i.e. the memory space) of a single shared instance of `DLLHost.exe`. This protects the web server executable (`Inetinfo.exe`) from the risk of any one of the executables or components failing. However, one failed executable or component can cause the `DLLHost.exe` process to fail, and with it all the other hosted executables and components.
High (Isolated)	All application executables and components for an ASP virtual application with this setting are run in the process (i.e. the memory space) of a single instance of `DLLHost.exe`, but each ASP application has its own instance of `DLLHost.exe` that is exclusive to that application. This protects the web server executable (`Inetinfo.exe`) from the risk of any one of the executables or components failing, and protects the virtual application from risk if an executable or component from another virtual application should fail. Microsoft suggests that a maximum of ten isolated virtual applications should be hosted on any one web server.

Microsoft recommends a configuration where mission-critical applications run in their own processes, i.e. **High (Isolated)**, and all remaining applications in a shared, pooled process, i.e. **Medium (Pooled)**.

Threading Issues and Object Scope

One other factor that affects the performance of instantiated objects and components is the threading model that they use. This also controls the scope in which they will perform successfully. There are five different threading models:

- **Single-threaded** components allow only one process to access the component at a time, and so each must wait in turn for the component to become available. Single-threaded components should **never** be used in ASP.

- **Apartment-threaded** components allow multiple instances of an object to be created, with each user getting their own instance. Object instances cannot be shared amongst processes, however. Apartment-threaded components are suitable for use in ASP with certain limitations as described in the table below.
- **Free-threaded** components allow multiple processes to access them concurrently, so a single instance can service more than one process. However, access is slower than with apartment-threaded objects as each access has to cross a process boundary. Free-threaded objects are suitable for use in ASP pages.
- **Both-threaded** components can act as though they are either apartment-threaded or free-threaded, depending on the context of the calling application. Both-threaded objects are suitable for use in ASP pages.
- **Neutral-threaded** components (new in Windows 2000 with COM+) allow multiple instances of the object to be created like apartment-threaded components. However, they do not limit each instance to always working in the same process, and so can be shared amongst requests. Neutral-threaded components are the best choice for ASP applications, although few tools are currently able to create this type of component.

This is only an overview of the different component types. Threading issues are covered further in Chapter 41, Optimizing ASP Performance. For an exhaustive technical discussion of threading issues see Beginning ASP Components from Wrox (ISBN 1-861002-88-2).

Object instances can be created in three different levels of scope:

- **Application-level** scope means that the object will be available to all pages within that virtual application (or the default web site). One instance of the object will service all requests from all users. For this reason, only both-threaded or neutral-threaded components should be used in application-level scope. However, if possible, avoid using any components at application-level scope at all as this always risks becoming a performance limitation.
- **Session-level** scope means that one object instance will service all requests from a single user within their ASP session. Both-threaded or neutral-threaded components work well at session-level scope, because they do not tie the session to a single process thread, as do apartment-threaded objects. Again, if possible, avoid using any components at session-level scope unless it is absolutely necessary.
- **Page-level** scope means that the object is created and destroyed within a single ASP page. While this seems to be inefficient, the COM+ Component Services within Windows 2000 are specially designed to make this fast and provide minimum use of resources. Objects can be pooled and/or recreated very quickly. With the exception of single-threaded components, any threading model is acceptable at page-level scope. However, apartment-threaded objects generally provide the best performance here.

ASP Directives

For each page that we put together we have several options which we can set which affect the way that the server processes it. A processing **directive** is always the first line of the ASP Page and is delimited by `<%@…%>`. The outside section you may recognize as the standard way of informing `asp.dll` that inside it there is content pertinent to it. The additional `@` sign denotes that it is the processing directive. This may contain all or any of the following keywords; if none are required this line can be omitted:

Processing Directives	**Description**
`CODEPAGE-"code-page"`	This defines the character set for this page. The code page is the numeric value of the character set. This value may differ between locales and languages to support both additional and alternative characters.
`ENABLESESSIONSTATE="True |False"`	This value can be set to `False` in which case no session cookie is set to the browser, this effectively disables sessions. The default is `True`. The main reason for doing this is to improve efficiency for pages that do not require state information.
`LANGUAGE="language-name"`	This sets the default language for the page. This does not preclude use of other languages within that page. The default language if this is not specified is VBScript, unless the default for the entire application has been changed.
`LCID="locale-identifier"`	An integer value which uniquely identifies the locale from which the page has been sent. This may affect such things as the currency symbol used.
`TRANSACTION="transaction _type"`	This directive specifies that the page will run under a transaction context. See Chapter 34, Transactions and Message Queuing.

What's New in ASP Version 3.0

If you're already familiar with ASP 2.0, and are looking for a concise list of what has actually changed in version 3.0, you'll find the information below:

Summary of New Features in ASP 3.0

These are the new, or substantially changed and improved, features which have been added to ASP in version 3.0. (Also see Chapter 2, ASP, Windows 2000 and Windows DNA for details of how Windows 2000 improves ASP 3.0.)

Scriptless ASP

ASP is now much faster at processing `.asp` pages that don't contain any script. If you are creating a site or Web application where the files may eventually use ASP, you can assign these files the `.asp` file extensions, regardless of whether they contain server–side script or only static (HTML and text) content.

New Flow Control Capabilities

A new feature to ASP 3.0 is an alternative to the `Response.Redirect` statement. Effectively this sent an instruction to the client browser to load an alternative page. Unfortunately, this is both error-prone and a slow process. In ASP 3.0, two new methods to the server object allow page transfers without browser intervention.

`Server.Transfer` transfers execution to another page, while `Server.Execute` will execute another page then return control to the original one. Inside the new page you can access the original page's context, including all the ASP objects like `Response` and `Request`, but you lose access to page scope variables. If the original page indicates that it is a transaction type in the processor directive (the opening `<%@...%>` element), the transaction context is passed to the new page. If this happens and the second ASP file's transaction flag indicates that transactions are supported or required, then an existing transaction will be used and a new transaction will not be started.

Error Handling and the New ASPError Object

Configurable error handling is now available, by providing a single custom ASP page that is automatically called if an error occurs with the `Server.Transfer` method. In that page, `Server.GetLastError` can be used to return an instance of the new `ASPError` object, which contains more details about the error including the error description and the relevant line number.

Encoded ASP Scripts

ASP script and client-side script can now be encoded using Base64 encryption, and higher levels of encryption are planned for future releases of ASP. (Note that this feature is implemented by the VBScript 5.0 and JScript 5.0 scripting engines, and so requires them.) Encoded scripts are decoded at run time by the script engine, so there's no need for a separate utility. Although not a secure encryption method, it does prevent casual users from browsing or copying scripts.

A New Way to Include Script Files

Rather than using the `<!-- #include ... -->` element to force IIS to server-side include a file containing script code, ASP 3.0 can do the 'including' itself. The `<SCRIPT>` element can be used with `RUNAT="SERVER"` and `SRC="`*`file_path_and_name`*`"` attributes to include files containing script code. The full and relative physical path or virtual path of the file can be used in the SRC attribute:

```
<SCRIPT LANGUAGE="language" RUNAT="SERVER" SRC="path_and_filename">
</SCRIPT>
```

Server Scriptlets

ASP 3.0 supports a powerful new scripting technology called **server scriptlets**. These are XML-format text files that are hosted on the server and become available to ASP as normal COM objects (i.e. Active Server Components). This makes it much easier to implement (or just prototype) your web application's business logic script procedures as reusable components, as well as using them in other COM-compliant programs.

Performance-Enhanced Active Server Components

Many of the Active Server Components that come with ASP have been improved to provide better performance or extra functionality. One example is the new Browser Capabilities component. In addition, there are some new components, such as the XML Parser that allows applications to handle XML formatted data on the server. Closer integration between ADO and XML is also provided (through the new version 2.5 of ADO that ships with Windows 2000), which opens up new opportunities for storing and retrieving data from a data store in XML format.

Performance

A great deal of work has been done to improve performance and scalability of ASP and IIS. This includes self-tuning features in ASP, which detect blocking situations and automatically increase the number of available process threads. ASP now senses when requests that are currently executing are blocked by external resources, and automatically provides more threads to simultaneously execute additional requests and to continue normal processing. If the CPU becomes overloaded, however, ASP reduces the number of available threads, to minimize the thread switching that occurs when too many non-blocking requests are executing simultaneously.

Changes from ASP Version 2.0

These are the features that have been changed or updated from version 2.0.

Buffering is On by Default

ASP has offered optional output buffering for some time. Since IIS 4.0, this has provided much faster script execution, as well as the ability to control the output that is streamed to the browser. In ASP 3.0, this improved performance has been reflected by changing the default setting of the `Response.Buffer` property to `True`, so that buffering is on by default. This means that the final output will be sent to the client only at the completion of processing, or when the script calls the `Response.Flush` or `Response.End` method.

Note that you should turn buffering off by setting the `Response.Buffer` *property to* `False` *when sending XML-formatted output to the client to allow the XML parser to start work on it as it is received. You may also want to use* `Response.Flush` *to send sections of very large pages, so that the user sees some output arrive quickly.*

Changes to Response.IsClientConnected

The `Response.IsClientConnected` property can now be read before any content is sent to the client. In ASP 2.0, this only returned accurate information after at least some content had been sent. This resolves the problem of IIS responding to every client request, even though the client might have moved to another page or site. Also, if the client is no longer connected after three seconds, the complete output that has been created on the server is dumped.

Query Strings with Default Documents

When a user accesses a site without providing the name of the page they require, the default document is sent back to them. However, if a query string is appended to that URL this is now passed to the default page. In previous versions this information was lost. For example, if the default page in a directory that has the URL `http://www.wrox.com/store/` is `default.asp`, then both the following will send the name/value pair `code=1274` to the `default.asp` page:

```
http://www.wrox.com/store/?code=1274
http://www.wrox.com/store/default.asp?code=1274
```

Server-side Include File Security

Server-side include files are often used for sensitive information, such as database connection strings or other access details. In ASP 2.0 specifying the virtual path of a file in a server side include to specify a file bypassed the security checking for the file. In other words the authenticated or anonymous account was not compared with the access control list entries for the file.

In ASP 3.0 on IIS 5.0, these credentials are now checked, and can be used to prevent unauthorized access.

Configurable Entries Moved to the Metabase

In IIS 5.0, the registry entries for `ProcessorThreadMax` and `ErrorsToNTLog` have been moved into the metabase. All configurable parameters for ASP can be modified in the metabase via Active Directory and the Active Directory Service Interface (ADSI).

Behavior of Both-Threaded Objects in Applications

For best performance in ASP, where there are often multiple concurrent requests, components should be **Both-Threaded** (Single Threaded Apartment (STA) and Multi-Threaded Apartment (MTA)) *and* support the COM Free-Threaded Marshaller (FTM). Both-Threaded COM objects that do not support the Free-Threaded Marshaller will fail if stored in the ASP `Application` state object.

Earlier Release of COM Objects

In IIS 5.0, instantiated objects or components are now released earlier. In IIS 4.0, COM objects were only released when ASP finished processing a page. In IIS 5.0, if a COM object does not use the `OnEndPage` method, and the reference count for the object reaches zero, then the object is released before processing completes.

COM Object Security

IIS uses the new **cloaking** feature provided by COM+ so that local server applications instantiated from ASP can run in the security context of the originating client. In previous versions, the security context assigned to the local server COM object depended on the identity of the user who created the instance.

Components Run Out-of-Process By Default

In earlier versions of ASP, all components created within the context of an ASP page ran **in-process** by default, i.e. within the memory space of the web server. In IIS 4.0, the ability to create a virtual application allowed components to be run **out-of-process**. In IIS 5.0 and ASP 3.0, components are now instantiated **out-of-process** by default. This is controlled by the metabase property `AspAllowOutOfProcComponents`, which now has a default value of `1`. Setting it to zero changes the default back to that of IIS 4.0.

To better fine-tune the component performance to Web server protection trade-off, you can now choose from the three options for Application Protection in the Properties dialog for a virtual application; see earlier in this chapter. The recommended configuration is to run mission-critical applications in their own processes – i.e. **High (Isolated) –** and all remaining applications in a shared, pooled process – i.e. **Medium (Pooled)**. It is also possible to set the **Execute Permissions** for the scripts and components that make up each virtual application. The three options are: None, Scripts only, or Scripts and Executables.

What's New in JScript 5.0

The only change to JScript is the long-awaited introduction of proper error handling.

Exception Handling

The Java-style `try` and `catch` constructs are now supported in JScript 5.0. For example:

```
function GetSomeKindOfIndexThingy() {

  try {
    // If an exception occurs during the execution of this
    // block of code, processing of this entire block will
    // be aborted and will resume with the first statement in its
    // associated catch block.
    var objSomething = Server.CreateObject("SomeComponent");
    var intIndex = objSomething.getSomeIndex();
    return intIndex;
  }

  catch (exception) {
    // This code will execute when *any* exception occurs during
    // the execution of this function
    alert('Oh dear, the object didn't expect you to do that');
  }

}
```

The built-in JScript `Error` object has three properties that define the last run-time error. We can use these in a `catch` block to get more information about the error:

```
alert(Error.number);  // Gives the numeric value of the error number
// AND the result with 0xFFFF to get a 'normal' error number in ASP

alert(Error.description);  // Gives an error desciption as a string
```

If you want to throw your own errors, you can raise an error (or **exception**) with a custom **exception object**. However there is no built-in exception object, so you have to define a constructor for one yourself:

```
// Define our own Exception object
function MyException(intNumber, strDescription, strInfo) {
  this.Number = intNumber;            // Set the Number property
  this.Description = strDescription;  // Set the Description property
  this.CustomInfo = strInfo;          // Set some 'information' property
}
```

An object like this can then be used to raise custom exceptions within our pages, by using the `throw` keyword and then examining the type of exception in the `catch` block:

```
function GetSomeKindOfIndexThingy() {
  try {
    var objSomething = Server.CreateObject("SomeComponent");
    var intIndex = objSomething.getSomeIndex();
    if (intIndex == 0) {
      // Create a new MyException object
      theException = new MyException(0x6F1, "Zero index not " +
                                  "permitted", "Index_Err");
      throw theException;
    }
    return intIndex;
  }

  catch (objException) {
    if (objException instanceof MyException) {
      // This is one of our custom exception objects
      if (objException.Category == "Index_Err") {
         alert('Index Error: ' + objException.Description);
      else
        alert('Undefined custom error:' + objException.Description);
      }
      else
        // Not "our" exception, display & raise to next high routine
        alert(Error.Description + ' (' + Error.Number + ')');
        throw exception;
    }
  }
}
```

What's New in VBScript 5.0

The features that are available in ASP include those provided by the scripting engines, which means that improvements there are also available in ASP. The changes to VBScript are as follows.

Using Classes in Script

The full Visual Basic `Class` model is implemented, with the obvious exception of events in ASP server-side scripting. You can create classes within your script, which make their properties and methods available to the remainder of the code in your page. For example:

```
Class MyClass

  'local variable to hold value of HalfValue
  Private m_HalfValue

  'executed to set the HalfValue property
  Public Property Let HalfValue(vData)
    If vData > 0 Then m_HalfValue = vData
  End Property

  'executed to return the HalfValue property
  Public Property Get HalfValue()
    HalfValue = m_HalfValue
  End Property

  'implements the GetResult method
  Public Function GetResult()
    GetResult = m_HalfValue * 2
  End Function

End Class

Set objThis = New MyClass

objThis.HalfValue = 21

Response.Write "Value of HalfValue property is " & _
               objThis.HalfValue & "<BR>"
Response.Write "Result of GetResult method is " & _
               objThis.GetResult & "<BR>"
...
```

This produces the result:

```
Value of HalfValue property is 21
Result of GetResult method is 42
```

The With Construct

The `With` construct is now supported, allowing more compact scripts to be written where the code accesses several properties or methods of one object:

```
...
Set objThis = Server.CreateObject("This.Object")

With objThis
  .Property1 = "This value"
  .Property2 = "Another value"
  TheResult = .SomeMethod
End With
...
```

String Evaluation

The `Eval` function (long available in JavaScript and JScript) is now supported in VBScript 5.0. This allows you to build a string containing script code that evaluates to `True` or `False`, and then execute it to obtain a result:

```
...
datYourBirthday = Request.Form("Birthday")
strScript = "datYourBirthday = Date()"

If Eval(strScript) Then
   Response.Write "Happy Birthday!"
Else
   Response.Write "Have a nice day!"
End If
...
```

Statement Execution

The new `Execute` function allows script code in a string to be executed in much the same way as the `Eval` function, but without returning a result as is usually the case with the `Eval` statement. It can be used to dynamically create procedures that are executed later in the code. For example:

```
...
strCheckBirthday = "Sub CheckBirthday(datYourBirthday)" & vbCrlf _
             & "  If Eval(datYourBirthday = Date()) Then" & vbCrlf _
             & "    Response.Write ""Happy Birthday!""" & vbCrlf _
             & "  Else" & vbCrlf _
             & "    Response.Write ""Have a nice day!""" & vbCrlf _
             & "  End If" & vbCrlf _
             & "End Sub" & vbCrlf
Execute strCheckBirthday
CheckBirthday(Date())
...
```

Either a carriage return (as shown) or a colon character '`:`' can be used to delimit the individual statements within the string.

Setting Locales

The new `SetLocale` method can be used to change the current locale of the script engine. This enables it to properly display special locale-specific characters, such as those with accents or from a different character set:

```
strCurrentLocale = GetLocale
SetLocale("en-gb")
```

Regular Expressions

VBScript 5.0 now supports regular expressions (again, long available in JavaScript, JScript and other languages). The `RegExp` object is used to create and execute a regular expression. For example:

```
strTarget = "test testing tested attest late start"
Set objRegExp = New RegExp      'create a regular expression

objRegExp.Pattern = "test*"     'set the search pattern
objRegExp.IgnoreCase = False    'set the case sensitivity
objRegExp.Global = True         'set the scope

Set colMatches = objRegExp.Execute(strTarget)  'execute the search

For Each Match in colMatches     'iterate the colMatches collection
  Response.Write "Match found at position " & Match.FirstIndex & "."
  Response.Write "Matched value is '" & Match.Value & "'.<BR>"
Next
```

This produces the result:

```
Match found at position 0. Matched value is 'test'.
Match found at position 5. Matched value is 'test'.
Match found at position 13. Matched value is 'test'.
Match found at position 22. Matched value is 'test'.
```

Setting Event Handlers in Client-side VBScript

While not applying directly to ASP scripting techniques, this new feature is useful when writing client-side VBScript. You can now assign a reference to a function or subroutine to an event dynamically. For example, given a function named `MyFunction()`, you can assign it to a button's `ONCLICK` event using:

```
Function MyFunction()
   ...
   'Function implementation code here
   ...
End Function
...
Set objCmdButton = document.all("cmdButton")
Set objCmdButton.onClick = GetRef("MyFunction")
```

This provides similar functionality to that existing in JavaScript and JScript, where functions can be assigned as properties of an object dynamically.

On Error Goto 0 in VBScript

Although this technique was not documented previously, it does in fact work in existing versions of VBScript (as those of you with a VB background and an inquisitive mind will have already discovered). It is now documented, and can be used to 'turn off' custom error handling in a page after an `On Error Resume Next` has been executed. The result is that any subsequent errors will raise a browser-level or server-level error and the appropriate dialog/response.

Other New Features

A couple of other features have been made available in IIS 5.0.

Distributed Authoring and Versioning (DAV)

This standard, created by the Internet Engineering Task Force (IETF) and now in version 1.0, allows authors in several locations to concurrently build and maintain Web pages and other documents. It is designed to provide upload and download access, and control versions so that the process can be properly managed. Internet Explorer contains features that integrate with DAV in IIS 5.0. However, in the IETF standard, and in the current release of IIS 5.0, the versioning capabilities are not yet implemented.

Referencing Type Libraries

In the past, it has been common practice to use a server-side include file to add constants from a type library (such as scripting objects, ADO, or MSMQ) to an ASP page. This is necessary, as ASP does not create a reference to the type library or component DLL as does, for example, Visual Basic. In IIS 5.0, you no longer need to use include files for constants. Instead, you can access the type library of a component directly using a new HTML comment-style element, placed in the `<HEAD>` section of the page:

```
<!-- METADATA TYPE="typelib" FILE="c:\WinNT\System32\scrrun.dll" -->
```

This makes all the constants in the specified file available within the current ASP page. (Although this is slated as being new in IIS 5.0, it was working but undocumented in IIS 4.)

FTP Download Restarts

The FTP service now (at last, some would say) provides a restart facility for downloads. If a file download stops part way through – perhaps because of a dropped connection at the user end – it can be resumed from that point. This means that failed file downloads do not require the client to download the entire file all over again.

HTTP Compression

IIS now automatically implements compression of the HTTP data stream for static and dynamically generated files, and caches compressed static files as well. This gives faster response and reduces network loading when communicating with suitably equipped clients.

New versions of the scripting engines, JScript 5.5 and VBScript 5.5 are currently available in beta. A description of the key new features of these can be downloaded from our web site.

Summary

In this chapter, we've very briefly looked at many of the major topics that you need to be aware of when working with ASP 3.0. We've purposely taken the point of view of the experienced web developer, assuming that either you have previous experience of ASP through using an earlier version, or at least you know how the Web works when clients and servers interact.

By now, you should have a good overall view of what ASP 3.0 offers – both in terms of the existing features in earlier versions, and the new features that are available in version 3.0. If you feel that you don't fully understand the concepts of the ASP object model, or the way that context is used to allow access to this object model from other sources, don't worry. Providing you have a broad understanding of the topics we've covered here, you will easily be able to follow the next chapter and the following section, which cover it all in more detail.