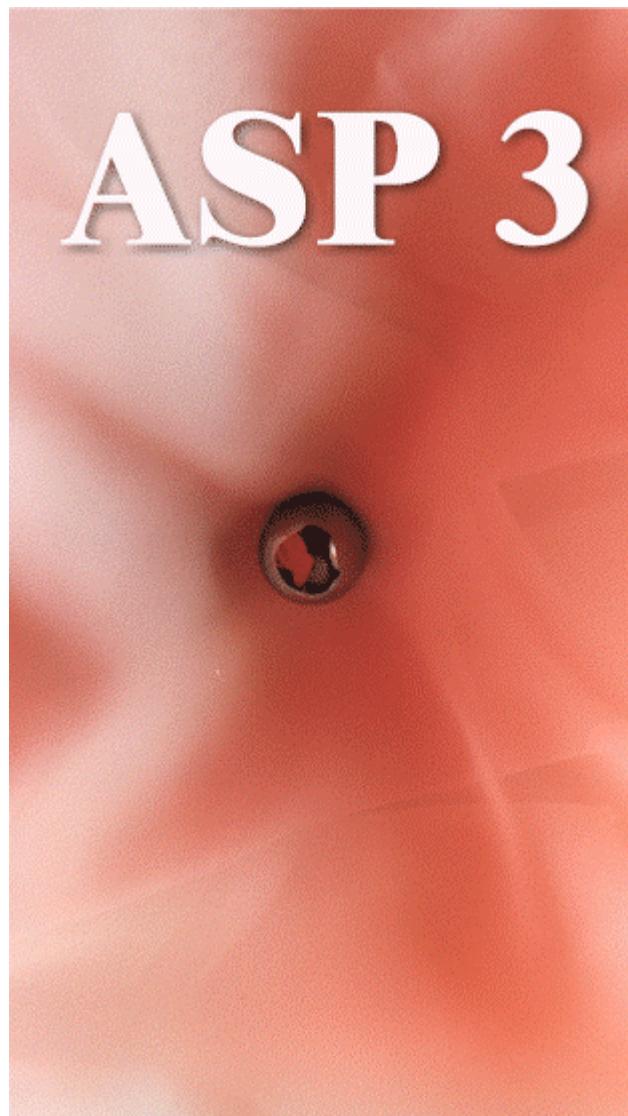


Texto diseñado para aquellos que deseen introducirse en el mundo del desarrollo de aplicaciones de negocio en Internet, utilizando para ello la tecnología ASP (Active Server Pages).

Se persigue adiestrar en el uso de: Internet Information Server 5, Visual InterDev como herramienta para la gestión de proyectos basados en ASP, Visual Basic Script, la jerarquía de objetos ASP, y las técnicas de acceso a datos a través de ADO.

Se requiere conocer los fundamentos de Internet/Intranet, estar familiarizado con la navegación por la web, conocer HTML y conocer el sistema operativo Windows a nivel de usuario.



# **PROGRAMACIÓN DE APLICACIONES PARA INTERNET CON ASP 3**



*Programación de aplicaciones para Internet con ASP 3*  
*Versión 1.0.0*  
2000 © Grupo EIDOS  
[www.LaLibreriaDigital.com](http://www.LaLibreriaDigital.com)



# Índice

<b>ÍNDICE.....</b>	<b>5</b>
<b>INTRODUCCIÓN A ASP, ACTIVE SERVER PAGES.....</b>	<b>11</b>
ANTECEDES DE ASP: LA ESPECIFICACIÓN CGI .....	11
DEFINICIÓN DE ASP .....	12
APLICACIONES ASP .....	13
APORTACIONES DE ASP.....	14
REQUERIMIENTOS DE ASP 3.0.....	15
SINTAXIS DE ASP.....	16
OBJETOS INTEGRADOS EN ASP 3.0.....	18
COMPONENTES DE SERVIDOR .....	19
VISIÓN GENERAL DE ASP .....	20
HOLA MUNDO CON ASP.....	21
<b>NOVEDADES DE ASP 3.0 .....</b>	<b>23</b>
¿PARA QUIÉN ES ESTE CAPÍTULO? .....	23
MEJORAS GENERALES EN ASP 3.0.....	23
EL OBJETO RESPONSE .....	24
EL OBJETO SERVER .....	25
EL OBJETO ASPERROR .....	28
COMPONENTE DE REGISTRO DE IIS (LOGGING UTILITY) .....	30
ACTIVEX DATA OBJECTS 2.5 (ADO 2.5) .....	34
APLICACIONES ASP CON IIS 5.0 .....	38
VBSCRIPT 5.0 .....	40
PÁGINAS ASP CODIFICADAS.....	42
OTROS CAMBIOS .....	42
<b>LENGUAJE DE SCRIPT: VBSCRIPT .....</b>	<b>45</b>
INTRODUCCIÓN .....	45

DIFERENCIAS ENTRE LOS SCRIPTS DE CLIENTE Y DE SERVIDOR .....	46
CARACTERÍSTICAS GENERALES DE VBSCRIPT .....	48
TIPOS DE DATOS DE VBSCRIPT .....	49
VARIABLES DE VBSCRIPT .....	50
CONSTANTES DE VBSCRIPT .....	52
OPERADORES DE VBSCRIPT .....	52
LITERALES DE VBSCRIPT .....	53
ESTRUCTURAS DE CONTROL EN VBSCRIPT .....	53
PROCEDIMIENTOS DE VBSCRIPT .....	58
TRATAMIENTO DE ERRORES EN VBSCRIPT .....	59
DIRECTIVAS DE PREPROCESAMIENTO .....	62
EXPRESIONES REGULARES EN VBSCRIPT .....	63
INTRODUCCIÓN A LA POO .....	67
<i>Programación Orientada a Objetos</i> .....	67
<i>Objetos</i> .....	67
<i>Clases</i> .....	68
CLASES Y OBJETOS EN VBSCRIPT .....	68
EVENTOS Y PROPIEDADES EN VBSCRIPT .....	70
OTRAS FUNCIONES EN VBSCRIPT .....	74
<i>Funciones para consultar variables</i> .....	74
<i>Funciones utilizadas para la manipulación de cadenas</i> .....	74
<i>Funciones matemáticas</i> .....	75
<i>Funciones para dar formatos</i> .....	75
<i>Funciones de fecha/hora</i> .....	75
<i>Funciones que devuelven información del motor de secuencias de comandos</i> .....	76
COMPONENTES DE VBSCRIPT .....	76
<b>PRIMEROS PASOS CON ASP 3.0 .....</b>	<b>79</b>
OBJETIVO DE ESTE TEMA .....	79
OBJETOS INTEGRADOS, MODELO DE OBJETOS DE ASP 3.0 .....	79
EQUIPO NECESARIO .....	82
EL SERVIDOR WEB .....	82
LA HERRAMIENTA DE DESARROLLO .....	84
<b>MODELO DE OBJETOS DE ASP: EL OBJETO RESPONSE .....</b>	<b>89</b>
DEFINICIÓN DEL OBJETO RESPONSE .....	89
COLECCIONES DEL OBJETO RESPONSE .....	89
PROPIEDADES DEL OBJETO RESPONSE .....	93
<i>Buffer</i> .....	94
<i>ContentType</i> .....	95
<i>Expires</i> .....	96
<i>Status</i> .....	96
<i>CacheControl</i> .....	97
<i>CharSet</i> .....	97
<i>PICS</i> .....	98
<i>IsClientConnected</i> .....	99
MÉTODOS DEL OBJETO RESPONSE .....	99
<i>Clear, Flush y End</i> .....	100
<i>Write</i> .....	101
<i>Redirect</i> .....	102
<i>AddHeader</i> .....	103
<i>AppendToLog</i> .....	103
<i>BynaryWrite</i> .....	104
<b>MODELO DE OBJETOS DE ASP: EL OBJETO REQUEST .....</b>	<b>105</b>

DEFINICIÓN DEL OBJETO REQUEST .....	105
COLECCIONES DEL OBJETO REQUEST .....	106
FORMULARIOS HTML .....	107
COLECCIONES DEL OBJETO REQUEST: FORM .....	108
COLECCIONES DEL OBJETO REQUEST: QUERYSTRING.....	111
COLECCIONES DEL OBJETO REQUEST: SERVER VARIABLES .....	112
COLECCIONES DEL OBJETO REQUEST: COOKIES .....	117
COLECCIONES DEL OBJETO REQUEST: CLIENTCERTIFICATE .....	120
MÉTODOS Y PROPIEDADES DEL OBJETO REQUEST.....	122
<b>MODELO DE OBJETOS DE ASP: EL OBJETO APPLICATION .....</b>	<b>123</b>
APLICACIONES ASP .....	123
DEFINICIÓN DEL OBJETO APPLICATION .....	127
COLECCIONES DEL OBJETO APPLICATION .....	129
MÉTODOS DEL OBJETO APPLICATION .....	132
EVENTOS DEL OBJETO APPLICATION .....	133
EL FICHERO GLOBAL.ASA .....	134
<b>MODELO DE OBJETOS DE ASP: EL OBJETO SESSION .....</b>	<b>139</b>
DEFINICIÓN DEL OBJETO SESSION .....	139
COLECCIONES DEL OBJETO SESSION.....	142
PROPIEDADES DEL OBJETO SESSION .....	144
MÉTODOS DEL OBJETO SESSION .....	147
EVENTOS DEL OBJETO SESSION .....	148
<b>MODELO DE OBJETOS DE ASP: EL OBJETO SERVER .....</b>	<b>151</b>
DEFINICIÓN DEL OBJETO SERVER .....	151
PROPIEDADES DEL OBJETO SERVER.....	151
MÉTODOS DEL OBJETO SERVER.....	153
<i>CreateObject</i> .....	153
<i>MapPath</i> .....	155
<i>HTMLEncode</i> .....	156
<i>URLEncode</i> .....	156
<i>URLPathEncode</i> .....	157
<i>Execute</i> .....	158
<i>Transfer</i> .....	160
<i>GetLastError</i> .....	161
<b>MODELO DE OBJETOS DE ASP: EL OBJETO OBJECTCONTEXT.....</b>	<b>165</b>
DEFINICIÓN DEL OBJETO OBJECTCONTEXT .....	165
MÉTODOS DEL OBJETO OBJECTCONTEXT .....	167
EVENTOS DEL OBJETO OBJECTCONTEXT .....	168
<b>MODELO DE OBJETOS DE ASP: EL OBJETO ASPERROR .....</b>	<b>171</b>
DEFINICIÓN DEL OBJETO ASPERROR .....	171
PROPIEDADES DEL OBJETO ASPERROR .....	172
TRATAMIENTO DE ERRORES CON EL OBJETO ASPERROR.....	172
<b>COMPONENTES DE VBSCRIPT .....</b>	<b>179</b>
COMPONENTES DE VBSCRIPT.....	179
EL OBJETO DICTIONARY .....	180
OBJETO FILESYSTEMOBJECT.....	184
<i>Métodos para trabajar con unidades de disco</i> .....	184
<i>Métodos para trabajar con carpetas</i> .....	185
<i>Métodos para trabajar con ficheros</i> .....	187

EL OBJETO DRIVE .....	189
EL OBJETO FOLDER.....	192
EL OBJETO FILE.....	195
OBJETO TEXTSTREAM.....	198
<b>COMPONENTES DE SERVIDOR .....</b>	<b>205</b>
INTRODUCCIÓN .....	205
COMPONENTE AD ROTATOR.....	207
COMPONENTE FUNCIONES DEL NAVEGADOR.....	211
COMPONENTE NEXTLINK.....	214
COMPONENTE CONTENT ROTATOR .....	217
COMPONENTE PAGECOUNTER .....	220
COMPONENTE COUNTERS .....	222
COMPONENTE MYINFO.....	224
COMPONENTE TOOLS.....	225
COMPONENTE PERMISSIONCHECKER .....	227
COMPONENTE STATUS .....	229
COMPONENTE DE REGISTRO DE IIS.....	230
COMPONENTES DE TERCERAS PARTES .....	235
<b>ASP E INTERNET INFORMATION SERVER 5.0 .....</b>	<b>241</b>
INTRODUCCIÓN .....	241
EL SERVIDOR WEB INTERNET INFORMATION SERVER 5.0 .....	242
INSTALANDO IIS 5.0 .....	242
NOVEDADES DE IIS 5.0 .....	244
EL ADMINISTRADOR DE SERVICIOS DE INTERNET.....	245
ELEMENTOS DE IIS 5.0.....	249
ADMINISTRACIÓN DEL SITIO WEB .....	250
<i>Sitio Web</i> .....	250
<i>Directorio particular</i> .....	252
<i>Documentos</i> .....	253
<i>Operadores</i> .....	254
<i>Errores personalizados</i> .....	255
<i>Rendimiento</i> .....	255
<i>Seguridad de directorios</i> .....	256
<i>Filtros ISAPI</i> .....	257
<i>Encabezados HTTP</i> .....	258
<i>Extensiones de servidor</i> .....	259
LA APLICACIÓN WEB.....	260
<b>CDONTS Y ASP .....</b>	<b>271</b>
INTRODUCCIÓN .....	271
MODELO DE OBJETOS DE CDONTS.....	273
EL OBJETO NEWMAIL.....	274
EL OBJETO SESSION .....	279
EL OBJETO FOLDER.....	282
EL OBJETO MESSAGE .....	285
<b>INTRODUCCIÓN A ACTIVEX DATA OBJECTS (ADO).....</b>	<b>295</b>
INTRODUCCIÓN .....	295
OLE DB .....	295
CARACTERÍSTICAS GENERALES DE ADO.....	298
MODELO DE OBJETOS DE ADO .....	299
PRINCIPALES OBJETOS DE ADO .....	300

UDA.....	303
INSTALACIÓN DE SQL SERVER 7 .....	304
<i>Standard</i> .....	304
<i>Enterprise</i> .....	304
<i>Desktop o SBS (Small Business Server)</i> .....	304
<b>ACCESO A DATOS CON ADO: CONNECTION .....</b>	<b>309</b>
INTRODUCCIÓN .....	309
EL OBJETO CONNECTION .....	309
REALIZACIÓN DE UNA CONEXIÓN.....	311
CONFIGURACIÓN DE LA CONEXIÓN.....	315
EJECUTANDO COMANDOS SOBRE LA CONEXIÓN.....	317
TRANSACCIONES.....	320
COLECCIONES DEL OBJETO CONNECTION.....	322
CERRANDO LA CONEXIÓN.....	324
<b>ACCESO A DATOS CON ADO: COMMAND.....</b>	<b>325</b>
EL OBJETO COMMAND .....	325
CREACIÓN DEL OBJETO COMMAND.....	326
LA COLECCIÓN PARAMETERS .....	329
EJECUCIÓN DE COMANDOS .....	333
TRATAMIENTO DE ERRORES EN ADO .....	342
<b>ACCESO A DATOS CON ADO: RECORDSET I.....</b>	<b>347</b>
EL OBJETO RECORDSET .....	347
TIPOS DE RECORDSET .....	350
CREACIÓN Y APERTURA DE OBJETOS RECORDSET .....	357
LA COLECCIÓN FIELDS.....	360
<b>ACCESO A DATOS CON ADO: RECORDSET II.....</b>	<b>367</b>
RECORRIENDO EL RECORDSET .....	367
MODIFICACIÓN DEL RECORDSET .....	373
CONSULTANDO EL RECORDSET .....	381
PAGINACIÓN DEL RECORDSET .....	387
<b>ADO 2.5: RECORD.....</b>	<b>393</b>
INTRODUCCIÓN .....	393
ACTIVEX DATA OBJECTS 2.5 (ADO 2.5).....	393
EL OBJETO RECORD .....	395
<i>Creación y apertura del objeto Record</i> .....	396
<i>La colección Fields</i> .....	401
<i>Manipulación de objetos Record</i> .....	406
EL MÉTODO GETCHILDREN.....	410
<b>ADO 2.5: STREAM.....</b>	<b>415</b>
INTRODUCCIÓN .....	415
EL OBJETO STREAM .....	415
CREACIÓN Y APERTURA DEL OBJETO STREAM .....	417
MANIPULACIÓN DEL OBJETO STREAM .....	420
<b>ASP Y VISUAL INTERDEV 6.0.....</b>	<b>427</b>
INTRODUCCIÓN .....	427
PROYECTOS Y SOLUCIONES .....	428
CREAR UN PROYECTO WEB.....	430
EL ACCESO A DATOS A TRAVÉS DE VISUAL INTERDEV .....	435

<i>Agregar una conexión a datos</i> .....	435
<i>La vista de datos</i> .....	440
<i>El Diseñador de Consultas</i> .....	441
<i>El Diseñador de bases de datos</i> .....	442
CONTROLES EN TIEMPO DE DISEÑO .....	443
PROYECTO DE BASE DE DATOS .....	448
DESARROLLO DE APLICACIONES WEB EN EQUIPO.....	450
DISTRIBUIR LA APLICACIÓN.....	452
<b>ASP Y SERVICIOS DE COMPONENTES .....</b>	<b>455</b>
INTRODUCCIÓN .....	455
CREANDO COMPONENTES ACTIVEX SERVER.....	455
INTRODUCCIÓN A COM .....	459
ANTECEDENTES: MICROSOFT TRANSACTION SEVER .....	460
CARACTERÍSTICAS DE SERVICIOS DE COMPONENTES .....	461
EL EXPLORADOR DE SERVICIOS DE COMPONENTES .....	463
GESTIÓN DE TRANSACCIONES.....	468
PÁGINAS ASP TRANSACCIONALES. EJEMPLO DE DESARROLLO .....	470
<i>Ejemplo de desarrollo transaccional con ASP y Servicios de componentes</i> .....	471

# 1

# Introducción a ASP, Active Server Pages

---

## Antecedentes de ASP: la especificación CGI

ASP no es una idea realmente nueva, encontramos un antecedente muy importante y muy utilizado en Internet denominado comúnmente scripts CGI.

Las siglas CGI se corresponden en inglés a Common Gateway Interface, es decir, interfaz de pasarela común. Vamos a ir viendo paso a paso que significan cada unas de estas palabras, que realmente son las que definen el concepto de CGI.

La especificación Common Gateway Interface permite a los servidores Web ejecutar y comunicarse con otros programas, llamados programas CGI, e incorporar la salida de los mismos a los gráficos, texto o audio enviados a un navegador Web.

La programación en CGI implica diseñar programas que se ejecutarán en el entorno de Internet, y más concretamente en el entorno World Wide Web.

El programa CGI se ejecutará dentro del entorno ofrecido por el servidor Web que lo contiene. El servidor Web creará una información especial para el CGI cuando pasa a ejecutarlo, y el servidor esperará una respuesta del programa CGI como resultado de su ejecución. Es esta comunicación o interacción entre el servidor Web y el programa CGI es lo que define realmente la especificación CGI.

Los programas CGI también se suelen denominar scripts CGI, esto es debido a que los primeros programas CGI fueron escritos utilizando scripts de la shell de UNIX y Perl.

Antes de que el programa CGI se ejecute, el servidor Web que lo contiene se encargará de crear un entorno con el que podrá interactuar el programa CGI. Este entorno comprende la traducción de

cabeceras de peticiones del protocolo HTTP (HyperText Transfer Protocol) en variables de entorno a las que podrá acceder nuestro programa CGI. Estas variables de entorno contendrán una información muy variada acerca del cliente que ha realizado la petición o del propio servidor Web en el que se ejecuta el programa CGI.

Una vez que el servidor ha iniciado la ejecución del programa CGI esperará un resultado de la ejecución del mismo. Este resultado suele ser una serie de encabezados de respuesta del protocolo HTTP y código HTML. Estos encabezados y código HTML serán recogidos por el servidor Web y enviados al cliente que realizó la petición, es decir, al navegador o cliente Web.

Después de ver esta pequeña introducción podemos definir un programa CGI como un programa que se encuentra en un servidor Web y que recibe peticiones desde un cliente Web través del servidor Web. Y gracias al entorno que le ofrece el servidor Web el programa CGI puede obtener información sobre la petición realizada, además de otra información útil, que le permitirá procesar la petición. La respuesta a esta petición será generada por el programa CGI en forma de cabeceras de respuesta del protocolo HTTP y etiquetas del lenguaje HTML (HyperText Markup Language), que serán enviadas por el servidor Web al navegador Web que realizó la petición.

CGI no es un lenguaje de programación sino que es una especificación. La especificación CGI va a realizar la función de interfaz o pasarela entre el servidor Web y los programas CGI, haciendo uso del protocolo HTTP y el lenguaje de hipertexto HTML.

Un programa CGI será aquel que cumpla la especificación CGI, es decir, interactuará con el servidor atendiendo a unos principios establecidos por la especificación CGI.

CGI ya lleva siendo utilizado muchos años en la red y todavía se sigue utilizando en muchos sitios Web a la hora de acceder a datos o construir páginas dinámicas, pero cada vez más los sitios Web van adoptando la utilización de Active Server Pages.

Active Server Pages (ASP) es el nombre que reciben las páginas activas de servidor, es decir, las páginas que se ejecutan en el servidor. ASP se basa en la especificación CGI, podemos considerar que ASP es una evolución de la especificación CGI.

## Definición de ASP

La filosofía de ASP resulta muy sencilla, en pocas palabras se puede definir de la siguiente forma: las páginas ASP, también llamadas páginas activas, son páginas que contienen código HTML, script de cliente y un script que se ejecuta en el servidor, dando como resultado código HTML. Por lo tanto al cargar una página ASP en nuestro navegador, en realidad no estamos cargando la página ASP como tal, sino el resultado de la ejecución de la página ASP, es decir la salida de la página ASP, y como se ha apuntado anteriormente se trata de código HTML. Es decir, son páginas que se ejecutan en el servidor enviando como resultado al cliente código HTML.

Antes de seguir vamos a definir de forma sencilla lo que se considera un lenguaje de script o de secuencia de comandos. Un lenguaje de script es un subconjunto de otro lenguaje más general y que se utiliza para un entorno muy determinado, en este caso el entorno es la Web.

Una página ASP podrá contener los siguientes elementos: texto, componentes ActiveX, código HTML y comandos de script. Este script puede ser de dos tipos: script de cliente o script de servidor. El script de servidor es la nueva idea que introduce ASP, se debe tener en cuenta que en el script de servidor se tiene acceso a diferentes objetos y no está orientado a eventos.

El script de servidor utilizado en ASP utiliza la misma sintaxis que el script de cliente, la diferencia está en que con ASP el script de servidor es compilado y procesado por el servidor Web antes de que la página sea enviada al navegador.

ASP no es un lenguaje de script, ASP ofrece un entorno para procesar scripts que se incorporan dentro de páginas HTML, es decir, un entorno de procesamiento de scripts de servidor.

La propia Microsoft define ASP de la siguiente manera: "...es un entorno de secuencias de comandos en el lado del servidor que puede utilizar para crear y ejecutar aplicaciones de servidor Web dinámicas, interactivas y de alto rendimiento...".

Realmente, ASP es un componente (asp.dll) que se instala en un servidor Web y cuya misión es la de procesar ficheros que terminan con la extensión .asp y transmitir el resultado al cliente que solicitó la página ASP.

El script de servidor incluido en una página ASP empieza a ejecutarse cuando un navegador solicita el archivo .asp al servidor Web. El servidor Web llama entonces a ASP, el cual lee el archivo solicitado de arriba a abajo, ejecuta los comandos y envía una página HTML al explorador. ASP incluye un motor de interpretación de scripts del lado del servidor.

Las páginas ASP son ficheros con la extensión .asp. Crear un fichero .asp resulta muy sencillo, se puede crear a partir de una página HTML existente, simplemente renombrando el fichero .html o .htm a un fichero .asp. Para hacer esta página ASP disponible para los usuarios de la Web, el fichero .asp se debe almacenar en un directorio de publicación en Internet, se debe tener en cuenta que el directorio virtual asociado debe tener permisos de ejecución de secuencias de comandos.

La última versión de la tecnología ASP es la versión 3.0. Esta versión es muy similar a su predecesora, y todas las nuevas características que presenta se deben a que se utiliza una nueva versión del servidor Web (Internet Information Services 5.0), recordemos que las páginas ASP son procesadas por el servidor.

En el tema siguiente se ofrece una comparativa de ASP 2.0 con ASP 3.0 comentando brevemente todas sus novedades, se recomienda la lectura del segundo capítulo sobretodo a los alumnos que ya conozcan ASP 2.0.

## Aplicaciones ASP

Una aplicación basada en ASP consta de un directorio virtual en un servidor Web y de todos los subdirectorios y archivos contenidos en él. Una aplicación puede ser una página principal sencilla, o bien puede estar formada por un conjunto completo de páginas interrelacionadas entre sí.

Al usar aplicaciones en ASP es posible mantener un estado, es decir, se tiene la capacidad de mantener información. Dentro de una aplicación ASP se pueden mantener dos tipos de estado:

- Estado de la aplicación, en la que toda la información relativa a una aplicación está disponible para todos los usuarios de la misma.
- Estado de sesión, en la que la información sólo está disponible para un usuario o sesión específicos. Una sesión por lo tanto, pertenece a un solo usuario.

Un ejemplo práctico de una aplicación ASP puede ser este mismo sitio Web. Almagesto está completamente realizado con páginas ASP constituyendo por lo tanto una aplicación ASP, este sitio

Web demuestra los diferentes usos que puede tener la tecnología ASP y las necesidades que puede cubrir.

Las aplicaciones ASP no son aplicaciones al uso, ya que en realidad no se dispone de un ejecutable sino de un conjunto de páginas, imágenes y recursos, por lo tanto se trata de aplicaciones muy particulares que requieren para su ejecución de un servidor Web que soporte las páginas ASP.

## Aportaciones de ASP

En este apartado se comentan las aportaciones que ofrece ASP desde su primera versión, es decir, se trata de aportaciones muy genéricas de la tecnología ASP.

Para entender las aportaciones que ofrecen las páginas ASP se deben tener en cuenta una serie de características del protocolo HTTP (HyperText Transfer Protocol). Se dice que el protocolo HTTP es un protocolo sin estado, es decir, no se puede mantener un estado entre diferentes peticiones. El protocolo HTTP se basa en el paradigma cliente/servidor o petición/respuesta.

Se deben tener en cuenta un par de puntos a la hora de establecer la comunicación entre clientes (navegadores Web) y servidores (servidores Web) del protocolo HTTP:

- Después de realizar una petición el cliente se desconecta del servidor y espera una respuesta. El servidor debe restablecer la conexión después de que haya procesado la petición.
- El servidor y el cliente sólo se tienen en cuenta durante la conexión, después, se olvidan el uno del otro. Por esta razón, ni el cliente ni el servidor pueden retener información entre diferentes peticiones o a través de diferentes páginas Web. Sin embargo, ASP permite al servidor almacenar información, o mantener el estado, entre las diferentes peticiones del cliente.

El cliente y el servidor Web se comunican utilizando cabeceras HTTP, estas cabeceras son colecciones de datos que intercambian el cliente y el servidor para asegurar que la transacción es coherente y completa. Como petición del usuario se envía una cabecera y el servidor interpreta esta cabecera y envía una respuesta HTTP cuyo cuerpo sería el contenido del recurso demandado por el cliente.

ASP permite al desarrollador intervenir en todo el proceso de comunicación del protocolo HTTP. Los objetos integrados dentro de ASP Request y Response interactúan con las peticiones y respuestas del protocolo HTTP, respectivamente.

Dentro de los objetos integrados de ASP podemos encontrar la forma de acceder al servidor, obtener información del mismo, así como del usuario. Y también se permite, como se había comentado anteriormente, mantener el estado entre diferentes peticiones del cliente.

Se puede considerar ASP como una nueva (aunque ya no tan nueva) aproximación a la creación de páginas web complejas que pueden acceder a bases de datos o a otros objetos del servidor. Ofrece lo siguiente:

- Independencia del navegador, ASP puede ejecutar complejas operaciones en el servidor y enviar solamente los resultados al cliente.
- Construcción de páginas basadas en bases de datos que permiten realizar operaciones sobre las bases de datos del servidor de forma bastante sencilla.
- Es una de las soluciones más versátiles para el desarrollo de aplicaciones en el entorno de Internet/Intranet.

- Desarrollo de complejas aplicaciones Web.
- Facilidad de uso de componentes de terceras partes ejecutándose en el servidor, es decir, se pueden utilizar componentes para liberarnos de realizar tareas complejas. Estos componentes se deben registrar en el servidor y podrán ser utilizados desde el script correspondiente. Estos componentes se denominan componentes ActiveX de servidor.
- Posibilidad de definir páginas ASP transaccionales para realizar todas las operaciones contenidas en la misma dentro de una transacción.
- Una tecnología en constante evolución y mejora.

A lo largo del curso se profundizará más en todos estos puntos, aquí se han comentado simplemente los más evidentes y también para poseer una visión general de lo que supone la tecnología ASP.

## Requerimientos de ASP 3.0

En este apartado se va a comentar los distintos requerimientos que presentaban cada una de las versiones de ASP, desde la más antigua hasta la presente versión.

La primera versión de las páginas activas (ASP 1.0), se incorporó como un añadido o ampliación al servidor Web del sistema operativo Microsoft Windows NT Server 4.0 llamado Internet Information Server 3.0 (IIS 3.0). Este servidor Web era bastante interesante pero todavía era demasiado rudimentario y presenta limitaciones y problemas.

La primera versión de ASP era bastante interesante ya que se pasaba de la complejidad de los CGIs (Common Gateway Interface) a la sencillez de las páginas activas. ASP 1.0 supuso el inicio del desarrollo de aplicaciones Web con productos basados en tecnología Microsoft.

La versión 2.0 de Active Server Pages la encontramos en el servidor Web de Microsoft Internet Information Server 4 (IIS 4) y en el servidor Personal Web Server 4 (PWS 4). Ambos servidores los podemos instalar desde la extensión del sistema operativo de Windows NT denominada Windows NT 4.0 Option Pack, o más comúnmente Option Pack. Esta extensión del sistema operativo no sólo es aplicable a Windows NT, sino que también la podemos utilizar para Windows 95/98.

Se debe señalar que el servidor IIS 4 es el servidor Web para plataformas Windows NT Server 4.0, y el servidor Personal Web Server 4.0 es el servidor Web para plataformas Windows 95/98 y Windows NT Workstation 4.0.

IIS 4 además de ofrecer la nueva versión de la tecnología ASP permite configurar y administrar de forma sencilla nuestras aplicaciones ASP. Además la figura de la aplicación ASP se encuentra mucho más clara que en la versión 1.0 de las páginas ASP, el servidor Web nos indicará claramente el alcance de una aplicación ASP determinada.

ASP 2.0 es una clara y necesaria evolución de ASP 1.0 incorporando la posibilidad de realizar páginas ASP transaccionales, añadiendo para ello un nuevo objeto integrado denominado ObjectConext (objeto de contexto). ASP 2.0 ofrece un entorno más robusto y potente que la versión anterior para el desarrollo de aplicaciones Web.

Y por fin llegamos al presente, ASP 3.0. Para poder utilizar ASP tenemos que disponer de cualquiera de las versiones del sistema operativo Windows 2000 (Professional, Server y Advanced Server). En este caso no se trata únicamente de una nueva versión del servidor Web sino también de una nueva versión del sistema operativo Windows.

ASP 3.0 se encuentra contenido en la nueva versión del servidor Web de Microsoft, llamado Internet Information Server 5.0 o también Internet Information Services 5.0 en cualquier caso lo llamaremos IIS 5.0. El servidor Web IIS 5.0 se encuentra formando parte del sistema operativo Windows 2000 como un componente más, de esta forma Microsoft integra la funcionalidad del servidor Web dentro de su plataforma Windows 2000.

ASP 3.0 podemos decir que es la evolución lógica de ASP 2.0, no supone ningún cambio radical, ofrece una serie de mejoras y novedades (que se comentarán el siguiente capítulo para los lectores que ya conozcan ASP 2.0). Se añade un nuevo objeto integrado llamado ASPError, este nuevo objeto es utilizado para el tratamiento de errores.

## Sintaxis de ASP

Como se ha comentado anteriormente ASP no es un lenguaje de script, sino que ofrece un entorno para la ejecución de estos lenguajes que se encuentran dentro de páginas ASP. ASP posee una sintaxis para poder distinguir cada uno de los elementos que nos podemos encontrar dentro de una página ASP.

Encerrado dentro de los delimitadores <%%> se va a encontrar todo el código de script de servidor, de esta forma el comando <%nombre="Pepe"%> asigna el valor Pepe a la variable nombre; y dentro de los delimitadores <%==%> se encuentran expresiones de salida, así por ejemplo la expresión <%=nombre%> enviará al navegador el valor Pepe, es decir, el valor actual de la variable, más adelante se verá una equivalencia de estos delimitadores con un método de un objeto integrado de ASP.

Entre los delimitadores <%%> se puede y debe incluir varias sentencias en distintas líneas de código del lenguaje de secuencias de comandos, sin embargo los delimitadores <%==%> sólo podemos encerrar una sentencia por línea.

Entre los delimitadores de ASP se puede incluir cualquier tipo de expresión válida en el lenguaje de script principal. Por ejemplo la línea que muestra el Código fuente 1 genera un texto que contiene la hora actual del servidor.

Esta página se actualizó a las <%=Now%>

Código fuente 1

En este caso el servidor Web devuelve al navegador el valor de la función Now de VBScript junto con el texto.

Dentro de los delimitadores de script de servidor se pueden encontrar también instrucciones del lenguaje de script correspondiente, así por ejemplo puede aparecer una instrucción If...Then...Else del lenguaje VBScript como se puede apreciar en el Código fuente 2.

```
<%
If nombre="" Then
    variable="Nombre desconocido"
Else
    variable="Hola amigo "&nombre
End If
%>
```

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

Código fuente 2

En el código anterior se comprueba si la variable nombre tiene algún valor, si lo tiene saludamos con el valor de la variable, mostrando el saludo en color verde.

También se puede incluir código HTML entre las instrucciones del script. Por ejemplo la secuencia de comandos del Código fuente 3, mezcla HTML con una instrucción condicional y produce el mismo resultado que la secuencia del Código fuente 2.

```
<FONT COLOR="GREEN">
<% If nombre="" Then%>
    Nombre desconocido
<%Else%>
    Hola amigo <%=nombre%>
<%End If%>
</FONT>
```

Código fuente 3

Para poder realizar una lectura más sencilla del código ASP se recomienda utilizar los delimitadores de script de servidor encerrando varias líneas de código en lugar de un par de delimitadores por cada línea. Así, en lugar de escribir el código que indica el Código fuente 4, se debería escribir lo que muestra el Código fuente 5.

```
<%strNombre=Session("nombre")%>
<%strApellidos=Session("apellidos")%>
<%strEdad=Session("edad")%>
```

Código fuente 4

```
<%
    strNombre=Session("nombre")
    strApellidos=Session("apellidos")
    strEdad=Session("edad")
%>
```

Código fuente 5

En el caso de tener línea simple de script, los delimitadores se deben encontrar en la misma línea.

```
<%strNombre=Session("nombre")%>
```

Código fuente 6

## Objetos integrados en ASP 3.0

ASP en su versión 3.0 contiene siete objetos integrados que liberan al programador de la realización de tareas complejas. Estos seis objetos no requieren que sean instanciados siempre se encuentran disponibles en nuestras páginas ASP.

Estos objetos son los siguientes: Application, Session, Request, Response, Server, *ASPError* y ObjectContext. Cada uno de estos objetos posee una serie de métodos y propiedades para poder ser utilizados por el script de servidor, además cada objeto posee una función determinada, básicamente estas funciones son las siguientes:

- Request: obtención de información del cliente.
- Response: envío de información al cliente.
- Server: acceso a los recursos del servidor, como puede ser la creación de componentes .
- Session: almacena información sobre la sesión de un usuario.
- Application: almacena información común para todos los usuarios de la aplicación ASP.
- ObjectContext: gestión de transacciones en páginas ASP.
- *ASPError*: contiene información detallada acerca del último error que se ha producido.

Cada uno de estos objetos se explicarán con una mayor profundidad en su capítulo correspondiente.

La sintaxis utilizada para poder acceder a los métodos y propiedades de los objetos depende del lenguaje de script que estemos utilizando. Debido que el lenguaje de script por defecto de ASP es VBScript (subconjunto de Visual Basic) en este curso nos vamos a centrar en este script.

Los objetos Request y Response contienen colecciones. Una colección es un conjunto de elementos de información relacionados y que se accede a ellos de una misma forma. Se puede acceder a cada elemento de una colección mediante el bucle For...Each. La utilización de colecciones se verá en detenimiento en los capítulos dedicados a estos dos objetos integrados.

Un método es un procedimiento que actúa sobre un objeto, la sintaxis para poder invocar un método de un objeto es la siguiente:

`Objeto.método parámetros`

Donde el tipo de parámetros dependerá del método invocado.

Una propiedad es un atributo de un objeto. Las propiedades son características de un objeto que describen su estado, así por ejemplo un objeto podría tener las características tamaño, nombre, color, etc. Para obtener el valor de una propiedad utilizamos la sintaxis siguiente:

`Objeto.propiedad`

Y para asignarle un valor a una propiedad de un objeto debemos utilizar la sintaxis

`Objeto.propiedad=valor`

Donde valor depende de la propiedad del objeto.

## Componentes de servidor

ASP incluye una serie de componentes ActiveX de servidor (o componentes de servidor), llamados componentes ActiveX Server, anteriormente conocidos como servidores de Automatización. Estos componentes están diseñados para ejecutarse en un servidor Web y contienen una serie de funciones bastante útiles para que el programador no tenga que construirlas, una de estas funciones puede ser el acceso a bases de datos. Estos componentes los invocaremos desde nuestras páginas ASP.

No se deben confundir los componentes de servidor con los objetos integrados en ASP.

Para poder tener acceso a alguno de los componentes ActiveX de servidor primero se deberá crear una instancia del componente correspondiente. Una vez creada la instancia, se pueden usar los métodos asociados al componente o establecer y leer sus propiedades.

Los componentes ActiveX Server que incluye ASP en su versión 3.0 son los siguientes:

- Componente de acceso a bases de datos, ADO (ActiveX Data Objects). A través de la utilización de este componente se puede ofrecer acceso a bases de datos desde una página ASP, así por ejemplo, se puede mostrar el contenido de una tabla, permitir que los usuarios realicen consultas y otras operaciones sobre una base de datos.
- Componente Ad Rotator. Este componente permite mostrar una serie de imágenes alternativas con un vínculo a otra dirección desde la imagen presentada. Este componente se suele utilizar para mostrar diferentes anuncios de forma alternativa dentro de una página ASP.
- Componente Funciones del explorador. A través de este componentes podemos recuperar datos acerca del tipo de navegador del cliente y que capacidades o funciones tiene.
- Componente vínculo de contenidos. Facilita el desplazamiento lógico entre las diferentes páginas ASP de una aplicación ASP.
- Componente Content Rotator (rotador de contenidos). Este componente permite hacer rotaciones de cadenas de contenido HTML en una página.
- Componente Page Counter (contador de páginas). Permite llevar una cuenta del número de veces que se ha accedido a una página determinada dentro de nuestro sitio Web.
- Componente Counters. A través de este componente podremos almacenar, crear, incrementar y consultar cualquier contador.
- Componente MyInfo. Nos permite almacenar información personal que será ofrecida por el administrador del sitio Web.
- Componente Tools. Es el denominado componente de utilidades. Ofrece una serie de funciones diversas, como la generación de números aleatorios o la comprobación de la existencia de un fichero en el servidor.
- Componente Permission Checker. A través de este componente podremos determinar si a un usuario se le ha dado permisos para acceder a un fichero determinado.
- Componente Status. Este componente, de momento, únicamente está disponible para el servidor Personal Web Server en plataformas Macintosh, resulta extraño pero es así. Nos ofrece una información variada acerca del estado del servidor Web.

- Componente de registro de IIS. Mediante este componente tenemos acceso a la información y manipulación de los ficheros de registro (log) generados por el servidor Web IIS 5.0.

En el curso se ofrece un capítulo monográfico en el que se muestra la utilización de todos estos componentes de servidor que vienen incluidos en ASP.

Además de todos estos componentes, el programador puede crear sus propios componentes ActiveX Server. Estos componentes se pueden desarrollar en lenguajes de programación como Visual Basic, Java o C++, una vez creado el componente se transforma a una DLL que se registrará en el servidor.

Todos los componentes de servidor que no se encuentran incluidos en ASP deben ser registrados. Una vez registrado el componente en el servidor Web lo podemos instanciar desde el lenguaje de secuencias de comandos de una página ASP, al igual que hacíamos con los componentes que vienen por defecto con ASP.

Veremos en el curso un capítulo dedicado completamente a la creación de nuestros propios componentes con Visual Basic 6.0.

También se puede adquirir estos componentes a terceros, existen empresas que se dedican al diseño de componentes para que sean utilizados desde páginas ASP.

## Visión general de ASP

En este apartado se muestra un esquema en el que se puede observar cómo se encuentra estructurada la arquitectura de ASP y los componentes que intervienen dentro de ella. Con la Figura 1. se pretende dar una visión global del entorno ASP.

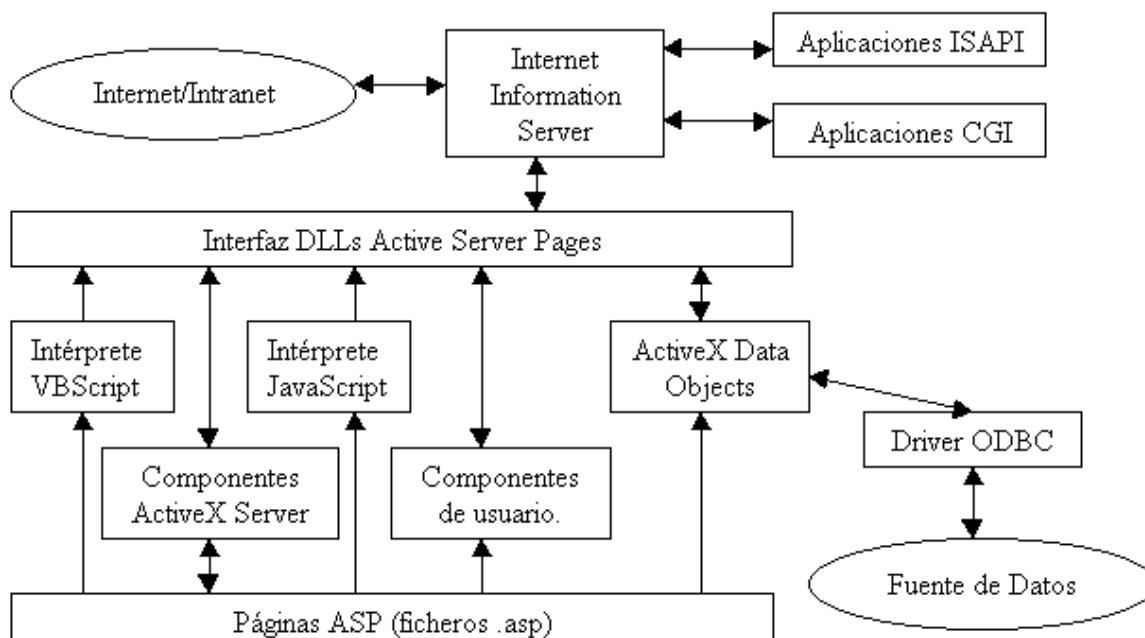


Figura 1. Esquema general del entorno ASP.

En el esquema podemos diferenciar las principales partes de las que se compone el entorno ASP:

- Internet Information Server: es el servidor Web en el que se ejecutarán las páginas ASP y devolverá, como resultado de la ejecución de las mismas, código HTML
- Los intérpretes de los lenguajes de script: estos intérpretes tratarán el script de servidor.
- Componentes ActiveX Server: son los componentes de servidor que se incluyen con ASP y que ya se han visto en el apartado anterior.
- Componentes de usuario: son componentes ActiveX Server desarrollados y creados por terceras partes o por nosotros mismos.
- ActiveX Data Objects: aunque estarían dentro de los componentes ActiveX Server se separan debido a su importancia y complejidad. Nos permiten realizar el acceso a bases de datos de forma potente y sencilla.

## Hola mundo con ASP

Una vez realizada la introducción a ASP y su filosofía, vamos a dejar la teoría un poco de lado y vamos a escribir nuestra primera página ASP completa, realizando para ello el famoso programa Hola Mundo.

Primero se va a mostrar como sería el código completo y a continuación se va a ir comentando cada una de las líneas.

```

1.<%@ LANGUAGE="VBSCRIPT" %>
2.<%Option Explicit%>
3.<HTML>
4.<HEAD>
5.<TITLE>Hola Mundo</TITLE>
6.</HEAD>
7.<BODY>
8.<div align="center">
9.<%Dim i
10.For i=1 to 5%>
11.    <FONT SIZE="<%i%>">Hola Mundo</font><br>
12.<%Next%>
13.</div>
14.</BODY>
15.</HTML>
```

Código fuente 7

La ejecución de esta página ASP produce la salida que muestra la Figura 2, es decir, genera el documento HTML, que vemos en ella.

Hola Mundo  
Hola Mundo  
Hola Mundo  
**Hola Mundo**  
**Hola Mundo**

Figura 2. Hola mundo en el navegador.

En la primera línea indicamos el lenguaje de script que se va a utilizar, en este caso VBScript, a continuación forzamos la declaración de las variables con la instrucción Option Explicit, ya que por defecto VBScript no obliga a declarar variables.

Las líneas de la 3 a la 8 son código HTML que no necesita ningún comentario. De las líneas 9 a la 12 nos encontramos con el script de servidor junto con el código HTML que va a mostrar utilizando un bucle el mensaje "*Hola mundo*", el cual en cada iteración se va aumentando de tamaño.

En la línea 9 se declara la variable que se va a incrementar en el bucle y que va a servir para indicar el tamaño de la letra, ayudándonos para ello de la etiqueta <FONT> de HTML

En las últimas líneas, es decir, de la 13 a la 14 vuelve a aparecer solamente código HTML, en este caso necesario para indicar la finalización del documento.

Si vemos el código fuente desde el navegador Web observaremos lo que muestra el Código fuente 8.

```
<HTML>
<HEAD>
<TITLE>Hola Mundo</TITLE>
</HEAD>
<BODY>
<div align="center">
<FONT SIZE="1">Hola Mundo</font><br>
<FONT SIZE="2">Hola Mundo</font><br>
<FONT SIZE="3">Hola Mundo</font><br>
<FONT SIZE="4">Hola Mundo</font><br>
<FONT SIZE="5">Hola Mundo</font><br>

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

Código fuente 8

Como se puede comprobar no existe ningún rastro del código ASP, ya que ha sido ejecutado por el servidor y lo que obtenemos el resultado de su ejecución.

Esta sencilla página ASP se puede descargar desde [aquí](#).

Para poder ejecutar esta página ASP se escribirá todo el código en un fichero con extensión .asp que residirá en un directorio de publicación en Internet que además posea el permiso de ejecución de scripts.

Para poder probar este primer ejemplo debemos tener instalado el servidor Web Internet Information Server 5.0. El directorio de publicación en Internet, por defecto, de este servidor es C:\Inetpub\wwwroot. Por lo tanto si creamos una subcarpeta llamada C:\Inetpub\wwwroot\prueba y copiamos la página ASP llamada HolaMundo.asp a este directorio, para ejecutarla escribiremos en el navegador <http://nombreServidor/prueba/HolaMundo.asp>. Se debe recordar que una página ASP debe ser siempre ejecutada e interpretada por el servidor Web.

Este ejemplo ha sido un primer contacto con ASP, más adelante, en los distintos apartados de los capítulos correspondientes se verá con más detalle las estructuras de control, la sintaxis de VBScript, los objetos integrados de ASP con sus métodos y propiedades, etc.

# 2

## Novedades de ASP 3.0

---

### ¿Para quién es este capítulo?

Este capítulo está indicado para aquellas personas que ya conocen ASP 2.0 o por lo menos tienen un ligero conocimiento de la versión anterior de ASP. De todas formas este capítulo también puede ser válido para alumnos que no tengan ningún conocimiento de ASP, aunque muchos conceptos e ideas pueden escaparse, pero que no cunda el pánico, en los siguientes capítulos se entrará en más detalle.

Todo aquel que no se sienta con ganas o no lo vea necesario puede saltarse el presente capítulo, ya que tampoco existe examen para el mismo. Además el contenido de este capítulo se volverá a tratar de una forma más detallada a lo largo de todo el curso.

Aquí se pretende mostrar de forma muy general las mejoras y novedades que aporta ASP 3.0 sobre la versión anterior de las páginas activas ASP 2.0, no se va a entrar en detalles y se supone que el lector tiene ya algún conocimiento de ASP 2.0.

A continuación vamos a ir comentando en cada apartado cada uno de los cambios y mejoras que aporta ASP 3.0, para ello se apoya en Internet Information Server 5.0.

### Mejoras generales en ASP 3.0

En este epígrafe se va a reunir una serie de cambios y novedades que ofrece ASP 3.0, que son de carácter general y que afectan al rendimiento y funcionamiento de las aplicaciones ASP.

En esta nueva versión de ASP se ofrece un mejor funcionamiento y escalabilidad de la tecnología ASP, basándose en las nuevas características y mejoras de Internet Information Server 5.0.

- Se ha producido una mejora en el procesamiento de las páginas ASP por parte de la librería ASP.DLL.
- Se ofrece lo que se denomina ajuste automático, que consiste en detectar cuándo una petición está bloqueada por recursos externos, en ese caso se proporcionan automáticamente más subprocessos para ejecutar peticiones adicionales y continuar de esta forma con el procesamiento normal de forma simultánea.
- Los objetos COM se liberan más rápidamente y por defecto los componentes COM se ejecutan out-of-process, es decir, en un espacio de memoria distinto al del servidor Web.
- Con ASP 3.0 se ofrecen los objetos COM que se ofrecían con ASP 2.0 (componentes de servidor, como Content Rotator) pero con su rendimiento mejorado, es decir, aparecen versiones mejoradas de los componentes anteriores.
- El servidor transaccional Microsoft Transaction Server (MTS) ya no existe como una entidad separada en Windows 2000, y pasa a formar parte de Servicios de componentes (Microsoft Component Services). IIS 5.0 y Servicios de componentes funcionan conjuntamente para formar la arquitectura básica para la creación de aplicaciones Web.

## El objeto Response

Los únicos objetos integrados dentro de ASP que han sufrido alguna modificación han sido el objeto Response, que vemos en este apartado, y el objeto Server.

Por defecto la propiedad Buffer del objeto Response tiene el valor True (verdadero), en ASP 2.0 y 1.0 esta propiedad del objeto Response tenía por defecto el valor de False (falso). Debido a esto, en ASP 3.0 el resultado de la ejecución de una página ASP únicamente es enviado al cliente cuando se termina de procesar la página ASP correspondiente, o bien cuando se utilizan los métodos Flush o End del objeto Response.

Por lo tanto, a no ser que se indique otra cosa, de forma predeterminada el resultado de la ejecución de la página ASP se enviará al búfer. Según afirma Microsoft la técnica del búfer ofrece una entrega de páginas más eficiente al cliente.

En el objeto Response también cambia la forma de utilizar la propiedad IsClientConnected, mediante esta propiedad podemos consultar si un cliente se encuentra todavía conectado a nuestro servidor o por el contrario si ha finalizado su sesión con el mismo. En ASP 2.0 podíamos consultar esta propiedad sólo si antes habíamos enviado ya alguna salida o contenido al cliente, ahora con ASP 3.0 podemos utilizar IsClientConnected antes de enviar cualquier contenido al navegador.

En los siguientes capítulos veremos en profundidad este objeto y el resto de los objetos integrados dentro de ASP, como ya se ha dicho este capítulo pretende ser simplemente una comparativa entre ASP 3.0 y ASP 2.0 mostrando las mejoras y novedades que existen entre ambas versiones.

## El objeto Server

Este es otro de los objetos de ASP que ha experimentado cambios. Presenta dos nuevos métodos: Transfer y Execute, que permiten controlar el control de flujo del programa, ampliando las capacidades de control de flujo de las páginas ASP, anteriormente sólo se disponía del método Redirect del objeto Response.

En ASP 2.0 si queríamos transferir la ejecución a otra página ASP teníamos que utilizar el método Redirect del objeto Response, pero esto suponía enviar una respuesta al cliente para indicarle la carga de una nueva página, que es la página a la que pasamos la ejecución.

La utilización del método Redirect es bastante costosa y problemática ya supone un envío de información más del servidor al cliente para indicarle mediante una cabecera HTTP de redirección que la página ha cambiado de localización, siendo la nueva localización la página que deseamos cargar. Esto es problemático ya que en algunos navegadores como Netscape Communicator aparece un mensaje del tipo El objeto requerido se ha movido y se puede encontrar aquí, esto también ocurre cuando la conexión la realiza el cliente a través de proxy.

Pero ahora con ASP 3.0 podemos evitar esta redirección, que como hemos visto, tiene lugar en el cliente, mediante los métodos Execute y Transfer del objeto Server que permiten que la redirección tenga lugar en el servidor, quedando el cliente completamente ajeno. Ambos métodos reciben como parámetro la ruta de la página a la que queremos redirigir al cliente.

La utilización del método Execute es muy similar a realizar una llamada a un procedimiento o función. Cuando lanzamos el método Execute se empieza a ejecutar la página que indicamos por parámetro, y cuando termina la ejecución de esta nueva página, el control pasa a la siguiente sentencia después de la llamada al método Execute en la página inicial, siguiendo a partir de aquí con la ejecución de la página, es decir, el navegador del cliente recibe una salida formada por la combinación de la ejecución de ambas páginas.

El método Transfer se comporta de distinto modo, al lanzar este método se pasa la ejecución a la nueva página, pero una vez que finaliza la ejecución de la misma no se vuelve a la página inicial, como ocurría con el método Execute.

En ambos métodos se mantiene el contexto de la página inicial, es decir, en la nueva página tenemos acceso a las variables, objetos y a todos los objetos intrínsecos de ASP (Request, Session, Response...) de la página inicial o página de origen. También se mantienen las transacciones entre distintas páginas, siempre que proceda, atendiendo a la directiva @TRANSACTION.

De esta forma como la redirección entre páginas se produce en el servidor, el navegador cree que sigue recibiendo todavía la página original que había demandado, incluso en la barra de direcciones del navegador sigue apareciendo la misma URL y los botones Atrás y Adelante funcionan correctamente.

Vamos a ofrecer un sencillo código de una página ASP que utiliza los métodos Transfer y Execute para ejecutar otra página, y así se puede ver más claramente la utilización de estos dos nuevos métodos del objeto Server.

Nuestra página, llamada PaginaInicial.asp, va a constar de un formulario con dos botones, y según el botón que se pulse se lanzará el método Execute o Transfer para ejecutar la página OtraPagina.asp. El código de estas dos páginas se ofrece a continuación.

Primero el código de la página PAGINAINICIAL.ASP.

```

<%If Request.Form("Execute")<>"" Then
    Response.Write "Se está ejecutando la página " _
        & Request.ServerVariables("SCRIPT_NAME") & "<br>"
    Server.Execute "OtraPagina.asp"
    Response.Write "Se está ejecutando de nuevo la página " _
        & Request.ServerVariables("SCRIPT_NAME") & "<br>"_
ElseIf Request.Form("Transfer")<>"" Then
    Response.Write "Se está ejecutando la página " _
        & Request.ServerVariables("SCRIPT_NAME") & "<br>"
    Server.Transfer "OtraPagina.asp"
    Response.Write "Se está ejecutando de nuevo la página " _
        & Request.ServerVariables("SCRIPT_NAME") & "<br>"_
End if%>
<FORM ACTION="PaginaInicial.asp" METHOD="POST">
    <input type="submit" name="Execute" value="Lanza Server.Execute"><br>
    <input type="submit" name="Transfer" value="Lanza Server.Transfer">
</FORM>

```

Código fuente 9

Y ahora el código de la página ASP llamada OTRAPAGINA.ASP.

```

<hr>
Se está ejecutando la página OtraPagina.asp<br>
Esta página se ha cargado con el método
<%If Request.Form("Execute")<>"" Then%>
    <b>EXECUTE</b>
<%ElseIf Request.Form("Transfer")<>"" Then%>
    <b>TRANSFER</b>
<%End If%>
<br>La variable Request.ServerVariables("SCRIPT_NAME") sigue teniendo el valor:
<%=Request.ServerVariables("SCRIPT_NAME")%><br>
Termina la ejecución de OtraPagina.asp<br>
<hr>

```

Código fuente 10

Si ejecutamos la página PAGINAINICIAL.ASP y pulsamos cada uno de sus botones, vemos el distinto comportamiento de los método Execute y Transfer, en el primer caso se intercala el resultado ejecución de ambas páginas, y en el segundo paso una vez que se ha terminado de ejecutar la segunda página finaliza la ejecución de la secuencia de comandos, sin retornar a la página inicial. Las siguientes figuras muestran estas dos situaciones.

La Figura 3 muestra la página PAGINAINICIO.ASP cuando todavía no se ha pulsado ningún botón. En la Figura 4 se muestra cuando se ha pulsado el botón Execute. Y en la Figura 5 cuando se ha pulsado el botón Transfer.

En este [enlace](#) se pueden obtener las páginas ASP de este ejemplo.

Otro nuevo método que ofrece el objeto Server, y que está relacionado con el tratamiento de errores, es el método GetLastError. Mediante el uso del método GetLastError podemos tener acceso a toda la información referente al último error que se ha producido en la página ASP actual. Pero es necesario aclarar que su utilización no es similar al tratamiento de errores que realizábamos con la sentencia On Error Resume Next y el objeto Err de VBScript, que preguntábamos por la propiedad Number del objeto Err para averiguar si se había producido algún error, el método GetLastError se puede utilizar

únicamente dentro de una página de error personalizada, es decir, cuando el error ya se ha producido y lo ha detectado el servidor Web.

Mediante Internet Information Services 5 podemos indicar las páginas de error personalizadas y es en estas páginas dónde podemos hacer uso de este método.

El método GetLastError devuelve un nuevo objeto de ASP llamado ASPError, son las propiedades de este nuevo objeto las que nos permiten acceder de forma detallada a toda la información referente al error que se ha producido. Este nuevo objeto lo trataremos con más detalle en el siguiente apartado.

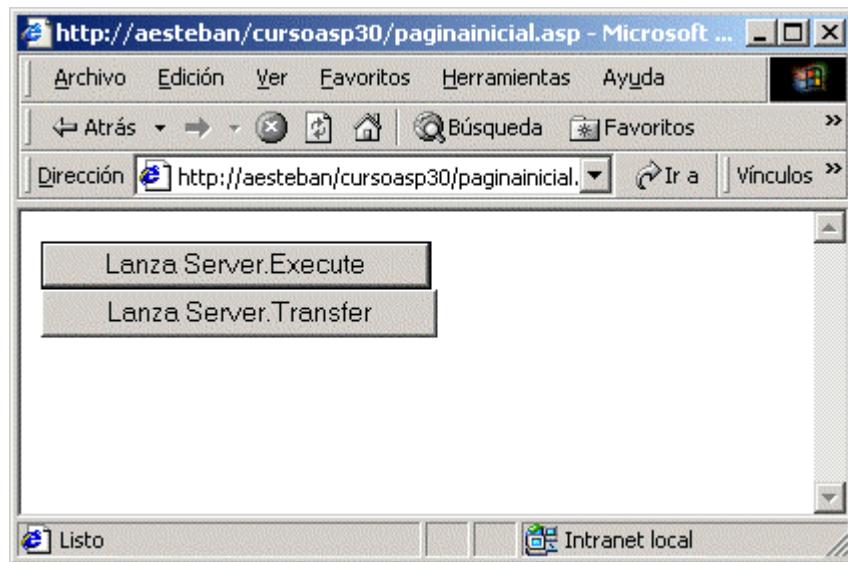


Figura 3

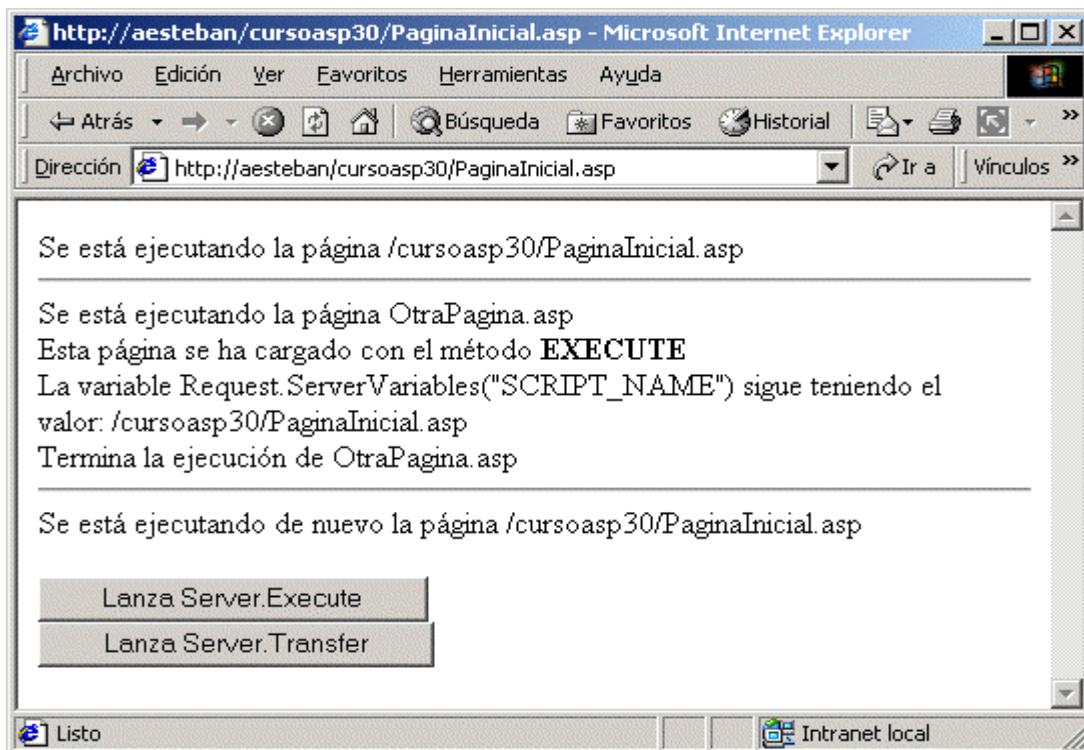


Figura 4

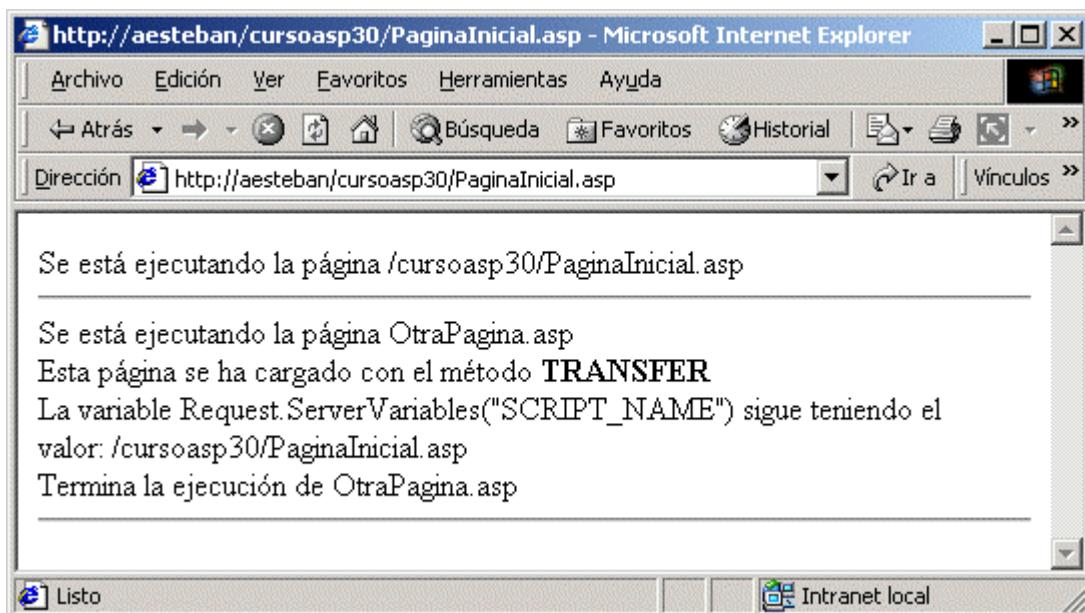


Figura 5

## El objeto ASPError

Como ya hemos visto en el apartado anterior, este es un nuevo objeto del modelo de objetos incluido dentro de ASP 3.0. Tendremos acceso al objeto ASPError a través de la llamada al método GetLastError del objeto Server. La función de este objeto es la de ofrecer de forma detallada toda la información disponible del último error que se ha producido.

El objeto ASPError únicamente dispone de una serie de propiedades de sólo lectura, que contienen la información relativa al último error que se ha producido. Estas propiedades se comentan de forma breve en la Tabla 1.

Propiedad	Descripción
ASPCODE	Un entero generado por IIS.
ASPDescription	Una cadena que es una descripción detallada del error si está relacionado con ASP.
Category	Cadena que indica si se trata de un error interno de ASP, del lenguaje de secuencia de comandos o de un objeto.
Column	Entero que indica la posición de la columna del archivo ASP que generó el error.
Description	Breve descripción del error.
File	Nombre del archivo ASP que se estaba procesando cuando se produjo el error.
Line	Línea del archivo ASP que generó el error.

Number	Código de error estándar de COM.
Source	Devuelve el código fuente real, si está disponible, de la línea que causó el error.

Tabla 1

En el Código fuente 11 se muestra un sencillo código de ejemplo que hace uso del objeto ASPError, y que podría pertenecer a una página de error personalizada de IIS 5.0. Primero se obtiene una referencia al objeto ASPError mediante el método GetLastError del objeto Server, y a continuación se muestra los valores de las propiedades del objeto ASPError para informar al cliente acerca del error que se ha producido.

```
<%@ language="VBScript" %>
<html>
<body>
<head>
<title>The page cannot be displayed</title>
</head>
<%Set objASPError=Server.GetLastError%>
<b>Detalles del error que se ha producido</b><br>
Código de error ASP: <i><b><=%objASPError.ASPCode%></b></i><br>
Número de error: <i><b><=%objASPError.Number%></b></i><br>
Código fuente que lo ha producido:
    <i><b><=%Server.HTMLEncode(objASPError.Source)%></b></i><br>
Categoría del error: <i><b><=%objASPError.Category%></b></i><br>
Fichero en el que se producido el error: <i><b><=%objASPError.File%></b></i><br>
Línea y columna en la que se ha producido el error:
    <i><b><=%objASPError.Line%, <=%objASPError.Column%></b></i><br>
Descripción del error: <i><b><=%objASPError.Description%></b></i><br>
</body>
</html>
```

Código fuente 11

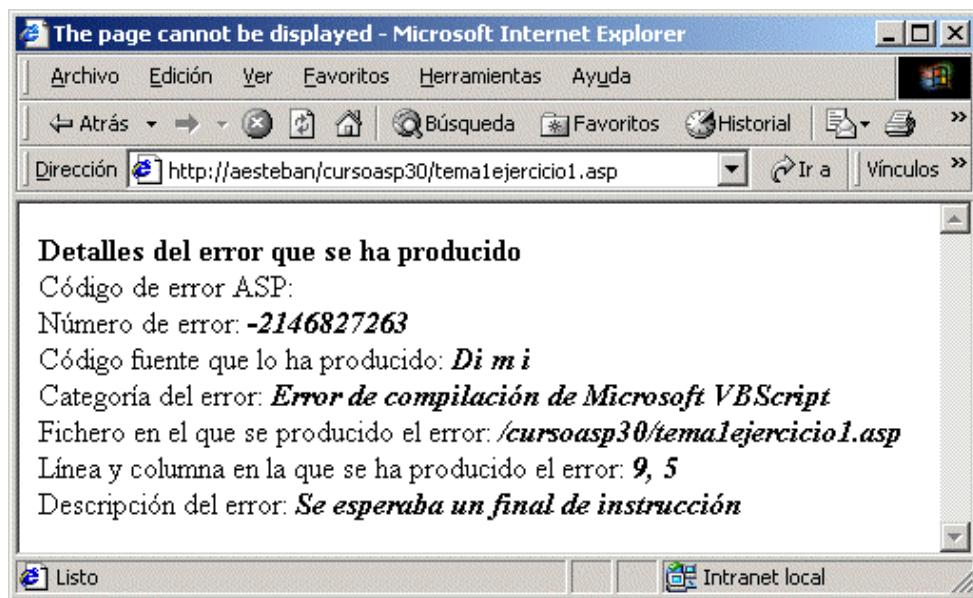


Figura 6

Un ejemplo de la ejecución del código anterior se puede ver en la Figura 6, y se produce cuando hay un error de ASP, es decir un error interno de servidor del tipo 500;100.

En el capítulo dedicado al objeto ASPError comentaremos en detalle el esquema de tratamiento de errores desde ASP 3.0 e IIS 5.0.

La página para el tratamiento de errores se puede obtener en este [enlace](#).

## Componente de registro de IIS (Logging Utility)

Otras de las novedades de ASP 3.0, abandonado ya los objetos integrados en ASP, es que ofrece un nuevo componente de servidor (componente ActiveX de servidor). El nuevo componente de servidor es denominado componente de registro o programa de registro de IIS. Mediante este componente tenemos acceso a la información y manipulación de los ficheros de registro (log) generados por el servidor Web IIS 5.0.

Este componente, al igual que todos los existentes en ASP 2.0, se instala conjuntamente con el servidor Web Internet Information Server 5.0. El fichero DLL que contiene a este nuevo componente es logscript.dll.

Para instanciar un componente de registro debemos utilizar la sentencia que muestra el Código fuente 12.

```
Set objRegistro=Server.CreateObject ("MSWC.IISLog")
```

Código fuente 12

Es importante señalar que el usuario que tiene acceso a la secuencia de comandos ASP que crea la instancia del componente de registro debe autenticarse como Administrador u Operador en el servidor donde se ejecuta IIS, si es un usuario anónimo, el componente de registro de IIS no funcionará correctamente. Para manipular los ficheros de registro de IIS el componente IISLog ofrece una serie de métodos que se muestran en la Tabla 2.

Método	Descripción
AtEndOfLog	Indica si se leyeron o no todos los registros del archivo.
CloseLogFile	Cierra todos los archivos de registro abiertos.
OpenLogFile	Abre un archivo de registro para lectura o escritura.
ReadFilter	Filtira los registros del archivo según la fecha y la hora.
ReadLogRecord	Lee el siguiente registro disponible del archivo de registro actual.
WriteLogRecord	Escribe un registro en el archivo actual.

Tabla 2

Para obtener la información del registro actual el componente IISLog ofrece veinte propiedades de sólo lectura que se corresponden con los distintos campos de un registro de un archivo de registro. Para obtener la información del registro actual el componente IISLog ofrece veinte propiedades de sólo lectura que se corresponden con los distintos campos de un registro de un archivo de registro.

<b>Propiedad</b>	<b>Descripción</b>
BytesReceived	Número de bytes recibidos del navegador como una petición.
BytesSent	Número de bytes enviados al navegador como una respuesta.
ClientIP	Dirección IP del cliente.
Cookie	Indica los contenidos de cualquier cookie enviada en la petición.
CustomFields	Un vector de cabeceras personalizadas que se añadieron a la petición.
DateTime	La fecha y hora de la petición en formato GMT.
Method	El tipo de operación, tal como puede ser GET o POST.
ProtocolStatus	El mensaje de estado devuelto al cliente, por ejemplo 200 OK.
ProtocolVersion	Una cadena con la versión del protocolo utilizado, por ejemplo HTTP/1.1.
Referer	La URL de la página que contiene el enlace que inició la petición, si está disponible.
ServerIP	La dirección IP del servidor Web.
ServerName	El nombre del servidor Web.
ServerPort	El número de puerto por el que se recibió la petición.
ServiceName	Nombre del servicio, como puede ser el servicio FTP (MSFTPSVC) o Web (W3SVC).
TimeTaken	El tiempo de procesamiento total para devolver y crear la página devuelta.
URIQuery	Cualquier parámetro añadido a la cadena de consulta (QueryString) de la URL en la petición.
URIStem	La URL que demandó el cliente.
UserAgent	La cadena de agente de usuario (tipo de navegador) enviada por el cliente.
UserName	Nombre de inicio de sesión del usuario si no ha accedido de forma anónima.
Win32Status	Código de estado Win32 después de haber procesado la petición.

Tabla 3

Se puede configurar el tipo de registro que queremos en nuestro servidor a través de IIS 5.0, de esta forma podremos añadir o eliminar de nuestro fichero de registro los campos descritos anteriormente. Para ello acudiremos a las propiedades del sitio Web y en la pestaña sitio Web pulsaremos el botón Propiedades contenido en el epígrafe de Habilitar registro. Se debe seleccionar uno de los formatos de registro que se corresponden con un fichero de registro, por lo tanto la opción registro ODBC no sería válida.

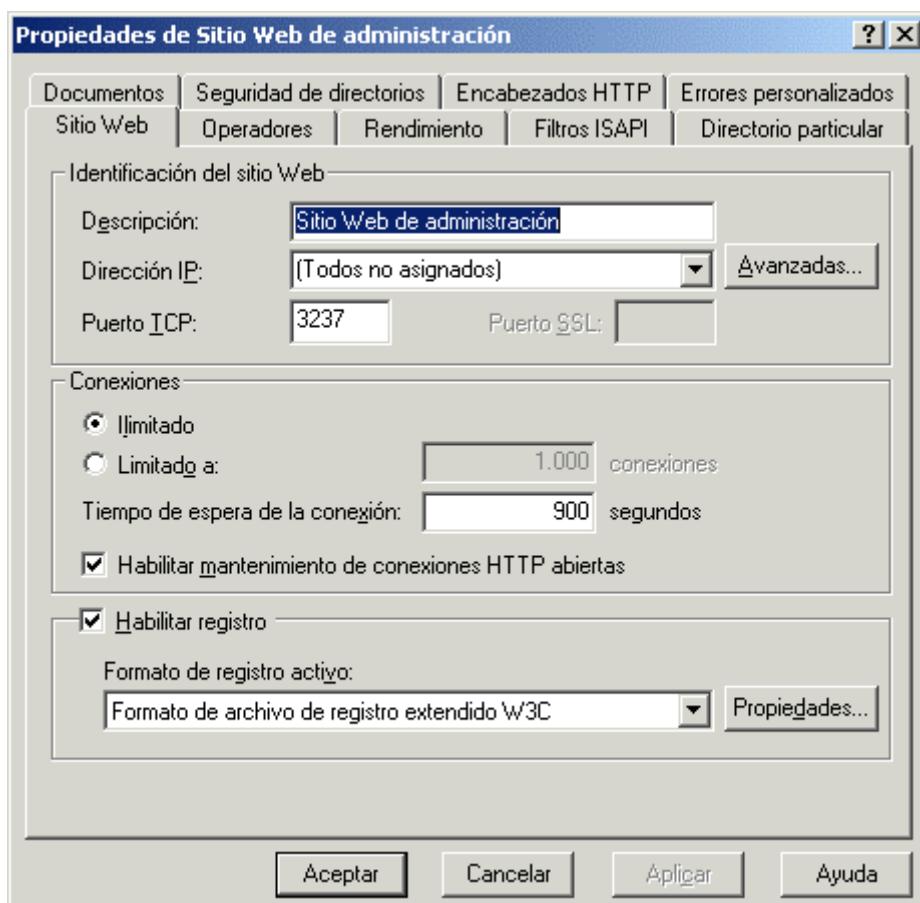


Figura 7

En el Código fuente 13 se muestra la utilización de este nuevo objeto ActiveX de servidor. En este sencillo código se utiliza el componente de registro para mostrar algunos de los campos contenidos en el fichero de registro.

```

<html>
<head>
<!--METADATA TYPE="TypeLib" FILE="c:\winnt\system32\inetsrv\logscript.dll"-->
</head>
<body>
<%Set objRegistro=Server.CreateObject("MSWC.IISLog")%
objRegistro.OpenLogFile "c:\winnt\system32\logfiles\w3svc1\ex000517.log"_
,ForReading,"W3SVC",1,0
objRegistro.ReadFilter DateAdd("d",-1,Now),Now%>
<table align="center" border="0" cellspacing="2" cellpadding="5">
<tr>
    <th>Fecha/Hora</th>
    <th>IP del cliente</th>

```

```

<th>Método</th>
<th>URL</th>
</tr>
<%While Not objRegistro.AtEndOfLog
    objRegistro.ReadLogRecord%>
    <tr>
        <td><%=objRegistro.DateTime%></td>
        <td><%=objRegistro.ClientIP%></td>
        <td><%=objRegistro.Method%></td>
        <td><%=objRegistro.URIStem%></td>
    </tr>
<%Wend
objRegistro.CloseLogFile(ForReading)%>
</body>
</html>

```

Código fuente 13

Se ha utilizado un filtro para recuperar la información del fichero de registro referente al servicio Web y únicamente de las últimas 24 horas. También se puede observar que se utiliza una directiva METADATA, más tarde comentaremos su utilidad y sintaxis, de momento diremos únicamente que nos permite incluir las constantes definidas en la librería que contiene al componente de registro.

La información que se va a mostrar del fichero de registro va a ser la fecha y hora de la petición, la dirección IP del cliente que ha realizado la petición, el método que se ha utilizado y la URL correspondiente. En la Figura 8 se puede ver un ejemplo de ejecución de la página anterior.

Fecha/Hora	IP del cliente	Método	URL
17/05/2000 8:02:22	192.168.4.41	OPTIONS	/
17/05/2000 8:02:23	192.168.4.41	GET	/_vti_inf.html
17/05/2000 8:02:23	192.168.4.41	POST	/_vti_bin/shtml.dll
17/05/2000 8:02:23	192.168.4.41	POST	/_vti_bin/shtml.dll
17/05/2000 8:02:24	192.168.4.41	POST	/cursoASP30/_vti_bin/_vti_aut/author.dll
17/05/2000 8:02:24	192.168.4.41	POST	/cursoASP30/_vti_bin/_vti_aut/author.dll
17/05/2000 8:02:24	192.168.4.41	POST	/cursoASP30/_vti_bin/_vti_aut/author.dll
17/05/2000 8:02:24	192.168.4.41	OPTIONS	/cursoASP30/paginainicial.asp

Figura 8

El nombre del fichero de registro variará según sea nuestra configuración del registro en el sitio Web correspondiente, la ubicación de estos ficheros de registro suele ser el directorio c:\winnt\system32\logfiles\w3svc1 para el servicio Web.

Esta página ASP que utiliza el componente de registro se puede utilizar únicamente restringiendo el acceso anónimo a la propia página o al directorio que la contiene a nivel de permisos de NTFS, en caso contrario no podremos acceder al fichero de registro, ya sea para leer o escribir datos.

## ActiveX Data Objects 2.5 (ADO 2.5)

Junto con ASP 3.0 se ofrece la nueva versión de los componentes de servidor para el acceso a datos, es decir, ActiveX Data Objects 2.5 (ADO 2.5).

Lo más destacable es que ADO 2.5 amplía su modelo de objetos con dos objetos más: Record y Stream.

Hasta ahora mediante ADO accedíamos sobre todo a datos estructurados como puede ser un conjunto de registros de una tabla en una base de datos, pero también nos puede interesar acceder a datos que no sean tan homogéneos como puede ser un sistema de ficheros o un sistema de correo, en este caso estaremos ante datos semi-estructurados.

Una característica nueva que forma parte de los datos semi-estructurados es lo que se denomina Internet Publishing. La versión 2.1 de ADO ya ofrecía el proveedor OLE DB para Internet Publishing, pero con una funcionalidad muy limitada. Ahora en su nueva versión, ADO ofrece la funcionalidad completa.

Mediante el proveedor OLE DB para Internet Publishing podemos manipular recursos Web desde ADO, es decir, podemos construir nuestras propias aplicaciones para manipular sitios Web. Todas estas innovaciones entran dentro de la estrategia de acceso a datos de Microsoft, UDA (Universal Data Access).

Veamos ahora como se modela en objetos este nuevo acceso a datos semi-estructurados por parte de ADO.

Normalmente el almacenamiento semi-estructurado sigue una organización de árbol, con nodos, subnodos y archivos. Si consideramos un sitio Web vemos que presenta carpetas, subcarpetas y archivos. Si pensamos que este sistema de almacenamiento debe ser modelado mediante ADO, podemos elegir como candidato el objeto RecordSet, ya que puede contener un conjunto de datos.

Pero si lo pensamos más detenidamente vemos que un objeto RecordSet puede contener un conjunto de carpetas, pero luego cada una de las carpetas tendrá distintos archivos de distintos tipos cada uno con unas características.

Es en este momento cuando entra en juego el objeto Record. En la situación vista anteriormente, la colección de directorios y/o archivos será representada por un objeto Recordset pero cada directorio o archivo será representado mediante un objeto Record, ya que pueden tener propiedades únicas y distintas. Como se puede ver este es un nuevo concepto que puede ser necesario madurar por parte del lector.

Los valores de las propiedades que contiene el objeto Record referentes al archivo o directorio que representa, se pueden recuperar a modo de campos del objeto Record. Además, el objeto Record ofrece una serie de métodos que permiten manipular el registro: CopyRecord, MoveRecord y DeleteRecord.

Una vez visto la necesidad del objeto Record, para describir y representar las características únicas y variadas de cada elemento, es sencillo ver la utilidad del nuevo objeto Stream.

El objeto Stream permite acceder a los contenidos de cada elemento, que puede ser un directorio, un archivo, un mensaje de correo, etc. Para ello este objeto posee una serie de métodos como pueden ser: ReadText, WriteText, LoadFromFile, etc.

Para instanciar estos dos nuevos objetos de ADO utilizaremos las dos sentencias que muestra el Código fuente 14.

```
Set objRecord=Server.CreateObject ("ADODB.Record")
Set objStream=Server.CreateObject ("ADODB.Stream")
```

Código fuente 14

En primer lugar se va a ofrecer un código ASP que muestra los campos que posee un objeto Record que representa a un directorio de un sitio Web y otro que representa a un fichero, junto con sus valores correspondientes. Mediante el objeto Record vamos a abrir en primer lugar una URL y mostrar los campos que posee con sus valores y a continuación se hace lo mismo con un fichero.

```
<!--#INCLUDE FILE="ADOVBS.INC"-->
<%Set objRecord=Server.CreateObject ("ADODB.Record")
objRecord.Open "", "URL=http://aesteban/cursoASP30%">
<html>
<body>
<div align="center"><b>Campos de un directorio</b></div>
<table align="center" border="2" cellspacing="2" cellpadding="5">
<tr>
<th>Campo</th>
<th>Valor</th>
<%For Each campo in objRecord.Fields%>
<tr>
<td><%=campo.Name%></td>
<td><%=campo.Value%></td>
</tr>
<%Next%>
</table>
<%objRecord.Close
objRecord.Open "paginainicial.asp", "URL=http://aesteban/cursoASP30%">
<div align="center"><b>Campos de un fichero</b></div>
<table align="center" border="2" cellspacing="2" cellpadding="5">
<tr>
<th>Campo</th>
<th>Valor</th>
<%For Each campo in objRecord.Fields%>
<tr>
<td><%=campo.Name%></td>
<td><%=campo.Value%></td>
</tr>
<%Next%>
</table>
<%objRecord.Close
Set objRecord=Nothing%>
</body>
</html>
```

Código fuente 15

El resultado de la ejecución del código anterior se puede ver en la Tabla 4 y en la Tabla 5.

### Campos de un directorio

Campo	Valor
RESOURCE_PARSENAMESPACE	cursoASP30
RESOURCE_PARENTNAME	<a href="http://aesteban">http://aesteban</a>
RESOURCE_ABSOLUTEPARSENAME	http://aesteban/cursoASP30
RESOURCE_ISHIDDEN	
RESOURCE_ISREADONLY	
RESOURCE_CONTENTTYPE	
RESOURCE_CONTENTCLASS	
RESOURCE_CONTENTLANGUAGE	
RESOURCE_CREATIONTIME	
RESOURCE_LASTACCESSTIME	
RESOURCE_LASTWRITETIME	
RESOURCE_STREAMSIZE	
RESOURCE_ISCOLLECTION	True
RESOURCE_ISSTRUCTUREDDOCUMENT	
DEFAULT_DOCUMENT	
RESOURCE_DISPLAYNAME	cursoASP30
RESOURCE_ISROOT	True
RESOURCE_ISMARKEDFOROFFLINE	False

Tabla 4

### Campos de un fichero

Campo	Valor
RESOURCE_PARSENAMESPACE	PaginaInicial.asp
RESOURCE_PARENTNAME	http://aesteban/cursoASP30
RESOURCE_ABSOLUTEPARSENAME	http://aesteban/cursoASP30/PaginaInicial.asp

RESOURCE_ISHIDDEN	
RESOURCE_ISREADONLY	
RESOURCE_CONTENTTYPE	
RESOURCE_CONTENTCLASS	
RESOURCE_CONTENTLANGUAGE	
RESOURCE_CREATIONTIME	10/05/2000 14:29:24
RESOURCE_LASTACCESSTIME	
RESOURCE_LASTWRITETIME	10/05/2000 14:38:34
RESOURCE_STREAMSIZE	861
RESOURCE_ISCOLLECTION	False
RESOURCE_ISSTRUCTUREDDOCUMENT	
DEFAULT_DOCUMENT	
RESOURCE_DISPLAYNAME	PaginaInicial.asp
RESOURCE_ISROOT	False
RESOURCE_ISMARKEDFOROFFLINE	False

Tabla 5

El objeto Stream lo vamos a utilizar en un ejemplo que consiste en mostrar el contenido de un fichero en un área de texto. El objeto Stream representa el contenido del fichero y permite acceder al mismo.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<!-- INCLUDE FILE="ADOVBS. INC"-->
<%Set objStream=Server.CreateObject ("ADODB.Stream")>
objStream.Charset="ascii"
objStream.Open
"URL=http://aesteban/cursoASP30/adovbs.inc", adModeRead, adOpenStreamUnspecified%
<textarea cols="50" rows="20">
<%=objStream.ReadText%>
</textarea>
<%objStream.Close
Set objStream=Nothing%>
</BODY>
</HTML>
```

Código fuente 16

Para abrir un objeto Stream lo podemos hacer a partir del objeto Record que representa el fichero que queremos abrir, o como es el caso que nos ocupa, lo podemos utilizar directamente indicando la URL que indica el fichero que deseamos abrir.

En la Figura 9 se puede observar la ejecución del código anterior.

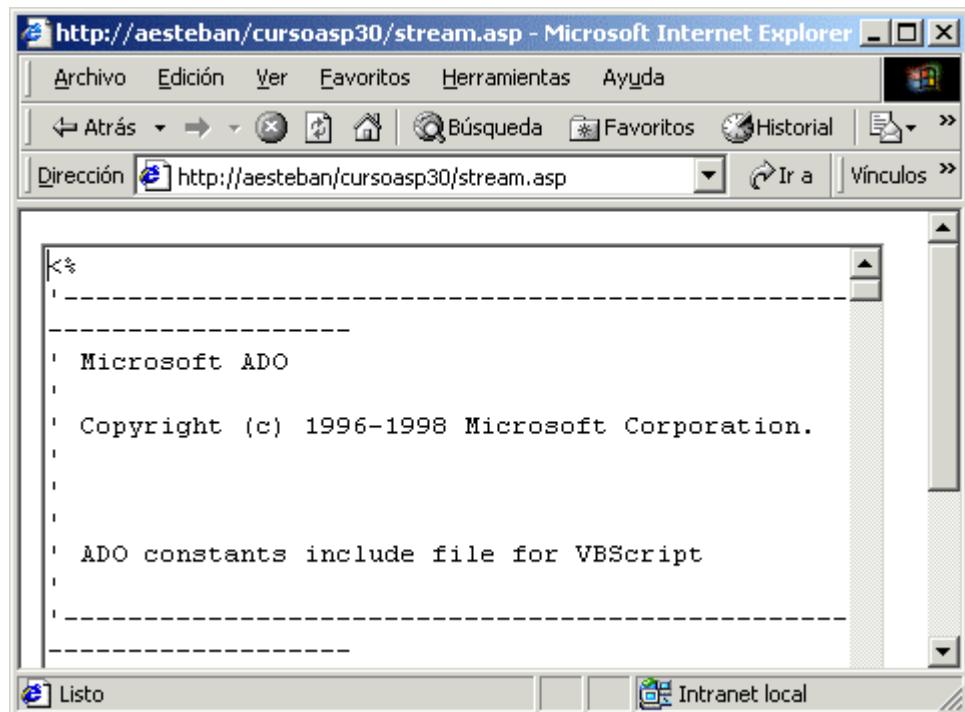


Figura 9

Se debe señalar que para utilizar con éxito los objetos Record y Stream desde ASP 3.0 se debe incluir la cuenta del usuario anónimo de Internet (IUSR\_nombreMaquina) dentro del grupo de Administradores. Esto supone un grave problema de seguridad y puede ser válido para servidores Web de prueba o de desarrollo, pero para servidores Web en producción resulta inadmisible. Supongo que se trata de un error en la versión 3.0 de ASP o bien un error del proveedor OLEDB Internet Publishing, pero lamentablemente hasta la fecha no existe otra solución.

## Aplicaciones ASP con IIS 5.0

En IIS 4.0 ya podíamos definir aplicaciones ASP y su directorio de inicio, también podíamos indicar a través de la configuración de los directorios de un sitio Web si la aplicación se ejecutaba en otro espacio de memoria como un proceso aislado.

Con IIS 5.0 el concepto de aplicación ASP no ha variado, es decir, una aplicación es un conjunto de páginas ASP que se ejecutan en un conjunto de directorios definidos dentro de un sitio Web, tampoco ha variado excesivamente la forma de configurar las aplicaciones ASP. Lo más destacable que ofrece IIS 5.0 con respecto a las aplicaciones ASP es la posibilidad de definir tres niveles de protección distintos para dichas aplicaciones.

En IIS 4.0 podíamos indicar que nuestra aplicación se ejecutara en el mismo espacio de memoria que el servidor Web o que se ejecutara en un proceso aislado, pero con IIS 5.0 tenemos otra posibilidad

intermedia, que consiste en que la aplicación se ejecuta en un proceso agrupado con el resto de las aplicaciones ASP.

Los tres grados de protección que ofrece IIS 5.0 para las aplicaciones ASP se denominan bajo, medio y alto.

Protección	Descripción
Baja (proceso IIS)	<p>Las aplicaciones ASP se ejecutan todas en el mismo espacio de memoria que el servidor Web IIS 5.0. Si una aplicación ASP falla afectará a todo el servidor Web, poniendo en peligro la ejecución de la aplicación InetInfo.exe, que es el ejecutable del servidor Web.</p> <p>Ofrece la ejecución más rápida y eficiente de las aplicaciones ASP, pero también la que ofrece más riesgos.</p>
Media (agrupada)	<p>Esta es la protección por defecto, todas las aplicaciones ASP se ejecutan agrupadas en un espacio de memoria distinto que el del servidor Web, en este caso todas las aplicaciones ASP del servidor Web utilizan una instancia compartida del ejecutable DLLHost.exe.</p> <p>De esta forma se protege al ejecutable InetInfo.exe de los posibles fallos de las aplicaciones ASP. Si se produce un fallo en una aplicación ASP no afecta al servidor Web, pero sí a resto de las aplicaciones ASP.</p>
Alta (aislada)	<p>Cada aplicación ASP se ejecuta en un espacio de memoria distinto, es decir, cada aplicación se ejecuta en una instancia distinta y exclusiva del ejecutable DLLHost.exe. De esta forma si una aplicación falla no afectará al resto de las aplicaciones ASP ni tampoco al servidor Web, ya que se ejecuta en su propio espacio de memoria.</p> <p>Microsoft recomienda que por cada servidor Web no se definan más de diez aplicaciones aisladas. Este tipo de protección es recomendable para aplicaciones ASP de alto riesgo o críticas.</p>

Tabla 6

Por defecto el sitio Web predeterminado se define como una aplicación ASP agrupada o con grado de protección medio, este es el modo de protección de la aplicación más usual y recomendable, ya que ofrece una buena relación en lo que a rendimiento y seguridad se refiere, con el grado de protección alto comprometemos el rendimiento y con el grado de protección bajo se compromete la seguridad del funcionamiento del servidor Web.

Como resumen de los distintos grados de protección con los que podemos configurar nuestras aplicaciones ASP diremos que lo recomendable es ejecutar el servidor Web (InetInfo.exe) en su propio proceso, ejecutar aplicaciones decisivas en sus propios procesos y ejecutar el resto de aplicaciones en un proceso agrupado y compartido.

En el capítulo monográfico dedicado a Internet Information Server 5.0 se volverá a comentar en detalle la configuración de aplicaciones ASP.

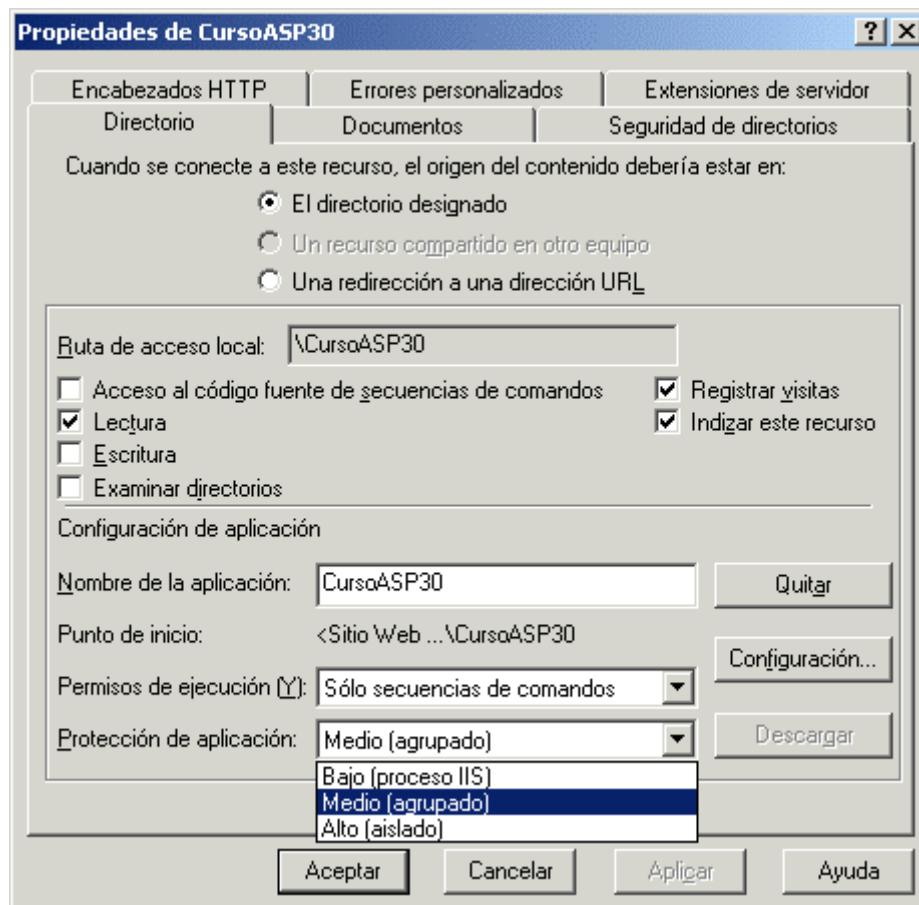


Figura 10

## VBScript 5.0

Hasta ahora hemos visto que en ASP 3.0 nos podemos encontrar nuevos objetos integrados, nuevos componentes de servidor, la nueva versión de los componentes de acceso a datos, y también, como vamos a ver en este apartado, una nueva versión del lenguaje de secuencia de comandos, VBScript 5.0.

Lo más destacable que ofrece VBScript 5.0 es la posibilidad de la utilización de clases, de la misma forma que lo hacíamos con su hermano mayor, Visual Basic, a excepción, claro está, de los eventos. Ahora ya podemos crear objetos de una clase determinada y definida por nosotros con sus métodos y propiedades correspondientes.

En el Código fuente 17 se ofrece la creación y utilización de una sencilla clase en VBScript.

```
<%Class MiClase
Private valor

Public Property Let MiValor(dato)
    valor=dato
End Property

Public Property Get MiValor()
    MiValor=valor
End Property
```

```

Public Function Cuadrado()
    Cuadrado=valor*valor
End Function
End Class
Set objMiClase=New MiClase
objMiClase.MiValor=4
Response.Write "El cuadrado de "&objMiClase.MiValor&" es "&objMiClase.Cuadrado()%>

```

Código fuente 17

Como podemos observar a la vista de la nueva capacidad de VBScript para la utilización de clases, existe una intención de equiparar (o por lo menos acercar) el lenguaje de secuencias de comandos VBScript con su hermano mayor Visual Basic. Esto también se hace patente en otra nueva característica de VBScript, que es la incorporación de la construcción With.

```

Set objeto=Server.CreateObject("Este.Objeto")
With objeto
    .Propiedad1="Valor 1"
    .Propiedad2="Valor 2"
End With

```

Código fuente 18

También se incorpora la función Eval dentro de VBScript para evaluar una expresión de cadena y devolver el valor verdadero o falso de dicha evaluación.

Otra nueva función de VBScript es Execute, que es utilizada para ejecutar el código contenido en una cadena, puede ser utilizada esta función para crear procedimientos de forma dinámica y ejecutarlos más tarde en el código de la secuencia de comandos.

Mediante el método nuevo método SetLocale podemos cambiar la localización de la secuencia de comandos en lo que se refiere a la utilización del conjunto de caracteres específicos de una localización.

Y para finalizar con las novedades que aporta VBScript 5.0 diremos que también soporta la utilización de expresiones regulares. Para ello se ha incorporado un nuevo objeto dentro de VBScript, denominado RegExp.

```

<%cadena="prueba nmd dsdjjdh sd dshsdjhd prueba ddsdnmn pruebadrtt"
Set objExpresion=New RegExp 'se crea la expresión regular

objExpresion.Pattern="prueba*" 'se configura la expresión
objExpresion.IgnoreCase=False
objExpresion.Global=True
Set coincidencias=objExpresion.Execute(cadena) 'se ejecuta la búsqueda

For each Match in coincidencias 'se muestran las coincidencias de la expresión
    Response.Write "Coincidencia encontrada en la posición "&Match.FirstIndex&"<br>"
Next%>

```

Código fuente 19

El resultado de la ejecución del Código fuente 19 es:

Coincidencia encontrada en la posición 0  
Coincidencia encontrada en la posición 31  
Coincidencia encontrada en la posición 46

## Páginas ASP codificadas

Gracias a la nueva versión del servidor Web de Microsoft, Internet Information Services 5.0, disponemos de una nueva característica bastante interesante, que consiste en poder codificar el código fuente de nuestras páginas ASP.

Si pretendemos distribuir una aplicación ASP y entregársela a un cliente, no ocurre como con las aplicaciones tradicionales, que entregamos una instalación que instalará en el cliente varios ejecutables o librerías, pero nuestros ficheros fuente quedan protegidos, ya que el cliente no puede acceder a los mismos. Sin embargo con nuestra aplicación ASP los ficheros fuente, que son nuestras páginas ASP, están a la vista del cliente sin ningún tipo de protección. Por lo tanto existe la necesidad de codificar nuestras secuencias de comandos de ASP.

Mediante la herramienta Windows Script Encoder podemos codificar las páginas ASP que escribamos y de esta forma podremos proteger nuestro código fuente para que no sea copiado o manipulado por terceras personas. Aunque se debe señalar que esta codificación no supone una solución segura y decifrable, pero puede evitar que muchos usuarios ocasionales exploren y copien secuencias de comandos.

Al aplicar la codificación las secuencias de comandos de cliente y de servidor aparece la lógica de programación como caracteres ASCII ilegibles, será el motor de secuencias de comandos el que descodifique el código fuente en tiempo de ejecución.

La herramienta Windows Script Encoder se puede encontrar en la siguiente dirección de Microsoft: <http://msdn.microsoft.com/scripting/default.htm?/scripting/vbscript/download/vbsdown.htm>, al instalar esta herramienta también se instala la ayuda correspondiente con ejemplos de uso, se trata de una sencilla herramienta que se debe ejecutar desde la línea de comandos con los parámetros correspondientes.

Se debe señalar y advertir, que una vez codificadas las páginas ASP no se puede recuperar el código original, es decir, no se pueden decodificar, por lo tanto es recomendable conservar una copia de las páginas ASP originales para el desarrollo de la aplicación ASP y tener otra copia para la aplicación ASP en producción con las páginas ASP codificadas.

## Otros cambios

En este último apartado se recogen algunos cambios que presenta ASP 3.0 y que no se han clasificado en ninguno de los apartados anteriores.

En la versión anterior de ASP si necesitábamos utilizar las constantes definidas en una librería de componentes, como puede ser ADO, teníamos que incluir un fichero con la definición de dichas constantes mediante la también conocida directiva INCLUDE de ASP, en el caso de ADO se trataba del famoso fichero ADOVBS.INC. Pero con ASP 3.0 esta situación ha cambiado, podemos incluir las constantes definidas en una librería de forma directa desde la propia librería, sin tener que crear un fichero de definición de constantes diferenciado.

Para incluir una referencia a una librería un componentes utilizamos la nueva directiva METADATA, cuya sintaxis se muestra a continuación.

```
<!-- METADATA TYPE="TypeLib"
    FILE="camino y nombre del fichero"
    UUID="identificador de la librería"
    VERSION="numVersionMayor.numVersionMenor"
    LCID="identificador de localización"
-->
```

Las propiedades que son obligatorias son FILE o UUID, siempre deberemos indicar uno u otro para identificar la librería, el resto de las propiedades son de tipo opcional De esta forma para incluir las constantes de ADO y poder utilizarlas, escribiremos lo que indica el Código fuente 20.

```
<!-- METADATA TYPE="TypeLib"
    FILE="c:\Archivos de Programa\Archivos
comunes\System\ado\msado15.dll"
-->
```

Código fuente 20

Se debe señalar que o bien podemos utilizar la directiva METADATA en cada una de las páginas ASP en las que necesitemos incluir las constantes o también se puede incluir en el fichero GLOBAL.ASA y de esta forma estar disponible la definición de constantes y la referencia a la librería para todas las páginas de la aplicación ASP. Como curiosidad hago notar que le nombre de la librería de ADO sigue siendo msado15.dll cuando lógicamente debería ser msado25.dll, ya que ya nos encontramos en la versión 2.5 de ADO, aunque de todas formas esta librería contiene la última versión de ADO.

Para incluir archivos en nuestras páginas ASP ya hemos visto que utilizamos la directiva INCLUDE, pero en ASP 3.0 hay otra alternativa que es la utilización de la etiqueta <SCRIPT> como muestra el Código fuente 21.

```
<SCRIPT RUNAT="SERVER" SRC="ruta relativa, física o virtual al fichero de
scrip"></SCRIPT>
```

Código fuente 21

El fichero que incluimos, a diferencia de la directiva INCLUDE, únicamente puede contener secuencias de comandos, no puede contener texto ni código HTML, además no debe existir ningún elemento entre las etiquetas <SCRIPT></SCRIPT>.

Si un usuario accede a un sitio Web indicando únicamente el nombre del mismo, sin indicar ninguna página, se enviará al usuario el documento o página por defecto. Sin embargo, si se añadía una cadena de consulta (QueryString) a esta URL en la versión anterior de ASP esta cadena era ignorada. Pero ahora con ASP 3.0 y combinación con IIS 5 la cadena de consulta si es considerada por la página predeterminada. De esta forma en ASP 3.0 escribir <http://www.eidos.es/?prueba=true> es equivalente a escribir la URL <http://www.eidos.es/default.asp?prueba=true>, siendo default.asp la página predeterminada del sitio Web del Grupo EIDOS.



# 3

## Lenguaje de Script: VBScript

---

### Introducción

En este capítulo vamos a tratar el lenguaje de secuencias de comandos, o lenguaje de script, utilizado para crear páginas activas de servidor (Active Server Pages, ASP).

Para desarrollar aplicaciones ASP se debe utilizar un lenguaje de script como puede ser VBScript, JavaScript/Jscript, Perl, etc; no debemos olvidar que ASP sólo nos ofrece un entorno para que se ejecute el script de servidor, ASP por si sólo no es un lenguaje de script.

Los lenguajes de script que se pueden utilizar directamente y que son interpretados por ASP sin necesidad de ningún intérprete adicional son VBScript y JScript, si se quiere utilizar otro lenguaje se deberá instalar el intérprete correspondiente.

El lenguaje de script utilizado por la página ASP correspondiente se especifica a través de la sentencia `<%@ LANGUAGE="VBSCRIPT" %>`, esta sentencia se debe situar al comienzo de la página ASP. En el caso de que no se indique que lenguaje de script se va a utilizar, ASP tomará el lenguaje por defecto, que es VBScript.

Debido a que el lenguaje por defecto de las páginas activas es VBScript, ha sido este lenguaje de script el elegido para este tema, y durante todo el presente curso se utilizará VBScript.

VBScript (Visual Basic Script) es un subconjunto de lo que se denomina VBA (Visual Basic for Application) que, a su vez es un subconjunto del lenguaje Visual Basic. VBScript fundamentalmente se emplea para integrarse dentro de páginas HTML para ampliar las características de un sitio Web.

Si el lector ya conoce Visual Basic verá que aprender la sintaxis del lenguaje de secuencias de comandos VBScript es muy sencillo.

## Diferencias entre los Scripts de cliente y de servidor

Como indica el título de este apartado podemos distinguir dos tipos de scripts: script de cliente y script de servidor, no sólo se diferencian en el lugar en que se ejecutan (máquina del cliente y máquina servidor) sino también en las funciones que desempeñan cada uno de ellos.

En el script de cliente las secuencias de comandos se ejecutan en la máquina local del usuario, este tipo de script puede ser utilizado para la animación de páginas Web o para la validación local de datos y recoger eventos proporcionados por la interacción del usuario con la página. Por lo tanto el script de cliente será interpretado por el navegador Web del cliente y debido a esto el script de cliente depende del navegador Web.

El script de servidor es compilado y procesado por el servidor Web antes de que la página sea enviada al navegador, el uso de este script está orientado a tareas más complejas como puede ser páginas Web basadas en acceso a bases de datos.

Otra diferencia importante es que no se tiene acceso a los eventos del usuario, ya que cuando la página llega al usuario el script ya ha sido ejecutado por el servidor Web, y el usuario visualiza en el navegador el resultado de la ejecución del script de servidor. De esta forma, el script de servidor depende del servidor Web para su ejecución.

El script de cliente se encuentra encerrado entre las etiquetas <SCRIPT></SCRIPT> de HTML y el script de servidor se encuentra entre los delimitadores <% %> o también <%=%>, precisamente son estos delimitadores los que indican al servidor Web que el script debe ser ejecutado en el servidor.

Aunque también nos podemos encontrar el script de servidor entre las etiquetas <SCRIPT RUNAT="SERVER"></SCRIPT>. Como se puede observar con la propiedad RUNAT de la etiqueta <SCRIPT> indicamos que el script se ejecute en el servidor al asignarle el valor "SERVER". Esta forma de indicar el script de servidor es menos común, lo más normal es encerrar el script de servidor entre los delimitadores <%=%> ó <%/%>.

Las diferencias entre los dos tipos de scripts se desprenden de las diferentes funciones que cubren cada uno de ellos, así por ejemplo el script de cliente lo utilizaremos en los siguientes casos:

- Validación de formularios dinámica.
- Desarrollo de páginas Web interactivas.
- Proveer al cliente con avanzados controles ActiveX.
- Controlar el navegador del usuario.
- Reaccionar ante los eventos proporcionados por el usuario, es decir, gestión de eventos.

Y por otro lado, el script de servidor lo utilizaremos en los siguientes otros casos:

- Nuestro sitio Web debe funcionar en cualquier navegador del mercado, y no se requiere realizar una versión diferente por cada navegador. Se debe tener en cuenta que VBScript utilizado como script de cliente únicamente es reconocido por Internet Explorer, aunque se puede añadir un plug-in a Netscape Communicator para que lo interprete también.

- Deseamos proteger nuestro código fuente. En el caso del script de cliente, todo el código fuente es enviado al navegador.
- Si queremos conservar y seguir información del usuario a través de varias páginas Web.
- Interacción con bases de datos en el servidor.
- Necesitamos instanciar componentes ActiveX de servidor para realizar ciertas tareas.

La principal desventaja del script de servidor, o más bien inconveniente, es la siguiente:

- No hay un control directo sobre el interfaz de usuario. Por ejemplo, desde el script de servidor no se puede llamar a la función MsgBox para mostrar un mensaje, tampoco se pueden atrapar los eventos provocados por el usuario, por ejemplo la pulsación de un botón o el cierre de una ventana.

Y por otro lado tenemos las ventajas:

- El script de servidor es independiente del navegador, ya que el script es procesado completamente por el servidor Web. Una de las funciones que puede desempeñar un script de servidor es la de formatear el código HTML que va a ser enviado al navegador, es decir, se puede modificar las páginas HTML antes de que sean transmitidas al cliente.
- Se pueden utilizar variables globales para mantener el estado. Una de las desventajas del script de cliente es que no existe una forma sencilla de mantener variables entre una página y otra. Pero el script de servidor permite almacenar variables durante toda la sesión del usuario con nuestro sitio Web, esto se consigue a través de uno de los objetos intrínsecos de ASP, el objeto Session, es un proceso bastante sencillo que puede ocupar media línea de código: Session("nombre")="Pepe".
- Creación dinámica de páginas HTML basadas en la entrada del usuario. Por ejemplo, se puede modificar el aspecto o color de la página según haya sido la elección del usuario, también se puede insertar el contenido de las variables del script dentro del código HTML. También se puede enviar al usuario a diferentes páginas basándonos en los datos ofrecidos por el mismo.
- Es más seguro que el script de cliente, ya que todo lo que se encuentre entre los delimitadores <%%> será ejecutado por el servidor y nunca enviado al navegador.
- A través del script de servidor se puede acceder a bases de datos de forma muy sencilla a través de ADO (ActiveX Data Objects). Todo el proceso con la base de datos es realizado por el script de servidor.
- Se pueden utilizar controles o componentes ActiveX en el servidor sin que tengan que ser enviados al cliente. Con IIS y Visual Interdev se incluyen algunos controles ActiveX de servidor (ADO, Advertisement Rotator, Browser Capabilities, Text Stream...). Estos controles están listos para ser utilizados y son muy fáciles de usar, pero también se pueden crear controles ActiveX de servidor propios.
- Los scripts de servidor son más rápidos que los programas CGI (Common Gateway Interface). Esto es debido a que la DLL (Dynamic Link Library) utilizada por ASP está ya cargada en memoria esperando ejecutar los scripts. Los programas CGI son normalmente ejecutables que se deben cargar cada vez desde el disco.

- Existen numerosas variables HTTP con las que se comunican entre sí el servidor Web y el navegador. Estas variables van desde la dirección IP del usuario hasta el tipo de navegador que está utilizando. Esta información puede ser utilizada por el script de servidor a través del modelo de objetos de ASP.
- Se pueden identificar las propiedades del navegador mediante el componente Browser Capabilities incluido en ASP, y además ofrece una descripción de las capacidades del navegador. Utilizando este componente se podrá determinar si el usuario dispone del navegador Internet Explorer o Netscape Communicator, además, se podrá conocer la versión del navegador, si soporta o no Java, si soporta ActiveX, VBScript, etc.

Centrándonos ahora en el lenguaje de script que vamos a utilizar, es decir, VBScript, en lo que se refiere a la sintaxis y a las funciones que se utilizan en VBScript en el lado del cliente y en el lado del servidor su utilización es idéntica, la única diferencia es que en el script de servidor no se pueden utilizar instrucciones que presentan elementos de interfaz de usuario como pueden ser las funciones InputBox y MsgBox, tampoco se admiten las funciones CreateObject y GetObject, y en el script de cliente no se puede tener acceso al modelo de objetos (objetos integrados) de ASP.

En este curso nos vamos a centrar en las secuencias de comandos de servidor, ya que son con las que crearemos nuestras páginas activas.

La versión de VBScript utilizada por ASP 3.0 es la versión 5.1 del lenguaje de secuencias de comandos.

## Características generales de VBScript

En este apartado se enumeran brevemente algunas de las características más destacables del lenguaje de secuencias de comandos VBScript:

- Manejo de arrays: se pueden manipular arrays de hasta 60 dimensiones.
- Colecciones: se crean colecciones definidas por el usuario.
- Tipos de datos: sólo existe el tipo Variant, aunque también existen subtipos. Un tipo Variant es una clase especial de tipo de datos que puede contener diferentes tipos de información, dependiendo de cómo se utilice. El tipo Variant se comportará como un tipo determinado dependiendo del contexto en que se utilice.
- Depuración: no existe propiamente un entorno de depuración para VBScript, aunque las últimas versiones de Internet Explorer incluyen un depurador.
- Gestión de errores: se puede acceder al objeto Err y usar la instrucción On Error Resume Next.
- Acceso a archivos locales: no está permitido para evitar los posibles daños en la máquina del cliente, pero si se puede acceder al sistema de archivos del servidor mediante el objeto FileSystemObject.
- Tipos definidos por el usuario: no se permiten.
- Soporta la definición de clases con propiedades (atributos) y métodos, pero no permite definir eventos como ocurre con Visual Basic.

- Ofrece una serie de objetos dentro de la librería de script: Dictionary, FileSystemObject, RegExp..., estos objetos los veremos con detalle en siguientes capítulos.

## Tipos de datos de VBScript

Como ya se ha comentado anteriormente, sólo existe un tipo de datos, el tipo Variant. Este tipo de datos puede contener diferentes subtipos. El subtipo dependerá del contexto en el que se esté utilizando el tipo Variant.

La Tabla 7 muestra varios subtipos de datos que puede contener un tipo Variant.

Subtipo	Descripción
Empty	Variant está sin inicializar. El valor es 0 para las variables numéricas o una cadena de longitud cero("") para variables de cadena.
Null	Variant contiene datos no válidos.
Boolean	Contiene True o False.
Byte	Contiene un entero desde 0 hasta 255
Integer	Contiene un entero desde -32.768 hasta 32.767
Currency	Valores monetarios desde -922.337.203.685.477,5808 hasta 922.337.203.685.477,5807.
Long	Contiene un entero desde -2.147.483.648 hasta 2.147.483.647.
Single	Contiene un número de signo flotante y precisión simple desde -3,402823E38 hasta -1,401298E-45 para valores negativos y desde 1,401298E-45 hasta 3,402823E38 para valores positivos.
Double	Contiene un número de signo flotante y precisión doble desde -1,79769313486232E308 hasta -4,94065645841247E-324 para valores negativos y desde 4,94065645841247E-324 hasta 1,79769313486232E308 para valores positivos.
Date(Time)	Contiene un número que representa una fecha entre el 1 de enero de 100 y el 31 de diciembre de 9999.
String	Contiene una cadena de longitud variable que puede contener hasta 2 mil millones de caracteres de longitud.
Object	Contiene la referencia a un objeto.
Error	Contiene un número de error

Tabla 7

Se pueden realizar conversiones entre los diferentes subtipos utilizando una serie de funciones de conversión (CDate, CBool, CInt,CStr, etc.).

Para obtener el subtipo de una variable determinada poseemos utilizar las funciones VarType y TypeName. La primera de ellas devuelve un entero que se corresponde con la constante que representa al tipo de variable, y la segunda función devuelve una cadena que especifica el subtipo de la variable.

## Variables de VBScript

Una variable es un puntero que hace referencia a una posición de memoria a la que se le puede asignar un valor o recuperar un valor. En VBScript, las variables son siempre del tipo de datos Variant.

La declaración de cualquier variable se produce con la sentencia Dim. VBScript por defecto no obliga a declarar variables, aunque es aconsejable hacerlo para mejorar el código. Para forzar que VBScript obligue a declarar todas las variables, se debe utilizar la sentencia Option Explicit. En el Código fuente 22 se puede ver como se declaran las variables, se les asigna y un valor y se concatenan:

```
<%  
'Declaración de las variables  
Dim saludo1, saludo2, saludoCompleto  
'Asignación de valores a las variables  
saludo1="Buenas "  
saludo2="tardes."  
'Concatenación de las dos variables  
saludoCompleto=saludo1 & saludo2  
%>
```

Código fuente 22

En el ejemplo anterior también se puede observar como se indican los comentarios en VBScript, con el apóstrofe (') o bien con la palabra reservada REM. Los comentarios son únicamente para una línea de código.

Para declarar una variable como un array (vector) se debe indicar entre paréntesis el número de elementos que va a contener. Así por ejemplo, si queremos un array de una dimensión con 11 elementos deberemos escribir la sentencia que muestra el Código fuente 23.

```
Dim vector(10)
```

Código fuente 23

Todos los arrays en VBScript son de base cero, es decir, el primer elemento es el cero, por lo que si escribimos 10 en realidad se está indicando que va a tener el array 11 elementos. Para asignar valores al array se deben utilizar índices para acceder a sus elementos debemos escribir lo que muestra el Código fuente 24.

```
vector(0)=10  
vector(1)=20  
vector(2)=30
```

Código fuente 24

Del mismo modo, también se pueden recuperar los valores del elemento deseado del array.

```
elemento=vector(3)
```

Código fuente 25

Si queremos declarar una array de dos dimensiones separaremos por comas el número de filas y el de columnas que tendrá la matriz. Se pueden declarar arrays de hasta 60 dimensiones, aunque normalmente lo máximo son 3 o 4 dimensiones. Así por ejemplo, si queremos declarar una matriz bidimensional con 6 filas y 11 columnas escribiremos la sentencia que muestra el Código fuente 26.

```
Dim matriz(5,10)
```

Código fuente 26

También se pueden declarar arrays cuyo tamaño puede cambiar en tiempo de ejecución, para ello deberemos dejar los paréntesis utilizados en la declaración vacíos y utilizar Dim o ReDim.

```
Dim vector()  
ReDim vector()
```

Código fuente 27

Para utilizar el array dinámico se deberá utilizar ReDim para indicar el número de dimensiones y de elementos. El tamaño del array se puede cambiar tantas veces como sea necesario, pero si se quiere conservar el contenido del array deberá utilizar la palabra clave Preserve, pero si el nuevo tamaño del array es menor que el que poseía antes se perderán los datos de los elementos eliminados. Si utilizamos Preserve sólo podremos cambiar el tamaño de la última dimensión del array y no pueden cambiar el número de dimensiones.

```
ReDim vector(25)  
...  
ReDim Preserve vector(30)
```

Código fuente 28

Aunque este apartado está dedicado a las variables en general, al hablar de los array vamos a comentar de forma breve algunas de las sentencias y funciones que ofrece VBScript para manipular y utilizar variables de tipo array.

- IsArray: función que devuelve verdadero o falso dependiendo si la variable es un array o no (ej: If IsArray(variable) Then).

- Erase: sentencia que elimina el contenido de un array liberando la memoria ocupada por cada uno de sus elementos (ej: Erase variable).
- LBound: función que devuelve el índice menor de un array.
- UBound: función que devuelve el índice mayor de un array.
- Array: función que devuelve una variable array con todos los elementos que se le pasan por parámetro. Ej.:

```
Dim A
A = Array(10,20,30)
B = A(2)  ' el valor de B es 30
```

Retomemos de nuevo el tema principal del presente apartado.

En cuanto a la visibilidad de las variables, de momento vamos a distinguir dos niveles, uno a nivel de la página ASP y otro a nivel de procedimiento o función. Una variable declarada en un script es visible desde el momento de su declaración hasta el fin del script, es decir, en toda la página ASP, y una variable declarada dentro de un procedimiento o función sólo es visible dentro del procedimiento. En próximos capítulos veremos que hay dos niveles más de visibilidad de una variable, a nivel de sesión y a nivel de aplicación.

## Constantes de VBScript

Una constante es un nombre significativo que contiene un número o una cadena que nunca cambia. Para crear constantes se debe utilizar la palabra reservada Const, una vez creada la constante se le da un valor que puede ser numérico o un literal. Los literales de cadena se encierran entre comillas ("") y los de fecha entre almohadillas (#).

```
Const Cadena
Cadena="Una cadena"
Const num
num=30
Const fecha
fecha=#1-3-98#
```

Código fuente 29

## Operadores de VBScript

VBScript tiene un completo conjunto de operadores, incluyendo operadores aritméticos, de comparación y lógicos. Cuando se producen varias operaciones en una expresión, cada parte se evalúa y se resuelve en un orden predeterminado. Este orden se conoce como prioridad de los operadores. Se pueden utilizar paréntesis para invalidar el orden de la prioridad y obligar a que se evalúen algunas partes de una expresión antes que otras.

Cuando las expresiones contienen operadores de más de una categoría, se evalúan primero los aritméticos, a continuación los de comparación y por último los lógicos.

En la Tabla 8 aparecen los diferentes operadores.

Aritmético		De comparación		Lógicos	
Descripción	Símbolo	Descripción	Símbolo	Descripción	Símbolo
Exponenciación	$^$	Igualdad	=	Negación lógica	Not
Negación unaria	-	Desigualdad	$\diamond$	Conjunción lógica	And
Multiplicación	*	Menor que	<	Disyunción lógica	Or
División	/	Mayor que	>	Exclusión lógica	Xor
División Entera	\	Menor o igual que	$\leq$	Equivalencia lógica	Eqv
Módulo aritmético	Mod	Mayor o igual que	$\geq$	Implicación lógica	Imp
Suma	+	Equivalencia de objeto	Is		
Resta	-				

Tabla 8

Además también tenemos el operador de concatenación de cadenas: &

## Literales de VBScript

VBScript presenta los literales que se comentan a continuación.

- Empty: esta palabra reservada es utilizada para indicar el valor no inicializado de una variable.
- Nothing: esta palabra reservada s utilizada para destruir una variable que representa a un objeto y de esa forma liberar la memoria que utiliza.
- Null: se utiliza para indicar que una variable no tiene datos válidos, no se debe confundir con el literal Empty.
- True: representa al valor booleano verdadero y su valor es -1.
- False: representa al valor booleano falso y su valor es 0.

## Estructuras de control en VBScript

Utilizando instrucciones condicionales y bucles se puede escribir código VBScript que tome decisiones y repita acciones, se puede comprobar que estas estructuras de control son muy similares a las de otros lenguajes de programación.

If...Then...Else, se utiliza para evaluar si una condición es verdadera o falsa y después especificar qué instrucciones ejecutar, dependiendo del resultado. Las instrucciones If...Then...Else se pueden anidar en tantos niveles como necesite.

Su sintaxis es la siguiente:

```
If<condicion> Then  
    <Instrucciones>  
ElseIf <condicion> Then  
    <Instrucciones>  
    .  
    .  
Else  
    <Instrucciones>  
End If
```

Si sólo se necesita ejecutar una instrucción cuando una condición se cumple se puede utilizar la sintaxis de línea única y se omite la palabra clave Else.

```
<%If edad<18 Then MenorEdad=True%>
```

Código fuente 30

Si se desea ejecutar más de una línea de código se debe utilizar la sintaxis de múltiples líneas y añadir al final la instrucción End If.

```
<%If edad<18 Then  
    MenorEdad=True  
    Adulito=False  
    CarnetConducir=False  
End If%>
```

Código fuente 31

Utilizando la sintaxis completa, es decir If...Then...Else, se puede indicar un bloque de instrucciones si la condición es cierta y otro bloque si la condición es falsa.

```
<%If edad<18 Then  
    MenorEdad=True  
    Adulito=False  
    CarnetConducir=False  
Else  
    MenorEdad=False  
    Adulito=True  
End If%>
```

Código fuente 32

También existen instrucciones que ofrecen una condición múltiple mediante la sentencia Select...Case. Según sea el valor de una variable se realizará unas acciones u otras, existirá un bloque de instrucciones por cada valor que nos interese. También se realizará un bloque de acciones por defecto, es decir, si no coincide el valor de la variable con ninguno de los casos especificados se ejecutarán las instrucciones incluidas en la rama Case Else, aunque esta alternativa se puede omitir. La sintaxis que presenta esta estructura de control es la siguiente:

```
Select Case <variable>
    Case<valor1>
        <instrucciones>
    Case<valor2>
        <instrucciones>
    ...
    Case<valor n>
        <instrucciones>
    Case Else
        <instrucciones>
End Select
```

En el Código fuente 33 se diferencian tres casos para el valor de la variable operación y se ejecuta un bloque de acciones por defecto en el caso de que no se den ninguno de los tres casos anteriores.

```
<%
    Select Case operacion
        Case "Suma"
            x=x+1
            y=y+1
        Case "Resta"
            y=y-1
            x=x-1
        Case "Multiplicacion"
            x=x*2
            y=y*2
        Case Else
            x=0
            y=0
    End Select
%>
```

Código fuente 33

VBScript proporciona bucles para permitir ejecutar un grupo de instrucciones de forma repetida. Algunos bucles se repiten hasta que una condición es falsa, otros hasta que una condición es verdadera y otros un número de veces determinado.

En VBScript están disponibles las siguientes instrucciones de bucles:

- Do...Loop: ejecuta el bucle mientras o hasta que una condición es verdadera.
- While...Wend: ejecuta el bucle mientras una condición es verdadera.
- For...Next: utiliza un contador para ejecutar instrucciones un número de veces determinado.
- For Each...Next: permite recorrer colecciones.

Las Instrucciones Do...Loop se utilizan para ejecutar un bloque de instrucciones un número de veces indefinido, las instrucciones se repiten mientras una condición sea verdadera o hasta que una condición sea verdadera.

Si elegimos la opción mientras una condición es verdadera, deberemos utilizar la palabra reservada While.

Se comprobará la condición antes de entrar en el bucle o después de que el bucle se haya ejecutado, esto dependerá de donde coloquemos la instrucción While con la condición.

```
<%
    'Comprueba la condición antes de entrar en el bucle
    numero=20
    Do While numero>10
        numero=numero-1
        contador=contador+1
    Loop
    'Comprueba la condición después de la ejecución del bucle
    numero=20
    Do
        numero=numero-1
        contador=contador+1
    Loop While numero>10
%>
```

Código fuente 34

También se puede repetir un conjunto de instrucciones hasta que una condición sea verdadera, para ello se deberá utilizar la instrucción Until; y al igual que ocurría en el caso anterior la comprobación de la condición puede ir antes de la ejecución del bucle o después de la ejecución.

```
<%
    numero=20
    'Comprueba la condición antes de entrar en el bucle
    Do Until numero=10
        numero=numero-1
        contador=contador+1
    Loop
    numero=20
    'Comprueba la condición después de la ejecución del bucle
    Do
        numero=numero-1
        contador=contador+1
    Loop Until numero=10
%>
```

Código fuente 35

Con al instrucción While...Wend conseguimos repetir un bloque de instrucciones mientras una condición sea verdadera.

```
<%
    numero=20
    While numero>10
```

```

numero=numero-1
contador=contador+1
Wend
%>

```

Código fuente 36

Para ejecutar un bloque de instrucciones un número de veces determinado se deberá utilizar la instrucción For...Next. Se debe utilizar una variable numérica cuyo valor aumente o disminuya con cada repetición del bucle. Por defecto la variable contador se incrementa de uno en uno, si queremos modificar este comportamiento lo deberemos indicar utilizando la palabra reservada Step.

```

<%
    'Se ejecuta 50 veces
    j=0
    For x=1 To 50
        j=j+1
    Next
    'La variable contador se incrementa de 2 en 2
    For x=2 To 10 Step 2
        j=j+1
    Next
    'La variable contador se decrementa de 2 en 2
    For x=20 To 2 Step -2
        j=j+1
    Next
%>

```

Código fuente 37

Para forzar la salida de una instrucción Do...Loop o For...Next se utilizará las sentencias Exit Do y Exit For respectivamente.

```

<%
    Do Until numero=10
        numero=numero+1
        If error Then Exit Do
    Loop
%>

```

Código fuente 38

Con la insctrucción For Each...Next repetiremos una serie de sentencias por cada uno de los elementos de un array o colección. La sintaxis de esta estructura de control es la siguiente:

```

For Each elemento In grupo
    [sentencias]
    [sentencias]
Next

```

De esta forma si queremos recorrer la colección ServerVariables del objeto Request, escribiremos el Código fuente 39.

```
<%For Each elemento IN Request.ServerVariables  
    Response.Write elemento & "=" & Request.ServerVariables(elemento) &"<BR>"  
Next%>
```

Código fuente 39

Mediante este código escribimos el nombre de cada elemento de la colección ServerVariables igualándolo al su valor correspondiente. Los objetos Request y Response son objetos integrados en ASP que pertenecen al modelo de objetos de las páginas activas y que veremos en el capítulo correspondiente, como se puede ver el acceso del modelo de objetos de ASP desde el lenguaje de secuencias de comandos es muy sencillo.

Mediante la instrucción With ejecutaremos una serie de sentencias que se encuentran todas ellas relacionadas con un objeto, se suele utilizar para manipular las propiedades de un objeto y lanzar sus métodos. Su sintaxis es:

```
With objeto  
    sentencias  
End With
```

Así por ejemplo si queremos modificar las propiedades de un objeto determinado utilizaremos el Código fuente 40.

```
Set objeto=Server.CreateObject ("Este.Objeto")  
With objeto  
    .Propiedad1="Valor 1"  
    .Propiedad2="Valor 2"  
End With
```

Código fuente 40

De momento no vamos a entrar en detalles en la forma en la que se crean los objetos, se verá más adelante en el curso.

## Procedimientos de VBScript

En VBScript hay dos tipos de procedimientos: Sub y Function. Un procedimiento Sub, o procedimiento propiamente dicho, es un conjunto de instrucciones de VBScript incluidas entre las palabras reservadas Sub y End Sub que ejecutan acciones, pero que no devuelven ningún valor. A un procedimiento Sub se le pueden pasar parámetros, si no posee parámetros en la instrucción Sub se deberán escribir dos paréntesis vacíos.

```
Sub InicializaVariables()  
    nombre=""  
    contador=0  
End Sub
```

Código fuente 41

Un procedimiento Function, o función, es una serie de instrucciones incluidas entre las palabras clave Function y End Function. Un procedimiento Function es muy similar a un procedimiento Sub, pero puede devolver un valor. La función devuelve el valor que se le asigna a su nombre dentro de la implementación de la misma. El tipo de datos devuelto por una función siempre es de tipo Variant.

La función que muestra el Código fuente 42 devuelve el cuadrado del valor que se le pasa como parámetro.

```
Function cuadrado(valor)
    cuadrado=valor^2
End Function
```

Código fuente 42

Para utilizar una función dentro del código se debe situar la llamada al procedimiento Function en la parte derecha de una operación de asignación o bien dentro de una expresión.

```
<%
    'En una asignación
    valor=cuadrado(2)
    'En una expresión
    Response.write("El cuadrado de 2 es: " & cuadrado(2))
%>
```

Código fuente 43

Para realizar la llamada a un procedimiento basta con utilizar su nombre y pasarle los parámetros necesarios. También se puede utilizar la palabra reservada Call para realizar la llamada a un procedimiento, aunque no es necesario utilizarla pero si se usa se deben situar los parámetros entre paréntesis.

```
Call Procedimiento (arg1,arg2)
Procedimiento arg1,arg2
```

Código fuente 44

Se puede forzar la salida de un procedimiento o función utilizando las instrucciones Exit Sub o Exit Function respectivamente, al ejecutarse cualquiera de las dos instrucciones anteriores, la ejecución del programa continua en la siguiente instrucción a la que había invocado al procedimiento o a la función.

Si el procedimiento devuelve un valor los parámetros del procedimiento siempre deben ir encerrados entre paréntesis, es decir, las funciones siempre llevarán paréntesis.

## Tratamiento de errores en VBScript

Si queremos atrapar errores dentro de nuestro script, se debe indicar dentro del código mediante la sentencia On Error. El tratamiento de errores dentro de VBScript se encuentra bastante limitado, no

existe soporte para un tratamiento centralizado de errores, simplemente se dispone de dos opciones a través de On Error Resume Next y de On Error Goto 0, por lo tanto no existe la opción On Error Goto EtiquetaTratamientoError con la que podríamos tratar los errores dentro de un procedimiento.

Para atrapar errores se debe indicar mediante la sentencia On Error Resume Next. Una vez ejecutada esta sentencia, después de cada línea de código del script que pueda contener o producir posibles errores, se debe comprobar el valor de la propiedad Number del objeto Err para averiguar si es mayor que cero. El objeto Err contiene información sobre los errores que se producen en tiempo de ejecución.

Cuando se produce un error en tiempo de ejecución, las propiedades del objeto Err se llenan con información que identifica el error de forma única y que es útil para el procesamiento del error. Las propiedades del objeto Err se restablecen a cero o a una cadena vacía después de una sentencia On Error Resume Next.

En el Código fuente 45 se muestra como se comprobaría si se ha producido un error.

```
<%  
    On Error Resume Next  
    objeto.MetodoInvocado  
    If Err>0 Then mensaje="Se ha producido el error: "&Err.Description  
%>
```

Código fuente 45

Al utilizar On Error Resume Next si se da un error en tiempo de ejecución no se detendrá la ejecución del script sino que pasará a la siguiente instrucción. Por lo tanto, la instrucción On Error Resume Next hace que la ejecución continúe con la instrucción que sigue inmediatamente a la que causó el error en tiempo de ejecución, o con la que sigue inmediatamente a la última llamada desde el procedimiento que contiene la instrucción On Error Resume Next.

Si el tratamiento de errores lo hemos activado con On Error Resume Next dentro de un procedimiento, el tratamiento de errores se desactivará cuando se llame a otro procedimiento. Por lo tanto se deberá utilizar la sentencia On Error Resume Next en cada procedimiento si se quieren atrapar todos los errores.

Para desactivar la rutina de tratamiento de errores se debe utilizar la instrucción On Error Goto 0, una vez ejecutada esta instrucción, cualquier error en tiempo de ejecución detendrá la ejecución del script.

El objeto Err es un objeto intrínseco con alcance global y no es necesario crear un instancia de él dentro del código. Este objeto tiene varios métodos y propiedades, la propiedad predeterminada del objeto Err es la propiedad Number. Esta propiedad identifica el código de error que tiene asignando el error que se ha producido. Si no se ha producido ningún error la propiedad Number tendrá el valor 0, por lo tanto para averiguar si se ha producido un error, las dos líneas de código, que aparecen en el Código fuente 46, son equivalentes.

```
If Err.Number>0 Then  
If Err>0 Then
```

Código fuente 46

Otra propiedad del objeto Err es Description. Esta propiedad ofrece una descripción del error que se ha producido, esta propiedad se puede utilizar para mostrar la descripción del error al usuario.

Una propiedad más del objeto Err que puede resultar de utilidad es la propiedad Source. La propiedad Source indica el nombre del objeto o aplicación que causó el error.

Para mostrar la información completa acerca de un error deberemos escribir el Código fuente 47.

```
<%If Err>0 Then infoErr="Código de error: "&Err.number&" Descripción del error: "-_
&Err.Description&" Fuente del error:
"&Err.Source%>
```

Código fuente 47

En este ejemplo se puede observar también de que forma en VBScript partimos una línea de código que pertenece a una misma sentencia, se utiliza el guión bajo (\_) para indicar que la sentencia sigue en la siguiente línea de código.

El objeto Err posee dos métodos: Clear y Raise. El método Clear vacía el contenido del objeto Err, asignando a sus propiedades el valor cero o una cadena vacía. Es necesario llamar al método Clear siempre que hayamos atrapado un error de ejecución y nuestro script continúe con su ejecución. Si no se llama al método Clear, la próxima vez que se compruebe una condición de error, las propiedades del objeto Err contendrán todavía los valores del error previo. Por ejemplo, el Código fuente 48 podría causar problemas.

```
<%
On Error Resume Next
objeto.metodo
If Err>0 Then InfoErr1=Err.Description
objeto.otroMetodo
if Err>0 Then infoErr2=Err.Description
%>
```

Código fuente 48

Si la sentencia objeto.metodo provoca un error de ejecución los valores del objeto Err se asignarán de acuerdo con este error, y la primera sentencia If Err>0 Then... se ejecutará. Sin embargo, la segunda sentencia If Err>0 Then... también se ejecutará aunque la sentencia objeto.otroMetodo no produzca ningún error de ejecución. Para solucionar este problema deberemos utilizar el método Clear del objeto Err como indica el Código fuente 49.

```
<%
On Error Resume Next
objeto.metodo
If Err>0 Then
    InfoErr1=Err.Description
    Err.Clear
End If
objeto otroMetodo
if Err>0 Then
    infoErr2=Err.Description
    Err.Clear
```

```
End If  
%>
```

Código fuente 49

Por último, el método Raise nos permite crear un error de ejecución en el script. Para crear un error de ejecución mediante el método Raise se utiliza la siguiente sintaxis:

```
Err.Raise(numero, origen, descripcion, ficheroayuda,  
contextoayuda)
```

Todos los argumentos son opcionales menos numero, que indica el código de error. El parámetro origen es una cadena que especifica el proceso que provocó el error, descripcion es una cadena que describe el error, los parámetros ficheroayuda y contextoayuda no se suelen utilizar en VBScript, pero indican el nombre de un fichero de ayuda local y el contexto.

Si se utiliza Raise, sin especificar algunos argumentos, y los valores de las propiedades del objeto Err contienen valores que no se han borrado, éstos pasarán a ser los del error creado. Al establecer en el parámetro número nuestro propio código de error, se debe agregar el número de código de error a la constante vbObjectError. Por ejemplo, para generar el número de error 1150, se escribir vbObjectError + 1150 en el argumento numero. El Código fuente 50 muestra como se crearía un error.

```
<%Err.Raise(vbObjectError+5555, "TextStream", "Error creado")%>
```

Código fuente 50

## Directivas de preprocesamiento

Dentro de las páginas ASP se pueden realizar inclusiones de archivos del servidor, es decir, se puede insertar información en una página ASP antes de su procesamiento, de hay el nombre del presente apartado. Para ello se utiliza la directiva de preprocesamiento #INCLUDE que tiene la siguiente sintaxis:

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

VIRTUAL y FILE indican el tipo de ruta de acceso que se utiliza para incluir el archivo, virtual y relativa respectivamente. Por ejemplo, si el archivo a incluir, llamado incluido.inc, se encuentra en el directorio virtual /miWeb se debería escribir el Código fuente 51.

```
<!--#INCLUDE VIRTUAL="/miWeb/incluido.inc"-->
```

Código fuente 51

Pero si queremos expresar la inclusión del archivo utilizando su ruta relativa utilizaremos FILE, la ruta relativa comienza en el directorio en el que se encuentra el fichero .asp en el que queremos realizar la inclusión, así por ejemplo, si el fichero .asp se encuentra en el directorio miWeb y el fichero incluido.inc se encuentra en el directorio miWeb\subdirectorio deberemos escribir el Código fuente 52.

```
<!--#INCLUDE FILE="subdirectorio/incluido.inc"-->
```

Código fuente 52

Los archivos incluidos no requieren una extensión especial.

Un archivo incluido puede, a su vez, incluir otros archivos. Un archivo no puede incluirse así mismo.

Es recomendable utilizar rutas relativas, es decir, utilizar la propiedad FILE de la directiva INCLUDE, ya que supone menos trabajo para el servidor Web al no tener que resolver direcciones virtuales.

ASP incluye los archivos antes de ejecutar cualquier script, ya que, como se ha dicho anteriormente, es una directiva de preprocesamiento. De esta forma el Código fuente 53 generará un error ya que la variable nombreFichero no tendrá ningún valor.

```
<% nombreFichero="/miWeb/incluido.asp"%>
<!-- INCLUDE VIRTUAL="<%nombreFichero%>" -->
```

Código fuente 53

Existe otra alternativa para realizar inclusiones de ficheros, en este caso se trata para incluir ficheros que únicamente contengan código perteneciente a script de servidor. La otra forma de incluir ficheros es la que muestra el Código fuente 54.

```
<SCRIPT RUNAT="SERVER" SRC="ruta relativa, física o virtual al fichero de
script"></SCRIPT>
```

Código fuente 54

El fichero que incluimos, a diferencia de la directiva INCLUDE, únicamente puede contener secuencias de comandos, no puede contener texto ni código HTML, además no debe existir ningún elemento entre las etiquetas <SCRIPT></SCRIPT>.

Esta forma de incluir ficheros puede ser útil a la hora de incluir páginas ASP que contengan una colección de funciones y procedimientos para utilizarla a modo de librería de utilidades dentro de otras páginas activas.

## Expresiones regulares en VBScript

Para manejar expresiones regulares VBScript ofrece un objeto llamado RegExp. Este objeto ofrece el soporte básico para la utilización de expresiones regulares. Para ello ofrece una serie de métodos y propiedades.

Las propiedades del objeto RegExp definen el tipo de búsqueda que se va a realizar y son las siguientes:

- Pattern: representa el patrón de búsqueda de la expresión regular. Se pueden utilizar caracteres especiales para especificar el patrón de búsqueda, los más usuales son los que se muestran en la Tabla 9.

Carácter	Descripción
*	Representa al carácter precedente cero o más veces. Por ejemplo, "zo*" coincide con "z" o "zoo".
+	Representa al carácter anterior una o más veces. Por ejemplo, "zo+" coincide "zoo" pero no con "z".
?	Representa al carácter anterior ninguna vez o una. Por ejemplo, "a?ve?" coincide con "ve" en "never".
{n}	n es un entero no negativo. Representa un carácter n veces. Por ejemplo, "o{2}" no coincide con "o" in "Bob," pero sí con las dos primeras o's es "fooooo".
[xyz]	Un conjunto de caracteres. Coincide con cualquiera de los caracteres. Por ejemplo, "[abc]" coincide con la "a" en "plan".
[^xyz]	Un conjunto de caracteres "negativo". Coincide con cualquier carácter que no se encuentre en el conjunto. Por ejemplo, "[^abc]" coincide con la "p" in "plan".
[a-z]	Un rango de caracteres. Coincide con cualquier carácter que se encuentre en el rango especificado. Por ejemplo, "[a-z]" coincide con cualquier carácter alfanumérico en minúsculas entre a y z.
[^m-z]	Un rango de caracteres negativo, es decir, coincide con cualquier carácter que no se encuentre en el rango indicado.
\d	Coincide con un dígito.
\D	Coincide con un carácter que no sea dígito.
\n	Coincide con un carácter de nueva línea
\s	Coincide con un espacio.
\t	Coincide con un tabulador.

Tabla 9

- IgnoreCase: tiene los valores verdadero o falso (True/False) e indica si la búsqueda en la expresión regular es sensible a mayúsculas y a minúsculas (case-sensitive). Por defecto su valor es False.
- Global: presenta los valores verdadero o falso e indica si la búsqueda de la expresión regular es en la cadena completa o únicamente hasta la primera ocurrencia. Por defecto su valor es False.

Los métodos que tiene son tres y son los siguientes:

- Execute: ejecuta la búsqueda de un expresión en una cadena determinada. El parámetro de este método es la cadena en la que se va a realizar la búsqueda de la expresión, la expresión de encontraría definida por el objeto RegExp correspondiente. Y este método devuelve una colección Matches con todas las coincidencias del resultado de la búsqueda, esta colección está compuesta de objetos Match que representan cada una de las coincidencias. Veamos un ejemplo de utilización de este método, en el Código fuente 55.

```

cadena="prueba nmd dsdjjdh sd dshsdjhd Prueba ddsdnmm pruebabadrtt"
Set objExpresion=New RegExp 'se crea la expresión regular
objExpresion.Pattern="prueba" 'se configura la expresión
objExpresion.IgnoreCase=True
objExpresion.Global=True
Set coincidencias=objExpresion.Execute(cadena) 'se ejecuta la búsqueda
For each Match in coincidencias 'se muestran las coincidencias de la
expresión
    Response.Write "Coincidencia encontrada en la posicion
"&Match.FirstIndex"<br>"
Next

```

Código fuente 55

En este caso se nos devuelven todas las coincidencias con la palabra prueba, y se obtiene el siguiente resultado

```

Coincidencia encontrada en la posicion 0
Coincidencia encontrada en la posicion 31
Coincidencia encontrada en la posicion 46

```

Pero si modificamos la propiedad Global del objeto RegExp asignandole el valor False, obtenemos este otro resultado.

```
Coincidencia encontrada en la posicion 0
```

Como se puede observar las posiciones dentro de la cadena que queremos buscar comienzan con el índice cero.

Y si modificamos la propiedad IgnoreCase asignándole el valor False, se obtiene este otro resultado.

```

Coincidencia encontrada en la posicion 0
Coincidencia encontrada en la posicion 46

```

Como se puede ver en el código para recorrer todas las coincidencias encontradas utilizamos la sentencia For Each..Next sobre la colección Matches, que contiene objetos Match. Cada objeto Match ofrece información relativa a cada una de las coincidencias, en nuestro caso hemos recuperado su propiedad FirstIndex para obtener la posición en la que se encuentra localizada la coincidencia correspondiente.

El objeto Match dispone de dos propiedades más, Value, que contiene el valor de la cadena que se corresponde con la coincidencia, y Length, que es la longitud de dicha cadena. Si modificamos el Código fuente 55 como indica el Código fuente 56.

```

For each Match in coincidencias 'se muestran las coincidencias de la
expresión
    Response.Write "Coincidencia encontrada en la posición
    "&Match.FirstIndex&
        ". Valor de la coincidencia: "&Match.value&", longitud:
    "&Match.length"<br>"
Next

```

Código fuente 56

Obtenemos este resultado:

```

Cocincidencia encontrada en la posición 0. Valor de la
coincidencia: prueba, longitud: 6
Cocincidencia encontrada en la posición 31. Valor de la
coincidencia: Prueba, longitud: 6
Cocincidencia encontrada en la posición 46. Valor de la
coincidencia: prueba, longitud: 6

```

- Replace: este método devuelve una cadena como resultado de reemplazar la cadena que indiquemos por parámetro y atendiendo a la expresión regular definida en el objeto RegExp. En el Código fuente 57 se puede ver como utilizar este método.

```

Set objExpresion=New RegExp
objExpresion.Pattern="r+"
objExpresion.IgnoreCase=True
objExpresion.Global=True
TextoReemplazado=objExpresion.Replace("El perro de San Roque no tiene
rabo", "x")
Response.write TextoReemplazado

```

Código fuente 57

Y el resultado de su ejecución es:

El pexo de San xoque no tiene xabo

- Test: ejecuta la búsqueda en una cadena de la expresión regular indicada y devuelve verdadero si ha encontrado alguna coincidencia. En el Código fuente 58 tenemos un ejemplo que utiliza este método del objeto RegExp.

```

Set objExpresion=New RegExp
objExpresion.Pattern="perro"
objExpresion.IgnoreCase=True
objExpresion.Global=True
resultado=objExpresion.Test("El perro de San Roque no tiene rabo")
If resultado Then
    Response.Write "Se ha encontrado una o más coincidencias"
Else
    Response.Write "No se han encontrado coincidencias"
End if

```

Código fuente 58

Y su resultado es:

Se ha encontrado una o más coincidencias

Para terminar este apartado vamos a comentar una función de VBScript que ya no tiene que ver con expresiones regulares, pero si con expresiones en general, se trata de la función Eval. Esta función evalúa la expresión que se pasa como parámetro en forma de cadena y devuelve el resultado de su evaluación. En el Código fuente 59 se muestran algunos ejemplos.

```
Response.Write Eval("3+4") 'devuelve 7
Response.Write Eval("3>5") 'devuelve falso(False)
Response.Write Eval("True OR False") 'devuelve verdadero (True)
```

Código fuente 59

## Introducción a la POO

En la versión 5.1 de VBScript se implementa una nueva característica que consiste en la posibilidad de utilizar clases en nuestras páginas ASP. Antes de comentar la sintaxis de la creación de clases y la forma de utilizarlas dentro de nuestras páginas activas (lo veremos en detalle a lo largo de los siguientes apartados), vamos a realizar una pequeña introducción a las clases comentando para ello conceptos de la programación orientada a objetos (OOP, Object Oriented Programming).

No vamos a entrar a explicar todos los conceptos de la programación orientada a objetos ni tampoco vamos a entrar en mucho detalle, sólo vamos a comentar algunos conceptos que se consideran necesarios para comprender la utilización de objetos y clases dentro de VBScript.

## Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) trata de utilizar una visión real del mundo dentro de nuestros programas. La visión que se tiene del mundo dentro de la POO es que se encuentra formado por objetos. Para comprender bien la POO debemos olvidar un poco la Programación Estructurada, que si nos fijamos bien es algo artificial, la POO es una forma de abordar los problemas más natural. Aquí natural significa más en contacto con el mundo real que nos rodea, de esta forma si queremos resolver un problema determinado, debemos identificar cada una de las partes del problema con objetos presentes en el mundo real.

## Objetos

Como ya hemos adelantado un objeto es la pieza básica de la POO, es una representación o modelización de un objeto real perteneciente a nuestro mundo, por ejemplo, podemos tener un objeto perro que represente a un perro dentro de nuestra realidad, o bien un objeto factura, cliente o pedido.

Los objetos en la vida real tienen todos dos características: estado y comportamiento. El estado de un objeto viene definido por una serie de parámetros que lo definen y que lo diferencian de objetos del mismo tipo. En el caso de tener un objeto perro, su estado estaría definido por su raza, color de pelo, tamaño, etc. Y el comportamiento viene definido por las acciones que pueden realizar los objetos, por ejemplo, en el caso del perro su comportamiento sería: saltar, correr, ladrar, etc.

Si tomamos un ejemplo que tiene que ver más con el mundo de la informática se pueden ver más claros estos dos conceptos. Si tenemos un objeto pantalla que representa la pantalla de nuestro ordenador, el estado de la misma estaría definido por los siguientes parámetros: encendida o apagada, tamaño, resolución, número de colores, etc.; y su comportamiento podría ser: imprimir, encender, apagar, etc.

Los parámetros o variables que definen el estado de un objeto se denominan atributos o miembros o variables y las acciones que pueden realizar los objetos se denominan métodos.

## Clases

Una clase es un molde o prototipo que define un tipo de objeto determinado. Una clase define los atributos y métodos que va a poseer un objeto. Mediante las clases podremos crear o instanciar objetos de un mismo tipo, estos objetos se distinguirán unos de otros a través de su estado, es decir, el valor de sus atributos.

Los atributos y métodos de una clase pueden ser públicos o privados. Si son privados sólo se puede acceder a ellos desde dentro de la clase y si son públicos se puede acceder desde otras clases distintas.

La clase la vamos a utilizar para definir la estructura de un objeto, es decir, estado (atributos) y comportamiento (métodos). La clase es un concepto abstracto que generalmente no se va a utilizar directamente en nuestros programas o aplicaciones. Lo que vamos a utilizar van a ser objetos concretos que son instancias de una clase determinada.

La clase es algo genérico y abstracto, es similar a una idea. Cuando decimos piensa en un coche todos tenemos en mente la idea general de un coche, con puertas, ruedas, un volante, etc., sin embargo cuando decimos "ese coche que está aparcado ahí fuera", ya se trata de un coche determinado, con una matrícula, de un color, con un determinado número de puertas, y que podemos tocar y utilizar si es necesario. Sin embargo como ya hemos dicho la clase es la idea que define al objeto concreto.

Un ejemplo que se suele utilizar para diferenciar y relacionar clases y objetos es el ejemplo del molde de galletas. El molde para hacer galletas sería una clase, y las galletas que hacemos a partir de ese molde ya son objetos concretos creados a partir de las características definidas por el molde.

Una vez implementada una clase podremos realizar instancias de la misma para crear objetos que pertenezcan a esa clase.

Si comparamos las clases y objetos de la POO con la programación estructurada tradicional, se puede decir que las clases son los tipos de datos y los objetos las variables de esos tipos de datos. De esta forma si tenemos el tipo entero, en la POO diríamos que es la clase entero, y si tenemos una variable de tipo entero, en la POO diríamos que tenemos un objeto de la clase entero.

## Clases y objetos en VBScript

Una vez realizada esta breve pero interesante aproximación a la programación orientada a objetos, vamos a pasar a ver como crear clases en VBScript.

Los lectores que ya conozcan la utilización y creación de clases en Visual Basic comprobarán que en VBScript es todo bastante similar, existen métodos, propiedades, atributos de la clase, pero la principal diferencia es que en VBScript no podemos declarar eventos para las clases.

Para crear una clase en VBScript se utiliza la siguiente sintaxis general:

```
Class nombre
    sentencias
End Class
```

Dónde *nombre* es el nombre de la clase y *sentencias* son todas las declaraciones de atributos de la clase y la implementación de los métodos de la misma.

Para declarar los atributos de una clase utilizamos la siguiente sintaxis:

```
Private|Public nombreVariable
```

Con Public o Private indicamos si el atributo es visible o no desde fuera de la clase, respectivamente. Normalmente en VBScript los atributos de las clases se denominan variables.

Los métodos de una clase pueden ser tanto procedimientos (Sub) como funciones (Function) y que se pueden definir como privados o públicos, al igual que ocurría con las variables de la clase. Si nos indica Private o Public el método será tomado como privado, sin embargo en las variables de la clase si es necesario indicar Private o Public.

Un método si es público (Public) también puede ser el método por defecto (Default). Un método por defecto es aquel que se lanza sobre el objeto sin necesidad de especificarlo, sólo se puede especificar un método por defecto, si se intenta indicar más de uno se producirá un error. La sintaxis general para declarar un método es la siguiente.

Si es un procedimiento:

```
[Public [Default] | Private] Sub nombre [(lista de parámetros)]
    [sentencias]
    [Exit Sub]
    [sentencias]
End Sub
```

Si es una función:

```
[Public [Default] | Private] Function nombre [(lista de parámetros)]
    [sentencias]
    [nombre = expresión]
    [Exit Function]
    [sentencias]
    [nombre = expresión]
End Function
```

Veamos un ejemplo, en el Código fuente 60, de una clase con un atributo o variable privado y tres métodos públicos, uno de los métodos es una función y los otros dos son procedimientos, siendo uno de estos últimos un método por defecto.

```
Class Sencilla
    Private numero
    Public Function devuelveNumero()
        devuelveNumero=numero
    End Function
    Public default Sub incrementa(inc)
        numero=numero+inc
    End Sub
    Public Sub decrementa(dec)
        numero=numero-dec
    End Sub
```

```
End Sub
End Class
```

Código fuente 60

Ya tenemos creada nuestra primera clase, pero en realidad todavía no podemos hacer nada con ella, para poder hacer uso de la clase en nuestro código debemos instanciar un objeto de la clase. Esto ya lo hemos hecho anteriormente al utilizar el objeto RegExp para utilizar expresiones regulares. En realidad RegExp es una clase de la que nosotros creamos una instancia, y para ello utilizamos la sintaxis:

```
Set nombreObjeto= New nombreClase
```

Así si queremos crear un objeto de nuestra clase *Sencilla* y utilizar sus métodos utilizaremos las sentencias que muestra el Código fuente 61.

```
Set objSencilla=New Sencilla
objSencilla(2)
Response.Write objSencilla.devuelveNumero &"<br>"
objSencilla.decrementa(4)
Response.Write objSencilla.devuelveNumero
```

Código fuente 61

El resultado de la ejecución de estas líneas es 2 y -2, ya que en un principio el atributo *numero* es como si tuviera asignado el valor cero. También se puede observar la forma en la que se utiliza el método por defecto *incrementa*.

Para destruir un objeto y de esa forma liberar la memoria que utiliza debemos asignarle el literal Nothing. La sintaxis general para destruir objetos es:

```
Set nombreObjeto= Nothing
```

## Eventos y propiedades en VBScript

Hemos comentado que una diferencia que tiene VBScript con Visual Basic en cuanto a la implementación de clases se refiere es que en VBScript no se permite declarar eventos para las clases, pero existe una excepción.

Hay dos eventos especiales que están asociados a la clase y se trata de los eventos Initialize y Terminate. El evento Initialize se lanza cuando se crea una instancia de una clase, es decir, cuando se crea un objeto. El evento Terminate se lanza cuando se destruye un objeto de la clase. La sintaxis de los eventos Initialize y Terminate es la que se muestra a continuación:

```
Private Sub Class_Initialize
    [sentencias]
End Sub

Private Sub Class_Terminate
    [sentencias]
End Sub
```

El método Initialize se puede utilizar para tareas de inicialización, como puede ser inicializar los atributos de una clase con unos valores determinados. El método Terminate se puede utilizar para realizar funciones de "limpieza" cuando se destruye un objeto de una clase.

Pasemos ahora a la segunda parte de este apartado que tal y como se indica en el título del mismo son las propiedades.

Existe un tipo de método especial denominado propiedad (Property), las propiedades se utilizan para devolver o acceder a atributos de la clase de forma controlada. Existen dos tipos Property Get, para obtener un valor y Property Let para asignar un valor.

Mediante las propiedades podemos encapsular los atributos de la clase para que permanezcan ocultos desde fuera de la clase, es decir, podemos ocultar la implementación interna de la clase mediante las propiedades que serán las que accedan a los atributos que hemos declarados como privados en la clase.

Las propiedades pueden ser de sólo lectura, si sólo tienen un método Property Get, de sólo escritura si sólo tienen Property Let y de lectura y escritura si tienen Property Get y Property Let.

Las propiedades tienen los mismos modificadores de acceso que los métodos, es decir, Public (en combinación con Default) y Private. También podemos definir una propiedad por defecto para la clase. La propiedad por defecto sólo se puede indicar en el método Property Get, es decir, una propiedad por defecto únicamente tiene sentido a la hora de recuperar el valor de la propiedad.

Vamos a ver la sintaxis del método especial Property Get, que define la lectura de una propiedad.

```
[Public | Private] Property Get nombre [(lista de parámetros)]
[sentencias]
[[Set] nombre = expresión]
Exit Property
[sentencias]
[[Set] nombre = expresión]
End Property
```

A la vista de la sintaxis de la sentencia Property Get, podemos asegurar que en la misma siempre debe existir una asignación del nombre de la propiedad para poder recuperar el valor de la propiedad correspondiente, que en realidad estará representado por un atributo privado de la clase correspondiente. Normalmente las sentencias Property Get no presentan lista de parámetros.

Y la sintaxis del método Property Let, que define la escritura o modificación de una propiedad es la siguiente.

```
[Public | Private] Property Let nombre ([lista de parámetros,] valor)
[sentencias]
Exit Property
[sentencias]
End Property
```

Como se puede comprobar, en una sentencia Property Let siempre es necesario especificar un valor, que será el que se le asigne a la propiedad.

Cuando la propiedad se trata de un objeto en lugar de utilizar la sentencia Property Let se utiliza Property Set con la siguiente sintaxis.

```
[Public | Private] Property Set nombre([lista de parámetros,]
referencia)
[sentencias]
[Exit Property]
[sentencias]
End Property
```

Vamos a ver con un sencillo ejemplo la forma de definir propiedades y utilizarlas junto con los atributos privados de una clase. También utilizaremos y definiremos los eventos Initialize y Terminate.

En el siguiente ejemplo se define una clase *Rectangulo* que tiene una propiedad llamada *Color* de sólo lectura, otra propiedad de lectura/escritura llamada *Base*, y una propiedad por defecto llamada *Altura*. La propiedad por defecto de la clase *Rectangulo* es la propiedad *Base*.

Al definir una propiedad por defecto, ya no podremos definir ningún método por defecto, no se debe olvidar que una propiedad es un tipo de método especial que definimos con la sentencia Property Get.

Nuestra clase *Rectangulo* va a tener tres atributos privados (atributoColor, atributoAltura y atributoBase) para representar internamente las propiedades que expone la clase.

Para evitar que se asignen valores negativos a las propiedades *Altura* y *Base* de la clase *Rectangulo*, vamos a aprovechar las sentencias Property Let correspondientes y en caso de asignación de un número negativo a estas propiedades se les asignará el valor cero.

También vamos a tener un sencillo método llamado *area()* y que nos calcula el área del objeto *Rectangulo* correspondiente.

En el evento Initialize de esta clase inicializamos el atributo atributoColor al valor rojo, y en el evento Terminate simplemente indicamos que el objeto ha sido destruido.

Pasemos a ver el código que define a la clase *Rectángulo* (Código fuente 62).

```
Class Rectangulo
    'atributos de la clase
    Private atributoColor, atributoBase, atributoAltura
    'Evento de inicialización de la clase
    Private Sub Class_Initialize
        atributoColor="rojo"
    End Sub
    'Evento de destrucción de la clase
    Private Sub Class_Terminate
        Response.Write "Objeto destruido"
    End Sub
    'propiedades de la clase
    'propiedad de sólo lectura
    Public Property Get Color
        Color=atributoColor
    End Property
    Public Default Property Get Base
        Base=atributoBase
    End Property
    Public Property Let Base(valor)
        'comprobamos el valor que se va a asignar a la propiedad
        If valor<0 Then
            atributoBase=0
        Else
            atributoBase=valor
        End If
    End Property
End Class
```

```

        End if
    End Property
    Public Property Get Altura
        Altura=atributoAltura
    End Property
    Public Property Let Altura(valor)
        If valor<0 Then
            atributoAltura=0
        Else
            atributoAltura=valor
        End if
    End Property
    Public Function area()
        area=atributoBase*atributoAltura
    End Function
End Class

```

Código fuente 62

Como se puede ver en el código del ejemplo estas propiedades, que son todas públicas, utilizan los tres atributos privados de la clase rectángulo, de esta forma obtenemos la encapsulación de nuestra clase, los usuarios de la misma únicamente van a poder ver las propiedades y métodos que están declarados como públicos.

Ahora se muestra el Código fuente 63, el cual, hace uso de la clase Rectangulo instanciando un objeto de la clase y utilizando las propiedades y métodos de la misma.

```

'se instancia el objeto
Set objRectangulo=New Rectangulo
'damos un valor negativo
objRectangulo.Base=-4
'pero observamos que se la ha dado el valor cero
Response.Write "Base del rectángulo: "&objRectangulo&"<br>"
objRectangulo.Base=3
objRectangulo.Altura=4
'obtenemos las propiedades
Response.Write "Color del rectángulo: "&objRectangulo.Color&"<br>"
'obtenemos la propiedad por defecto
Response.Write "Base del rectángulo: "&objRectangulo&"<br>"
Response.Write "Altura del rectángulo: "&objRectangulo.Altura&"<br>"
'ejecutamos el método de la clase
Response.Write "Área del rectángulo: "&objRectangulo.area&"<br>"
'se destruye el objeto
Set objRectangulo=Nothing

```

Código fuente 63

El resultado de la ejecución de este código es el siguiente:

```

Base del rectángulo: 0
Color del rectángulo: rojo
Base del rectángulo: 3
Altura del rectángulo: 4
Área del rectángulo: 12
Objeto destruido

```

## Otras funciones en VBScript

En este apartado se recogen una serie de funciones de VBScript que pueden resultar bastante útiles, no se entra en detalle sino que se ofrece una visión general de las mismas, está realizado al estilo de una referencia rápida, agrupando cada una de las funciones atendiendo a al papel que desempeñan y a su utilidad.

### Funciones para consultar variables

Funciones que devuelven un valor booleano tras consultar el tipo y contenido de la de variable que se pasa como parámetro. Por ejemplo IsDate() devolverá verdadero si la variable que se pasa como parámetro es una fecha.

`IsEmpty, IsDate, IsNumeric, IsObject, IsNull`

### Funciones utilizadas para la manipulación de cadenas

Todas estas funciones reciben como parámetro una cadena de caractes y realizan alguna operación con ella devolviendo el resultado correspondiente.

- `InStr`: devuelve la posición de la primera ocurrencia de una cadena dentro de otra.
- `Len`: devuelve el número de caracteres de una cadena.
- `LCase`: devuelve una cadena convertida en minúsculas.
- `UCase`: devuelve una cadena convertida en mayúsculas.
- `Left`: devuelve los caracteres especificados desde la parte izquierda de la cadena.
- `Righth`: igual que la función anterior pero desde la parte derecha de la cadena.
- `Mid`: devuelve los caracteres indicados de una cadena desde una posición incial determinada.
- `Split`: devuelve un array de cadenas resultantes de dividir una cadena atendiendo a un separador definido.
- `Space`: devuelve una cadena de caracteres blancos según el parámetro indicado, esta es la única función de este apartado que no recibe una cadena como parámetro.
- `StrReverse`: devuelve una cadena con los caracteres un orden inverso.
- `LTrim`: devuelve una cadena a la que se le eliminan los espacios a la izquierda.
- `RTrim`: devuelve una cadena a la que se le eliminan los espacios a la derecha.
- `Trim`: devuelve una cadena a la que se le eliminan los espacios a la derecha y a la izquierda.

## Funciones matemáticas

- Cos: devuelve el coseno de un ángulo.
- Sin: devuelve el seno de un ángulo.
- Tan: devuelve la tangente de un ángulo.
- Exp. devuelve el número e elevado a una potencia.
- Log: devuelve el logaritmo de un número.
- Sqr: devuelve la raíz cuadrada de un número.
- Randomize: incializa la generación de números aleatorios.
- Rnd: devuelve un número aleatorio.

## Funciones para dar formatos

- FormatCurrency: da formato de tipo monetario.
- FormatDateTime: para dar formato a fecha o a la hora.
- FormatPercent: da formato a los tantos por ciento.
- FormatNumber: da formato a valores numéricos.

## Funciones de fecha/hora

- Date: devuelve la fecha actual del sistema.
- Time: devuelve la hora actual del sistema.
- DateAdd: devuelve una fecha a la que se le ha añadido un intervalo de tiempo especificado.
- DateDiff: devuelve el número de intervalos entre dos fechas.
- DatePart: devuelve una parte determinada de la fecha indicada.
- DateValue: devuelve una variable del subtipo Date a partir de la cadena especificada por parámetro.
- Day: devuelve el día del mes de la fecha que se pasa por parámetro.
- Month: devuelve el mes de la fecha que se pasa por parámetro.
- MonthName: devuelve el nombre del mes de la fecha que se pasa por parámetro.
- Now: devuelve la fecha y hora actual del sistema.

- WeekDay: devuelve el día de la semana que se corresponde con la fecha indicada.
- WeekDayName: devuelve el nombre del día de la semana que se corresponde con la fecha indicada.
- Year: devuelve el año de la fecha indicada.
- Hour: devuelve la hora del tiempo indicado.
- Minute: devuelve los minutos del tiempo indicado.
- Second: devuelve los segundos del tiempo indicado.
- TimeValue: devuelve una variable de subtipo Date a partir de la cadena especificada por parámetro.

## Funciones que devuelven información del motor de secuencias de comandos

- ScriptEngine: devuelve una cadena que describe el lenguaje de secuencias de comandos utilizado.
- ScriptEngineMinorVersion, ScriptEngineMajorVersion, ScriptEngineBuildVersion: todas estas funciones devuelven información relativa a la versión del motor de secuencias de comandos en uso.

## Componentes de VBScript

En el primer tema de este curso ya habíamos visto que podíamos distinguir varios tipos de objetos dentro de ASP: los que pertenecen al modelo de objeto de ASP llamados objetos integrados (Request, Response, Application, Session, etc.), los componentes de servidor que se ofrecen con ASP (Content rotator, componente para funciones del explorador, Tools, registro de IIS, etc.), y por último los componentes desarrollados por nosotros mismos o por terceras partes y que podemos registrar en el servidor Web.

Pero existe un cuarto tipo o grupo de objetos que son los componentes de la librería del intérprete de comandos, es decir, componentes de VBScript. Estos componentes los encontramos junto con VBScript y son los que se comentan a continuación:

- Dictionary: este objeto ofrece un sistema de almacenamiento de valores, los valores se pueden acceder referenciados por su nombre. Lo podríamos comparar con las colecciones de Visual Basic. Es un array bidimensional que contiene valores referenciados por sus claves.
- FileSystemObject: permite acceder al sistema de archivos del servidor Web, nos permite manipular ficheros de texto, carpetas y unidades de disco. Los siguientes objetos que vamos a comentar dependen de este objeto y se utilizan en colaboración con el mismo.
- Drive: permite acceder a las unidades de disco del servidor Web. Un objeto Drive, al igual que todos los que vienen a continuación, se crea a partir de un objeto FileSystemObject.

- Folder: ofrece acceso a las carpetas del sistema de archivos de servidor Web. El objeto Folder nos va a permitir acceder a todas las propiedades de una carpeta determinada y , a través de su métodos, copiar, mover y borrar carpetas.
- File: permite acceder a los ficheros de una carpeta en el sistema de archivos del servidor Web. El objeto File nos va a permitir acceder a todas las propiedades de un fichero y, por medio de su métodos, copiar, mover o borrar un fichero determinado.
- TextStream: ofrece acceso al contenido de los ficheros de texto almacenados en el servidor Web. Mediante este objeto podremos modificar y crear ficheros de texto.

En un tema monográfico trataremos detenidamente cada uno de estos objetos que forman parte como hemos dicho de la librería del motor de secuencias de comandos de VBScript.



# 4

## Primeros pasos con ASP 3.0

---

### Objetivo de este tema

Este tema tiene como función la de aproximar al alumno a la programación aplicaciones ASP comentando los diferentes "elementos" que podemos encontrar dentro de las páginas activas de servidor.

Además de comentar el modelo de objetos de ASP y los distintos tipos de recursos que podemos utilizar en una página ASP, también se hace una introducción al material (herramienta de desarrollo, equipo y sistema operativo) que resulta necesario para seguir el presente curso y realizar los distintos ejemplos que se proponen en cada tema.

Este es un tema de enlace entre los tres primeros, que podrían resultar un poco más teóricos, y los siguientes temas que hacen uso del modelo de objetos de ASP y que resultan mucho más prácticos, existirá un tema para cada uno de los objetos del modelo de objetos de ASP.

### Objetos integrados, modelo de objetos de ASP 3.0

ASP nos proporciona una serie de objetos integrados, a los que siempre tenemos acceso sin necesidad de instanciarlos, son objetos que constituyen lo que se denomina el modelo de objetos de ASP. Estos objetos son bastante interesantes ya que gran parte de la programación de aplicaciones ASP se basa en la utilización de los mismos.

Estos objetos ponen a disposición del programador una serie de métodos y propiedades que pueden ser utilizados desde el script de servidor, es decir, son directamente accesibles y manipulables desde VBScript. Cada uno de estos objetos cubre unas funciones determinadas.

Antes de comenzar a explicar cada uno de los objetos integrados se considera necesario repasar el concepto de objeto, método y propiedad dentro del entorno de la programación orientada a objetos y desde el punto de vista que nos interesa, es decir desde las páginas activas de servidor.

Un objeto es un componente que posee una serie de comportamientos y que tiene un estado. El estado de un objeto se encuentra definido por sus propiedades y sus comportamientos a través de los métodos. Un objeto puede contener otros objetos.

Cada objeto tiene unas propiedades que definen sus atributos. Las propiedades de un objeto nos indican algo sobre el objeto o sus contenidos. Las propiedades diferencian un objeto de otro y determinan su estado y características. Los métodos que poseen los objetos definen su comportamiento interno y frente a otros objetos.

Los objetos integrados además de poseer métodos y propiedades, también ofrecen colecciones. Una colección es un grupo de objetos del mismo tipo. Un objeto y una colección son ambos contenedores, pero de distinto tipo. Un objeto contendrá cero o más colecciones de diferente tipo, mientras que una colección contendrá cero o más objetos de naturaleza similar.

Por ejemplo, ASP ofrece el objeto Request que contiene diferentes propiedades y colecciones de distinto tipo. Una de estas colecciones es la colección Form, esta colección contiene información sobre los elementos del formulario que se ha enviado. Aquí se puede observar la diferencia entre objetos y colecciones en ASP, un objeto puede siempre contener otro nivel de información variada pero las colecciones no.

La mayoría de los objetos integrados de ASP proporcionan colecciones. Una colección es una estructura de datos, similar a una matriz, que almacena cadenas, números, objetos y otros valores. A diferencia de las matrices, las colecciones se amplían y reducen automáticamente al recuperar o almacenar elementos. La posición de un elemento también cambia al modificar la colección. Es posible tener acceso a un elemento de una colección por su clave de cadena única, por su índice (posición) en la colección o si se iteran todos los elementos de la colección.

El acceso a las colecciones, su sintaxis y utilización la veremos según vayamos avanzando en el temario con distintos ejemplos.

Todas las colecciones ofrecen una serie de métodos comunes a todas ellas y son los siguientes:

- Count: devuelve número de elementos contenidos en la colección.
- Item: devuelve el elemento que se corresponde con el índice o cadena clave que se pasa por parámetro.
- Key: devuelve el nombre de una clave dado su índice.
- Remove: elimina un elemento determinado de la colección, del que indicamos su clave o índice. La colección debe ser de escritura.
- RemoveAll: elimina todos los elementos presentes en una colección. Al igual que el método anterior requiere que la colección sea de escritura.

Las colecciones comienzan en el índice 1, a diferencia de los arrays que empiezan en el índice cero.

Antes de poder utilizar un objeto se debe instanciar, es decir, crear el objeto y asignárselo a una variable que va a representar una instancia de ese objeto, en realidad, hasta que no se instancia un objeto, el objeto no existe. Pero en el caso de los objetos integrados no hay que instanciarlos, se puede considerar que son creados al iniciar la ejecución la aplicación ASP.

Los objetos que pertenecen al modelo de objetos de ASP 3.0 son siete, y a continuación se comenta de forma breve el significado de cada uno de ellos.

- Response: mediante este objeto podremos enviar datos al cliente que recibirá en HTML y controlar la salida que el usuario va a recibir en su navegador Web. En el tema anterior ya hemos utilizado este objeto para escribir diferentes resultados en la página ASP.
- Request: este objeto lo vamos a utilizar para recuperar información del usuario. El objeto Request se utiliza sobretodo para recuperar información de formularios y de las cadenas de consulta (QueryString) del cliente (el navegador Web), toda esta información se guarda dentro de las colecciones del objeto Request, que veremos detenidamente en su momento. Podemos decir que el objeto Response y Request son complementarios, ya que el primero lo utilizamos para enviar información al navegador y el segundo para obtener información del mismo.
- Application: este objeto representa a la aplicación ASP y es utilizado para almacenar información a nivel de aplicación, es decir, variables globales que son comunes a toda la aplicación ASP. En el capítulo correspondiente veremos de forma detallada el significado de una aplicación ASP y la forma en la que podemos crearla desde Internet Information Server 5.0 (IIS 5.0). Este objeto tiene dos eventos asociados al inicio de la aplicación ASP y a la finalización de la ejecución de la aplicación.
- Session: representa una sesión de un usuario (navegador Web) con una aplicación ASP, es utilizado para almacenar información específica de cada usuario conectado a una aplicación ASP. La información contenida en el objeto Session es particular para cada usuario y no se puede intercambiar.
- Server: mediante este objeto podemos acceder a los recursos del servidor Web, permite crear objetos registrados en el servidor y que pueden ser de diverso tipo, desde objetos pertenecientes a componentes de servidor que se ofrecen con ASP hasta componentes desarrollados por nosotros mismos o terceras partes. Este objeto es de difícil clasificación y muchas veces se define como un objeto de utilidades.
- ObjectContext: este objeto nos permite gestionar nuestras páginas ASP transaccionales, este objeto viene de la integración del servidor Web Internet Information Server 5.0 con el Servicio de componentes de Windows 2000. Este objeto posee dos eventos que se corresponden con el éxito o fallo de la transacción. A través del objeto ObjectContext podremos abortar una transacción o llevarla a cabo, en el tema correspondiente trataremos en profundidad las transacciones.
- ASPError: es un nuevo objeto de ASP 3.0 y va a contener toda la información detallada del último error que se haya producido en la aplicación ASP correspondiente. Este objeto es el único al que no se puede acceder directamente sino que se obtiene a partir de un método otro objeto integrado, el objeto Server.

## Equipo necesario

En el primer tema, dónde se realizaba una introducción a ASP, ya se comentaba los requerimientos necesarios para la ejecución y utilización de páginas ASP, aquí lo volvemos a recordar brevemente y también lo ampliamos con los requerimientos de hardware.

Para seguir el curso satisfactoriamente se debe tener los siguientes requisitos de hardware y software:

- Cualquier versión de Windows 2000 (Professional, Server, Advanced Server).
- Microsoft SQL Server 7 como sistema gestor de bases de datos.
- Procesador Pentium II a 400 MHZ o superior.
- 1 GB de espacio libre en disco.
- 64 MB de RAM para Windows 2000 Professional y 128 de RAM para Windows 2000 Server/Advanced Server.

Se debe tener en cuenta que estos requisitos no son los de un equipo que vaya a realizar la función de un servidor Web con aplicaciones ASP, estos requisitos se refieren a una máquina que va a realizar la función de desarrollo de aplicaciones ASP, no es un servidor en producción. En el tema dedicado al servidor Web IIS 5.0 detallaremos los requisitos del servidor Web.

Para quién interese esta información voy a detallar la configuración del equipo utilizado para la realización de este curso.

- Pentium II a 400 MHZ con 128 de RAM.
- Windows 2000 Server.
- Microsoft SQL Server 7.

## El servidor web

Al instalar Windows 2000, en cualquiera de sus versiones, deberemos indicar que deseamos instalar también el componente de Windows 2000 llamado Servicios de Internet Information Server (IIS). Este componente va a convertir nuestro equipo en un servidor Web que soporte aplicaciones ASP.

Para averiguar si tenemos instalado IIS 5.0 en nuestra máquina podemos realizar una sencilla prueba que consiste en escribir la siguiente URL <http://nombreEquipo> en el navegador Web. Si IIS 5.0 está instalado aparece una ventana similar a la Figura 11.

Otra forma de comprobarlo es acudiendo al menú de inicio de Windows y en el grupo de programas Herramientas administrativas, debemos tener el Administrador de servicios de Internet (Figura 12).

Si observamos que IIS 5.0 no se encuentra instalado en nuestra máquina deberemos ir al Panel de Control y en la opción Agregar/quitar programas seleccionar componentes de Windows. Una vez hecho esto marcaremos el componente Servicios de Internet Information Server y procederemos a su instalación.

En el capítulo dedicado a IIS 5.0 se entrará en más detalle, explicando como configurar el servidor Web y como crear y configurar nuestras aplicaciones ASP.



Figura 11

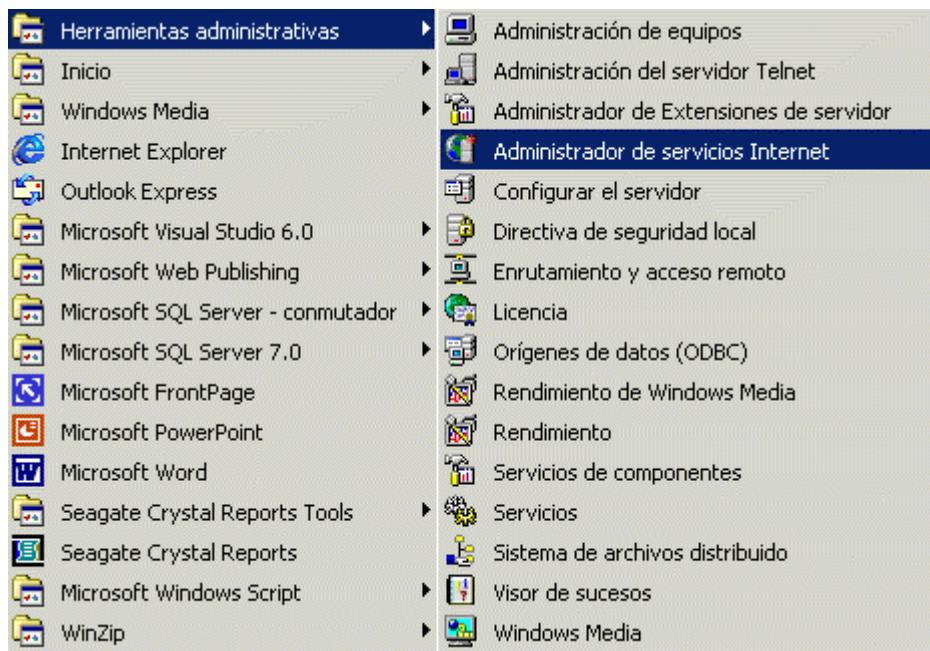


Figura 12

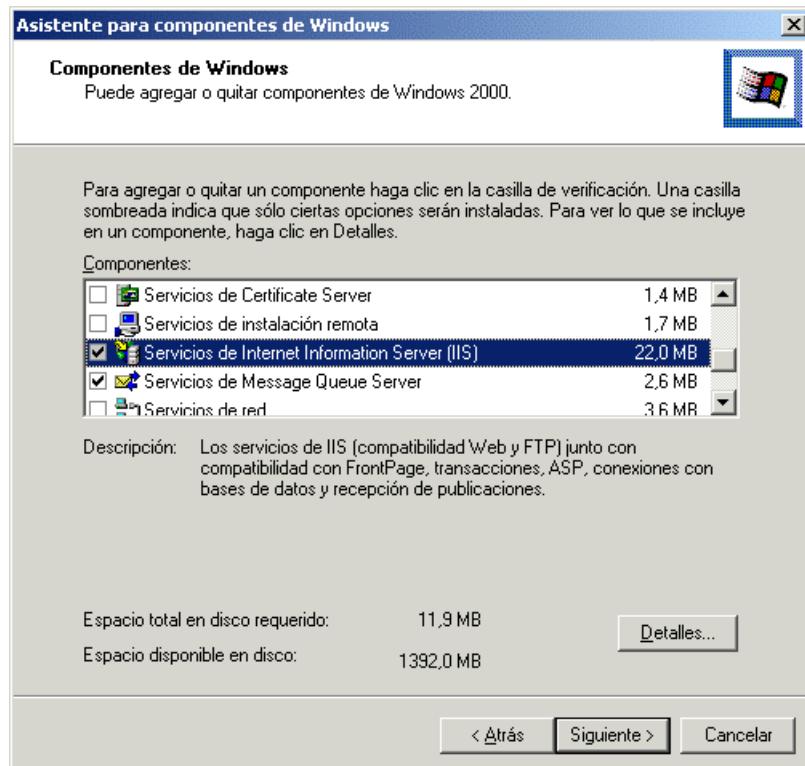


Figura 13

## La herramienta de desarrollo

La herramienta de desarrollo que se va a utilizar para crear las páginas ASP va a ser Visual InterDev 6, que forma parte del conjunto de herramientas de desarrollo de Microsoft Visual Studio 6.

Aunque dentro del curso existe un tema específico dedicado por completo a Visual InterDev 6, aquí vamos a comentar unos breves pasos para que nos sirva para crear nuestras páginas ASP, como mínimo necesitaremos crear un proyecto en nuestro servidor Web.

Se da por supuesto que tenemos instalado el servidor Web Internet Information Server 5.0. Este es el servidor Web que viene junto con cualquier versión de Windows 2000, como ya hemos visto en el apartado anterior.

Una vez instalado Visual InterDev 6 desde Visual Studio 6, lanzamos la ejecución del mismo y escribimos el nombre que va a tener nuestro proyecto y Seleccionamos Nuevo Proyecto Web. En la siguiente pantalla debemos seleccionar le servidor Web el que se va a crear la carpeta que va a contener las páginas ASP, debemos elegir en nuestro caso el nombre de nuestra máquina, indicando que vamos a trabajar en modo maestro. Pulsamos siguiente.

En el segundo paso vamos a indicar el nombre de la aplicación Web que vamos a crear, se debe aclarar que una cosa es el proyecto y otra cosa la aplicación Web. El proyecto simplemente es una serie de ficheros que guardan referencias de los ficheros contenidos en la aplicación Web. En nuestro caso el cliente y el servidor son el mismo equipo, pero en el caso de tener una máquina de desarrollo y otra distinta que hace la función de servidor Web de desarrollo tendremos que el proyecto está en la máquina local del desarrollador y que apunta a la estructura de la aplicación Web contenida en el servidor Web.

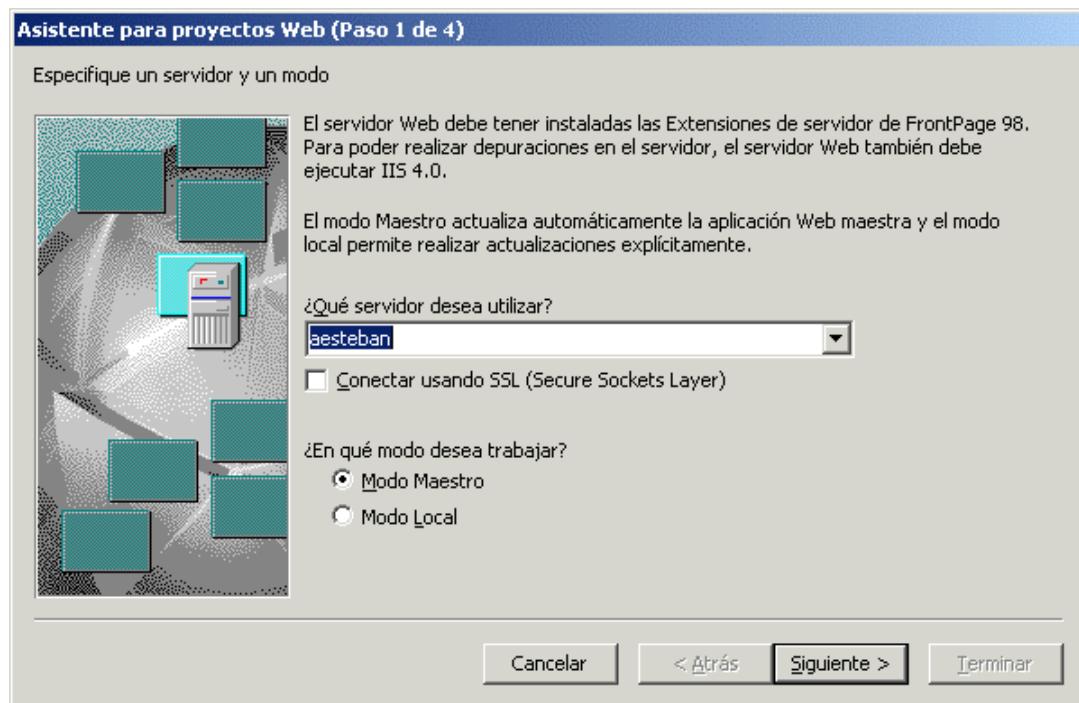


Figura 14

Los conceptos de proyecto, aplicación Web y modo de trabajo los veremos con detalle en el capítulo dedicado a la herramienta Visual InterDev 6, no debemos olvidar que la finalidad de este apartado es la de saber lo mínimo de Visual InterDev para probar los ejemplos de los capítulos venideros.

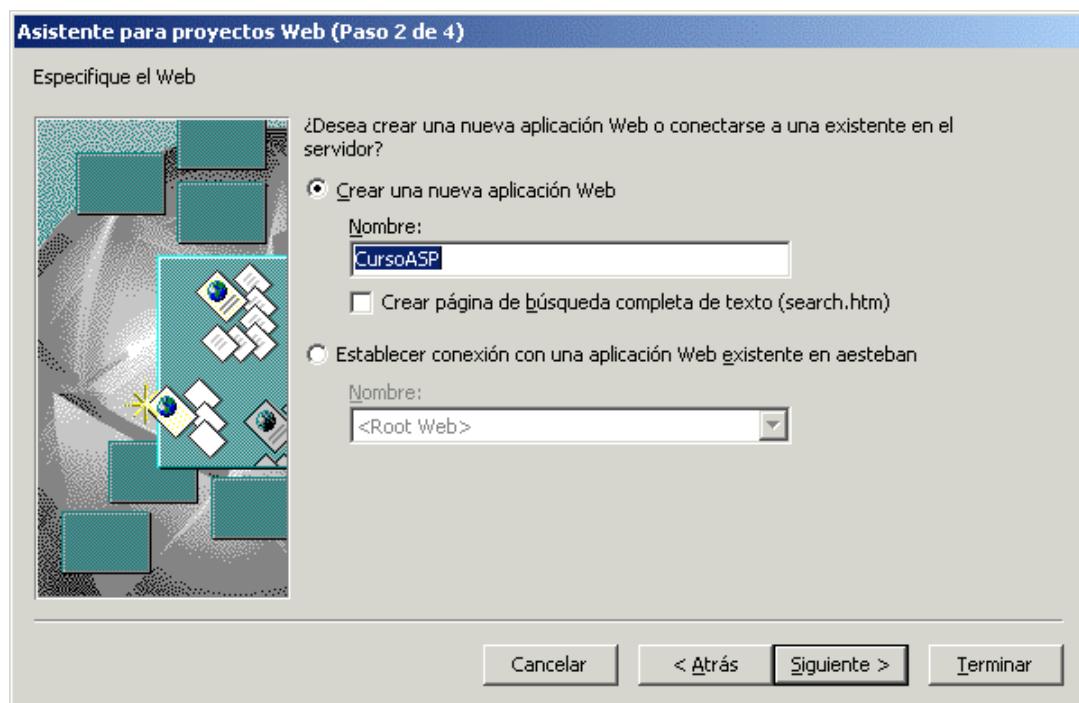


Figura 15

Una vez que hemos pulsado siguiente en este paso, las nuevas pantallas simplemente nos permiten aplicar temas de diseño sobre nuestro sitio Web, por lo que directamente podemos pulsar Terminar para finalizar con la creación del proyecto y la aplicación Web.

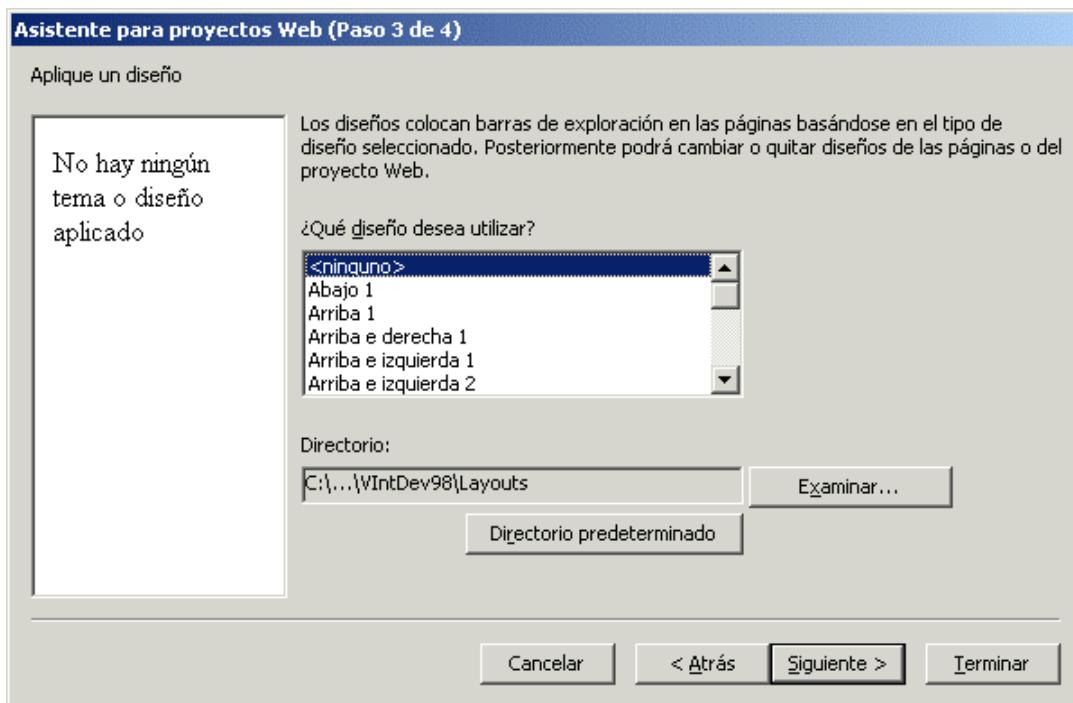


Figura 16

De esta forma ya habríamos terminado de crear nuestro proyecto y en la ventana del explorador del proyecto en Visual InterDev deberíamos tener algo parecido a lo que aparece en la Figura 17.

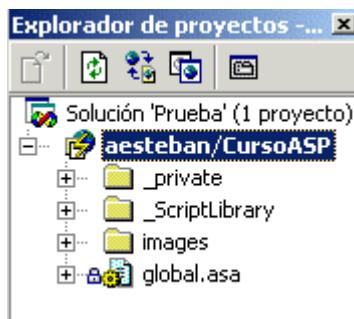


Figura 17

El explorador del proyecto muestra el contenido de la carpeta existente en el servidor Web y que va a contener la aplicación Web.

Ahora que ya hemos creado el proyecto y la aplicación Web vamos a añadir una página ASP que contenga el sencillo código que veímos en el tema uno con el ejemplo de "Hola mundo". Para ello pulsamos con el botón derecho del ratón sobre el proyecto (en el explorador de proyectos), y seleccionamos la opción de menú Agregar|Pagina Active Server.

Damos un nombre identificativo a la página ASP y ya estamos en disposición de escribir el código de la página. Para que el lector no tenga que volver al primer tema volvemos a reproducir el código del ejemplo "Hola mundo", en el Código fuente 64.

```
1.<%@ LANGUAGE="VBSCRIPT"%>
2.<%Option Explicit%>
3.<HTML>
4.<HEAD>
5.<TITLE>Hola Mundo</TITLE>
6.</HEAD>
7.<BODY>
8.<div align="center">
9.<%Dim i
10.For i=1 to 5%>
11.    <FONT SIZE="<%=i%>">Hola Mundo</font><br>
12.<%Next%>
13.</div>
14.</BODY>
15.</HTML>
```

Código fuente 64

Visual InterDev nos ofrece una ayuda denominada intellisense, que consiste en ir dándonos información de los distintos objetos que utilizamos en nuestras secuencias de comandos. Al escribir nuestro código no lo hemos notado, ya que aunque al escribir la línea 11 estamos utilizando el objeto Response mediante los delimitadores `<%=%>`, no aparece su nombre como tal. Pero si cambiamos la línea 11 por la línea de código que muestra el Código fuente 65, que es equivalente, veremos el mecanismo intellisense en acción.

```
<FONT SIZE="<%Response.Write i%>">Hola Mundo</font><br>
```

Código fuente 65

Una vez terminada nuestra página la grabamos y ahora debemos probarla en el navegador Web. Para ver la ejecución de una página ASP no debemos hacer doble click sobre ella como si abriéramos cualquier tipo de archivo, sino que siempre debemos abrirla mediante un navegador Web utilizando el protocolo HTTP para que el servidor Web pueda ejecutar e intretar dicha página. La ejecución de la página ASP la podemos realizar de dos formas:

- Abriendo el navegador Web y escribiendo la URL que identifica a la página de la siguiente manera: `http://nombreMaquina/carpeta/pagina.asp`. Así en mi caso deberé escribir lo siguiente `http://aesteban/CursoASP/holaMundo.asp`.
- También podemos ver el resultado de la ejecución de la página ASP directamente desde Visual InterDev, pulsando con el botón derecho del ratón sobre la página en el explorador de proyectos y seleccionando la opción de menú Ver en el explorador.

En cual quiera de los casos el resultado es el mismo, y debería ser como el de la Figura 18.

Hola Mundo  
Hola Mundo  
Hola Mundo  
**Hola Mundo**  
**Hola Mundo**

Figura 18

Como se puede comprobar en la nuestra máquina también debemos disponer de un navegador Web para poder probar los ejemplos, cosa que resulta evidente sino el lector no podrá estar viendo estas páginas. Para probar los ejemplos del curso se ha utilizado el navegador Web de Microsoft Internet Explorer 5, aunque como ya sabemos las páginas ASP son completamente independientes del navegador utilizado.

# 5

## **Modelo de objetos de ASP: el objeto Response**

---

### **Definición del objeto Response**

Este objeto integrado en el modelo de objetos de ASP tiene la función de enviar datos al cliente, es decir, al navegador que ha cargado la página ASP. A través de este objeto podremos escribir en la página que visualizará el usuario e incluso podremos redirigir al usuario a otra dirección en Internet. También a partir de este objeto podremos definir cookies para el usuario y asignarles un valor.

A lo largo de este capítulo vamos a comentar y ver con ejemplos las colecciones, propiedades y métodos que ofrece el objeto Response.

### **Colecciones del objeto Response**

El objeto Response posee una única colección llamada Cookies, que le permite crear y asignar valores a una cookie. Una cookie, físicamente, es un fichero que se escribe en la máquina local del cliente que se conecta a un sitio Web y que contiene información relativa a la conexión.

Una cookie es utilizada para mantener información entre diferentes conexiones HTTP. Se debe recordar que el protocolo HTTP es un protocolo sin estado, es decir, no se retiene información entre las diferentes conexiones que se realicen. Por esta razón, ni el cliente ni el servidor pueden mantener información entre diferentes peticiones o a través de diferentes páginas Web.

Este mecanismo para mantener información entre diferentes conexiones HTTP fue propuesto e implementado en un principio por la compañía Netscape.

Existen varios usos prácticos de las cookies, a continuación se van a comentar los más destacados:

- Para almacenar información acerca de las preferencias del cliente que se conecta a nuestro sitio Web, por ejemplo el color seleccionado de la página, el tipo de letra, etc.
- Para conservar información personal, no sensible, del usuario, como puede ser el nombre, el país de origen, código postal, el número de veces que ha accedido a nuestro sitio Web, etc.

Por lo tanto el uso de cookies nos puede permitir personalizar las páginas ASP según el cliente que se haya conectado atendiendo a sus preferencias y datos personales. Por ejemplo podemos saludar al usuario con su nombre y asignar al color de fondo de la página su color favorito o también podremos indicarle el número de veces que ha accedido a nuestro sitio Web.

De esta forma podemos evitar preguntar al usuario sus preferencias o datos personales cada vez que entra en nuestro sitio Web.

Siempre debe haber una primera ocasión en la que el cliente conectado especifique el valor de la cookie, una vez especificado este valor ya puede ser utilizada la cookie en las diferentes conexiones que realice ese cliente, ya que la información ha quedado almacenada en la cookie. Esta información se almacena físicamente en un fichero del disco duro local del cliente. En el caso del navegador Internet Explorer de Microsoft cada cookie se corresponde con un fichero.txt que se encuentra en el directorio Documents and Settings (en el caso de Windows 2000) y en el subdirectorio Cookies, y en el caso del Communicator de Netscape las cookies se almacenan todas en un fichero llamado cookies.txt en el subdirectorio Netscape\User\usuario.

Hay una serie de consideraciones que se deben tener en cuenta a la hora de utilizar cookies en nuestra aplicación ASP:

- Las cookies se pueden perder, por lo tanto nunca se debe depender de las cookies para almacenar una información que no se pueda volver a generar. Este tipo de información se debería almacenar en una base de datos del servidor. No se debe olvidar que las cookies se almacenan en el disco local del cliente en ficheros, y por lo tanto estos ficheros se pueden dañar, ser borrados o sobreescritos.
- Las cookies pueden ser modificadas por el cliente, por lo tanto nunca se debe considerar que la información ofrecida por una cookie es auténtica. Si estamos usando una cookie para determinar la fecha de la última vez que un usuario visitó nuestro sitio Web podemos considerar como auténtica esta información sin ningún problema, pero sin embargo no es nada recomendable considerar auténtica una cookie que posee el número de cuenta de un usuario. Como regla general nunca se debe utilizar una cookie para almacenar información confidencial, este tipo de información se debería almacenar en una base de datos en el servidor Web.
- Las cookies pueden ser copiadas sin el consentimiento del propietario, nunca se debería utilizar una cookie para identificar a un usuario, una solución mejor es utilizar una contraseña y validarla con una base de datos del servidor.
- Algunos navegadores no reconocen las cookies. Incluso los más populares como el Internet Explorer y el Communicator, se pueden configurar para no utilizar cookies.

Se debe señalar que ASP para mantener estados, es decir, valores de variables, objetos..., entre diferentes páginas de una aplicación ASP utiliza cookies desde los objetos integrados Session y Application.

El uso de estas cookies por parte de ASP, se oculta completamente al programador para que se abstraiga de la utilización lógica de las cookies, todo ello gracias a los objetos Session y Application que las gestionan internamente (estos dos objetos los trataremos en los capítulos correspondientes). Así por ejemplo, cuando se inicia una sesión se crea la cookie ASPSESSIONID, aunque esta cookie no se grabará en el disco duro del cliente, sino que permanecerá en memoria, por lo tanto podemos asegurar al usuario que no vamos a acceder a su disco.

De esta forma si un navegador no acepta cookies, la aplicación ASP no podrá mantener el estado entre diferentes páginas. Si tenemos configurado nuestro navegador para que nos avise cuando recibe una cookie, al iniciar una sesión con una aplicación ASP aparecerá una ventana similar a la que se muestra en la Figura 19.

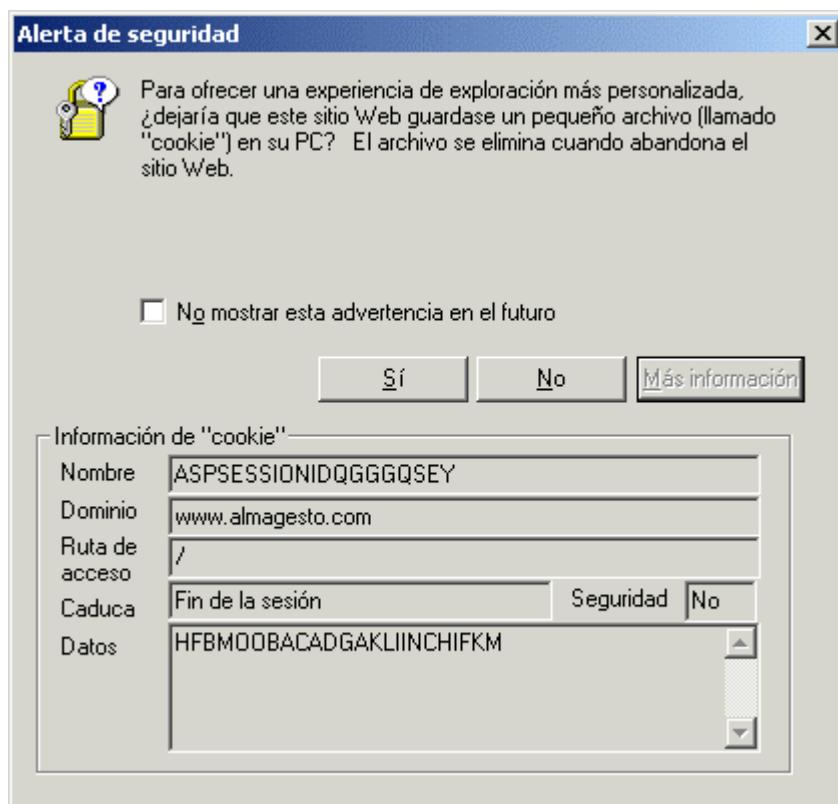


Figura 19. Cookie de inicio de sesión.

La colección Cookies también aparece en el objeto Request, pero en el objeto Request la colección es sólo de lectura, mientras que en el objeto Response es sólo de escritura. De esta forma en el objeto Response la colección Cookies será utilizada para crear las cookies o para modificar su valor, y en el objeto Request para recuperar el valor de las cookies.

Para crear una cookie se utiliza la siguiente sintaxis:

```
Response.Cookies(NombredelaCookie)=ValordelaCookie
```

Si la cookie ya existe modifica su valor. La sintaxis general de una cookie es la que se muestra a continuación:

```
Response.Cookies(cookie) [(clave)|atributo]=valor
```

Donde cookie, es el nombre de la cookie a la que hacemos referencia, clave es un parámetro opcional que se utiliza cuando la cookie es un diccionario, es decir puede contener diferentes datos, y atributo especifica una propiedad de la cookie. Las propiedades que poseen las cookies son:

- Expires: sólo de escritura, indica la fecha en la que caduca la cookie. Si no se especifica ningún valor, por defecto caduca cuando termina la sesión.
- Domain: sólo de escritura, especifica a qué dominio es enviada la cookie. Por defecto será el nombre del dominio del servidor del que proviene la cookie.
- Path: sólo de escritura, indica la ruta del servidor de la que proviene la cookie. Si no se especifica, por defecto se toma la ruta de la página ASP que generó la cookie.
- Secure: sólo de escritura, para indicar si la cookie es segura.
- HasKeys: sólo de lectura, especifica si la cookie tiene claves, es decir, si es un diccionario.

Si queremos crear una cookie que tenga claves, por ejemplo que contenga los datos nombre, apellidos y edad, y que caduque a finales del 2000 se debería escribir lo que indica el Código fuente 66.

```
<%
  Response.Cookies("datosPersonales").Expires= #December 31, 2000#
  Response.Cookies("datosPersonales")("nombre")="Angel"
  Response.Cookies("datosPersonales")("apellidos")="Esteban Núñez"
  Response.Cookies("datosPersonales")("edad")=25
%>
```

Código fuente 66

En el caso de que se quiera eliminar una cookie que ya exista, basta con cambiar el valor de su propiedad Expires, asignándole una fecha anterior a la actual.

Las cookies se envían desde el servidor al navegador del cliente como un encabezado, de esta forma, el encabezado de la cookie del ejemplo tendría el aspecto del Código fuente 67.

```
Set-Cookie:DATOSPERSONALES=EDAD=25&APELLODOS=Esteban+N%FA%Flez&NOMBRE=Angel
```

Código fuente 67

Y si tenemos el navegador configurado para que nos advierta sobre la utilización de cookies veríamos el mensaje que muestra la Figura 20.

Esto sería con Internet Explorer 5, pero si hacemos lo mismo con Netscape 4.5, además nos muestra el contenido de nuestra cookie como se puede apreciar en la Figura 21.

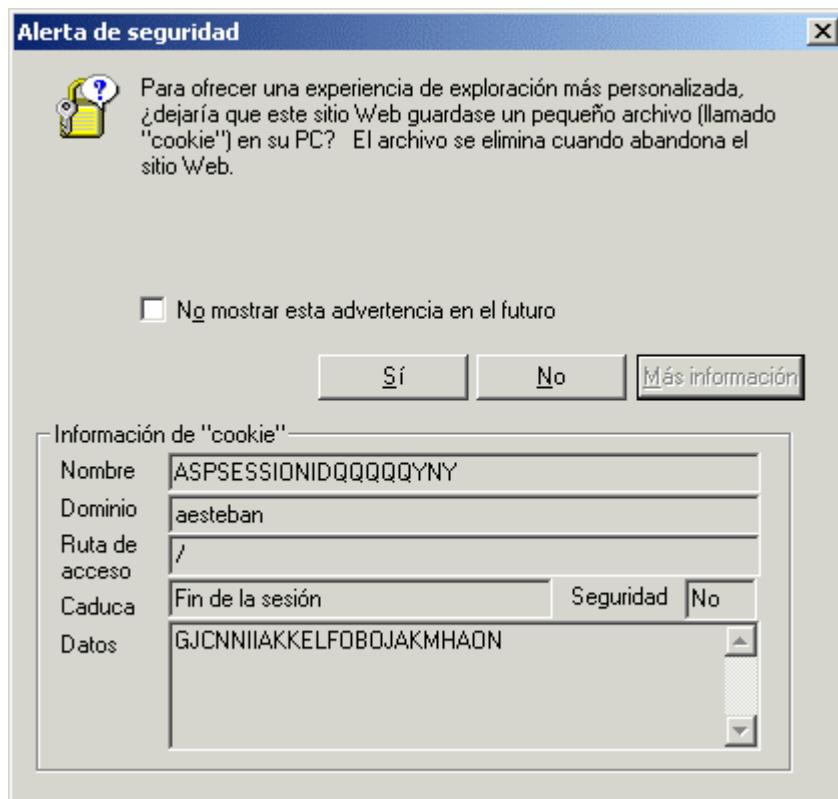


Figura 20. Recibiendo una cookie.

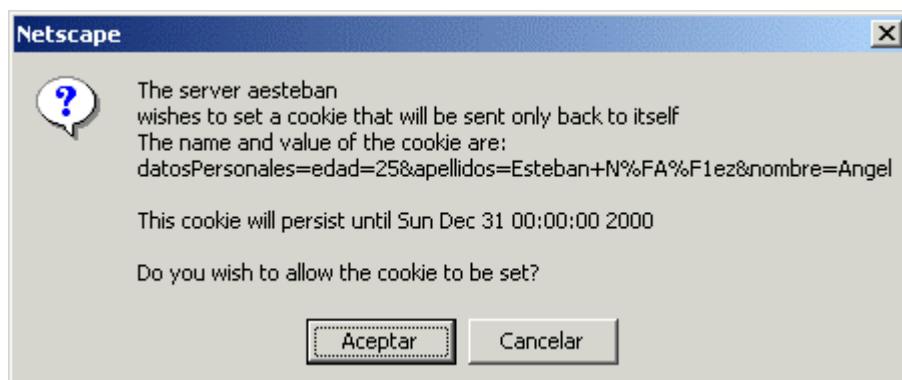


Figura 21. Una cookie en Netscape

## Propiedades del objeto Response

En este apartado vamos a comentar cada una de las propiedades del objeto Response, y como podemos utilizar las más comunes.

El objeto Response posee las siguientes propiedades:

- Buffer: indica si los datos de la página se almacenan en un búfer.
- ContentType: especifica el tipo de contenido HTTP de la respuesta.

- Expires: especifica el intervalo de tiempo que debe transcurrir para que caduque una página almacenada en la caché del navegador.
- ExpiresAbsolute: tiene el mismo sentido que la propiedad anterior, pero aquí se especificar la hora y la fecha en la que caducará la página en la caché del navegador.
- Status: valor de la línea de estado que devuelve el servidor.
- CacheControl: determinará si los servidores proxy van a almacenar o no en su caché la salida generada por una página ASP.
- CharSet: indicará el juego de caracteres que está utilizando la página ASP.
- PICS: mediante esta propiedad indicaremos el tipo de contenido que posee nuestra página ASP.
- IsClientConnected: propiedad de sólo lectura con la que podremos averiguar si el cliente se ha desconectado del servidor.

## Buffer

Si utilizamos la propiedad Buffer del objeto Response asignándole el valor True, conseguiremos que ASP procese todo el script en el servidor antes de enviar algo al usuario, por defecto esta propiedad tiene el valor True. El contenido del búfer no es enviado al navegador hasta que no se haya terminado de procesar la página o hasta que los métodos End o Flush del objeto Response no son invocados.

Una vez activado el almacenamiento en búfer, es decir, la propiedad Buffer posee el valor True, y una vez que ya se ha enviado contenido al navegador del cliente, no se puede modificar su valor, tampoco se puede modificar ningún valor de las propiedades del objeto Response. Así por ejemplo el Código fuente 68 generaría un error.

```
<%Response.Buffer=False%>
<HTML>
<HEAD>
<%Response.Expires=10%>
```

Código fuente 68

El error devuelto por el navegador sería similar al siguiente.

```
objeto Response error 'ASP 0156 : 80004005'
Error de encabezado
/cursoasp30/pruebas.asp, line 4
Los encabezados HTTP ya están escritos en el explorador cliente.
Cualquier cambio en el encabezado HTTP se debe hacer antes de
escribir
el contenido de la página.
```

En las versiones anteriores de ASP la propiedad Buffer tenía por defecto la propiedad False, de esta forma según se iba procesando la página ASP se enviaba el resultado de la ejecución al cliente. Según

Microsoft con este cambio se obtiene un mejor rendimiento en la ejecución de páginas ASP. En realidad lo que se tiene también es un mayor control sobre la salida que va a recibir el usuario.

El uso de este búfer puede ser útil en el caso de que sea necesario procesar el contenido de una página ASP antes de enviar ningún contenido previo al navegador del cliente. De esta forma se puede redireccionar al navegador hacia otra página con el método Redirect del objeto Response, este método se comentará en su apartado correspondiente, pero se puede adelantar que una llamada a este método producirá un error en el caso de que se haya enviado anteriormente algún contenido al usuario.

La propiedad Buffer también puede ser utilizada para verificar errores antes de enviar información al navegador, de esta forma si se detecta un error se puede detener el proceso de la página y redireccionar al navegador del cliente hacia una página determinada.

Cuando la propiedad Buffer, tiene su valor por defecto, es decir, el valor True se pueden utilizar una serie de métodos del objeto Response, estos métodos, cuya función se verá en el apartado correspondiente, son: Clear, End y Flush y básicamente lo que permiten es controlar el resultado o salida que se le envía al cliente.

## ContentType

La propiedad ContentType sirve para definir el tipo de contenido que tiene la página, y tiene la siguiente sintaxis:

```
Response.ContentType=TipoContenido
```

Donde TipoContenido es una cadena que describe el tipo de contenido y que normalmente tiene el formato tipo/subtipo, donde tipo es la categoría general del contenido y subtipo es el tipo de contenido específico. Si no se especifica el valor de la propiedad ContentType, su valor por defecto es "text/html", es decir código HTML. Estos encabezados son utilizados por el servidor para indicar al cliente que tipo de información le envía, esta información se denomina tipos MIME.

Los valores más comunes de estos tipos y subtipos son los mostrados en la Tabla 10.

Tipo/subtipo	Descripción
text/html	Código HTML. Especifica que el navegador del cliente debería interpretar el código HTML a mostrar
text/plain	Indica un texto ASCII. Especifica que el navegador debería únicamente mostrar texto sin interpretar código.
image/gif	Un fichero gráfico GIF. El navegador debería mostrar la imagen o iniciar una aplicación que pueda visualizar la imagen.
image/jpeg	Un fichero gráfico JPEG.
text/xml	Código XML. Especifica que el navegador cliente debería interpretar el código XML.

Tabla 10

El valor de esta propiedad la podemos cambiar, por ejemplo, para que en lugar de que el navegador interprete el código HTML, lo muestre tal como es, esto lo conseguimos con el Código fuente 69.

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

Código fuente 69

## Expires

Mediante Expires podemos indicar en minutos el tiempo que debe pasar para que una página caduque en la caché del navegador, su sintaxis es:

```
Response.Expires= minutos
```

Si se quiere que la página caduque inmediatamente se deberá asignar el valor 0 a esta propiedad, aunque la práctica demuestra que es más efectivo asignarle un valor negativo alto como puede ser -1000.

Con la propiedad ExpiresAbsolute indicamos en qué fecha y a qué hora caducará la página en la caché del navegador.

Por ejemplo si queremos que una página caduque el día 1 de Mayo de 2000 a las cuatro de la tarde, escribiremos el Código fuente 70.

```
<% Response.ExpiresAbsolute=#May 1,2000 16:00# %>
```

Código fuente 70

## Status

Esta propiedad indica un código de estado del protocolo de transferencia de hipertexto HTTP.

Los valores de la propiedad Status se definen en la especificación del protocolo HTTP y su sintaxis es la siguiente:

```
Response.Status=DescripcionEstado
```

Donde DescripcionEstado es una cadena formada por un número de tres dígitos, que es el código de estado, y una pequeña descripción del estado, por ejemplo el Código fuente 71.

```
<%Response.Status="401 Unauthorized"%>
```

Código fuente 71

Algunos de los códigos de estado HTTP más comunes son los que aparecen en la Tabla 11.

Código	Descripción
200	La petición se realizó con éxito y la información fue enviada. Este código se envía cada vez que una página se carga con éxito.
301	El contenido pedido ha sido movido permanentemente a otra localización.
302	La página solicitada se ha trasladado temporalmente a otra localización.
404	La información solicitada no ha podido ser localizada en el servidor.
500	En el servidor se produjo un error interno y la petición no pudo ser servida.

Tabla 11

## CacheControl

A través de la propiedad CacheControl podremos determinar si los servidores proxy van a almacenar en su caché la salida generada por una página ASP o no. Esta propiedad tiene dos valores posibles: Private y Public. Si queremos que la salida de la página ASP permanezca en la caché de un servidor proxy asignaremos a esta propiedad el valor Public, pero si por el contrario no deseamos que la salida de la página ASP permanezca en la caché del servidor proxy, asignaremos a la propiedad CacheControl el valor Private. Su sintaxis es la siguiente:

```
Response.CacheControl = "Public" / "Private"
```

## CharSet

Mediante la propiedad CharSet podremos indicar al navegador cliente el juego de caracteres que utilizará la página ASP. Gracias a esta propiedad podremos solventar problemas entre los diferentes idiomas, en el caso de que el navegador desconozca el juego de caracteres que debe utilizar para mostrar la página. Su sintaxis es:

```
Response.CharSet = JuegodeCaracteres
```

Así por ejemplo, si en una página .asp incluimos la sentencia que muestra el Código fuente 72, el encabezado de tipo de contenido correspondiente sería el Código fuente 73.

Se debe señalar que si en una misma página ASP incluimos varias sentencias Response.CharSet, con cada sentencia se reemplazará el juego de caracteres anterior. Por lo que el juego de caracteres quedará establecido al valor especificado por la última sentencia Response.CharSet de la página.

```
<% Response.CharSet = "ISO-LATIN-7" %>
```

Código fuente 72

```
content-type: text/html; charset=ISO-LATIN-7
```

Código fuente 73

## PICS

Mediante la propiedad PICS indicaremos el tipo de contenido que posee nuestra página ASP. PICS (Platform of Internet Content Selection) es una especificación definida por la entidad RSACI (Recreational Software Advisory Council on the Internet (<http://www.rsac.org>)) . Esta entidad clasifica los contenidos en Internet atendiendo a la violencia, sexo y lenguaje de las páginas. Lo cual es de utilidad a la hora de limitar el acceso a ciertos contenidos a los que no deberían tener acceso los menores.

A esta propiedad se le asigna una cadena de caracteres que debe cumplir con el formato especificado por las etiquetas PICS. Su sintaxis es:

```
Response.PICS(EtiquetaPICS)
```

Si incluimos en una página ASP la sentencia que muestra el Código fuente 74.

```
<% Response.PICS("(PICS-1.1 http://www.rsac.org/ratingv01.html labels on " &
chr(34) &
"1997.01.05T08:15-0500 " & chr(34) & " until " & char(34) & " 1999.12.31T23:59-0000
" &
chr(34) & "ratings (v 0 s 0 l 0 n 0))") %>
```

Código fuente 74

Se agregará el encabezado http que muestra el Código fuente 75.

```
PICS-label: (PICS-1.1 http://www.rsac.org/ratingv01.html labels on
"1997.01.05T08:15-0500" until "1999.12.31T23:59-0000"           ratings (v 0 s 0 l
0 n 0))
```

Código fuente 75

Nota: como las etiquetas PICS contienen comillas, se reemplaza cada comilla con "**& chr(34) &**"

Si en una misma página ASP incluimos varias sentencias Response.PICS, con cada sentencia se reemplazará la etiqueta PICS anterior. Por lo que la etiqueta PICS quedará establecida al valor especificado por la última sentencia Response.PICS de la página.

Esta propiedad es sólo de escritura, al contrario que todas las anteriores que eran de lectura/escritura.

## IsClientConnected

A través de la propiedad IsClientConnected podremos averiguar si el cliente se ha desconectado del servidor desde la última petición realizada. Esta propiedad es de sólo lectura.

La propiedad IsClientConnected devolverá un valor booleano a través del cual podremos controlar si el cliente se ha desconectado y por lo tanto no será necesario seguir procesando la página ASP y enviando información al cliente, esto es bastante interesante de cara a la mejora del rendimiento de nuestro sitio Web, ya que nos permite cancelar peticiones que ya no son esperadas por el cliente. La sintaxis es la siguiente:

```
Response.IsClientConnected ( )
```

Un ejemplo genérico de la utilización de esta propiedad es el que muestra el Código fuente 76.

```
<%If Not Response.IsClientConnected Then
    'Código a ejecutar en el caso de que el cliente no esté conectado
    'Esto supone detener la ejecución de la página ASP
End If
'Se sigue procesando la página ASP%>
```

Código fuente 76

En ASP 2.0 podíamos consultar esta propiedad sólo si antes habíamos enviado ya alguna salida o contenido al cliente, ahora con ASP 3.0 podemos utilizar IsClientConnected antes de enviar cualquier contenido al navegador.

## Métodos del objeto Response

A continuación, de la misma forma que se ha hecho con las propiedades del objeto Response, se va a hacer una breve descripción de los métodos que nos ofrece este objeto.

- AddHeader: agrega un nuevo encabezado HTML a la respuesta. Este método no es muy utilizado. Permite definir nuevos encabezados. Su sintaxis es: Response.AddHeader nombreEncabezado, valorEncabezado.
- AppendToLog: agrega una cadena al final de la entrada de registro del servidor Web para la petición. IIS se puede configurar para registrar las peticiones de los clientes en una base de datos o en un fichero de log. Este método permite agregar una cadena a la información que será almacenada por el usuario acerca de una petición de un cliente.
- BinaryWrite: este método es utilizado para escribir información directamente en la salida HTTP, es decir, el navegador del cliente, sin ningún tipo de interpretación. Este método es útil cuando se necesita enviar pequeños gráficos, multimedia u otros componentes que no son texto.
- Clear: elimina el contenido del código HTML del búfer.
- End: detiene el procesamiento de la página ASP y devuelve los resultados obtenidos hasta la llamada a End.

- Flush: envía al navegador el contenido del búfer antes de finalizar el procesamiento de la página ASP.
- Redirect: se redirecciona al navegador Web del cliente a la dirección que se le pasa como parámetro.
- Write: escribe en el navegador del cliente.

## Clear, Flush y End

Se ha decidido agrupar la explicación de estos tres métodos, ya que están íntimamente relacionados y además siempre se suelen usar en combinación. Los métodos Clear, Flush y End, deben ser utilizados cuando la propiedad Buffer del objeto Response esté activada, es decir, tiene como valor True, ya sabemos que este es el valor por defecto de esta propiedad.

Cuando está activado el almacenamiento en el búfer existen varias formas de enviar la información del búfer al navegador del cliente:

- Llamando al método Flush.
- Llamando al método End. De esta forma también se detiene la ejecución de la página ASP.
- Cuando finaliza la ejecución de la página ASP, es decir, cuando se ejecuta la última línea de las secuencias de comandos de servidor.

El método Clear se utiliza cuando se quiere eliminar todo el contenido que posee el búfer, pero esto no implica que se borre el contenido que ya se ha enviado al cliente, a ese contenido ya no se tiene acceso, así por ejemplo el Código fuente 77 produciría un error.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<b>Esto es contenido</b>
<%Response.Flush
Response.Clear
Response.Redirect "pruebas.asp"%>
</BODY>
</HTML>
```

Código fuente 77

La utilización de estos métodos junto con la propiedad Buffer nos permite controlar el procesamiento de la página ASP de una forma muy sencilla y eficaz. Para mostrar el uso de estos métodos junto con el almacenamiento en búfer vamos a ver un sencillo ejemplo que va mostrando poco a poco un texto en pantalla, Código fuente 78.

```
<%strTexto="Texto que se va mostrando poco a poco en el navegador"
For i=1 to Len(strTexto)
    'se realiza una pausa
    For j=1 To 100000
```

```

Next

Response.Write Mid(strTexto,i,1)
'se envía el contenido del búfer
Response.Flush
Next%>

```

Código fuente 78

El efecto de este código es que se va enviando el texto carácter a carácter mediante el método Flush, para que se produzca una pausa se realiza un bucle For adicional. Si situamos la sentencia Response.Flush al final del código, aparecerá todo el texto de una sola vez.

Este ejemplo no resulta muy útil pero muestra bastante bien el la utilización del método Flush y el concepto de búfer. De todas formas los ejemplos que se muestran todavía son bastante simples debido a que de momento sólo contamos con un objeto integrado de ASP, el objeto Response, en el siguiente tema, cuando veamos el objeto Request se podrán ver ejemplos más interesantes.

## Write

A lo largo de los distintos ejemplos que hemos visto hasta ahora en el curso, en alguno de ellos hemos utilizado la sentencia Response.Write, por lo tanto ya sabemos exactamente la función de este método del objeto Response.

Pero también para escribir se han utilizado los delimitadores <%=%>. Estos delimitadores son equivalentes a escribir Response.Write, se pueden considerar como su abreviatura.

El método Response.Write se utilizará dentro de una bloque de script de servidor, es decir, un bloque de código encerrado por los delimitadores <%=%>, y el uso de los delimitadores <%=%> será adecuado cuando se necesite intercalar una línea aislada entre código HTML:

Como ya se ha visto, el método Write se puede utilizar para escribir texto en el navegador. Las llamadas a métodos y las variables dentro del método Write serán evaluadas y sólo el resultado se escribirá en el navegador del cliente.

Por ejemplo, el Código fuente 79 mostrará el valor de la variable contador en el navegador del cliente.

```
<%Response.Write("La variable contador contiene el valor: "&contador)%>
```

Código fuente 79

El uso de paréntesis en el método Write es opcional. Si tenemos en cuenta los delimitadores <%=%>, de forma equivalente podríamos haber escrito lo que muestra el Código fuente 80, o bien, lo que muestra el Código fuente 81.

```
<%"La variable contador contiene el valor: "&contador%>
```

Código fuente 80

```
La variable contador contiene el valor: <%=contador%>
```

Código fuente 81

Si en nuestro código tenemos diferentes líneas con los delimitadores `<%=%>` intercaladas dentro de código HTML, para una mayor eficiencia es recomendable sustituirlos por una única línea mediante `Response.Write`. Además se consigue un código más legible, se debe tender a bloques de HTML y de script de servidor bien diferenciados, siempre que sea posible.

Vamos a realizar una sencilla página que utilizando el método `Write`, pero a través de la de los delimitadores `<%=%>`, muestre los valores por defecto que tienen todas las propiedades del objeto `Response` en nuestro servidor Web.

```
Response.Buffer: <%=Response.Buffer%><br>
Response.Expires: <%=Response.Expires%><br>
Response.CacheControl: <%=Response.CacheControl%><br>
Response.ExpiresAbsolute: <%=Response.ExpiresAbsolute%><br>
Response.IsClientConnected: <%=Response.IsClientConnected%><br>
Response.Status: <%=Response.Status%><br>
Response.CharSet: <%=Response.CharSet%><br>
Response.ContentType: <%=Response.ContentType%><br>
```

Código fuente 82

En mi caso el resultado de la ejecución del Código fuente 82 es el siguiente:

```
Response.Buffer: True
Response.Expires:
Response.CacheControl: private
Response.ExpiresAbsolute: 0:00:00
Response.IsClientConnected: True
Response.Status: 200 OK
Response.CharSet:
Response.ContentType: text/html
```

## Redirect

Este método recibe como parámetro una cadena de texto en forma de URL que especifica la nueva dirección a la que se va a redirigir el navegador del usuario. La URL puede ser absoluta, del tipo `www.almagesto.com` o del tipo `/directorio/pagina.asp`, o relativa del tipo `página.asp`. Al lanzar el método `Redirect` en realidad lo que estamos haciendo es enviar una cabecera HTTP al navegador Web del cliente para indicarle que la página se ha movido y le indicamos la nueva localización, que es la URL que hemos indicado al método `Redirect`. La información que se envía al navegador es similar a la que se muestra en el Código fuente 83.

```
HTTP/1.1 Object Moved
Location /directorio/pagina.asp
```

Código fuente 83

Como se puede observar la redirección se realiza en el cliente, indicándole que la página se ha movido a una nueva dirección, que en realidad es otra página completamente distinta.

Aquí nos hemos centrado en redireccionar el navegador a otra página (HTML o ASP), pero se puede generalizar diciendo que podemos redireccionar al navegador a cualquier recurso, como puede ser una imagen, un fichero zip, un documento, etc.

## AddHeader

Con este método podemos añadir cabeceras de respuesta al protocolo HTTP, así por ejemplo, una llamada al método Response.Redirect "pruebas" se puede sustituir mediante la utilización del método AddHeader que añadirá las cabeceras HTTP correspondientes, y la propiedad Status que indicará el código de estado correspondiente.

```
<%Response.Status="302 Object Moved"  
Response.AddHeader "Location", "pruebas.asp"%>
```

Código fuente 84

Como se puede ver es mucho más claro utilizar el método Redirect.

Al igual que ocurría con la modificación de las propiedades del objeto Response y su método Redirect una llamada al método AddHeader generará un error si ya se ha enviado algún contenido al navegador del cliente, ya que supone realizar un cambio al encabezado HTTP.

## AppendToLog

IIS permite habilitar el registro de la actividad en un sitio Web, es decir, las peticiones que realizan los clientes y las respuestas a las mismas.

Con el método AppendToLog podemos añadir al registro del servidor Web Internet Information Server entradas personalizadas. Para poder utilizar este método debemos configurar IIS 5.0 para que registre el acceso al sitio Web correspondiente en el formato de archivo de registro extendido W3C.

Esta forma de registro va añadiendo a un fichero de texto con un formato determinado las peticiones que se van realizando y sirviendo. Con AppendToLog se añade una cadena en la última línea del fichero de registro. En el Código fuente 85, se puede ver la utilización de este método. Si vemos el fichero de registro podría tener el contenido que se muestra a continuación.

```
<%Response.AppendToLog "Esto es una línea en el fichero de registro"%>
```

Código fuente 85

```
2000-05-22 10:52:48 192.168.4.41 -  
192.168.4.41 80 GET /_vti_inf.html - 200  
Mozilla/2.0+(compatible;+MS+FrontPage+3.0)
```

```
2000-05-22 10:52:54 192.168.4.41 -
192.168.4.41 80 POST /CursoASP30/_vti_bin/shtml.dll - 200
MSFrontPage/3.0
2000-05-22 10:52:54 192.168.4.41 -
192.168.4.41 80 POST /CursoASP30/_vti_bin/_vti_aut/author.dll - 200
MSFrontPage/3.0
2000-05-22 10:52:55 192.168.4.41 -
192.168.4.41 80 POST /CursoASP30/_vti_bin/_vti_aut/author.dll - 200
MSFrontPage/3.0
2000-05-22 10:53:37 192.168.4.41 -
192.168.4.41 80 POST /CursoASP30/_vti_bin/_vti_aut/author.dll - 200
MSFrontPage/3.0
2000-05-22 10:53:48 192.168.4.41 -
192.168.4.41 80 GET /cursoasp30/add.asp
Esto+es+una+línea+en+el+fichero+de+registro 200
Mozilla/4.0+(compatible;+MSIE+5.01;+Windows+NT+5.0)
```

## BynaryWrite

Este método es utilizado para escribir datos binarios en la respuesta al navegador Web del cliente. Este método no se suele utilizar, excepto para crear recursos que no son HTML como puede ser para mostrar una imagen que se encuentra almacenada en un campo binario de una base de datos.

# 6

## **Modelo de objetos de ASP: el objeto Request**

---

### **Definición del objeto Request**

Este objeto integrado es utilizado para obtener información del usuario, por lo tanto realiza las funciones contrarias al objeto Response.

Cuando un navegador cliente se comunica con el servidor Web a través del protocolo HTTP, el navegador manda un gran número de información además de la página que se quiere cargar, esta información puede ser: variables de entorno, información sobre certificados, cookies, formularios, etc. Toda esta información es codificada y enviada a través de cabeceras del protocolo HTTP y en muchos casos no es sencilla de extraer. ASP realiza las funciones de extraer, decodificar y organizar toda esta información a través del objeto Request y sus diferentes colecciones.

Esta información enviada por el navegador, se puede utilizar para: validar un usuario, obtener información sobre el usuario el navegador, almacenar información para su posterior uso, enviar información al servidor a través de formularios, etc.

ASP almacena toda esta información en colecciones del objeto Request. Mediante llamadas del tipo Request.colección se puede obtener de forma sencilla la información que sea necesaria. El objeto Request presenta cinco colecciones que veremos en el siguiente apartado

## Colecciones del objeto Request

Las cinco colecciones que ofrece el el objeto Request del modelo de objetos de ASP contienen toda la información enviada por el navegador cliente al servidor Web. Estas colecciones se pasan a comentar de forma breve:

- ClientCertificate: contiene los valores de los campos de certificación (especificados en el estándar X.509) de la petición emitida por el navegador.
- Cookies: valores de las cookies del cliente.
- Form: valores de elementos de un formulario HTML.
- QueryString: valores de las variables de la cadena de consulta HTTP enviada.
- ServerVariables: esta colección almacena información sobre el navegador, el servidor y el usuario.

Para tener acceso a la información de las colecciones del objeto Request se puede utilizar la siguiente sintaxis general:

```
Request.NombreColeccion(variable)
```

Donde NombreColección es el nombre de la colección que se quiere consultar y variable es el nombre de la variable de la colección a la que se quiere tener acceso y recuperar su valor.

Las variables contenidas en las colecciones del objeto Request son únicamente de lectura.

También se puede acceder a una variable de una colección sin especificar el nombre de la colección (Request(variable)). En este caso se realizará una búsqueda por todas las colecciones del objeto Request hasta que se obtenga la variable deseada. El orden de búsqueda entre las colecciones es el siguiente: QueryString, Form, Cookies, ServerVariables y por último ClientCertificate. Se obtendrá la primera variable que coincida con el parámetro variable. Esto no es muy recomendable ya que es más lento y pueden existir variables con el mismo nombre en distintas colecciones.

Para obtener todos los valores de las variables de una colección se puede utilizar la instrucción For Each...in..., el Código fuente 86, muestra como se obtendría el nombre y el valor de cada variable de la colección ServerVariables.

```
<%For each variable in Request.ServerVariables  
    Response.Write("La variable: "&variable&" tiene el valor: "&  
                  Request.ServerVariables(variable)&"<br>")  
Next%>
```

Código fuente 86

Antes de empezar a explicar cada una de las colecciones del objeto Request en más detalle se va a comentar la función de los formularios HTML y en que consisten. Se va a ofrecer una visión muy básica y general sobre este tema, sólo comentaremos lo que resulte útil para utilizarlo dentro del entorno ASP.

En los siguientes apartados se comentará cada una de las colecciones de Request en profundidad

## Formularios html

Un formulario HTML es uno de los medios más utilizados para obtener información de un usuario (con un navegador) a través de la Web. Un formulario se puede componer de cajas de texto, botones de opción, casillas de verificación, áreas de texto, etc, el formulario se desplegará en el navegador cliente dentro del código HTML, no se debe olvidar que un formulario forma parte de la especificación HTML y viene delimitado por las etiquetas <FORM></FORM>.

Los formularios suelen tener un botón del tipo Submit, es decir, enviar. Si el usuario pulsa sobre este botón, toda la información incluida dentro del formulario asociado se enviará al servidor, es decir, se enviarán los valores de todos los controles que posean un nombre, esto es, la propiedad NAME de cada elemento del formulario debe tener un valor. También se puede realizar el envío de un formulario desde JavaScript de cliente mediante el método submit(), pero para ello es necesario también dar un nombre al formulario con la propiedad NAME de la etiqueta <FORM>.

El cliente puede enviar información al servidor de dos maneras, con el método POST o el método GET. El método utilizado se indica en el atributo METHOD de la etiqueta <FORM>.

Antes de que el navegador envíe la información al servidor, se codifica en un formato llamado codificación URL. En esta codificación los pares nombre/valor (nombre del elemento del formulario y su valor) se unen con símbolos de igual y los diferentes pares se separan con el ampersand. Los espacios son sustituidos por el carácter +, y otros caracteres no alfanuméricos son reemplazados con valores hexadecimales de acuerdo con la especificación RFC 1738.

nombre1=valor1&nombre2=valor2&nombre3=valor3

Cuando se utiliza el método GET se envía la información codificada del usuario añadida a la petición de la página. La página y la información codificada se separa mediante un signo de interrogación (?), así por ejemplo si se quiere acceder a la dirección www.eidos.es/default.htm y se envía también el nombre y la edad proporcionados por el usuario veríamos la siguiente cadena:

<http://www.eidos.es/default.htm?nombre=pepe&edad=23>

El método GET muestra esta cadena en la caja del navegador donde se muestra la dirección de la página que se ha cargado. Esta cadena recibe el nombre de cadena de consulta (QueryString).

El método GET posee una limitación de tamaño, el máximo de caracteres que puede haber en una cadena de consulta es de 1024, es decir, como máximo se pueden enviar 1024K, sin embargo el método POST no posee ninguna limitación de tamaño.

La utilización del método GET muchas veces es útil para a través de un enlace pasar parámetros sin la necesidad de utilizar botones de tipo submit de formularios o incluso ni siquiera utilizar un formulario, por ejemplo, si es necesario enviar a una página ASP llamada procesaDatos.asp algún dato, se podría hacer mediante el siguiente enlace:

<http://www.eidos.es/procesaDatos?nombreDato=valorDato>

El método GET envía la información a través de cabeceras del protocolo HTTP. La información es codificada como se mostró anteriormente y enviada dentro de una cabecera llamada QUERY\_STRING. Por el contrario, si utilizamos el método POST, la información no irá en una cabecera sino que irá en el cuerpo de la petición HTTP.

El método POST sólo se puede utilizar cuando la información es enviada a un fichero ejecutable, como puede ser un CGI (Common Gateway Interface) o página ASP que puedan extraer la

información del cuerpo de la petición HTTP, en caso contrario se producirá un error. El destino al que se envía la información de un formulario se indica en el atributo ACTION de la etiqueta <FORM>.

Si consideramos que tenemos un formulario cuya función es recoger el nombre y edad del usuario y que se crea con el código HTML que muestra el Código fuente 87.

```
<form action="procesaInfo.asp" method="POST">
    Nombre: <input type="Text" name="nombre" value="">
    Edad: <input type="Text" name="edad" value="">
    <input type="Submit" name="boton" value="Enviar">
</form>
```

Código fuente 87

Si se cambia la primera línea por <form action="pagina.html" method="POST"> se producirá un error ya que la información no se envía a un ejecutable. Pero si la cambiamos por <form action="pagina.html" method="GET"> no dará error ya que no se utiliza el método POST.

Una vez enviada la información al servidor debe ser extraída y decodificada a un formato útil para poder ser tratada, pero esto no supone ningún problema ASP realiza esta tarea a través de sus dos colecciones: Form y QueryString. Estas dos colecciones contienen la información de los formularios enviados por el usuario a través de su navegador cliente. La colección Form se utiliza cuando el formulario se envía con el método POST y la colección QueryString cuando se envía con el método GET.

## Colecciones del objeto Request: form

El objeto Request mediante su colección Form permite acceder a los datos enviados a través de un formulario mediante el método POST.

Para obtener los datos del formulario del ejemplo visto en el apartado anterior se debería escribir el Código fuente 88 dentro de la página ASP llamada PROCESAINFO.ASP.

```
Nombre usuario: <%=Request.Form("nombre")%><br>
Edad usuario: <%=Request.Form("edad")%>
```

Código fuente 88

Si se quiere obtener los datos sin decodificar bastaría con escribir el Código fuente 89.

```
Datos: <%=Request.Form%>
```

Código fuente 89

Y en el navegador aparecería la siguiente salida:

```
Datos: nombre=Pepe&edad=30&boton=Enviar
```

Se puede observar que además de los campos de texto del formulario también se envía el botón de Submit.

Pero no es obligatorio utilizar un botón de tipo submit para enviar un formulario, se puede hacer a través de script de cliente con JavaScript, por ejemplo si tenemos el siguiente formulario que se muestra en la Figura 22.



Figura 22. Formulario enviado mediante JavaScript

Queda bastante mejor que con un botón de tipo submit de un formulario. Veamos ahora el Código fuente 90.

```

<html>
<head>
<meta NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</head>
<body>
<form method="POST" action="ProcesaInfo.asp" name="formulario">
<table border="1" align="center" width="20%" bgcolor="#C0C0C0" height="117">
<tr>
  <td align="center" width="100%" height="111">
    <font face="Arial" size="1">Nombre:<br>
    </font>
    <input type="text" maxlength="20" name="nombre" size="15">
    <font face="Arial" size="1"><br>Edad:<br></font>
    <input type="text" name="edad" maxlength="2" size="15"><br>
    <a href="javascript:document.formulario.submit()">
      </a>
    </td>
  </tr>
</table>
</form>
</body>
</html>

```

Código fuente 90

La clave está en la línea que indica un enlace, es decir, la línea con la etiqueta `<a href>`. Como se puede observar en lugar de indicar una URL en el enlace se indica una sentencia de JavaScript, esta sentencia utiliza el modelo de objetos de JavaScript y lanza el método `submit()` sobre el formulario. Esta sentencia se ejecutará al pulsar sobre el enlace, al igual que ocurre cuando pulsamos un enlace convencional y se carga la nueva dirección en el navegador.

De esta forma el botón submit lo podemos personalizar y podemos darle un aspecto más adecuado a nuestra página. La página ASP ProcesaInfo.asp recuperará la información de la misma forma que habíamos visto.

En algunos casos los elementos de un formulario pueden ser un array, basta con, por ejemplo, dar a varias cajas de texto el mismo nombre, de esta forma para acceder a la información se deberá utilizar un índice. Por ejemplo si tenemos el formulario que aparece en el Código fuente 91, para poder recuperar la información enviada se debería tratar como un array, y se realizaría como indica el Código fuente 92.

```
<form action="procesaInfo.asp" method="POST">
    Nombre: <input type="Text" name="datosPersonales" value="">
    Apellidos: <input type="Text" name="datosPersonales" value="">
    Edad: <input type="Text" name="datosPersonales" value="">
    <input type="Submit" name="boton" value="Enviar">
</form>
```

Código fuente 91

```
<%For i=1 To Request.QueryString("datosPersonales").Count
    Response.write(Request.Form("datosPersonales")(i)&"<br>")
Next%>
```

Código fuente 92

Como se puede observar los índices comienzan en 1, y para saber el número de elementos del array se utiliza el método Count que devuelve el número de elementos de un array.

Las páginas ASP se pueden utilizar para recoger y procesar valores de formularios HTML de varias maneras:

- Un archivo htm o html estático puede contener un formulario que envíe sus valores a una página ASP, como ha sido el caso de los ejemplos anteriores.
- Una página ASP puede contener un formulario que envíe información a otra página ASP.
- Una página ASP puede contener un formulario que envíe información así misma, es decir, a la página ASP que contiene el formulario.

La última opción es la más potente y la más recomendable, de esta forma una página ASP se llama así misma. Para poder utilizar esta estrategia lo que se debe tener en cuenta es si la página ASP se carga por primera vez o se carga de nuevo porque ha sido llamada por sí misma. Esto se puede conseguir preguntando si existe alguno de los campos del formulario, si existe es que se ha realizado la llamada desde la misma página y por lo tanto se validarán o mostrarán los datos; y si no existe, la página se está cargando por primera vez y se mostrará el formulario. En el Código fuente 93 se puede observar esta técnica.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
    <TITLE>Misma página ASP</TITLE>
</HEAD>
<BODY>
<%If Request.Form("botonEnviar")<>"" Then%>
    <u>Datos enviados</u><br>
    Nombre: <%=Request.Form("nombre")%><br>
```

```

    Edad: <%=Request.Form("edad")%>
<%Else%>
    <form action="mismaPagina.asp" method="POST">
        Nombre: <input type="Text" name="nombre" value="">
        Edad: <input type="Text" name="edad" value="">
        <input type="Submit" name="botonEnviar" value="Enviar">
    </form>
<%End If%>
</BODY>
</HTML>

```

Código fuente 93

En este caso se pregunta por el valor del botón de envío, ya que siempre que se envíe el formulario se enviará también el botón de tipo submit. Además se supone el código mostrado pertenece a una página ASP llamada mismaPagina.asp tal como se indica en la propiedad ACTION de la etiqueta <FORM>.

Si cargamos por primera vez esta página pasaremos por la rama de la instrucción Else y veríamos la Figura 23 en el navegador.

Nombre:  Edad:  Enviar

Figura 23. La página ASP se carga por primera vez.

Una vez cargada la página, si rellenamos los campos de texto y pulsamos sobre el botón etiquetado como Enviar, pasaríamos por el primer bloque del If, ya que la página ASP se volvería a cargar así misma y la variable botonEnviar de la colección Form ya contendría un valor.

Como resultado se obtendría la Figura 24, suponiendo que en el campo nombre habíamos escrito "Pepe" y en edad "21".

Datos enviados  
Nombre: Pepe  
Edad: 21

Figura 24. La página ASP se carga así misma, al enviar el formulario.

Como se puede observar da la impresión que son dos páginas distintas, pero en realidad es la misma página ASP que genera diferente código HTML según la ejecución del script de servidor.

## Colecciones del objeto Request: Querystring

Para obtener los valores de un formulario que se ha enviado con el método GET o para recuperar los valores de una cadena de consulta de un enlace, utilizaremos esta otra colección del objeto Request: QueryString.

Todo lo visto en la sección anterior para la colección Form es válido para la colección QueryString, la única diferencia es que los datos aparecen visibles en la barra de direcciones del navegador y existe un tamaño máximo de 1024K.

La decisión de utilizar una colección u otra depende del tamaño de los datos a enviar y del número de ellos. Cuando el tamaño es mayor de 1024K sólo tendremos la opción de usar la colección Form.

Yo personalmente sigo la siguiente regla, para los formularios de HTML siempre utilizo el método de envío POST y por lo tanto para recuperar los valores de los campos uso Form, y la colección QueryString la suelo utilizar para recuperar los valores de los datos facilitados desde un enlace.

Se debe tener en cuenta que el usuario puede manipular la cadena de consulta enviada mediante un enlace, por lo tanto habrá algunos datos que no podremos enviar desde una cadena de consulta, pero hay algunos casos en que no representa ningún problema que el usuario modifique la cadena de consulta.

## Colecciones del objeto Request: server variables

Una vez vistas las colecciones Form y QueryString, vamos a pasar a comentar otra colección del objeto Request: la colección ServerVariables. Esta colección proporciona información de los encabezados HTTP enviados junto con una petición del cliente, así como determinadas variables de entorno del servidor Web.

El nombre de esta colección puede despistar un poco, ya que las variables que ofrece no sólo muestran información del servidor sino también del cliente. Algunas de las variables que contiene esta colección son las que se muestran en la Tabla 12.

Variable	Descripción
AUTH_TYPE	Método de autenticación que utiliza el servidor para validar los usuarios cuando intentan el acceso a una secuencia de comandos protegida.
CONTENT_LENGTH	Longitud del contenido.
CONTENT_TYPE	Tipo de datos del contenido.
GATEWAY_INTERFACE	Especificación CGI que utiliza el servidor.
HTTP_<NombreCabecera>	Valor almacenado en esta cabecera. Son valores de las distintas cabeceras del protocolo HTTP.
LOGON_USER	Cuenta de Windows NT con la que inició la sesión el usuario.
PATH_INFO	Información sobre la ruta de acceso.
PATH_TRANSLATED	Ruta física que se corresponde con la virtual que aparecía en PATH_INFO.
QUERY_STRING	Información de consulta almacenada en la URL a partir del signo de interrogación (?).
REMOTE_ADDR	Dirección IP de la máquina cliente que realiza la petición.
REMOTE_HOST	Nombre de la máquina que realiza la petición.

REQUEST_METHOD	Método empleado para realizar la petición.
SCRIPT_MAP	Porción de base de la URL.
SCRIPT_NAME	Ruta virtual del script que se está ejecutando.
SERVER_NAME	Nombre del servidor, DNS o dirección IP.
SERVER_PORT	Número de puerto al que se envió la petición.
SERVER_PORT_SECURE	Si se está atendiendo la petición en el puerto seguro, el valor será 1, en caso contrario 0.
SERVER_PROTOCOL	Nombre y versión del protocolo que realiza la petición.
SERVER_SOFTWARE	Nombre y versión del software del servidor que responde a la petición.
URL	Porción de base de la URL.

Tabla 12

Si escribimos el Código fuente 94, al ejecutarse mostrará todas las variables con sus valores de la colección ServerVariables.

```
<%For each variable in Request.ServerVariables
    Response.Write("La variable: "&variable&" tiene el valor: "&
                    Request.ServerVariables(variable) &"<br>")
Next%>
```

Código fuente 94

El resultado que podría dar este código se muestra a continuación, se debe tener en cuenta que es para un caso concreto, el valor de las variables dependerá de dónde ejecutemos la página ASP.

```
La variable: ALL_HTTP tiene el valor: HTTP_ACCEPT:image/gif,
image/x-bitmap, image/jpeg, image/pjpeg, application/vnd.ms-
powerpoint, application/msword, /* HTTP_ACCEPT_LANGUAGE:es
HTTP_CONNECTION:Keep-Alive HTTP_HOST:aesteban
HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
HTTP_COOKIE:datosPersonales=nombre=Angel&edad=25&apellidos=Esteban+N
%FA%F1ez HTTP_ACCEPT_ENCODING:gzip, deflate
La variable: ALL_RAW tiene el valor: Accept: image/gif, image/x-
bitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint,
application/msword, /* Accept-Language: es Connection: Keep-Alive
Host: aesteban User-Agent: Mozilla/4.0 (compatible; MSIE 5.01;
Windows NT 5.0) Cookie:
datosPersonales=nombre=Angel&edad=25&apellidos=Esteban+N%FA%F1ez
Accept-Encoding: gzip, deflate
La variable: APPL_MD_PATH tiene el valor:
/LM/W3SVC/1/Root/CursoASP30
La variable: APPL_PHYSICAL_PATH tiene el valor:
```

```
c:\inetpub\wwwroot\CursoASP30\  
La variable: AUTH_PASSWORD tiene el valor:  
La variable: AUTH_TYPE tiene el valor:  
La variable: AUTH_USER tiene el valor:  
La variable: CERT_COOKIE tiene el valor:  
La variable: CERT_FLAGS tiene el valor:  
La variable: CERT_ISSUER tiene el valor:  
La variable: CERT_KEYSIZE tiene el valor:  
La variable: CERT_SECRETKEYSIZE tiene el valor:  
La variable: CERT_SERIALNUMBER tiene el valor:  
La variable: CERT_SERVER_ISSUER tiene el valor:  
La variable: CERT_SERVER_SUBJECT tiene el valor:  
La variable: CERT SUBJECT tiene el valor:  
La variable: CONTENT_LENGTH tiene el valor: 0  
La variable: CONTENT_TYPE tiene el valor:  
La variable: GATEWAY_INTERFACE tiene el valor: CGI/1.1  
La variable: HTTPS tiene el valor: off  
La variable: HTTPS_KEYSIZE tiene el valor:  
La variable: HTTPS_SECRETKEYSIZE tiene el valor:  
La variable: HTTPS_SERVER_ISSUER tiene el valor:  
La variable: HTTPS_SERVER_SUBJECT tiene el valor:  
La variable: INSTANCE_ID tiene el valor: 1  
La variable: INSTANCE_META_PATH tiene el valor: /LM/W3SVC/1  
La variable: LOCAL_ADDR tiene el valor: 192.168.4.41  
La variable: LOGON_USER tiene el valor:  
La variable: PATH_INFO tiene el valor: /cursoasp30/form.asp  
La variable: PATH_TRANSLATED tiene el valor:  
c:\inetpub\wwwroot\cursoasp30\form.asp  
La variable: QUERY_STRING tiene el valor:  
La variable: REMOTE_ADDR tiene el valor: 192.168.4.41  
La variable: REMOTE_HOST tiene el valor: 192.168.4.41  
La variable: REMOTE_USER tiene el valor:  
La variable: REQUEST_METHOD tiene el valor: GET  
La variable: SCRIPT_NAME tiene el valor: /cursoasp30/form.asp  
La variable: SERVER_NAME tiene el valor: aesteban  
La variable: SERVER_PORT tiene el valor: 80  
La variable: SERVER_PORT_SECURE tiene el valor: 0  
La variable: SERVER_PROTOCOL tiene el valor: HTTP/1.1  
La variable: SERVER_SOFTWARE tiene el valor: Microsoft-IIS/5.0  
La variable: URL tiene el valor: /cursoasp30/form.asp  
La variable: HTTP_ACCEPT tiene el valor: image/gif, image/x-xbitmap,  
image/jpeg, image/pjpeg, application/vnd.ms-powerpoint,  
application/msword, */*  
La variable: HTTP_ACCEPT_LANGUAGE tiene el valor: es  
La variable: HTTP_CONNECTION tiene el valor: Keep-Alive  
La variable: HTTP_HOST tiene el valor: aesteban  
La variable: HTTP_USER_AGENT tiene el valor: Mozilla/4.0  
(compatible; MSIE 5.01; Windows NT 5.0)  
La variable: HTTP_COOKIE tiene el valor:  
datosPersonales=nombre=Angel&edad=25&apellidos=Esteban+N%FA%Flez  
La variable: HTTP_ACCEPT_ENCODING tiene el valor: gzip, deflate
```

Algunas de estas variables pueden ser de utilidad dentro de nuestra página ASP, en las siguientes líneas del presente apartado se va a comentar con pequeños ejemplos la utilidad de algunas de las variables de la colección ServerVariables.

Por ejemplo para obtener la ruta completa de la página actual y no tener que modificar nuestro código cuando cambie el nombre de la página. Esto puede ser útil a la hora de hacer referencia a la propia página en un formulario que llame a la misma página, así por ejemplo podemos escribir lo que indica el Código fuente 95.

```
<form action="<% =Request.ServerVariables("PATH_INFO") %>" method="POST">
```

Código fuente 95

O también la línea que muestra el Código fuente 96.

```
<form action="<% =Request.ServerVariables("SCRIPT_NAME") %>" method="POST">
```

Código fuente 96

Otra posibilidad es construir una URL completa de la página actual a partir de una serie de variables de la colección ServerVariables, como se puede observar en el Código fuente 97. Este código generaría el Código fuente 98.

```
<%strURL="http://" & Request.ServerVariables("SERVER_NAME")_
&" : "&Request.ServerVariables("SERVER_PORT")_
&Request.ServerVariables("PATH_INFO")%>
URL:<a href="<% =strURL%>">Enlace</a>
```

Código fuente 97

```
URL:<a href="http://aesteban:80/cursoasp30/pruebas.asp">Enlace</a>
```

Código fuente 98

Otras funciones que podemos realizar con la colección ServerVariables es obtener el navegador Web utilizado por el usuario que realizó la petición y su versión correspondiente. Hemos extraído el número de la versión de la cadena que representa al nombre del navegador porque en algunos casos nos puede interesar realizar comparaciones con el número de versión.

```
<%strNavegador=Request.ServerVariables("HTTP_USER_AGENT")
'comprobamos si es trata del navegador Internet Explorer
If InStr(strNavegador,"MSIE") Then
    intVersion=Cint(Mid(strNavegador,InStr(strNavegador,"MSIE")+5,1))
Else
    intVersion=Cint(Mid(strNavegador,InStr(strNavegador,"/")+1,1))
End if%>
Navegador:<b><% =strNavegador%></b><br>
Versión:<b><% =intVersion%></b><br>
```

Código fuente 99

En el caso de ejecutar esta página con Internet Explorer 5 obtendríamos el siguiente resultado:

Navegador:Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)  
 Versión: 5

Y si la ejecutamos con Netscape 4.5, obtenemos este otro:

Navegador:Mozilla/4.5 [en] (WinNT; I)  
 Versión: 4

Como se puede observar el navegador Netscape no nos ofrece la información relativa al nombre del navegador, si lo hace Internet Explorer mediante la cadena MSIE.

Siguiendo con información referente al navegador, también en algún caso nos puede interesar obtener el código del lenguaje utilizado en el navegador del cliente. De esta forma podemos detectar el idioma utilizado y redirigir el navegador del cliente a la versión del sitio Web correspondiente a su idioma. Un ejemplo podría ser el Código fuente 100.

```
<%strIdioma=LCase(Left(Request.ServerVariables("HTTP_ACCEPT_LANGUAGE"),2))
Select Case strIdioma
    Case "en": Response.Redirect "http://sitioWeb.uk"
    Case "de": Response.Redirect "http://sitioWeb.de"
    Case "fr": Response.Redirect "http://sitioWeb.fr"
    Case "es": Response.Redirect "http://sitioWeb.es"
    '...etc
    Case Else: Response.Redirect "http://sitioWeb.us"
End Select%>
```

Código fuente 100

Una utilidad más de las variables de la colección ServerVariables es que nos permite averiguar el número de puerto que ha utilizado un usuario para conectarse a la página actual. Esto puede ser útil si queremos distinguir si la conexión se ha realizado a través de un puerto seguro SSL (Secure Socket Layer). En el Código fuente 101 se verifica si el usuario se ha conectado utilizando el puerto seguro 443, por defecto sino indica nada se el usuario se conecta a través del puerto 80. Si el usuario se conecta por el puerto seguro se le envía a la sección segura del sitio Web.

```
<%If Request.ServerVariables("SERVER_PORT")="443" Then
    Response.Redirect "sitioseguro/default.asp"
Else
    Response.Redirect "/sitionormal/default.asp"
End if%>
```

Código fuente 101

Por último otro uso de estas variables lo tenemos cuando queremos verificar el usuario que se ha autenticado con el servidor Web. Para conseguir esta información debemos forzar que el usuario se autentifique con el servidor Web, para ello podemos deshabilitar el acceso anónimo o bien restringir permisos a nivel del sistema de archivos NTFS.

En el Código fuente 102 se comprueba si el usuario es administrador del dominio del servidor Web, en caso afirmativo se muestran una serie de enlaces que serían enlaces sólo de administrador. Esto en

realidad no es demasiado seguro, pero es un ejemplo que muestra la utilidad de la colección ServerVariables.

```
<%If Request.ServerVariables("AUTH_USER")=_  
    UCase(Request.ServerVariables("SERVER_NAME")) & "/Administrador" Then%>  
    <a href="informacion.asp">Información sólo para administradores</a>  
<%End If%>
```

Código fuente 102

## Colecciones del objeto Request: Cookies

La siguiente colección que se va a comentar también aparecía en el objeto Response, es la colección Cookies.

En el objeto Request esta colección contiene los valores de las cookies del cliente, pero sólo son de lectura, no se pueden modificar ni crear, para ello se debe utilizar el objeto Response, como habíamos visto anteriormente.

En este objeto, la colección Cookies únicamente posee la propiedad HasKeys para indicar si la cookie contiene claves. Si queremos recuperar todas las claves de una cookie deberemos utilizar el Código fuente 103.

```
<%If Request.Cookies("datosPersonales").HasKeys Then  
    For Each clave In Request.Cookies("datosPersonales")  
        Response.Write clave& "="  
        "&Request.Cookies("datosPersonales") (clave) &"<br>"  
    Next  
End If%>
```

Código fuente 103

Si la cookie la hemos construido con la colección Response.Cookies, Código fuente 104,

```
<%  
    Response.Cookies("datosPersonales").Expires= #December 31, 2000#  
    Response.Cookies("datosPersonales")("nombre")="Angel"  
    Response.Cookies("datosPersonales")("apellidos")="Esteban Núñez"  
    Response.Cookies("datosPersonales")("edad")=25  
%>
```

Código fuente 104

veremos el siguiente resultado:

```
nombre = Angel  
edad = 25  
apellidos = Esteban Núñez
```

Pero si lo que queremos hacer es recorrer por completo la colección Cookies del objeto Request, utilizaremos el Código fuente 105.

```
<%For Each cookie In Request.Cookies
    Response.Write cookie & " = " & Request.Cookies(cookie) & "<BR>"
    If Request.Cookies(cookie).HasKeys Then
        For Each clave In Request.Cookies(cookie)
            Response.Write ">" & cookie & "(" & clave & ") = " & _
                Request.Cookies(cookie)(clave) & "<BR>"
        Next
    End If
Next%>
```

Código fuente 105

Como resultado de la ejecución podríamos obtener la siguiente salida:

```
datosPersonales = apellidos=Esteban+N%FA%Flez&edad=25&nombre=Angel
->datosPersonales(apellidos) = Esteban Núñez
->datosPersonales(edad) = 25
->datosPersonales(nombre) = Angel
nombreUsuario = Angel Esteban
```

Vamos a realizar un ejemplo que consiste en crear una cookie a partir del valor indicado en un formulario. En este ejemplo se puede ver la utilización de las colecciones Cookies, Form y ServerVariables del objeto Request, y la utilización de la colección Cookies del objeto Response.

Veamos el Código fuente 106 que resulta bastante claro ya que se encuentra comentado.

```
<%@ LANGUAGE="VBSCRIPT" %>
<%'Nos aseguramos de tener la última versión de la página ASP
Response.Expires=-1000
'Se procede al borrado de la cookie si se ha pulsado el botón de borrado
'del segundo formulario
If Request.Form("borrar")<>"" Then
    Response.Cookies("nombreUsuario").Expires=#December 31, 1998#
    'se redirecciona al usuario a la misma página para que se vuelva a cargar
    Response.Redirect Request.ServerVariables("SCRIPT_NAME")%
<%End if
'Preguntamos si ya existe la cookie, si no existe se pide su valor
If Request.Cookies("nombreUsuario")="" AND _
    Request.Form("boton")="" Then%
    <form method="post" action="cookies.asp">
        Nombre:<input type="text" name="nombre" value="">
        <input type="submit" name="boton" value="Enviar">
    </form>
<%ElseIf Request.Form("boton")<>"" Then
    'El botón se ha pulsado y se ha enviado la información
    'Se indica la caducidad de la cookie
    Response.Cookies("nombreUsuario").Expires=#December 31, 2000#
    Response.Cookies("nombreUsuario")=Request.Form("nombre")
End If%>
<h1>Valor Cookie:<%=Request.Cookies("nombreUsuario")%>
<%If Request.Cookies("nombreUsuario")<>"" Then%
    <form method="post" action="cookies.asp" id=form1 name=form1>
        <input type="submit" name="borrar" value="Borrar Cookie">
    </form>
<%End If%>
```

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

Código fuente 106

Así al cargar la página por primera vez veríamos una página como la de la Figura 25, es decir, la cookie no existe todavía.



Figura 25. Cookie todavía no creada

Al pulsar el botón de enviar del formulario, se crea la cookie con el valor especificado, y veríamos algo parecido a la Figura 26.



Figura 26. La cookie se ha creado

Si pulsamos el botón de borrado de la cookie, lo que haremos será asignar a la propiedad Expires de la cookie un valor de fecha antiguo para que de esta forma se borre la cookie. Acto seguido se redirecciona el navegador del usuario a la misma página y tendríamos la situación de la Figura 27, que como se puede ver es equivalente a la situación inicial.



Figura 27. La cookie se ha borrado

Si no borramos la cookie y volvemos a conectarnos a la página ASP correspondiente vemos que el valor de la cookie se ha conservado. El código de esta página se puede obtener [aquí](#).

## Colecciones del objeto Request: Clientcertificate

La última colección del objeto Request es la colección ClientCertificate. Esta colección contiene la información del certificado del cliente (definida por el estándar X.509) para peticiones SSL3/PCT1. SSL3 y PCT1 son protocolos que codifican la información y la mantienen protegida a través de la Web. Los certificados son códigos encriptados que identifican únicamente a un cliente o a un servidor. Los certificados se obtienen de una autoridad (empresa) que garantiza que son únicos y auténticos, una de estas empresas es VeriSign.

IIS debe ser configurado para obtener información de certificados. Normalmente, si no se está utilizando información altamente confidencial, los certificados de cliente no se suelen utilizar.

La colección ClientCertificate contiene en sus variables toda la información que define el certificado del cliente. Estas variables son únicamente de lectura.

Si un explorador Web utiliza el protocolo SSL3.0/PCT1 (es decir, utiliza una dirección URL que comienza con https:// en lugar de http://) para conectarse a un servidor y éste pide la certificación, el explorador envía los campos de certificación. Si no se envía ningún certificado, la colección ClientCertificate devuelve Empty. Para poder utilizar la colección ClientCertificate debe configurar el servidor Web para que pida certificados de cliente.

La sintaxis para acceder a cada una de las variables de la colección ClientCertificate es la siguiente.

```
Request.ClientCertificate( Clave [SubCampo] )
```

Las variables que contiene la colección ClientCertificate se comentan en la Tabla 13.

Clave	Descripción
Certificate	Una cadena que contiene la secuencia binaria con todo el contenido del certificado.

Flags	<p>Un conjunto de indicadores que proporcionan información adicional del certificado de cliente. Podemos ver los siguientes indicadores:</p> <p>ceCertPresent: existe un certificado de cliente.</p> <p>ceUnrecognizedIssuer: el último certificado de la cadena es de un emisor desconocido.</p> <p>Para utilizar los indicadores anteriores debe incluir en la página ASP el archivo de inclusión de certificado de cliente. Si utiliza VBScript, debe incluir cervbs.inc. Este archivo se instala en el directorio \Inetpub\ASPSamp\Samples.</p>
Issuer	Una cadena que contiene una lista de valores de subcampos que contienen información acerca del emisor del certificado. Si se especifica este valor sin un subcampo, la colección ClientCertificate devuelve una lista de subcampos separados por comas. Por ejemplo, C=US, O=Verisign, etc. Más adelante veremos el significado de los subcampos.
SerialNumber	Una cadena que contiene el número de serie de certificación.
Subject	Una cadena que contiene una lista de valores de subcampos. Los valores de subcampo pueden contener información acerca del asunto del certificado. Si se especifica este valor sin un subcampo, la colección ClientCertificate devuelve una lista de subcampos separados por comas. Por ejemplo, C=US, O=Msft, etc.
ValidFrom	Una fecha que especifica desde cuándo es válido el certificado. Utiliza el formato de VBScript y varía según la configuración internacional.
ValidUntil	Especifica la fecha de caducidad del certificado. El valor del año se muestra como un número con cuatro dígitos.

Tabla 13

Llegó el momento de definir un subcampo. Un subcampo es un parámetro opcional que puede utilizar para recuperar un campo individual en las claves Subject o Issuer. Este parámetro se agrega como un sufijo del parámetro de la clave. Por ejemplo, IssuerO o SubjectCN. En la Tabla 14 se enumeran algunos de los valores habituales que pueden presentar los subcampos.

Subcampo	Descripción
C	Especifica el nombre del país/ región de origen.
CN	Especifica el nombre común del usuario. (Este subcampo sólo se utiliza con la clave Subject.)
GN	Especifica un nombre dado.
I	Especifica un conjunto de iniciales.
L	Especifica una localidad.

O	Especifica el nombre de la organización o la empresa.
OU	Especifica el nombre del departamento.
S	Especifica una provincia o un estado.
T	Especifica el puesto de la persona o la organización.

Tabla 14

Veamos ahora unos sencillos ejemplos de la utilización de la colección ClientCertificate.

Podemos utilizar la variable o clave Subject para averiguar si se ha enviado un certificado de cliente.

```
<%If Len(Request.ClientCertificate("Subject")) = 0 Then
    Response.Write("No hay certificado de cliente")
End if%>
```

Código fuente 107

El Código fuente 108 obtiene el nombre de la entidad certificadora que emitió el certificado del cliente.

```
<%=Request.ClientCertificate("IssuerCN")%>
```

Código fuente 108

En el Código fuente 109 se muestra la fecha de caducidad del certificado.

```
<%=Request.ClientCertificate("ValidUntil")%>
```

Código fuente 109

## Métodos y propiedades del objeto Request

Este objeto carece de métodos y de propiedades.

# 7

## **Modelo de objetos de ASP: el objeto Application**

---

### **Aplicaciones ASP**

Una aplicación ASP se puede definir de manera sencilla como un conjunto de páginas activas de servidor que tiene una función común y que comparten un lugar de almacenamiento de variables común.

Una aplicación ASP se define a partir de un directorio o directorio virtual en un sitio Web. Se debe indicar el punto de inicio de la aplicación ASP, a partir de ese punto de inicio todas las páginas ASP formarán parte de la aplicación ASP y compartirán un mismo contexto. Un ejemplo de una completa aplicación ASP es el Campus Virtual Almagesto.

En el primer tema del presente curso abordábamos el concepto de aplicación ASP, aquí lo volvemos a tratar para ampliarlo y ver como crear realmente una aplicación ASP.

En este capítulo además vamos a tratar el objeto integrado en el modelo de objetos de ASP que representa a la aplicación ASP, es decir, el objeto Application.

Cuando creamos en su momento nuestro proyecto con Visual InterDev para crear una aplicación Web, ya estábamos creando sin saberlo una aplicación ASP. Se puede decir que una aplicación ASP es un tipo de aplicación Web.

Una aplicación Web es un conjunto de páginas Web y otros recursos que se encuentran diseñados para realizar un tipo de tarea determinado para obtener unos resultados. Las aplicaciones Web no son

aplicaciones al uso, ya que no disponen de un proceso de instalación real, sino que se encuentran contenidas en un servidor Web. En nuestro caso, para las aplicaciones ASP el servidor Web que vamos a utilizar es Internet Information Server 5 (IIS 5).

Al crear la aplicación Web con Visual InterDev vimos también que se creaba un fichero llamado global.asa. Este es un fichero especial en el que se pueden definir una serie de eventos relativos a la aplicación ASP y las sesiones que los usuarios tienen con la aplicación. Cada aplicación ASP suele tener su fichero global.asa, más adelante volveremos a dar más información sobre este fichero.

Veamos el proceso de creación y configuración de nuestra propia aplicación ASP.

Si ya hemos creado una aplicación Web con Visual InterDev, la aplicación Web ya estará creada y su punto de inicio será el directorio que contiene nuestras páginas ASP en el sitio Web. Por ejemplo si hemos creado la aplicación Web llamada CursoASP, el punto de inicio de la aplicación ASP será el directorio Curso ASP. En el punto de inicio de la aplicación ASP es dónde se tiene en cuenta el fichero global.asa.

Veamos el aspecto que tiene una aplicación ASP desde el Administrador de servicios de Internet.

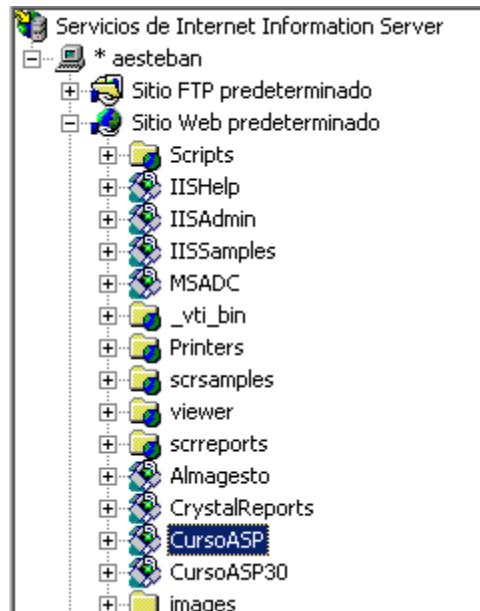


Figura 28. Una aplicación ASP en IIS 5.

Como se puede ver es algo similar a una bola metida en una caja, esto es lo que identifica gráficamente a una aplicación ASP en el Administrador de servicios de Internet.

Hemos nombrado una herramienta administrativa de Windows 2000 que quizás a algunos de nuestros lectores les suene y a otros no, definiremos el Administrador de servicios de Internet como una herramienta administrativa que nos permite configurar todos los servicios incluidos en el servidor Web Internet Information Server 5.

En este apartado sólo vamos a adelantar como crear una aplicación ASP y como configurarla de forma básica. En el capítulo dedicado a IIS 5 veremos la creación y configuración de aplicaciones ASP de forma más completa.

Si la aplicación ASP no queremos crearla desde Visual InterDev pedimos hacerlo desde el Administrador de servicios de Internet siguiendo pasos que vamos a comentar a continuación.

Primero debemos crear físicamente el directorio que va a contener la aplicación ASP y que va a ser su punto de inicio. Puede ser un directorio contenido dentro del directorio de publicación en Internet (c:\Inetpub\wwwroot) o bien un directorio virtual fuera de esta estructura. Yo he decidido crear el siguiente directorio c:\Inetpub\wwwroot\AplicacionASP.

A continuación ejecutamos el Administrador de servicios de Internet que lo encontramos en el grupo de Herramientas administrativas del menú de inicio. Vemos que nuestra carpeta se encuentra dentro del sitio Web predeterminado del servidor Web, el sitio Web predeterminado es también una aplicación ASP, es la aplicación ASP que se crea por defecto al instalar IIS 5. La seleccionamos y pulsamos sobre ella con el botón derecho del ratón para seleccionar la opción de menú Propiedades, y aparecerán las hojas de propiedades de este directorio.

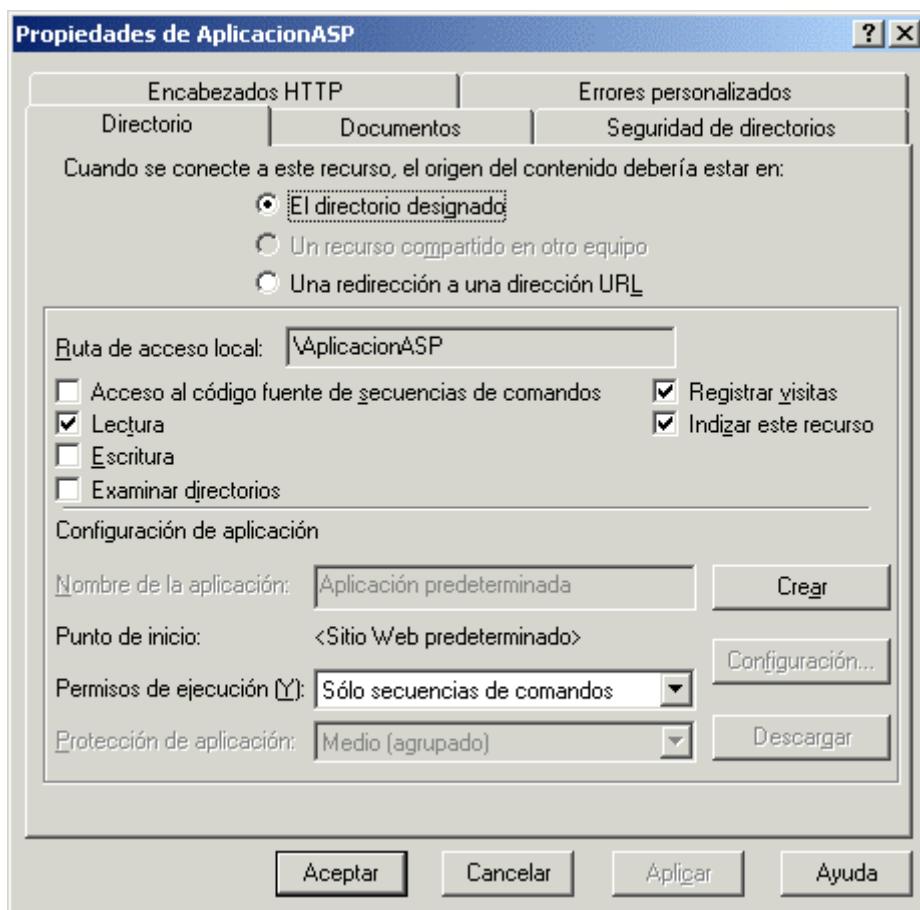


Figura 29. Propiedades de un directorio.

Vemos que existe un epígrafe denominado configuración de la aplicación y también que se encuentra habilitado un botón etiquetado como Crear. Crear la aplicación ASP es tan sencillo como pulsar este botón. Si pulsamos este botón la hoja de propiedades toma un nuevo aspecto, activándose el botón Configurar y cambiando el botón Crear por el de Quitar.

En el capítulo dedicado a IIS 5 volveremos a tratar el tema de las aplicaciones ASP explicando el significado de los permisos de ejecución y la protección de la aplicación, además de otros conceptos.

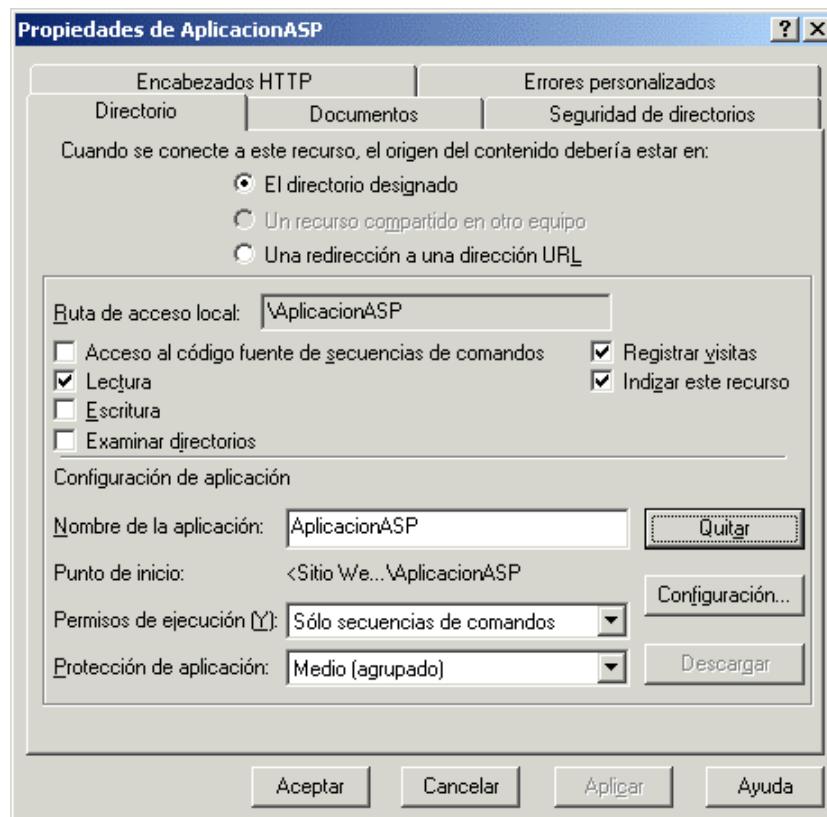


Figura 30. Propiedades de un directorio que es punto de inicio de una aplicación ASP

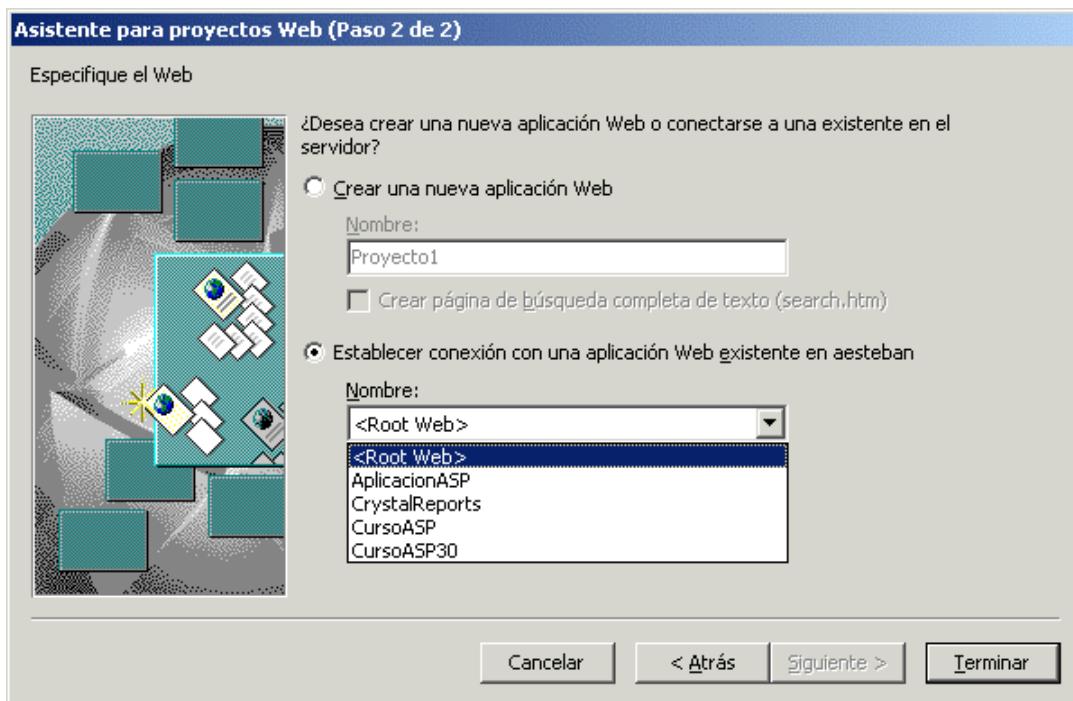


Figura 31. La aplicación ASP en la lista de aplicaciones existentes

Ya hemos creado nuestra aplicación ASP, pero si queremos que sea visible desde Visual InterDev, debemos indicar que el directorio es un subWeb de FrontPage. Para ello pulsamos con el botón derecho del ratón sobre nuestro directorio y seleccionamos la opción Todas las tareas|Configurar las

extensiones de Servidor, y se lanzará un asistente en el que aceptaremos todas las opciones por defecto.

Este paso no es necesario si no necesitamos acceder a la aplicación ASP desde Visual InterDev.

Ahora si creamos un proyecto Web con Visual InterDev deberíamos ver en la lista de aplicaciones Web existentes en el servidor Web nuestra nueva aplicación ASP.

Cada aplicación ASP tendrá en conjunto de variables globales que podrán ser utilizadas por todos los usuarios de esa aplicación. No es recomendable anidar distintas aplicaciones ASP, cada aplicación ASP debe tener su directorio bien diferenciado.

A lo largo de este apartado inicial hemos comentado el significado de una aplicación ASP y como crearla desde el Administrador de servicios de Internet.

## Definición del objeto Application

Un objeto Application representa una aplicación ASP. Una aplicación basada en ASP, como ya hemos visto en el apartado anterior, consta de un directorio en un servidor Web y de todos los subdirectorios y archivos contenidos en él. Una aplicación puede ser una página principal sencilla, o bien puede estar formada por un conjunto completo de páginas interrelacionadas entre sí.

Se debe recordar que el protocolo HTTP es un protocolo sin estado, es decir, no se puede almacenar información entre diferentes conexiones HTTP. No se puede mantener el estado entre diferentes páginas Web a través del protocolo HTTP, sino que se deben utilizar otros mecanismos como las cookies. Pero el objeto Application junto con el objeto Session (que se comentará en su capítulo correspondiente) nos permite de forma sencilla y directa almacenar información abstracta y desligada del uso de cookies y de encabezados HTTP.

La información almacenada en los objetos Session y Application difieren en el ámbito de la misma, una información tendrá el ámbito de la sesión de un usuario concreto y la otra el ámbito de la aplicación general, respectivamente.

Las variables almacenadas dentro del objeto Application son visibles para todos los usuarios que están utilizando la misma aplicación ASP, es decir son compartidas por varios usuarios. En contraposición al objeto Session, cuyas variables son para cada uno de los usuarios conectados, es decir, no se comparten y son propias de cada sesión. Podremos acceder a una variable a nivel de aplicación en cualquiera de las páginas ASP contenidas en la aplicación ASP actual.

En este punto se debe señalar que una variable en una aplicación ASP puede tener cuatro ámbitos diferentes, se va a ir del más global al más particular. La creación de una variable con un ámbito u otro dependerá del uso que queramos hacer de ella.

- Ámbito de aplicación: esta variable la podrán manipular todos los usuarios de la aplicación ASP, es común a todos ellos. Se almacena dentro del objeto Application.
- Ámbito de sesión: las variables con este ámbito son propias de cada uno de los usuarios de la aplicación, es decir, cada usuario tendrá acceso a las variables de su sesión. Estas variables se almacenan dentro del objeto integrado Session.
- Ámbito de página: estas variables son las que se crean dentro del script de servidor de una página ASP, sólo tienen vigencia dentro de la página en la que se declararon, al abandonar la página se destruirán.

- Ámbito de procedimiento: a este ámbito pertenecen las variables declaradas dentro de un procedimiento (Sub o Function). Estas variables sólo existirán dentro del procedimiento en el que son declaradas, se dice que son variables locales al procedimiento.

Para almacenar una variable dentro de un objeto Application, es decir, crear una variable cuyo ámbito sea la aplicación se debe utilizar la siguiente sintaxis:

```
Application("NombreVariable")=valorVariable
```

También se pueden almacenar objetos dentro del objeto Application, para ello se debe utilizar esta otra sintaxis:

```
Set Application("NombreObjeto")=Server.CreateObject("Componente")
```

Aquí vemos una sentencia Server.CreateObject, Server es un objeto integrado en ASP y lo veremos en su capítulo correspondiente, y CreateObject es un método de este objeto que nos permite crear un objeto que va a ser la instancia de un componente.

Hay una serie de consideraciones acerca de las variables que se pueden almacenar en el objeto Application y que pasamos a comentar a continuación.

En cuanto a tipos de variable se refiere, una variable de aplicación puede contener cualquier tipo de valor Variant, y en cuanto a objetos, puede contener cualquier tipo de objeto o componente que se encuentre preparado para tener ámbito de aplicación, a excepción de los objetos del modelo de objetos de ASP (ASPError, Request, Server, Response, Application, Session, ObjectConext) de esta forma la instrucción que muestra el Código fuente 110, generaría un error.

```
<%Set Application("objeto")=Request%>
```

Código fuente 110

Y el error sería:

```
Objeto Application error 'ASP 0180 : 80004005'
Uso no autorizado de objeto
/cursoasp30/aplicacion.asp, line 29
No se puede almacenar un objeto intrínseco en el objeto Application.
```

Si almacenamos una variable array a nivel de aplicación, para modificar sus valores no podremos hacerlo directamente, sino que se debe hacer a través de una variable auxiliar, que será la que se modificará y luego asignará a la variable de aplicación correspondiente. En el Código fuente 111 se muestra la utilización de una variable de aplicación que es un array de cadenas.

```
<%Dim vector(3)
vector(0)="Primer valor"
vector(1)="Segundo"
vector(2)="Tercero"
vector(3)="Ultimo Valor"
Application("valores")=vector
'-----
'-----
'Para modificar los valores recuperamos la variable
vectorAux=Application("valores")
```

```

vectorAux(2)="Elemento modificado"
Application("valores")=vectorAux
For i=0 to UBound(Application("valores"))
    Response.Write
    "Application('valores') ("&i&")=&Application("valores") (i)&"<br>"
Next%>

```

Código fuente 111

Y el resultado de este código es:

```

Application('valores')(0)=Primer valor
Application('valores')(1)=Segundo
Application('valores')(2)=Elemento modificado
Application('valores')(3)=Ultimo Valor

```

Sin embargo los valores de cada elemento de array si que se pueden recuperar directamente de la variable de aplicación.

No es recomendable almacenar objetos a nivel de aplicación, ya que esto supone que el objeto va a existir durante toda la vida de la aplicación ASP y puede ser accedido por distintos usuarios de manera concurrente. Lo ideal para almacenar en variables de aplicación son variables de tipo entero, booleano o cadenas de caracteres.

## Colecciones del objeto Application

En la versión 1.0 de ASP el objeto Application no poseía ninguna colección, sin embargo en ASP 2.0 se añadieron dos colecciones, y son:

- **Contents:** contiene las variables a nivel de aplicación a excepción de los objetos creados con la etiqueta <OBJECT>
- **StaticObjects:** contiene únicamente los objetos a nivel de aplicación creados con la etiqueta <OBJECT>.

Ambas colecciones nos van a permitir recuperar o acceder a las variables y objetos creados a nivel de aplicación, aunque ya hemos visto en el apartado anterior que podemos acceder directamente a ellos.

La colección Contents contiene todos los objetos y variables creados a nivel de aplicación, excluyendo aquellos objetos que han sido creados utilizando la etiqueta de HTML <OBJECT>. Esta etiqueta se utiliza para la creación de objetos, de todas formas veremos con detalle el uso de la etiqueta <OBJECT> en este mismo apartado a la hora de comentar la colección StaticObjects.

En esta colección se incluirán, por tanto, los objetos creados a nivel de aplicación con la sentencia Server.CreateObject y las variables creadas con Application("variable")=valor. Así por ejemplo, en el Código fuente 112 vamos a crear una serie de variables a nivel de aplicación y también un objeto, y vamos a todas ellas inspeccionando la colección Contents con la iteración For...Each.

```

<%'creación de las variables
Application("variableUno")="valor"
Application("variableDos")=2
Application("variableTres")=True
Set Application("objConexion")=Server.CreateObject ("ADODB.Connection")
'se abre la conexión

```

```

Application("objConexion").Open "DSN=BD;UID=sa;PWD="
For Each variable in Application.Contents
    Response.Write variable & " = "
    'comprobamos si la variable de aplicación es un objeto
    If IsObject(Application.Contents(variable)) Then
        Response.Write Application.Contents(variable).DefaultDatabase & "<BR>"
    Else
        Response.Write Application.Contents(variable) & "<br>"
    End if
Next
'se cierra la conexión
Application("objConexion").Close%>

```

Código fuente 112

Como se puede observar en el código anterior, se crea un objeto Connection de ADO (ActiveX Data Objects), este objeto va a representar una conexión con una base de datos. Una vez asignado este objeto a una variable de aplicación vemos que podemos utilizar sus métodos y propiedades (Open, Close, DefaultDatabase, etc.) directamente sobre la variable de aplicación, ya que ésta contiene una referencia al objeto creado.

Y el resultado de la ejecución del código del ejemplo sería el siguiente:

```

variableUno = valor
variableDos = 2
variableTres = Verdadero
objConexion = Cursos

```

La colección Contents del objeto Application permite utilizar un par de métodos que son comunes a todas las colecciones de escritura y que son los siguientes:

- RemoveAll: elimina todas las variables de aplicación, a excepción de las creadas con la etiqueta <OBJECT>. Este método carece de parámetros.
- Remove: elimina una variable de aplicación determinada que esté contenida en la colección Contents. Este método recibe por parámetro el índice o nombre de la variable correspondiente.

Vemos un ejemplo de utilización de estos dos métodos en el Código fuente 113.

```

<%Application("variableUno")="valor"
Application("variableDos")=2
Application("variableTres")=True
For Each variable in Application.Contents
    Response.Write variable & " = "&Application.Contents(variable)&"<br>"
Next
Application.Contents.Remove("variableDos")%>
<b>Se borra una variable</b><br>
<%For Each variable in Application.Contents
    Response.Write variable & " = "&Application.Contents(variable)&"<br>"
Next
Application.Contents.RemoveAll%>
<b>Se borran todas las variables</b><br>
<%If Application.Contents.Count=0 Then Response.Write "Colección Contents vacía"%>

```

Código fuente 113

El resultado es el siguiente:

```
variableUno = valor
variableDos = 2
variableTres = Verdadero
Se borra una variable
variableUno = valor
variableTres = Verdadero
Se borran todas las variables
Colección Contents vacía
```

También podemos crear una variable de aplicación creando directamente un nuevo elemento de la colección Contents, es decir, se trata de una colección de lectura/escritura, ha diferencia de las colecciones vistas en el objeto integrado Request. En el Código fuente 114 se crea una nueva variable de aplicación creando un nuevo elemento de la colección Contents.

```
<%Application("variableUno")="valor"
Application("variableDos")=2
Application("variableTres")=True
Application.Contents("NuevaVariable")="su valor"
For Each variable in Application.Contents
    Response.Write variable & " = "&Application.Contents(variable)&"<br>"
Next%>
```

Código fuente 114

El resultado del código es:

```
variableUno = valor
variableDos = 2
variableTres = Verdadero
NuevaVariable = su valor
```

Pasemos ahora a tratar la segunda colección del objeto Application.

La colección StaticObjects contiene todos los objetos a nivel de aplicación que se han creado mediante la etiqueta <OBJECT>, por lo tanto la colección StaticObjects únicamente contendrá variables que son referencias a objetos.

De la misma forma que veíamos en el ejemplo anterior, podremos recorrer la colección StaticObjects para acceder a los objetos estáticos creados a nivel de aplicación.

La etiqueta <OBJECT> es una etiqueta de HTML que se suele utilizar para instanciar y crear objetos ActiveX en el navegador del cliente, sin embargo aquí su uso es muy distinto, ya que se utiliza para instanciar componentes del servidor, podemos decir que es equivalente a la sentencia Server.CreateObject. Pero esta etiqueta únicamente se puede utilizar en el fichero especial GLOBAL.ASA, en el apartado dedicado ha este fichero en este mismo capítulo comentaremos la sintaxis de la etiqueta <OBJECT> y sus particularidades a la hora de crear objetos.

En el Código fuente 115, se muestra como recorrer la colección StaticObjects.

```
<%For Each objeto in Application.StaticObjects
    Response.Write objeto&"<br>"
```

Next%>

Código fuente 115

Como se puede observar, sólo mostramos el nombre de los objetos, y en mi caso devuelve la siguiente salida.

```
conex  
recordset
```

La colección StaticObjects no ofrece ningún método como los de la colección Contents, únicamente ofrece los métodos que son comunes a todas las colecciones de lectura, que son Count, Key e Item. Veamos un ejemplo (Código fuente 116), suponiendo que tenemos creados los objetos del ejemplo anterior.

```
Número de variables de la colección StaticObjects:  
<%=Application.StaticObjects.Count%><br>  
<%'Se recupera la segunda variable  
Set obj=Application.StaticObjects.Item(2)  
If IsObject(obj) Then Response.Write "Objeto recuperado<br>"%>  
Clave de la variable 2: <%=Application.StaticObjects.Key(2)%><br>
```

Código fuente 116

Y como viene siendo costumbre ahora mostramos el resultado:

```
Número de variables de la colección StaticObjects: 2  
Objeto recuperado  
Clave de la variable 2: recordset
```

## Métodos del objeto Application

Como se ha comentado en el apartados anteriores, las variables almacenadas en el objeto Application son compartidas por todos los usuarios de la aplicación ASP. Debido a esto se pueden dar problemas de concurrencia, es decir, cuando dos usuarios acceden a la misma información y la quieren modificar al mismo tiempo, por lo tanto se debe tener un acceso exclusivo a las variables del objeto Application cuando se quiera modificar su valor.

Aquí entendemos por usuarios a sentencias del lenguaje de secuencias de comandos, es decir, serían dos sentencias en distintas páginas que intentaran modificar la misma variable de aplicación al mismo tiempo.

Para implementar este mecanismo de exclusión mutua el objeto Application ofrece dos métodos: Lock y UnLock. Cuando se llama al método Lock, éste impide que otros usuarios modifiquen el contenido de las variables de la aplicación. Y cuando se llama al método UnLock, se permite el acceso a las variables a todos los usuarios de la aplicación.

Es un método de exclusión mutua, sólo un usuario puede a la vez estar utilizando las variables del objeto Application, las variables quedarán libres para el resto de los usuarios cuando se lance sobre el objeto Application el método UnLock. El Código fuente 117 muestra un ejemplo de utilización de estos dos métodos:

```
<%Application.Lock
Application("NumVisitas")=Application("NumVisitas")+1
Application.UnLock%>
```

Código fuente 117

Este mecanismo puede ser ineficiente, ya que se bloquean todas las variables del objeto Application cuando, como en el ejemplo anterior, necesitamos que solamente se proteja una variable.

No se debe utilizar demasiadas sentencias entre una llamada al método Lock y la correspondiente llamada a UnLock, ya que las variables del objeto Application podrían quedar bloqueadas demasiado tiempo afectando de esta forma al correcto funcionamiento de la aplicación ASP correspondiente.

Lock y UnLock se utilizarán normalmente en el momento en el que se vaya a modificar el valor de una variable de aplicación.

Otro ejemplo de utilización de estos métodos lo podemos tener al reescribir algunos de los ejemplos vistos en este capítulo.

```
Application.Lock
Application("variableUno")="valor"
Application("variableDos")=2
Application("variableTres")=True
Application.Contents("NuevaVariable")="su valor"
Application.UnLock
For Each variable in Application.Contents
    Response.Write variable & " = "&Application.Contents(variable)&"<br>"
Next%>
```

Código fuente 118

Nunca se debe olvidar el corresponder cada llamada Lock con una llamada UnLock, ya que podríamos caer en el error de dejar bloqueada las variables de la aplicación de forma indefinida.

## Eventos del objeto Application

El objeto Application posee dos eventos, cuando comienza la ejecución de la aplicación ASP y cuando finaliza la ejecución de la aplicación ASP, estos dos eventos se llaman, respectivamente Application\_OnStart y Application\_OnEnd.

Estos dos eventos se corresponden con dos procedimientos del mismo nombre definidos en el fichero global.asa. Ya empezamos a ver una de las principales funciones del ya mencionado fichero global.asa.

Dentro de estos eventos se puede situar un script que realice las tareas que consideremos necesarias. El script que se debe ejecutar en cada uno de estos eventos se deberá indicar el fichero especial llamado GLOBAL.ASA (ASA, Active Server Application), más detalles sobre este fichero se pueden encontrar en el apartado dedicado al mismo.

El evento Application\_OnStart se produce cuando entra el primer usuario en la aplicación, es decir antes de producirse el primer inicio de sesión (Session\_OnStart), por ejemplo, cuando el primer usuario carga una página ASP perteneciente a una aplicación. Dentro de este evento se deberá indicar el código de inicialización de la aplicación ASP.

En el siguiente capítulo veremos que el objeto Session también posee dos eventos, para el inicio de sesión y finalización de la misma.

Y el evento Application\_OnEnd se produce cuando se apaga el servidor Web, se descarga la aplicación ASP, finaliza la última de las sesiones de un usuario o se modifica el fichero GLOBAL.ASA, estas son las distintas formas que tiene de finalizar una sesión. Dentro de este evento se deberá escribir el script que se desee que se ejecute antes de destruir la aplicación.

En los eventos Application\_OnEnd y Application\_OnStart únicamente se tiene acceso a dos de los objetos integrados de ASP, se trata de los objetos Server y Application, si intentamos utilizar algún otro objeto del modelo de objetos de ASP se producirá un error.

## El fichero GLOBAL.ASA

El fichero GLOBAL.ASA (ASA, Active Server Application, aplicación activa de servidor o mejor aplicación ASP), que tanto hemos mencionado hasta ahora en este capítulo, es un fichero opcional que se encuentra en el directorio raíz de la aplicación ASP y que está relacionado de forma directa con los objetos integrados Application y Session. Este fichero, que debe ser único para cada aplicación ASP, tiene varias funciones:

- Definir como son tratados los eventos de los objetos Session y Application. Como hemos comentado en el apartado anterior.
- Permitir crear objetos con el ámbito de aplicación y sesión.
- Inicializar y crear variables en el ámbito de la sesión y aplicación.

Este fichero no es un fichero cuyo contenido se muestre al usuario, si un usuario intenta cargar el fichero GLOBAL.ASA en su navegador recibirá el siguiente mensaje de error:

HTTP Error 500-15 - Requests for global.asa not allowed  
Servicios de Internet Information Server

El fichero GLOBAL.ASA podrán contener:

- Información acerca del tratamiento de los eventos de los objetos Session y Application.
- Declaraciones de objetos mediante la etiqueta <OBJECT>.

La estructura general de GLOBAL.ASA es la siguiente:

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Application_OnStart
    .....
END SUB

SUB Session_OnStart
    .....
```

```

END SUB

SUB Session_OnEnd
.....
END SUB

SUB Application_OnEnd
.....
END SUB
</SCRIPT>

```

Si se escribe script fuera de las etiquetas <SCRIPT></SCRIPT> o se declara un objeto que no tiene ámbito de sesión o de aplicación se producirá un error.

Tampoco se pueden incluir ficheros de ningún tipo mediante el uso de la directiva INCLUDE.

El lenguaje de secuencias de comandos a utilizar se indica en el atributo LANGUAGE de la etiqueta <SCRIPT>, y también se debe indicar en qué lugar se ejecutará el script, esto se consigue a través del atributo RUNAT de la etiqueta <SCRIPT>, en realidad este atributo sólo tiene un valor posible: Server, es decir, el script se ejecutará en el servidor.

Para declarar objetos mediante la etiqueta <OBJECT>, para que tengan el ámbito de sesión o aplicación se deberá seguir la siguiente sintaxis:

```

<OBJECT RUNAT=Server SCOPE=Ambito ID=Identificador
{ PROGID="IDprog" | CLASSID="IDclase" }>
.....
</OBJECT>

```

La declaración de los objetos deberá ir fuera de las etiquetas de script. Los objetos declarados de esta forma no se crearán hasta que el servidor procese una secuencia de comandos en el que se haga referencia a ellos. Así se ahorran recursos, al crearse sólo los objetos que son necesarios.

Estos objetos pasan a formar parte de la colección StaticObjects del objeto Application.

El parámetro Ambito especifica el ámbito del objeto, podrá tener los valores Session o Application; Identificador especifica un nombre para la instancia del objeto; IDprog e IDclase para identificar la clase del objeto.

La propiedad SCOPE de la etiqueta <OBJECT> de HTML, tiene un valor de ámbito más, además de los de sesión y de aplicación, y se trata del ámbito de página SCOPE="PAGE". El ámbito de página indica que el objeto que se instancie sólo existirá en la página actual, pero este ámbito no se puede utilizar en el fichero global.asa sino que se puede utilizar en cualquier página ASP. Sin embargo en las páginas ASP no se puede utilizar la etiqueta <OBJECT> con el ámbito de aplicación o de sesión, únicamente se pueden instanciar objetos con ámbito de página.

El Código fuente 119 dentro del fichero GLOBAL.ASA crearía un objeto Connection de ADO con ámbito de sesión.

```

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

```

Código fuente 119

Para utilizar este objeto posteriormente dentro de una página ASP, por ejemplo, para ejecutar una sentencia SQL sobre la conexión con la base de datos, bastaría con escribir lo que indica el Código fuente 120.

```
<%conexion.Execute "DELETE FROM Usuarios WHERE id<10%">
```

Código fuente 120

Es decir, accedemos directamente al nombre del objeto, que será el indicado en la propiedad ID de la etiqueta <OBJECT>.

Veamos el código de un fichero GLOBAL.ASA concreto.

```
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">
Sub Application_OnStart
    Application("variableUno")="valor"
    Application("variableDos")=2
    Application("variableTres")=True
End Sub
Sub Application_OnEnd

End Sub
</SCRIPT>
<OBJECT RUNAT="Server" SCOPE="Application" ID="conex" PROGID="ADODB.Connection">
</OBJECT>
<OBJECT RUNAT="Server" SCOPE="Application" ID="recordset" PROGID="ADODB.RecordSet">
</OBJECT>
```

Código fuente 121

Cuando se inicie la aplicación ASP correspondiente se crearán tres variables de aplicación que pasarán a formar parte de la colección Contents, también se crean dos objetos a nivel de aplicación que pasarán a formar parte de la colección StaticObjects.

El fichero global.asa sólo se ejecutará si se ha indicado el punto de inicio de la aplicación correspondiente.

En el Código fuente 122 se muestra una página ASP que accede a las colecciones del objeto Application para comprobar las variables de aplicación creadas en el fichero global.asa anterior, también se muestra como acceder y utilizar un objeto definido a nivel de aplicación.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
Número de elementos de la colección Contents:
<b><%=Application.Contents.Count%></b><br>
<%For Each variable in Application.Contents
    Response.Write variable & " = "&Application.Contents(variable)&"<br>
Next%>
```

```

<br>
Número de elementos de la colección StaticObjects:
<b><%=Application.StaticObjects.Count%></b><br>
<%For Each objeto in Application.StaticObjects
    Response.Write objeto&"<br>
Next%>
<br>
<%'accedemos la variable de aplicación definida con <OBJECT>
conex.Open "DSN=BD;UID=sa;PWD="
Response.Write "Base de datos predeterminada: "&conex.DefaultDatabase
conex.Close%>
</BODY>
</HTML>

```

Código fuente 122

El resultado de esta página ASP es:

Número de elementos de la colección Contents: 3

variableUno = valor

variableDos = 2

variableTres = Verdadero

Número de elementos de la colección StaticObjects: 2

conex

recordset

Base de datos predeterminada: Cursos

A continuación se va a comentar la secuencia de ejecución del fichero GLOBAL.ASA.

El objeto Application se crea cuando el primer usuario se conecta a una aplicación ASP y pide una sesión, es decir, carga la primera página de la aplicación ASP. Cuando se crea el objeto Application, el servidor busca el fichero GLOBAL.ASA en el directorio raíz de esa aplicación ASP, si el fichero existe se ejecuta el script del evento Application\_OnStart.

A continuación se crea el objeto Session. La creación del objeto Session ejecuta el script que se encuentra en el evento Session\_OnStart. El script asociado al tratamiento de este evento se ejecutará antes de cargar la página indicada por la petición del navegador cliente.

Cuando el objeto Session caduca o se lanza el método Session.Abandon, se ejecutará el script que se corresponde con el evento Session\_OnEnd, el código de este evento se procesará antes de destruir la sesión.

Cuando finaliza la última sesión de los usuarios, es decir, se ejecuta el código del último evento Session\_OnEnd, se lanza el evento Application\_OnEnd antes de que se destruya el objeto Application.

Si se modifica el fichero GLOBAL.ASA, el servidor lo recompilará, para ello el servidor deberá destruir el objeto Application actual y los objetos Session actuales. Primero, el servidor procesa todas las peticiones activas, el servidor no procesará más peticiones hasta que no se procese el evento Application\_OnEnd. Los usuarios que se intenten conectar durante este proceso recibirán un mensaje que le indica que la petición no puede ser procesada mientras la aplicación es reiniciada. Durante este proceso se dan los siguientes pasos:

- Las sesiones activas se destruyen, dando lugar al procesamiento del evento Session\_OnEnd.

- La aplicación se destruye, produciéndose el evento Application\_OnEnd.
- La primera petición reiniciará el objeto Application y creará un nuevo objeto Session, es decir, se darán los eventos Application\_OnStart y Session\_OnStart, respectivamente.

También se ejecutará la el evento Application\_OnEnd cuando se descargue la aplicación ASP desde el Administrador de servicios de Internet o cuando se apague el servidor Web.

El objeto Session junto con sus eventos se verá en profundidad en el próximo capítulo, se ha creído oportuno adelantar parte de su explicación debido a su íntima relación con el fichero global.asa.

En la Figura 32 se puede apreciar a modo de resumen el orden en el que los eventos de los objetos Session y Application se producen, y en que condiciones.

El primer usuario se conecta  
y pide una página de la  
aplicación ASP.

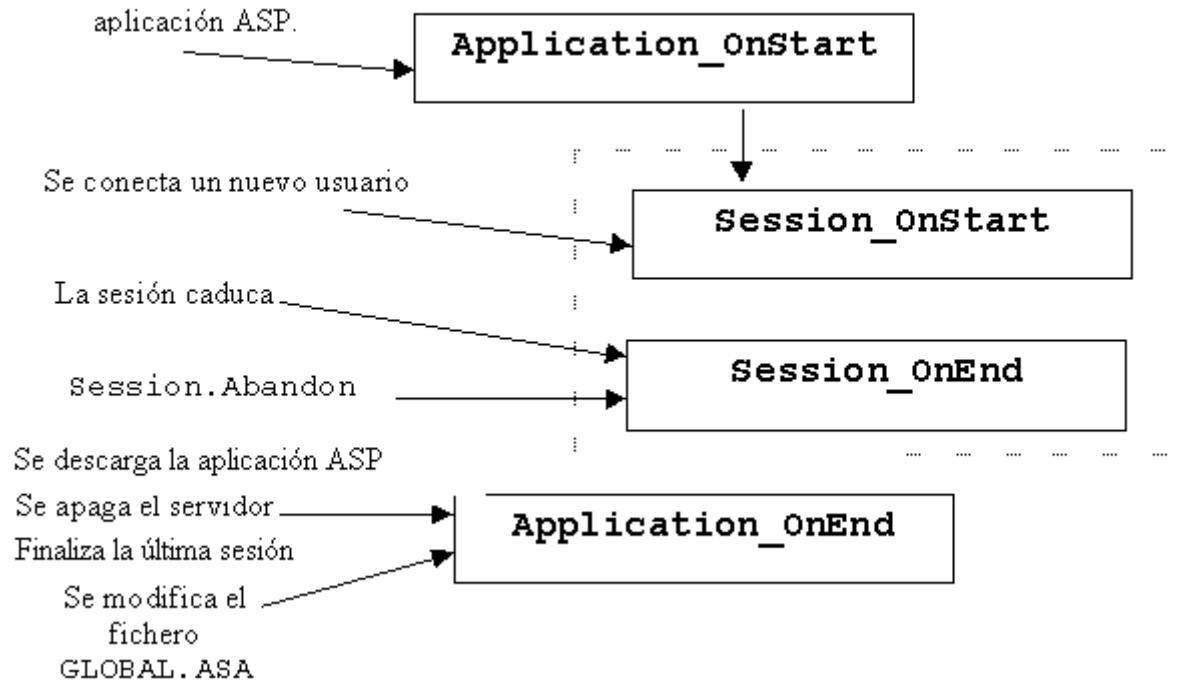


Figura 32. Eventos de los objetos Application y Session

# Modelo de objetos de ASP: el objeto Session

---

## Definición del objeto Session

Al igual que ocurría con el objeto Application, el objeto Session nos va a permitir almacenar información entre diferentes páginas ASP incluidas en una misma aplicación ASP. La diferencia con el objeto Application se encuentra en el ámbito de las variables, cada variable del objeto Session es particular a una sesión de un usuario determinado, no a toda la aplicación. De esta forma, cada usuario tendrá sus variables y sus valores, sin dar lugar a problemas de concurrencia, tampoco se podrá acceder a distintas variables de sesión, cada usuario tiene su espacio de almacenamiento.

Las variables de aplicación son valores globales y comunes a toda la aplicación, y las variables de sesión son particulares para cada usuario de la aplicación.

El servidor Web crea automáticamente un objeto Session, cuando un usuario que aún no estableció una sesión solicita una página ASP perteneciente a la aplicación ASP actual. El servidor Web sabe si un navegador tiene ya una sesión debido a que la cookie ASPSESSIONID es enviada junto con la petición de la página ASP. En la Figura 33 se puede ver el aspecto de esta cookie a través de navegador Internet Explorer.

Un uso común del objeto Session es almacenar las preferencias del usuario o información personal. Se debe señalar que se podrán almacenar variables dentro del objeto Session si el navegador acepta cookies, ya que el objeto Session para almacenar información se basa en la cookie ASPSESSIONID. Aunque la utilización de esta cookie es completamente transparente para el programador.

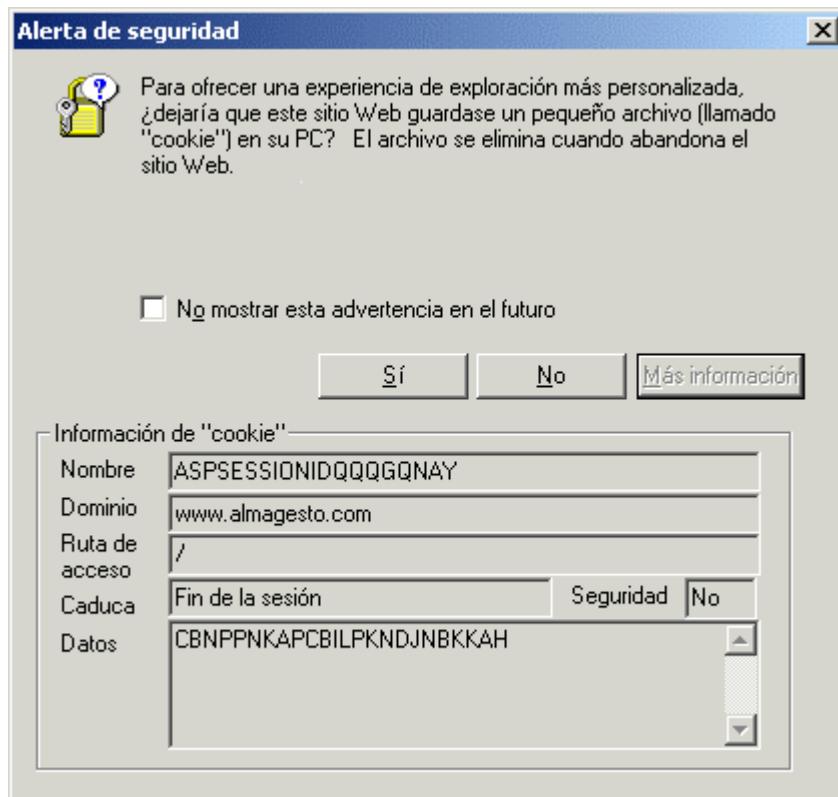


Figura 33. La cookie del objeto Session

Para una aplicación ASP tendremos tantas sesiones como usuarios conectados a la aplicación ASP.

Las recomendaciones realizadas para el objeto Application en cuanto a utilización de objetos y variables de tipo array, son también válidas para el objeto Session.

La sintaxis para manipular variables del objeto Session es la misma que en el objeto Application. El Código fuente 123 crea una variable del objeto Session con el nombre que el usuario facilita en un formulario.

```
<%Session("nombreUsuario")=Request.Form("nombre")%>
```

Código fuente 123

Vamos a mostrar un ejemplo de utilización de variables de sesión. Se trata de mostrar al usuario un formulario que le permite configurar el color de fondo de la página y el tipo, color y tamaño de la letra que se va a utilizar en la aplicación ASP. Al principio estas variables de sesión tienen un valor predeterminado que se le da en el evento de inicio de sesión Session\_OnStart, este evento lo trataremos un poco más adelante, ahora veamos el Código fuente 124.

```
<%@Language=VBScript %>
<%If Request.Form("configurar")<>"" Then
    Session("colorLetra")=Request.Form("colorLetra")
    Session("colorFondo")=Request.Form("colorFondo")
    Session("tamLetra")=Request.Form("tamLetra")
```

```

Session("tipoLetra")=Request.Form("tipoLetra")
End if%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY bgcolor="<%=>Session("colorFondo")%>">
<font face="<%=>Session("tipoLetra")%>" size="<%=>Session("tamLetra")%>" color="<%=>Session("colorLetra")%>">Este es un texto de muestra</font>
<br>
<br>
<table border="0">
<form action="<%=>Request.ServerVariables("SCRIPT_NAME")%>" method="post">
<tr>
    <td>
        <font face="<%=>Session("tipoLetra")%>" size="<%=>Session("tamLetra")%>" color="<%=>Session("colorLetra")%>">Color de fondo:</font>
    </td>
    <td><select name="colorFondo">
            <option value="green">Verde
            <option value="black">Negro
            <option value="white">Blanco
            <option value="red">Rojo
            <option value="blue">Azul
        </select>
    </td>
</tr>
<tr>
    <td>
        <font face="<%=>Session("tipoLetra")%>" size="<%=>Session("tamLetra")%>" color="<%=>Session("colorLetra")%>">Tipo de letra:</font>
    </td>
    <td><select name="tipoLetra">
            <option value="courier">Courier
            <option value="arial">Arial
            <option value="verdana">Verdana
            <option value="timesroman">TimesRoman
            <option value="Bookman Old Style">Bookman Old Style
        </select>
    </td>
</tr>
<tr>
    <td>
        <font face="<%=>Session("tipoLetra")%>" size="<%=>Session("tamLetra")%>" color="<%=>Session("colorLetra")%>">Color de letra:</font>
    </td>
    <td><select name="colorLetra">
            <option value="green">Verde
            <option value="black">Negro
            <option value="white">Blanco
            <option value="red">Rojo
            <option value="blue">Azul
        </select>
    </td>
</tr>
<tr>
    <td>
        <font face="<%=>Session("tipoLetra")%>" size="<%=>Session("tamLetra")%>" color="<%=>Session("colorLetra")%>">Tamaño de letra:</font>
    </td>
    <td><input type="text" name="tamLetra" size="2" maxlength="1"></td>
</tr>
<tr>
    <td colspan="2" align="center">
        <input type="submit" name="configurar" value="Enviar Cambios">
    </td>
</tr>

```

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

Código fuente 124

Para hacer uso de los valores de las variables de sesión tenemos que combinarlas con etiquetas HTML que definen el color de fondo de las páginas y el tipo de letra. Cada usuario tendrá su configuración personalizada. Cuando tratemos el acceso a datos desde ASP se retomará este ejemplo para guardar los valores de la configuración de la sesión del usuario en una tabla de una base de datos, ya que cuando el usuario finalice la sesión los valores seleccionados para las variables correspondientes se perderán. El código fuente de la página se encuentra en este [enlace](#).

Un ejemplo de ejecución de esta página se puede ver en la

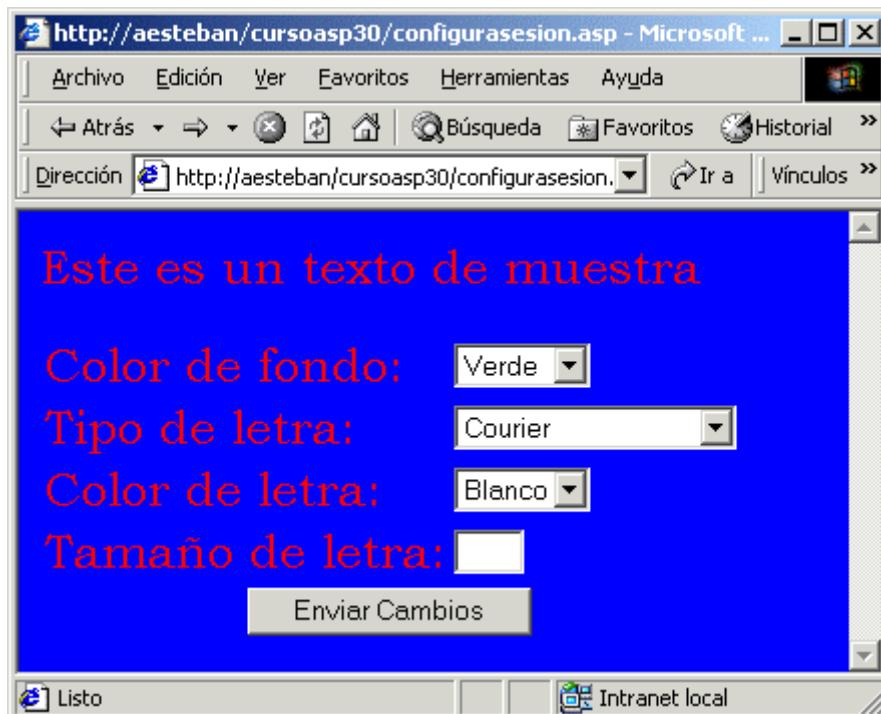


Figura 34. Configurando la sesión

## Colecciones del objeto Session

En la versión 1.0 de ASP el objeto Session ,al igual que ocurría con el objeto Application tampoco poseía ninguna colección, apareciendo en la versión de 2.0 ASP con dos nuevas colecciones con las mismas funciones que las del objeto Application pero a nivel de sesión claro:

- **Contents:** contiene todas las variables a nivel de sesión menos las creadas con al etiqueta <OBJECT>.
- **StaticObjects:** contiene las variables de la sesión creadas con la etiqueta <OBJECT>.

Ambas colecciones nos van a permitir recuperar o acceder a las variables y objetos creados, en este caso, a nivel de sesión.

La colección Contents contiene todos los objetos y variables creados a nivel de sesión, excluyendo a los objetos que han sido creados utilizando la etiqueta <OBJECT>.

En esta colección se incluirán, por tanto, los objetos creados a nivel de sesión con Server.CreateObject y las variables creadas con las sentencias del tipo Session("variable").

Así por ejemplo, en el Código fuente 125 vamos a crear una serie de variables a nivel de sesión y también un objeto, y vamos a todas ellas inspeccionando la colección Contents con la iteración For...Each.

```
<%'creación de las variables
Session("variableUno")="valor"
Session("variableDos")=2
Session("variableTres")=True
Set Session("objConexion")=Server.CreateObject ("ADODB.Connection")
'se abre la conexión
Session("objConexion").Open "DSN=BD;UID=sa;PWD="
For Each variable in Session.Contents
    Response.Write variable & " = "
    'comprobamos si la variable de aplicación es un objeto
    If IsObject(Session.Contents(variable)) Then
        Response.Write Session.Contents(variable).DefaultDatabase & "<BR>"
    Else
        Response.Write Session.Contents(variable) & "<br>"
    End if
Next
'se cierra la conexión
Session("objConexion").Close%>
```

Código fuente 125

Como bien puede observar el lector, se trata del mismo ejemplo que veíamos en el tema anterior. Las colecciones del objeto Session se utilizan de la misma forma que en el objeto Application, la única diferencia es el ámbito de las variables que contiene cada objeto.

La colección StaticObjects contiene todos los objetos que se han creado mediante la etiqueta <OBJECT> dentro del alcance de la sesión.

De la misma forma podremos recorrer la colección StaticObjects para acceder a los objetos estáticos creados a nivel de sesión. Si suponemos que tenemos el fichero GLOBAL.ASA que aparece en el

```
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">
Sub Session_OnStart
    Session("variableUno")="valor"
    Session("variableDos")=2
    Session("variableTres")=True
End Sub
</SCRIPT>
<OBJECT RUNAT="Server" SCOPE="Session"
ID="conex" PROGID="ADODB.Connection">
</OBJECT>
<OBJECT RUNAT="Server" SCOPE="Session"
ID="recordset" PROGID="ADODB.RecordSet">
</OBJECT>
```

Código fuente 126

La ejecución del Código fuente 127,

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
Número de elementos de la colección Contents:
<b><%=Session.Contents.Count%></b><br>
<%For Each variable in Session.Contents
    Response.Write variable & " = "&Session.Contents(variable)&"<br>
Next%>
<br>
Número de elementos de la colección StaticObjects:
<b><%=Session.StaticObjects.Count%></b><br>
<%For Each objeto in Session.StaticObjects
    Response.Write objeto&"<br>
Next%>
<br>
<%'accedemos la variable de aplicación definida con <OBJECT>
conex.Open "DSN=BD;UID=sa;PWD="
Response.Write "Base de datos predeterminada: "&conex.DefaultDatabase
conex.Close%>
</BODY>
</HTML>
```

Código fuente 127

producirá este resultado:

```
Número de elementos de la colección Contents: 3
variableUno = valor
variableDos = 2
variableTres = Verdadero

Número de elementos de la colección StaticObjects: 2
conex
recordset

Base de datos predeterminada: Cursos
```

## Propiedades del objeto Session

El objeto Session posee cuatro propiedades:

- SessionID: contiene la identificación de la sesión para el usuario. Cada sesión tiene un identificador único que genera el servidor al crearla. No se debe utilizar esta propiedad como clave de una tabla de una base de datos, ya que, al reiniciar el servidor Web, algunos de los valores de SessionID pueden coincidir con los generados antes de que se apague el servidor. Es únicamente de lectura.
- Timeout: la propiedad Timeout especifica el intervalo de inactividad para el objeto Session en minutos. Si el usuario no actualiza o solicita una página durante ese intervalo, la sesión

termina. El valor por defecto de esta propiedad es de 20 minutos, es decir, por defecto la sesión permanecerá inactiva 20 minutos. Una sesión se dice que está inactiva mientras el navegador cliente no realice una petición. El valor de esta propiedad se puede modificar dinámicamente a lo largo de la ejecución de la aplicación ASP.

- CodePage: indica la página de códigos de caracteres que va a ser utilizado. A esta propiedad se le puede asignar un valor entero para especificar la página de códigos que se va a utilizar. Así si queremos utilizar el juego de caracteres del alfabeto turco le asignaremos a esta propiedad el valor 1254 (`<% Session.CodePage = 1254 %>`). Esta propiedad es también de lectura/escritura.
- LCID (Locale Identifier): propiedad de lectura/escritura, es una abreviatura estándar e internacional que identifica de forma única la localización de los sistemas. Esta localización determina la forma en la que se le da formato a las horas y fechas, como se tratan las cadenas de caracteres y los diferentes elementos del alfabeto. Así si queremos establecer la propiedad LCID para la región de Rusia, le debemos asignar a esta propiedad el valor 1049 (`<% Session.LCID = 1049 %>`).

Nota: Las últimas dos propiedades han sido incluidas con la versión 2.0 de ASP

Vamos a ver el Código fuente 128 que muestra los valores que tienen asignados por defecto las distintas propiedades del objeto Session.

```
Session.TimeOut= <%=Session.Timeout%><br>
Session.SessionID= <%=Session.SessionID%><br>
Session.LCID= <%=Session.LCID%><br>
Session.CodePage= <%=Session.CodePage%><br>
```

Código fuente 128

En mi caso los resultados son estos:

```
Session.TimeOut= 20
Session.SessionID= 521229107
Session.LCID= 2048
Session.CodePage= 1252
```

Las propiedades del objeto Session CodePage y LCID se pueden modificar a través de dos directivas de procesamiento, al igual que indicábamos el lenguaje de secuencias de comandos con la directiva `<%@LANGUAGE>`. En este caso las directivas a utilizar son LCID y CODEPAGE. Así si escribimos el Código fuente 129,

```
<%@LANGUAGE=VBScript CODEPAGE="932" LCID="2055"%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
Session.TimeOut= <%=Session.Timeout%><br>
Session.SessionID= <%=Session.SessionID%><br>
Session.LCID= <%=Session.LCID%><br>
Session.CodePage= <%=Session.CodePage%><br>
```

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

Código fuente 129

obtenemos estos valores:

```
Session.TimeOut= 20
Session.SessionID= 521229110
Session.LCID= 2055
Session.CodePage= 932
```

Estos valores se mantienen a lo largo de la sesión de un usuario en concreto, pudiéndose modificar de nuevo en cualquier momento. También es posible deshabilitar el estado de la sesión, es decir, indicamos que no queremos utilizar la cookie de inicio de sesión y por lo tanto no podemos almacenar variables a nivel de sesión. Esto lo podemos hacer de dos formas distintas, mediante la directiva de procesamiento ENABLESESSIONSTATE y mediante la configuración de la aplicación ASP desde el Administrador de servicios de Internet.

Veamos la primera de las opciones. Consiste simplemente en asignar a la directiva de procesamiento ENABLESESSIONSTATE los valores verdadero o falso, dependiendo de si queremos habilitar el estado de la sesión o deshabilitarlo. Por defecto tiene el valor True.

La segunda forma de indicar si queremos utilizar el estado de sesión o no, consiste en acudir a la aplicación ASP que teníamos definida en el Administrador de servicios de Internet. Pulsamos sobre la aplicación con el botón derecho del ratón y seleccionamos la opción de menú propiedades, aparecerán las hojas de propiedades del directorio.

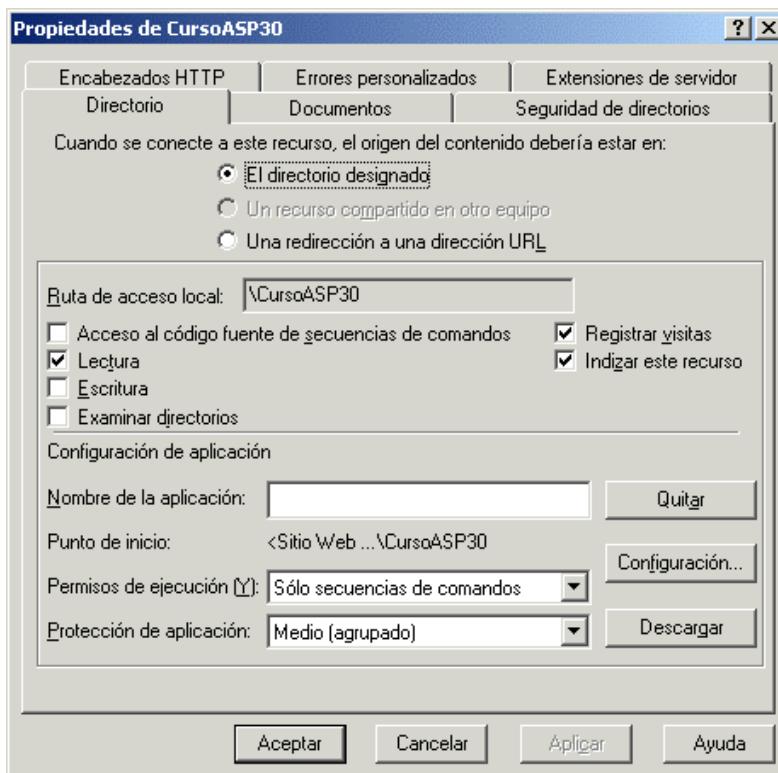


Figura 35. Página de propiedades de un directorio Web

En el epígrafe denominado configuración de aplicación podemos observar un botón etiquetado como Configuración, si lo pulsamos aparece una nueva ventana de la que elegiremos la pestaña etiquetada como opciones de aplicación. Desde esta hoja de propiedades podemos configurar algunos parámetros de nuestra aplicación ASP, entre ellos el que nos interesa, es decir, la posibilidad de habilitar o deshabilitar el estado de la sesión.

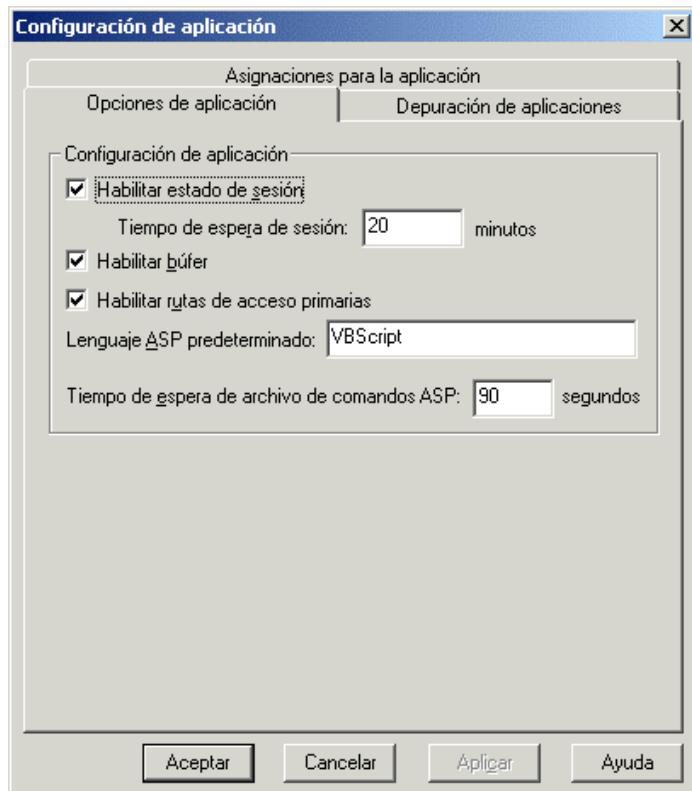


Figura 36. Configurando la aplicación ASP

Además podemos configurar la propiedad Timeout de la sesión y el lenguaje del intérprete de comandos por defecto. También podemos habilitar o deshabilitar el almacenamiento en búfer. Los valores que especifiquemos en esta pantalla se pueden sobreescribir desde el código ASP indicando los valores de las propiedades o directivas de procesamiento correspondientes.

En la Figura 35 se pueden observar los valores por defecto que tienen las opciones de configuración de la aplicación ASP.

## Métodos del objeto Session

Este objeto posee un único método: el método Abandon. Al lanzar sobre el objeto Session el método Abandon, se destruyen todas las variables de la sesión y se liberan sus recursos, finalizando la misma. Si no se llama explícitamente al método Abandon, el servidor destruirá los objetos cuando la sesión caduque, atendiendo al valor de la propiedad Timeout.

La destrucción del objeto Session no se hará efectiva hasta que el servidor no haya terminado de ejecutar la página ASP. Por lo tanto las variables del objeto Session existirán mientras no se cargue otra página.

El servidor creará un nuevo objeto Session al abrir una nueva página ASP después de abandonar la sesión.

Si tenemos el Código fuente 130, podemos ver que después de llamar al método Abandon, siguen existiendo las variables de sesión, al cargar una nueva página ASP será cuando realmente se destruyan.

```
<%Session("variablePrueba")="Esto es de prueba"  
Session.Abandon%>  
<%=Session("variablePrueba") %>
```

Código fuente 130

## Eventos del objeto Session

Al igual que ocurría en el objeto Application, el objeto Session objeto posee dos eventos: el inicio de sesión, Session\_OnStart, y el fin de sesión Session\_OnEnd.

El inicio de sesión se produce cuando el usuario carga la primera página se una aplicación ASP. Dentro de este evento se deberán indicar las acciones a llevar a cabo antes de que se cargue la primera página de una aplicación ASP.

Este evento se suele utilizar para inicializar las variables para toda la sesión, así por ejemplo, si la aplicación ASP va a manipular una serie de tablas de una base de datos, se podrá crear una conexión con la base de datos en el evento de inicio de sesión y almacenarla en el objeto Session para que esté disponible para ese usuario mientras su sesión permanezca activa.

También se puede utilizar para cargar una serie de variables que definan el perfil del usuario que se acaba de conectar.

El fin de la sesión, es decir, el evento Session\_OnEnd, se puede producir porque se haya caducado la sesión al permanecer inactiva el tiempo indicado por su propiedad Timeout, o bien, se puede forzar mediante a una llamada al método Abandon.

Cuando el usuario cierra el navegador no se ejecuta el evento Session\_OnEnd de forma inmediata, sino que se ejecutará cuando se cumpla el Timeout correspondiente, para la aplicación ASP es igual que el usuario se encuentre con la página cargada en el navegador sin realizar ninguna petición o que el usuario haya cerrado su sesión del navegador, lo que se tiene en cuenta es el tiempo de inactividad de la sesión del usuario.

Sin embargo una llamada a Session.Abandon lanza el evento Session\_OnEnd de manera inmediata.

Al igual que ocurría con el objeto Application, los procedimientos para el tratamiento de estos eventos se deben incluir en el ya conocido fichero de aplicación global.asa.

En el evento Session\_OnEnd sólo están disponibles los objetos Application, Server y Session, y tampoco se puede utilizar el método MapPath del objeto Server. En el evento Session\_OnStart se tiene acceso a todos los objetos integrados de ASP.

Podemos plantearnos un sencillo ejemplo para ilustrar la utilización de los eventos del objeto Session. Este ejemplo consiste en realizar un contador de visitas a nuestra aplicación Web, y un contador de número de usuarios conectados en ese momento.

Para guardar los dos contadores vamos a utilizar dos variables a nivel de aplicación. El contador de visitas se va a incrementar en el evento Session\_OnStart, es decir, cada vez que entre un usuario, lo mismo ocurrirá con el contador de conectados en ese momento. Además, el contador de conectados se deberá decrementar cuando el usuario se desconecte de la aplicación ASP, es decir, en el evento Session\_OnEnd.

Por lo tanto el fichero GLOBAL.ASA debería quedar como muestra el Código fuente 131.

```
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">
Sub Application_OnStart
    Application.Lock
    Application("numVisitas")=0
    Application("numConectados")=0
    Application.UnLock
End Sub
Sub Session_OnStart
    Application.Lock
    Application("numVisitas")=Application("numVisitas")+1
    Application("numConectados")=Application("numConectados")+1
    Application.UnLock
End Sub
Sub Session_OnEnd
    Application.Lock
    Application("numConectados")=Application("numConectados") -1
    Application.UnLock
End Sub
</SCRIPT>
```

Código fuente 131

Tendríamos una página para mostrar la información de los contadores y también con un enlace a una página de desconexión que tuviera una llamada al método Abandon del objeto Session.

La página que mostrara la información de número de sesiones y usuarios conectados podría ser la que muestra el Código fuente 132.

```
Número de visitas: <b><%=Application("numVisitas")%></b><br>
Número de conectados en <%=Now%>: <b><%=Application("numConectados")%></b><br>
<a href="Desconectar.asp">Desconectar</a>
```

Código fuente 132

Y la de desconexión es la más sencilla.

```
<%'terminamos la sesión
Session.Abandon
'se envia al raíz del Web al usuario
Response.Redirect"/%">
```

Código fuente 133



# Modelo de objetos de ASP: el objeto Server

---

## Definición del objeto Server

El objeto Server nos permite ampliar las capacidades de las páginas ASP mediante la posibilidad de la creación y utilización de objetos externos y componentes en el lenguaje de secuencias de comandos.

El objeto Server está diseñado para realizar tareas específicas en el servidor. Además de la posibilidad de instanciar componentes el objeto Server ofrece una serie de métodos muy útiles como pueden ser los que permiten dar formato URL o HTML a cadenas de caracteres, los que modifican la línea de ejecución de un script de una página con la posibilidad de ejecutar distintas páginas, y también existe un método utilizado para el tratamiento de errores dentro de ASP, etc. Todos estos métodos los veremos en el presente capítulo.

En algunas definiciones del objeto Server se suele indicar también que nos permite acceder a los recursos del servidor, en cierto modo esto es correcto ya que para poder instanciar un componente debe encontrarse registrado en el servidor Web.

## Propiedades del objeto Server

El objeto Server posee una única propiedad ScriptTimeout que es de lectura/escritura. La propiedad ScriptTimeOut expresa en segundos el periodo de tiempo durante el que puede ejecutarse una secuencia de comandos (script) antes de que termine su intervalo de espera.

El intervalo de espera de la secuencia de comandos no tendrá efecto mientras haya un componente del servidor en proceso.

Las modificaciones sobre la propiedad ScriptTimeOut del objeto Server se aplican únicamente a la página actual. Puede ser interesante aumentar el valor de esta propiedad en el caso de páginas que tarden mucho tiempo en ejecutarse debido a su complejidad, ya que si las secuencias de comandos de una página tardan en ejecutarse más tiempo que el especificado en la propiedad ScriptTomeOut se producirá un error.

La propiedad ScriptTimeOut es una forma de evitar que el servidor Web se sobrecargue con páginas ASP que presentan tiempo de ejecución excesivo.

El valor por defecto que posee esta propiedad es de 90 segundos, el valor por defecto se puede modificar a través del Administrador de servicios de Internet para que sea aplicable a todas las páginas ASP que componen una aplicación ASP determinada, es decir, se puede definir un valor por defecto de la propiedad ScriptTimeout para cada aplicación ASP.

La forma de modificar el valor por defecto de la propiedad ScriptTimeout es acudiendo a las propiedades de la carpeta que contiene la aplicación ASP dentro del Administrador de servicios de Internet. Pulsamos el botón de Configuración y accedemos a la pestaña Opciones de la aplicación. En esta ventana podemos ver un parámetro denominado Tiempo de espera de archivo de comandos ASP, este será el parámetro que defina el valor predeterminado de la propiedad ScriptTimeout.

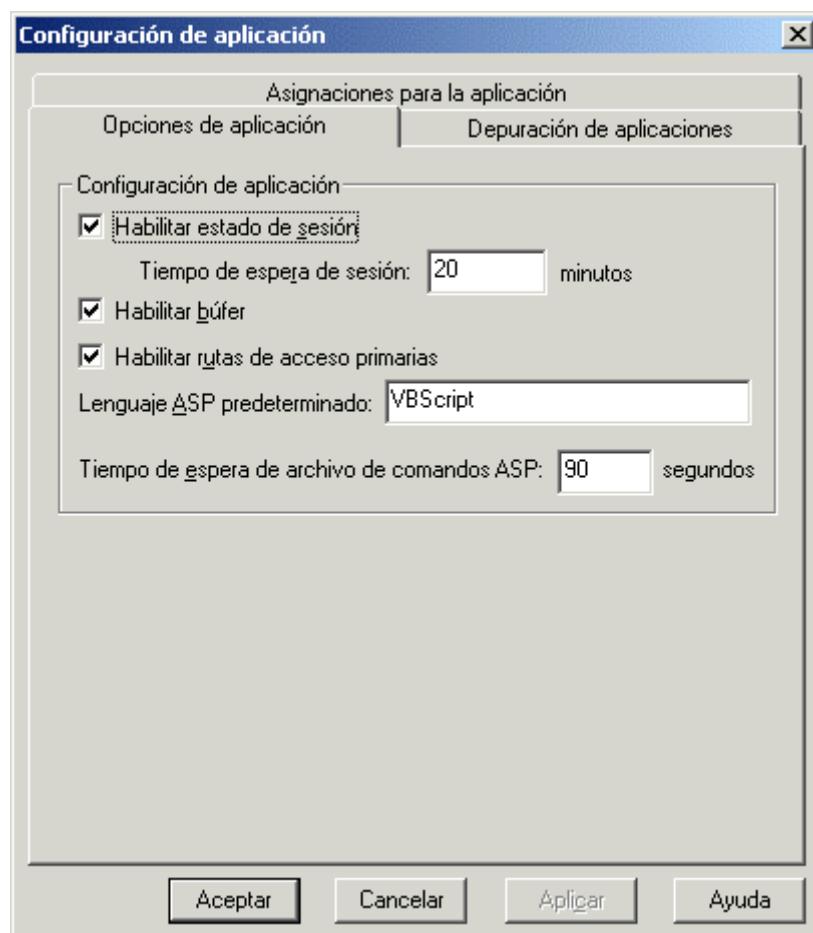


Figura 37. Opciones de una aplicación ASP

## Métodos del objeto Server

A continuación se ofrece una breve descripción de todos los métodos que presenta el objeto integrado Server, al lo largo de este apartado iremos profundizando en cada uno de ellos.

- CreateObject: método por excelencia del objeto Server, crea una instancia de un componente. Este componente debe estar registrado en el servidor Web.
- HTMLEncode: devuelve una cadena codificada en HTML a partir de la cadena que se le pasa como parámetro.
- MapPath: devuelve la ruta física de una ruta virtual dada a través de una cadena en formato URL.
- URLEncode: devuelve una cadena a la que se le ha aplicado la codificación URL correspondiente a las cadenas de consulta (QueryString).
- URLPathEncode: devuelve una cadena a la que se le ha aplicado la codificación URL correspondiente a las cadenas de rutas.
- Execute: para la ejecución de la página actual y transfiere la ejecución a la página indicada. Una vez que ha finalizado la ejecución de la nueva página, la ejecución continúa en la página inicial.
- Transfer: finaliza la ejecución de la página actual y transfiere la ejecución a la página indicada por parámetro. Sin volver en ningún caso a la página original.
- GetLastError: devuelve una referencia al objeto ASPError que contiene la información detallada del último error que se ha producido.

Los cuatro últimos métodos han sido añadidos en la versión 3.0 de ASP.

### CreateObject

Es el método más utilizado del objeto Server, posiblemente, sea el método CreateObject. Este método permite la instanciación de componentes de servidor, para que a continuación sean manipulados por las secuencias de comandos de nuestras páginas ASP. Para crear una instancia de un componente de servidor se debe utilizar la sintaxis general:

```
Set nombreObjeto=Server.CreateObject(ProgID)
```

La palabra reservada Set siempre se debe utilizar en la instanciación de un objeto y ProgID es el nombre con el que está registrado el objeto en el servidor, el formato para ProgID es: [Fabricante.]Componente[.Versión]. Así por ejemplo el ProgID de el objeto Recordset del modelo de objetos de acceso a datos ActiveX Data Objects (ADO) es ADODB.Recordset.

La variable nombreObjeto debe tener un nombre distinto de cualquiera de los objetos integrados de ASP. Se puede destruir un objeto asignándole Nothing, de esta forma se libera la memoria ocupada por el objeto.

Las instancias de un objeto, por defecto, se destruyen cuando la página ASP termina de ser procesada, aunque es más seguro destruirlas mediante la palabra reservada Nothing.

```
<%Set nombreObjeto=Nothing%>
```

Código fuente 134

Para crear un objeto con ámbito de sesión o de aplicación se debe almacenar el objeto en una variable de sesión o de aplicación, o bien utilizando la etiqueta de HTML <OBJECT> en el fichero global.asa y asignando al parámetro SCOPE los valores Session o Application, como vimos en el capítulo anterior. De esta forma el objeto se destruirá cuando haya finalizado la sesión o la aplicación.

Así por ejemplo, si queremos crear una conexión a una base de datos, utilizando un componente ActiveX Server de manejo de bases de datos, y que tenga el ámbito de una sesión, se deberá escribir el Código fuente 135.

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

Código fuente 135

No es demasiado recomendable utilizar objetos a nivel de sesión o aplicación, y si se utilizan se deben destruir cuanto antes, ya que pueden suponer una gran carga para la memoria del servidor Web. Es preferible crear y destruir un objeto varias veces que llevar su referencia almacenada en una variable de sesión o de aplicación.

En el Código fuente 136 se muestra la utilización del método CreateObject para utilizar un componente de VBScript. Se trata de crear una instancia de un componente FileSystemObject para mostrar los datos de un fichero del servidor.

```
<%Set objficheroSistema=Server.CreateObject("Scripting.FileSystemObject")
Set fichero=objficheroSistema.GetFile("e:\tmp\coleccio.zip")%
Fecha creación: <%=fichero.DateCreated%><br>
Nombre: <%=fichero.Name%><br>
Tamaño: <%=fichero.Size%><br>
Tipo: <%=fichero.Type%><br>
<%Set fichero=Nothing
Set objficheroSistema=Nothing%>
```

Código fuente 136

El resultado es:

```
Fecha creación: 25/05/2000 15:26:48
Nombre: coleccio.zip
Tamaño: 1043445
Tipo: WinZip File
```

Una vez creado el objeto podemos utilizar todos sus métodos, propiedades y colecciones. En el capítulo dedicado a los componentes de VBScript trataremos todos ellos en detalle.

## MapPath

El método MapPath devuelve la ruta física que se corresponde con la ruta virtual que se le pasa por parámetro. El objeto Server posee información sobre los directorios físicos, virtuales y relativos de la máquina servidor y sabe como debe realizar la traducción de rutas virtuales a físicas. Se puede necesitar obtener rutas físicas cuando se quieran crear directorios o manipular ficheros en el servidor. La forma de utilizar este método es la siguiente:

```
nombreVariable=Server.MapPath(rutaVirtual)
```

Dónde rutaVirtual contiene el ruta virtual del directorio o fichero del que se desea obtener su ruta física y nombreVariable contendrá la ruta física correspondiente.

Si la ruta que se le pasa como parámetro al método MapPath empieza con una barra (/ o \), el método traduce la ruta desde el directorio raíz del servidor hasta el directorio virtual especificado. Por otro lado, si no posee barra se traducirá de forma relativa al directorio donde reside la página ASP que ha realizado la llamada al método.

Así por ejemplo, si el directorio inicial del servidor (directorio de publicación en Internet) está establecido como el directorio c:\inetpub\wwwroot, y si tenemos una página ASP ejecutándose en el directorio c:\inetpub\wwwroot\cursoASP30 y escribimos el Código fuente 137 para mapear la ruta de un fichero llamado FICH.TXT que se encuentra en el mismo directorio que la página ASP:

```
<%=Server.MapPath("/scripts/fich.txt")%><br>
<%=Server.MapPath("scripts/fich.txt")%>
```

Código fuente 137

Se obtendrá en el navegador la siguiente salida:

```
C:\inetpub\wwwroot\scripts\fich.txt
C:\inetpub\wwwroot\cursoASP\scripts\fich.txt
```

El método MapPath no comprueba si la ruta virtual que le pasamos por parámetro existe en el servidor. Para saber la ruta virtual de la página actual el servidor utiliza una variable llamada PATH\_INFO incluida en la colección ServerVariables del objeto Request. Para obtener el valor de la ruta física de la página ASP actual deberemos escribir el Código fuente 138 dentro de la página ASP.

```
<%=Server.MapPath(Request.ServerVariables("PATH_INFO"))%>
```

Código fuente 138

Y si escribimos el Código fuente 139 obtendremos la ruta física del directorio inicial del servidor Web, es decir, el directorio del que cuelga toda la publicación en Internet.

```
<%=Server.MapPath("/")%>
```

Código fuente 139

Se recomienda por eficiencia evitar el uso del método MapPath, cada llamada a este método supone una nueva conexión al servidor Web para que IIS devuelva la ruta actual del servidor. En su lugar es más rápido utilizar la ruta literal, si se conoce, de esta forma en el ejemplo anterior, si queremos acceder al fichero fich.txt deberemos escribir su ruta completa: C:\inetpub\wwwroot\scripts\fich.txt.

## HTMLEncode

El método HTMLEncode devuelve una cadena a la que se le ha aplicado la codificación HTML a partir de la cadena que se le pasa por parámetro.

Este método se suele utilizar para mostrar texto en el navegador sin que sea interpretado por el mismo como si fuera HTML, veamos un sencillo ejemplo, en el Código fuente 140.

```
<%Response.Write "<b>Cadena en HTML</b>"%><br>
<%Response.Write Server.HTMLEncode("<b>Cadena en HTML</b>")%><br>
```

Código fuente 140

La salida de esta página será:

**Cadena en HTML**  
<b>Cadena en HTML</b>

Es decir, en el primer caso el navegador interpreta las etiquetas <B> de HTML, y por ello muestra el texto en negrita, y la segunda sentencia aplica el método HTMLEncode y de esta forma se evita que el navegador trate las etiquetas HTML, y muestra todo como un texto normal, si vemos el código de la página HTML generada podemos ver lo que muestra el Código fuente 141.

```
<b>Cadena en HTML</b><br>
&lt; b &gt; Cadena en HTML &lt; / b &gt; <br>
```

Código fuente 141

## URLEncode

El método URLEncode se utiliza para aplicar la codificación URL a la cadena que se le pasa como parámetro, devolviendo la cadena resultante de realizar dicha codificación. La codificación URL que se aplica es la perteneciente a las cadenas de consulta, es decir, la información que se sitúa en una URL a partir del carácter ?, lo que en ASP se representa mediante el objeto integrado QueryString.

Existen una serie de caracteres ASCII que se codificarán de forma hexadecimal dentro de la codificación URL, a estos códigos hexadecimales les precederá el símbolo %. Por ejemplo el carácter < se codificará como %3C, el espacio en blanco como %20 y la arroba (@) como %40.

Dentro de la codificación de las cadenas de consulta existen una serie de caracteres reservados que tienen un significado especial, en la Tabla 15 se muestran estos caracteres.

Carácter	Función en la codificación URL de la cadena de consulta
+	Separar datos.
=	Relaciona el nombre de un parámetro con su valor
&	Separa pares parámetro/valor
%	Indica que el carácter se ha codificado utilizando la codificación hexadecimal

Tabla 15

Este método se suele utilizar cuando debemos pasar una cadena a través de una cadena de consulta de una página y posee caracteres especiales como puede ser blancos, tildes, etc. El navegador Internet Explorer suele realizar la transformación a la codificación URL de forma automática, pero Netscape no lo hace y además puede producir errores.

Si queremos utilizar un enlace con un QueryString que posea un parámetro llamado nombre, deberemos hacer lo que indica el Código fuente 142.

```
<a href="Pagina.asp?nombre=<%=Server.URLEncode ("Angel Esteban Núñez") %>">Enlace</a>
```

Código fuente 142

En la barra de direcciones del navegador al pulsar sobre el enlace aparecerá la siguiente codificación URL:

`http://aesteban/cursoasp30/Pagina.asp?nombre=Angel+Esteban+N%FA%Flez`

## URLPathEncode

Si leemos la documentación en relación a ASP que ofrece Microsoft con el servidor Web Internet Information Server 5, vemos que este método del objeto Server no aparece, sin embargo si lo descubrimos utilizando la facilidad intellisense, que ofrece Visual InterDev, sobre el objeto Server.

Este método no documentado es muy similar a URLEncode, aplica la codificación URL pero no para cadenas de consulta, sino para caminos virtuales y direcciones URL. La diferencia entre el método anterior y éste se ve en el Código fuente 143.

```
Server.URLEncode: <%=Server.URLEncode("http://www.almagesto.com")%><br>
Server.URLPathEncode: <%=Server.URLPathEncode("http://www.almagesto.com")%><br>
```

Código fuente 143

Cuyo resultado es:

`Server.URLEncode: http%3A%2F%2Fwww%2Ealmagesto%2Ecom`  
`Server.URLPathEncode: http://www.almagesto.com`

Este método puede ser útil para utilizar con direcciones de Internet que posean caracteres especiales, como por ejemplo blancos o tildes, aunque esto no suele ser habitual.

## Execute

Cuando tratamos el objeto Response en su capítulo correspondiente vimos que presentaba un método llamado Redirect que nos permitía pasar a ejecutar una página distinta, pero esto suponía enviar una respuesta al cliente para indicarle la carga de una nueva página, que es la página a la que pasamos la ejecución, el aspecto de las cabeceras HTTP que se envían al cliente podría ser:

```
HTTP/1.1 Object Moved
Location /directorio/pagina.asp
```

La utilización del método Redirect es bastante costosa y problemática ya supone un envío de información más del servidor al cliente para indicarle mediante una cabecera HTTP de redirección que la página ha cambiado de localización, siendo la nueva localización la página que deseamos cargar. Esto es problemático ya que en algunos navegadores como Netscape Communicator aparece un mensaje del tipo *El objeto requerido se ha movido y se puede encontrar aquí*, esto también ocurre cuando la conexión la realiza el cliente a través de proxy.

Pero mediante los métodos Execute y Transfer podemos evitar esta redirección, que como hemos visto, tiene lugar en el cliente. Estos dos métodos permiten que la redirección tenga lugar en el servidor, quedando el cliente completamente ajeno. Ambos métodos reciben como parámetro la ruta de la página a la que queremos redirigir al cliente.

La utilización del método Execute es muy similar a realizar una llamada a un procedimiento o función. Cuando lanzamos el método Execute se empieza a ejecutar la página que indicamos por parámetro, y cuando termina la ejecución de esta nueva página, el control pasa a la siguiente sentencia después de la llamada al método Execute en la página inicial, siguiendo a partir de aquí con la ejecución de la página, es decir, el navegador del cliente recibe una salida formada por la combinación de la ejecución de ambas páginas.

Esto se ilustra con un sencillo ejemplo, la primera página llamada PAGINAORIGINAL.ASP, en un punto determinado hace una llamada a Server.Execute para ejecutar OTRAPAGINA.ASP, una vez terminada la ejecución de OTRAPAGINA.ASP, se sigue con la siguiente sentencia a Server.Execute en la página paginaORIGINAL.ASP. Veamos el Código fuente 144 y el Código fuente 145.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<b>Estás en la página inicial</b><br>
Server.ScriptTimeout=<%=Server.ScriptTimeout%><br>
<%Server.Execute "otraPagina.asp"
Response.Write "he vuelto de la ejecución"%>
</BODY>
</HTML>
```

Código fuente 144

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<table border="1">
<tr><td>
Estás en la otra página<br>
<b><%=Date%></b><br>
<b><%=Time%></b><br>
</td></tr>
</table>
</BODY>
</HTML>
```

Código fuente 145

El resultado de la ejecución de esta página se puede ver en la Figura 38.

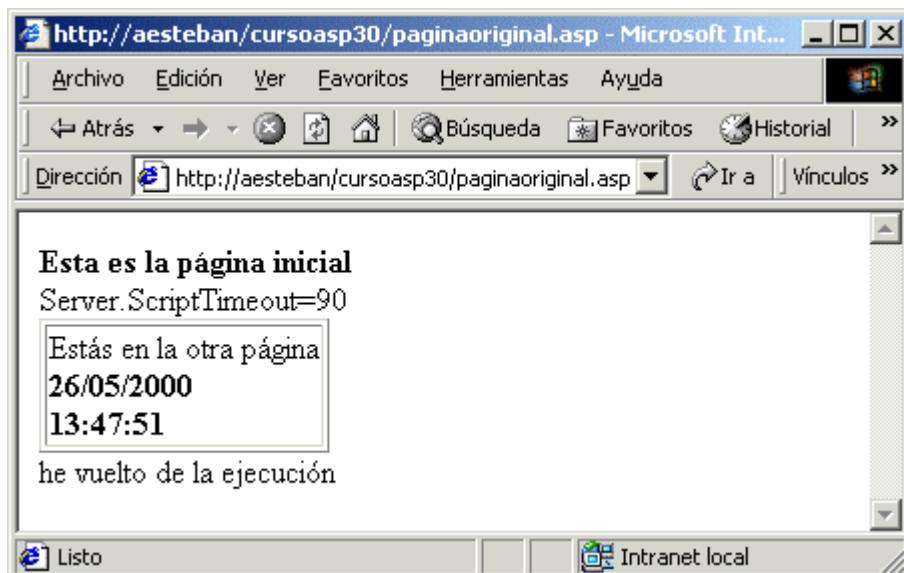


Figura 38. Utilización del método Execute

El cliente ni siquiera observa la URL correspondiente a la página otraPagina.asp ya que la redirección entre páginas se produce en el servidor, el navegador cree que sigue recibiendo todavía la página original que había demandado, incluso en la barra de direcciones del navegador sigue apareciendo la misma URL y los botones Atrás y Adelante funcionan correctamente.

En ambos métodos se mantiene el contexto de la página inicial, es decir, en la nueva página tenemos acceso a las variables, objetos y a todos los objetos intrínsecos de ASP (Request, Session, Response...) de la página inicial o página de origen. Si modificamos la página otraPagina.asp y mostramos el contenido de la variable SCRIPT\_NAME de la colección ServerVariables del objeto Request, veremos que tiene el valor que se corresponde con el de la página original.

El nuevo código de OTRAPAGINA.ASP es el que muestra el Código fuente 146.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<table border="1">
<tr><td>
Estás en la otra página<br>
<b><%=Date%></b><br>
<b><%=Time%></b><br>
Sin embargo Request.ServerVariables("SCRIPT_NAME") =
<%=Request.ServerVariables("SCRIPT_NAME")%>
</td></tr>
</table>
</BODY>
</HTML>
```

Código fuente 146

Y el nuevo resultado demuestra que se mantiene el contexto de la página ASP de origen.

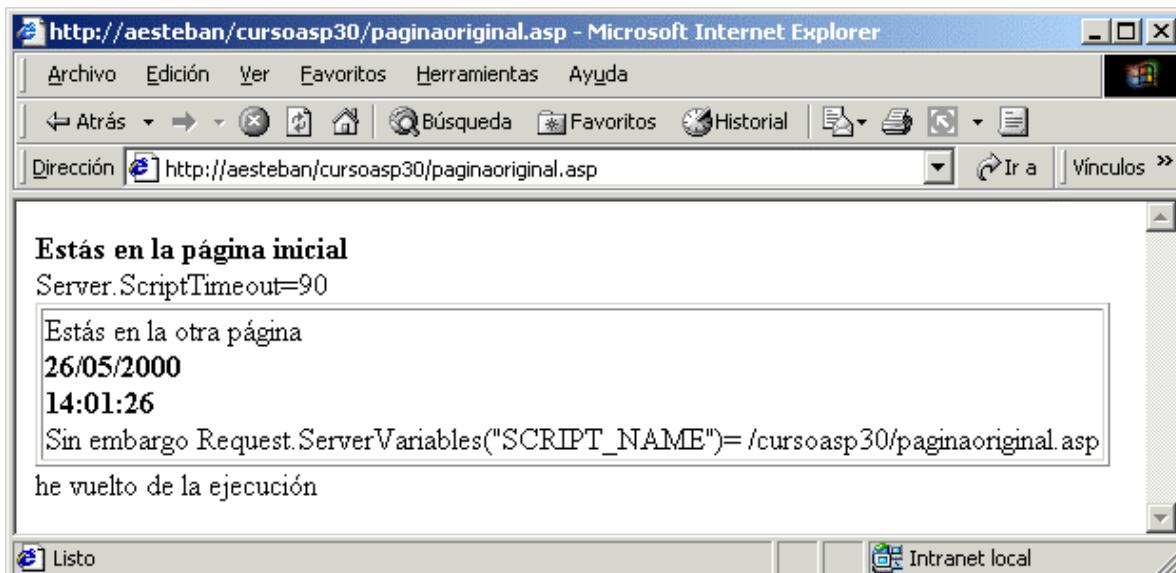


Figura 39. Se mantiene el contexto entre páginas

## Transfer

El método Transfer también permite la redirección entre páginas en el servidor, pero este método se comporta de distinto modo al método Execute, al lanzar este método se pasa la ejecución a la nueva página, pero una vez que finaliza la ejecución de la misma no se vuelve a la página inicial, como ocurría con el método Execute. El método Transfer es bastante más parecido al método Redirect que lo era el método Execute, ya que con Transfer al igual que con Redirect no se regresa a la página original.

Con el método Transfer se sigue manteniendo el contexto de la página inicial entre las distintas páginas. Para ver como funciona el método Transfer vamos a utilizar el ejemplo anterior y

simplemente vamos a cambiar la llamada a Execute por Transfer. El resultado que se obtiene es el que nos muestra la Figura 40.

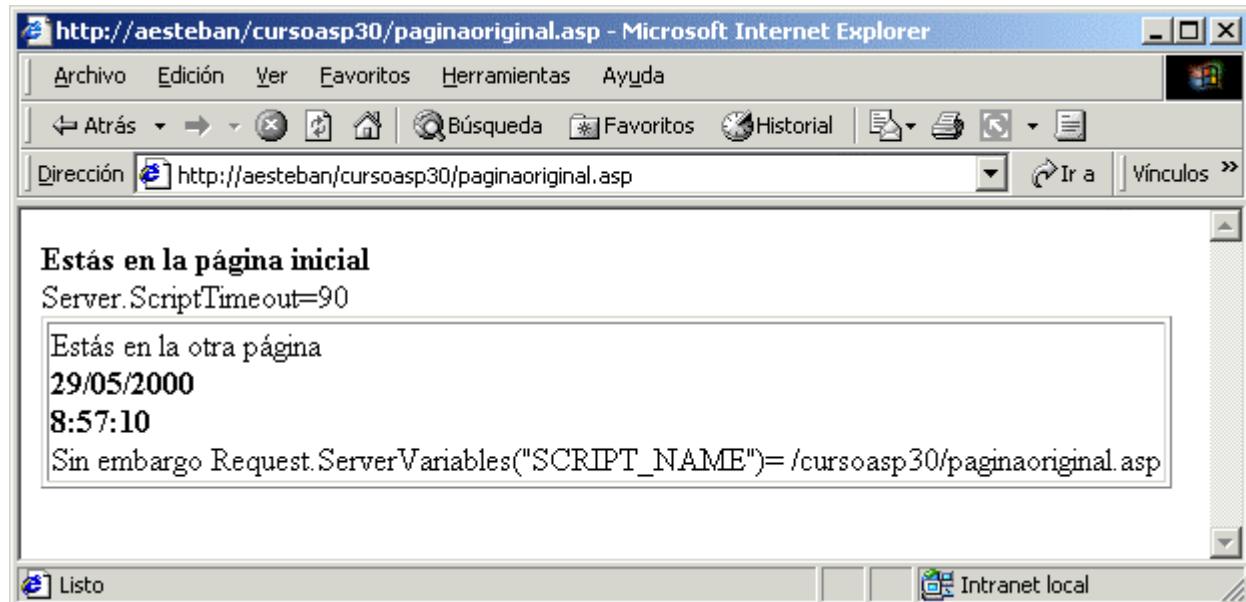


Figura 40. Utilizando el método Transfer

Como se puede comprobar la única diferencia es que la ejecución finaliza cuando termina la ejecución de la segunda página y no se vuelve a la inicial.

## GetLastError

Mediante el uso del método GetLastError podemos tener acceso a toda la información referente al último error que se ha producido en la página ASP actual. Pero es necesario aclarar que su utilización no es similar al tratamiento de errores que realizábamos con la sentencia On Error Resume Next y el objeto Err de VBScript, que preguntábamos por la propiedad Number del objeto Err para averiguar si se había producido algún error.

El método GetLastError se puede utilizar únicamente dentro de una página de error personalizada, es decir, cuando el error ya se ha producido y lo ha detectado el servidor Web.

Mediante el servidor Web Internet Information Server 5 podemos indicar las páginas de error personalizadas y es en estas páginas dónde podemos hacer uso de este método.

El método GetLastError devuelve un nuevo objeto del modelo de objetos de ASP llamado ASPError, son las propiedades de este nuevo objeto las que nos permiten acceder de forma detallada a toda la información referente al error que se ha producido. Este nuevo objeto lo trataremos con más detalle en el capítulo correspondiente, también veremos como configurar las páginas de error personalizadas desde IIS5.

Con este método terminamos la exposición del objeto Server, ya que no ofrece ninguna colección. Pero antes de terminar este capítulo vamos a ver un ejemplo que resume la utilización de todos los métodos del objeto Server.

Este ejemplo se trata de un formulario con diversos campos, cada uno de ellos se corresponde con un método del objeto Server. Para seleccionar el método que deseamos utilizar se debe seleccionar con el botón de radio correspondiente y acto seguido se debe pulsar el botón etiquetado como Ejecutar, que se trata de un campo de tipo submit del formulario.

En el caso de la ejecución del método Transfer y Execute se debe indicar la página que se desea ejecutar en cada caso, con los métodos URLEncode, URLPathEncode y HTMLEncode se debe indicar una cadena a la que se quiere aplicar la codificación correspondiente, para utilizar el método MapPath se debe indicar una ruta virtual para obtener la ruta física correspondiente, y en el caso del método CreateObject se debe indicar una cadena que represente un ProgID válido para el componente que se desea crear. El método GetLastError no aparece en este ejemplo, ya que como ya hemos comentado, este método se utiliza en las páginas de error personalizadas de IIS, en el capítulo dedicado al objeto integrado ASPError veremos su utilización.

Pasemos a ver el Código fuente 147.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Select Case Request.Form("metodo")
    Case "createobject"
        On Error Resume Next
        Set objeto=Server.CreateObject(Request.Form("progid"))
        On Error Goto 0
        If IsObject(objeto) Then%
            Se ha creado el objeto: <%=Request.Form("progid")%><br>
        <%Else%
            No se ha podido crear el objeto: <%=Request.Form("progid")%><br>
        <%End if
    Case "execute"%>
        Se pasa a ejecutar con Execute la página:
<%=Request.Form("paginaUno")%><br>
        <%Server.Execute(Request.Form("paginaUno"))%
    Case "transfer"%>
        Se pasa a ejecutar con Transfer la página:
<%=Request.Form("paginaDos")%><br>
        <%Server.Transfer(Request.Form("paginaDos"))%
    Case "mappath"%>
        Directorio virtual: <%=Request.Form("ruta")%>
        Directorio físico: <%=Server.MapPath(Request.Form("ruta"))%>
    <%Case "urlencode"%>
        La codificación URL del texto: <%=Request.Form("textoUno")%> es:
        <%=Server.URLEncode(Request.Form("textoUno"))%>
    <%Case "urlpathencode"%>
        La codificación de rutas URL del texto :<%=Request.Form("textoDos")%>
        es: <%=Server.URLPathEncode(Request.Form("textoDos"))%>
    <%Case "htmlencode"%>
        La codificación HTML del texto: <%=Request.Form("textoTres")%> es:
        <%=Server.HTMLEncode(Request.Form("textoTres"))%>
<%End Select%>
<br>
<form action="metodos.asp" method="post">
<b>Crea una instancia de un componente</b><br>
<input type="radio" name="metodo" value="createobject" checked>
Server.CreateObject("<input type='text' name='progid'>")<br>
<b>Ejecución de otra página ASP</b><br>
<input type="radio" name="metodo" value="execute">
Server.Execute("<input type='text' name='paginaUno'>")<br>
<input type="radio" name="metodo" value="transfer">
```

```

Server.Transfer("<input type='text' name='paginaDos'>") <br>
<b>Ruta física de una ruta virtual</b><br>
<input type='radio' name='metodo' value='mappath'>
Server.MapPath("<input type='text' name='ruta'>") <br>
<b>Aplicando codificaciones URL y HTML</b><br>
<input type='radio' name='metodo' value='urlencode'>
Server.URLEncode("<input type='text' name='textoUno'>") <br>
<input type='radio' name='metodo' value='urlpathencode'>
Server.URLPathEncode("<input type='text' name='textoDos'>") <br>
<input type='radio' name='metodo' value='htmlencode'>
Server.HTMLEncode("<input type='text' name='textoTres'>") <br>
<input type='submit' name='enviar' value='Ejecutar'>
</form>
</BODY>
</HTML>

```

Código fuente 147

Y el aspecto que presenta la página ASP se puede contemplar en la Figura 41.

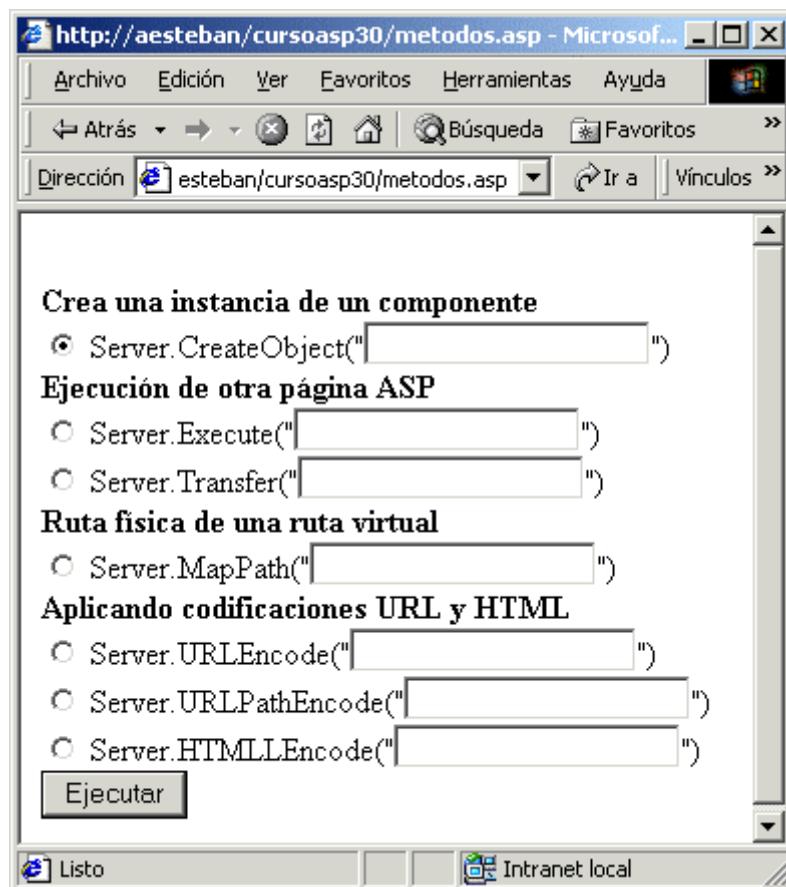


Figura 41. Métodos del objeto Server

Esta página se puede obtener en este [enlace](#).



# Modelo de objetos de ASP: el objeto Objectcontext

---

## Definición del objeto Objectcontext

Este objeto integrado se incluyó en la versión 2.0 de las páginas ASP. Este objeto se ofrece gracias a la integración que presentaban dentro de Windows NT el servidor Web IIS4 (Internet Information Server 4) con el servidor de transacciones MTS 2.0 (Microsoft Transaction Server).

A través del objeto integrado ObjectConext podremos tratar y gestionar las transacciones que se realicen en nuestras páginas ASP, pudiendo construir, por tanto, páginas ASP transaccionales. A lo largo de este capítulo comentaremos la utilización de este objeto.

Esta integración entre el servidor Web y el servidor de transacciones sigue existiendo, pero ahora el servidor Web se encuentra en su nueva versión IIS5 y el servidor de transacciones ya no se denomina MTS sino que se encuentra integrado en lo que en Windows 2000 se denomina Servicios de componentes, se puede decir que IIS y los Servicios de componentes funcionan conjuntamente para formar la arquitectura básica para la creación de aplicaciones Web y para coordinar el proceso de transacciones para las aplicaciones ASP transaccionales.

En el capítulo dedicado a la creación de componentes para ASP retomaremos los Servicios de componentes y los veremos en detalle.

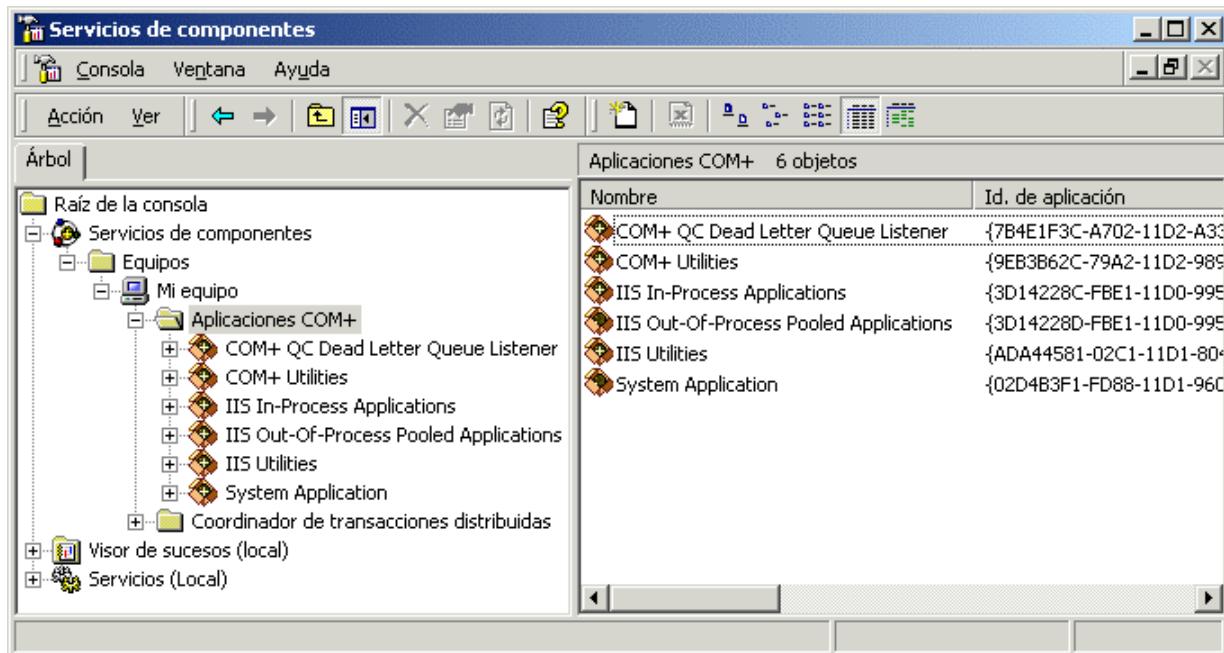


Figura 42. Servicios de componentes

A través del objeto ObjectContext podremos deshacer o llevar a cabo las transacciones gestionadas por los Servicios de componentes.

Recordemos que una transacción es una operación que se debe realizar de forma completa, es decir, si falla una parte de la transacción se deben deshacer todos los cambios. Un ejemplo típico de una transacción es la transferencia de una cuenta bancaria a otra, las operaciones de que implican restar el capital de una cuenta y añadirlo a otra se deben ejecutar dentro de una transacción, si falla alguna de las dos se debe devolver al sistema a su estado inicial. Las transacciones ofrecen una serie de características comunes conocidas como propiedades ACID (Atomicity, Consistency, Isolation, Durability).

Para indicar la naturaleza transaccional de una página ASP debemos incluir, como primera línea de la página, la directiva de procesamiento TRANSACTION.

```
<%@ TRANSACTION = valor %>
```

Código fuente 148

Donde el argumento valor define el comportamiento transaccional de la página ASP actual y puede tomar uno de los siguientes valores:

- Requires New: se requiere una nueva transacción, por lo que siempre se iniciará una nueva transacción para las páginas ASP con este valor. Esta configuración es recomendable para páginas que debe realizar transacciones pero que siempre deben estar separadas del resto.
- Required: se requiere una transacción, si no hay una iniciada, se iniciará una nueva. La página ASP siempre se ejecutará en una transacción ya sea iniciada por ella misma o bien aprovechando una transacción ya existente.

- Supported: soporta transacciones, pero no iniciará ninguna en el caso de que no exista. En algunos casos la página ASP se ejecutará en una transacción y en otros no, ya que sólo utiliza transacciones existentes.
- Not\_Supported: no soporta transacciones, es el valor por defecto. La página ASP nunca participará en ningún tipo de transacción.
- Disabled: se ignorarán los requerimientos transaccionales de la página ASP. La diferencia con el caso anterior, es que con Not\_Supported la página ASP siempre se ejecutará en un nuevo contexto, y con Disabled, si existe un contexto se utilizará el mismo.

A través del objeto `ObjectContext` podremos construir páginas ASP transaccionales, pero evidentemente, cuando se realicen operaciones que puedan tener una naturaleza transaccional, como puede ser el acceso a bases de datos. Además, una transacción no puede abarcar varias páginas ASP, a no ser que hagamos uso de los métodos `Transfer` o `Execute` del objeto `Server`, ya comentamos que con estos métodos de ejecución entre páginas se conserva el contexto de la página inicial, en este contexto se encuentra incluido las transacciones.

Hay que tener en cuenta, además, que la mayoría de las aplicaciones ASP sólo requieren hacer uso de un contexto transaccional en determinadas operaciones. Así por ejemplo, una aplicación de una Línea de Autobuses podría hacer uso de páginas ASP transaccionales para la venta de billetes y la reserva de asientos, dejando el resto de tareas fuera del contexto transaccional.

## Métodos del objeto Objectcontext

Para gestionar la transacción de una página ASP y poder indicar si se debe llevar a cabo la misma (`commit`) o bien se deben deshacer los cambios y volver al estado anterior (`rollback`) debemos hacer uso de los dos métodos del objeto `ObjectContext`.

En algún momento debemos determinar si las operaciones que debía realizar una página ASP se han realizado con éxito, y por tanto, se puede llevar a cabo la transacción correspondiente. Igualmente debemos disponer de algún método que nos permita anular una transacción en el caso de que falle alguna de las operaciones implicadas en la misma. Para ello disponemos de dos métodos del objeto `ObjectContext`.

- `SetComplete`: indica que se puede llevar a cabo la transacción correspondiente al realizarse con éxito las operaciones que debía llevar a cabo la página ASP. Es decir, una llamada a este método equivaldría a realizar un `commit` dentro de SQL Server o del gestor de bases de datos correspondiente.
- `SetAbort`: indica que algo ha fallado en las operaciones implicadas en la transacción, una llamada a este método cancela la transacción y deshace los cambios que se hayan podido realizar. Sería equivalente a realizar una llamada a `RollBack` en SQL Server.

En el caso de que finalice el procesamiento de la página ASP y no se haya realizado ninguna llamada a alguno de los dos métodos anteriores, se considerará que no se ha producido ningún problema en el procesamiento de la página y, por lo tanto, se llamará automáticamente al método `SetComplete`.

A la hora de crear páginas ASP transaccionales tenemos varias formas de abordar las transacciones:

1. Se crea una página ASP transaccional en la que las operaciones de acceso a datos se encuentran en la misma página ASP.

2. Se crea una página ASP transaccional en la que las operaciones de acceso a datos las realiza un componente o varios instanciados en la página ASP, pero son las secuencias de comandos de la página ASP quienes deciden el resultado de la transacción haciendo las llamadas a los métodos SetComplete o SetAbort cuando sea conveniente. Los componentes de deben registrar en Servicios de componentes, para que puedan participar de las transacciones.
3. Igual que el caso anterior pero es el componente quien decide el resultado de la transacción, la página ASP no llamaría a SetComplete o SetAbort. En este caso la página ASP se utiliza para agrupar a distintos componentes en una misma transacción ya que la página en realidad no decide sobre la transacción.

En el apartado anterior, en el que se muestran los eventos asociados al objeto ObjectContext, vamos a ver con ejemplos cada una de las situaciones anteriores. Ya que es mediante los eventos del objeto ObjectContext la forma de controlar si se ha finalizado la transacción con éxito o se ha cancelado la misma.

## Eventos del objeto Objectcontext

El objeto ObjectContext ofrece dos eventos que indican la forma en la que ha finalizado la transacción. Esto es bastante útil para obrar en consecuencia, es decir, si se ha abortado una transacción o se ha realizado con éxito se deberán hacer unas operaciones u otras. O si lo queremos podemos no utilizar estos eventos.

Los eventos del objeto ObjectContext se describen a continuación:

- OnTransactionCommit: este evento se dispara cuando se llevó a cabo de una transacción de manera satisfactoria con la llamada al método SetComplete.
- OnTransactionAbort: este evento se disparará tras la cancelación de una transacción con el método SetAbort.

El uso de estos dos eventos, nos permitirá realizar diferentes tratamientos según se haya finalizado la transacción, es decir, si se ha producido con éxito , o por el contrario se han producido errores y se ha tenido que deshacer.

Ahora es el momento adecuado para mostrar una serie de ejemplos que muestren las distintas formas de utilizar las transacciones desde las páginas ASP.

En este primer ejemplo se muestra una página ASP que requiere una transacción y que es ella misma la que realiza las operaciones y decide la finalización de la misma mediante a las llamadas a SetAbort y SetComplete. Primero se realiza un pedido y luego se actualiza la tabla de artículos para restarle las unidades que se han pedido, a continuación se realiza una comprobación para asegurar que siempre quedan como mínimo 200 unidades del artículo.

Como se puede ver en el Código fuente 149 es muy sencillo, se han puesto sentencias SQL de forma directa con constantes, ya que lo que nos interesa realmente es la utilización del objeto ObjectContext.

```
<%@Transaction=Required%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
```

```

<BODY>
<%'se realiza la conexión con la BD
Set Conex=Server.CreateObject("ADODB.Connection")
Conex.Open "Provider=SQLOLEDB.1;User ID=ejemplos;Password=ejemplos;" &
           "Initial Catalog=Ejemplos;Data Source=AESTEBAN"
'se realiza el pedido de artículos
Conex.Execute("INSERT INTO PEDIDOS (idpedido,idarticulo,idcliente,unidades) "&_
              "VALUES (1,2,3,100)")
'se decrementa el número de artículos disponibles
Conex.Execute("UPDATE ARTICULOS SET unidades=unidades-100 WHERE idarticulo=2")
'se recupera el número de artículos restantes
Set rs=Conex.Execute("Select unidades FROM ARTICULOS WHERE idarticulo=2")
unidades=rs("unidades")
rs.Close
Set rs=Nothing
If unidades<200 Then
   ObjectContext.SetAbort
Else
   ObjectContext.SetComplete
End if
Conex.Close
Set Conex=Nothing
'eventos que recogen la finalización de la transacción.
Sub OnTransactionCommit
    Response.Write "<b>Transacción realizada con éxito</b>"
End Sub
Sub OnTransactionAbort
    Response.Write "<b>La transacción se ha cancelado</b>"
End Sub%>
</BODY>
</HTML>

```

Código fuente 149

Para probar el ejemplo simplemente se deben ir modificando los valores que se insertan en la tabla de pedidos, y luego podemos consultar las tablas para comprobar realmente que las transacciones se cancelan y se realizan de forma adecuada.

En este nuevo ejemplo el acceso a datos lo realiza un componente que instanciamos dentro de nuestra página ASP, por lo que se incluirá también dentro de la transacción. Se supone que todos los componentes que se van a utilizar en las páginas ASP transaccionales se encuentran registrados adecuadamente en Servicios de componentes, la forma de hacerlo lo veremos en el tema dedicado a la construcción de componentes para ASP.

La página ASP sigue teniendo el control sobre la finalización de la transacción en este caso pregunta por el valor de un método para comprobar si la operación se ha realizado con éxito o se ha producido algún tipo de error, para así llamar al método SetComplete o SetAbort según corresponda en cada caso.

```

<%@Transaction=Required%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%'se instancia el componente
Set objeto=Server.CreateObject("Componente.Clase")
'se llama al método del componente y se comprueba el resultado
If objeto.AltaPedido(2,3,200) Then
    ObjectContext.SetComplete

```

```

Else
    ObjectContext.SetAbort
End if
Set objeto=Nothing
'eventos que recogen la finalización de la transacción.
Sub OnTransactionCommit
    Response.Write "<b>Transacción realizada con éxito</b>"
End Sub
Sub OnTransactionAbort
    Response.Write "<b>La transacción se ha cancelado</b>"
End Sub%>
</BODY>
</HTML>

```

Código fuente 150

En el último ejemplo, el encargado de indicar la forma en que finaliza la transacción es el componente que se instancia dentro de la página ASP. Por lo tanto es el propio componente el que llamará a los métodos SetComplete y SetAbort, la página se encarga de instanciar el componente, llamar a sus métodos y capturar con sus procedimientos para el tratamiento de eventos el resultado de la transacción. Veamos el Código fuente 151, que además será el más sencillo, ya que el componente hace gran parte del trabajo.

```

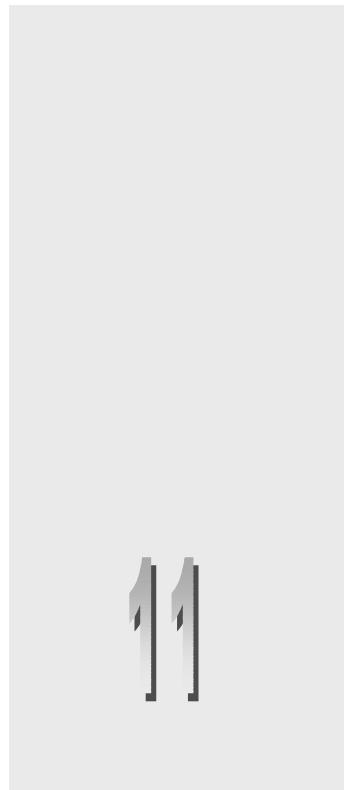
<%@Transaction=Required%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%'se instancia el componente
Set objeto=Server.CreateObject("Componente.Clase")
'se llama al método del componente
objeto.AltaPedido 5,3,200
'eventos que recogen la finalización de la transacción.
Sub OnTransactionCommit
    Response.Write "<b>Transacción realizada con éxito</b>"
End Sub
Sub OnTransactionAbort
    Response.Write "<b>La transacción se ha cancelado</b>"
End Sub%>
</BODY>
</HTML>

```

Código fuente 151

Utilizar una página ASP de esta forma puede ser muy interesante para agrupar distintos componentes dentro de una misma transacción.

El objeto ObjectContext y los Servicios de componentes los volveremos a tratar en el capítulo dedicado a la creación de nuestros propios componentes para ASP.



# Modelo de objetos de ASP: el objeto ASPError

---

## Definición del objeto ASPError

Este objeto es el nuevo objeto integrado que aparece en ASP 3.0. La función del objeto ASPError es la de ofrecer de forma detallada toda la información relativa al último error que se ha producido dentro de una aplicación ASP, describe el error que se ha producido, la naturaleza y fuente del mismo, y si es posible el código fuente que causó el error. Para ello el objeto ASPError consta de nueve propiedades de sólo lectura, y no ofrece ningún método.

Una referencia al objeto ASPError la obtenemos a través de un nuevo método del objeto Server, el método GetLastError que ya comentamos en el capítulo correspondiente, la sintaxis de este método la encontramos a continuación.

```
Set objASPErr=Server.GetLastError()
```

Ahora bien, el método GetLastError del objeto Server no se puede utilizar de forma indiscriminada en cualquier lugar de nuestro código para consultar si se ha producido un error, únicamente se puede utilizar de forma satisfactoria dentro de una página ASP de error personalizado. Las páginas de errores personalizados se tratan en detalle en el siguiente apartado del presente capítulo.

Si desactivamos el tratamiento de errores predeterminado que presenta ASP mediante la sentencia On Error Resume Next, al igual que se hacia anteriormente para consultar la propiedad Number del objeto Error, e intentamos lanzar el método GetLastError para obtener una referencia al objeto ASPErr, esta llamada fallará y el método no podrá acceder a los detalles del error que se ha producido.

## Propiedades del objeto ASPError

El objeto ASPError no ofrece ni eventos ni métodos únicamente ofrece propiedades y todas de lectura. Estas propiedades contienen la información detallada relativa al último error de ASP que se ha producido en una aplicación.

Las propiedades del objeto ASPError las consultaremos cuando se haya producido un error, cada una de ellas tiene un significado determinado que se pasa a exponer a continuación:

- ASPCode: un entero generado por IIS (Internet Information Server) y que representa un código de error de ASP.
- ASPDescription: una cadena que es una descripción detallada del error si está relacionado con ASP.
- Category: cadena que indica si se trata de un error interno de ASP, del lenguaje de secuencia de comandos o de un objeto.
- Column: entero que indica la posición de la columna del archivo ASP que generó el error.
- Description: cadena que contiene una breve descripción del error. Tiene el mismo significado que la propiedad Description del objeto Err.
- File: cadena que contiene el nombre del archivo ASP que se estaba procesando cuando se produjo el error.
- Line: entero que se corresponde con el número de línea del archivo ASP que generó el error.
- Number: entero que representa un código de error estándar de COM. Tiene el mismo significado que la propiedad Number del objeto Err.
- Source: cadena que contiene el código fuente real, si está disponible, de la línea que causó el error.

Como se puede observar podemos obtener una información bastante más precisa que con el objeto Err de VBScript.

Un ejemplo de utilización de las propiedades del objeto ASPError lo veremos en el siguiente apartado.

## Tratamiento de errores con el objeto ASPError

Anteriormente, en ASP 2.0, la forma más usual del tratamiento de errores era utilizar la sentencia de VBScript On Error Resume Next, esto causaba que el intérprete de secuencias de comandos ignorara los errores en tiempo de ejecución y continuara con la ejecución del código correspondiente. En nuestro código podíamos verificar la propiedad Number del objeto Err de VBScript, para comprobar si se había producido un error. También podíamos desactivar este tratamiento de errores con la sentencia On Error Goto 0. Todo esto lo vimos en el tema dedicado al lenguaje de secuencias de comandos.

Ahora con ASP 3.0 se encuentra disponible también el anterior tratamiento de errores, pero mediante la integración de ASP e IIS se ha añadido una forma nueva de tratar los errores. Esta nueva forma de tratamiento de errores se compone de dos partes: por un lado de la aparición del nuevo objeto

integrado de ASP, el objeto ASPError (que ya hemos comentado en este capítulo), y de la configuración de las páginas de error del servidor Web IIS (que será el siguiente punto a tratar).

Como ya hemos comentado, para obtener una referencia al objeto ASPError mediante el método GetLastError del objeto Server, debemos hacerlo en una página de error personalizada de IIS. Lo primero que vamos a hacer es comentar la utilidad de las páginas de error personalizadas y como podemos configurar nuestro servidor IIS para que las utilice.

Mediante la combinación del uso de las páginas de error personalizadas y del objeto ASPError, conseguimos un nuevo tratamiento de errores dentro de ASP 3.0 que anteriormente no se encontraba disponible.

Cuando el navegador de un cliente recibe una página de error con un mensaje, del estilo 404 Not Found, lo que está sucediendo es que el servidor Web le está enviando una página HTML que está asociada al tipo de error que se ha producido, es decir, existe una correspondencia entre código de error y páginas Web. Estas páginas es lo que llamamos páginas de error personalizadas.

Las páginas de error personalizadas no es una nueva característica del servidor Web IIS 5.0, sino que esta posibilidad ya se encontraba disponible en IIS 4.0. En el servidor IIS 4.0 podíamos especificar una página de error personalizada por cada distinto tipo de error del protocolo HTTP (HyperText Transfer Protocol) o error del servidor.

Particularmente, desde el punto de vista de este capítulo, el tipo de error que nos interesa configurar es el error 500:100 (Internal Server Error -ASP Error).

El error 500:100 es un error de servidor que se genera cuando se produce un error dentro de una página ASP. Se debe señalar que los errores de código 40x son siempre errores de cliente y los 50x de servidor. El código de error 500 es siempre generado por ASP, pero tiene varios subtipos que se corresponden con los errores de aplicación inválida (Invalid Application), el servidor se está apagando (Server Shutting Down), etc.

El subtipo de error específico 500:100 es el que se produce en una página ASP, ya sea por un error en el preproceso, un error al compilar las secuencias de comandos o bien un error en tiempo de ejecución.

El sitio Web predeterminado de Internet Information Server está configurado para utilizar como página de error del código 500:100, la URL /iishelp/common/500-100.asp. Cualquier sitio Web nuevo transferirá el procesamiento de errores de forma predeterminada al mensaje de error 500:100 en lugar de una página de tratamiento de errores.

Todas las páginas de errores predeterminadas que se ofrecen con IIS 5.0 se encuentran en el directorio WinNT\Help\iishelp\common. Nosotros vamos a crear una página ASP personalizada para el tratamiento de errores del tipo 500:100, será en esta página dónde tendremos acceso y utilizaremos el objeto ASPError.

Suponemos que ya tenemos creada la página de error ASP (en las siguientes líneas veremos el código de la misma y lo comentaremos), ahora debemos indicar al servidor IIS en qué aplicación ASP o sitio Web deseamos utilizar esta página de error personalizada. Para ello debemos acudir al Administrador de Servicios de Internet y configurar Internet Information Services.

En nuestro caso de ejemplo vamos a aplicar la página de error personalizada, ErrorASP.asp, a todo un directorio en concreto que se corresponde con una aplicación ASP.

Por lo tanto deberemos seleccionar el directorio deseado y pulsando con el botón derecho del ratón sobre el mismo seleccionaremos la opción Propiedades. Dentro de las propiedades del directorio seleccionaremos la pestaña de Errores personalizados. En esta pestaña aparecen todos los códigos de

error a los que podemos asignar una página de error personalizada, nosotros seleccionaremos el error 500:100 y pulsaremos el botón Modificar propiedades.

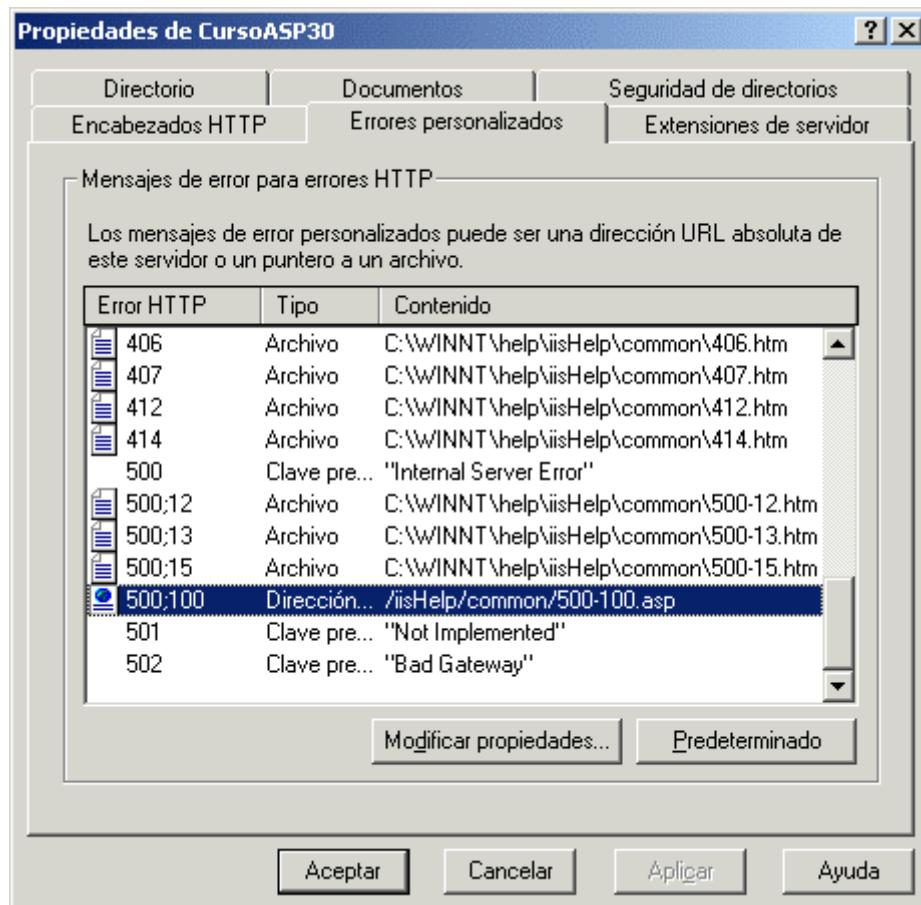


Figura 43. Errores personalizados

Ahora estamos ante el diálogo que nos permite modificar las propiedades del error 500:100. En la lista desplegable de tipo de mensaje existen tres opciones, en nuestro caso seleccionaremos el tipo de mensaje Dirección URL, los distintos tipos de mensaje se comentan a continuación:

- Dirección URL: si tenemos definida una página ASP para el tratamiento de errores deberemos indicar la ruta URL relativa a la misma.
- Archivo: si tenemos definidas páginas estáticas (HTML) para el tratamiento de errores debemos indicar su ruta física.
- Predeterminado: indicamos un texto que va a ser el mensaje que se muestre al cliente, en este caso no se utiliza ningún tipo de página de error personalizada.

Una vez seleccionado el tipo de mensaje Dirección URL escribiremos la URL al fichero ASP ErrorASP.asp, si esta página ASP la hemos guardado junto con el resto de páginas de error de IIS el contenido de esta caja de texto será /iishelp/common/ErrorASP.asp. Esto es así debido a que dentro del sitio Web predeterminado existe un directorio virtual llamado iishelp que apunta a la dirección física WinNT\Help\iishelp, que como ya sabemos contiene todos los mensajes de error personalizados de IIS.

Al seleccionar la opción de tipo de mensaje URL estamos obligados a utilizar una URL relativa al sitio Web en le que nos encontramos, es decir, no podemos indicarla de la siguiente forma <http://servidor/ruta/fichero.asp>.

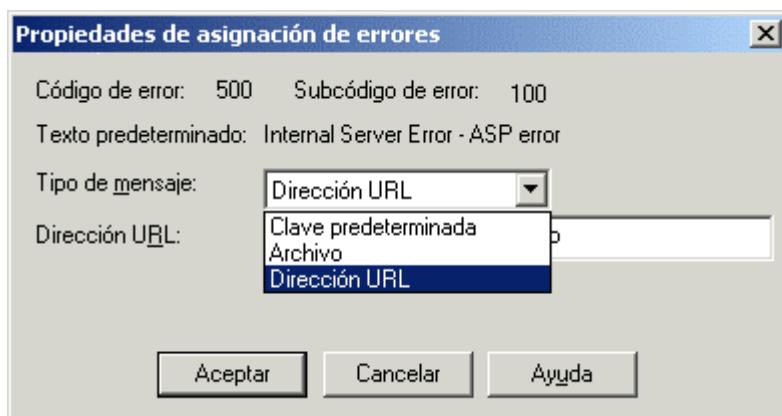


Figura 44. Tipos de mensaje

Una vez realizada esta configuración en nuestro sitio Web, cada vez que se produzca un error del tipo 500:100 se ejecutará la página ASP ERRORASP.ASP.

A continuación vamos a comentar el código que utilizaríamos para construir la página de error personalizada ERRORASP.ASP, que como ya hemos dicho es en esta página en la que utilizando el método GetLastError obtenemos un objeto ASPError que contiene toda la información detallada sobre el error que se ha producido.

Una vez que ya hemos configurado el servidor IIS, para que en nuestro directorio de nuestra aplicación ASP se utilice la página de error personalizada ErrorASP.asp, cuando se produzca un error en una página ASP, de forma completamente transparente al usuario el servidor Web invocará el método Transfer del objeto Server para que se pase a ejecutar la página de error.

El método Transfer es un nuevo método del objeto Server y permite realizar la redirección entre páginas en el servidor, en lugar de en el cliente, como ocurre con el método Redirect del objeto Response, si el lector necesita más información acerca de este método le remito al capítulo dedicado al objeto integrado Server.

Al invocarse la página ERRORASP.ASP mediante el método Transfer del objeto Server, tenemos acceso a todo el contexto en el que se encontraba la página ASP que ha producido el error.

Vamos ya a pasar a mostrar el contenido de la página de error personalizada ERRORASP.ASP. Lo primero que se puede observar en esta página es la llamada al método GetLastError del objeto Server para obtener el objeto ASPError con toda la información de error que se ha producido. El método GetLastError sólo está disponible antes de que la página ASP envíe contenidos al navegador del cliente.

Una vez que hemos obtenido el objeto ASPError, recuperamos los valores de sus propiedades para poder construir la cadena que enviaremos al cliente como parte de descripción del error que se ha producido.

Cuando se ha recopilado todos los datos del error, se lo mostramos al cliente. El código completo de la página ErrorASP.asp se encuentra disponible en el Código fuente 152. Y en el siguiente enlace se encuentra la página ASP [ERRORASP.ASP](#).

```

<%'Obtenemos el objeto ASPError
Set objASPError=Server.GetLastError()
'Se obtienen los valores de sus propiedades
strNumError=objASPError.Number
strCodigoASP=objASPError.ASPCode
If Len(strCodigoASP) Then
    strCodigoASP="'"&strCodigoASP&"'"
Else
    strCodigoASP=""
End if
strDescripcionError=objASPError.Description
strDescripcionASP=objASPError.ASPDescription
strCategoria=objASPError.Category
strNombreFichero=objASPError.File
strLinea=objASPError.Line
strColumna=objASPError.Column
If IsNumeric(strColumna) Then
    lngColumna=CLng(strColumna)
Else
    lngColumna=0
End if
strCodigoFuente=objASPError.Source
'Se crea la cadena del mensaje de error
strDetalles="Se ha producido un error ASP " & strCodigoASP & " a las " & Time&_
    " del día "&Date
If Len(strCategoria) Then strDetalles=strDetalles & vbCrLf& strCategoria
strDetalles=strDetalles & vbCrLf & "Número de error: " & strNumError _ 
    & "(OX" & Hex(strNumError) & ")" & vbCrLf
If Len(strNombreFichero) Then
    strDetalles=strDetalles & "Fichero: "&strNombreFichero
    If strLinea>"0" Then
        strDetalles=strDetalles & ", línea " & strLinea
        If lngColumna>0 Then
            strDetalles=strDetalles & ", columna " & lngColumna
            If Len(strCodigoFuente) Then
                'se marca la columna que ha producido el error
                strDetalles=strDetalles & vbCrLf & strCodigoFuente _ 
                    & vbCrLf & String (lngColumna-1,"-") & "^"
            End if
        End if
    End if
    strDetalles=strDetalles&vbCrLf
End if
strDetalles=strDetalles & strDescripcionError & vbCrLf
If Len(strDescripcionASP) Then
    strDetalles= strDetalles & "ASP informa: " & strDescripcionASP & vbCrLf
End if
'Se muestra el error al cliente%>
<html>
<head>
<title>Errores 500:100</title>
</head>
<body>
<PRE><%=Server.HTMLEncode(strDetalles)%></PRE>
</body>
</html>

```

Código fuente 152

Un ejemplo de ejecución de la página de error personalizado se puede ver en la Figura 45. Sería un error producido por la sentencia *Respose.Write "hola"* de una página ASP.

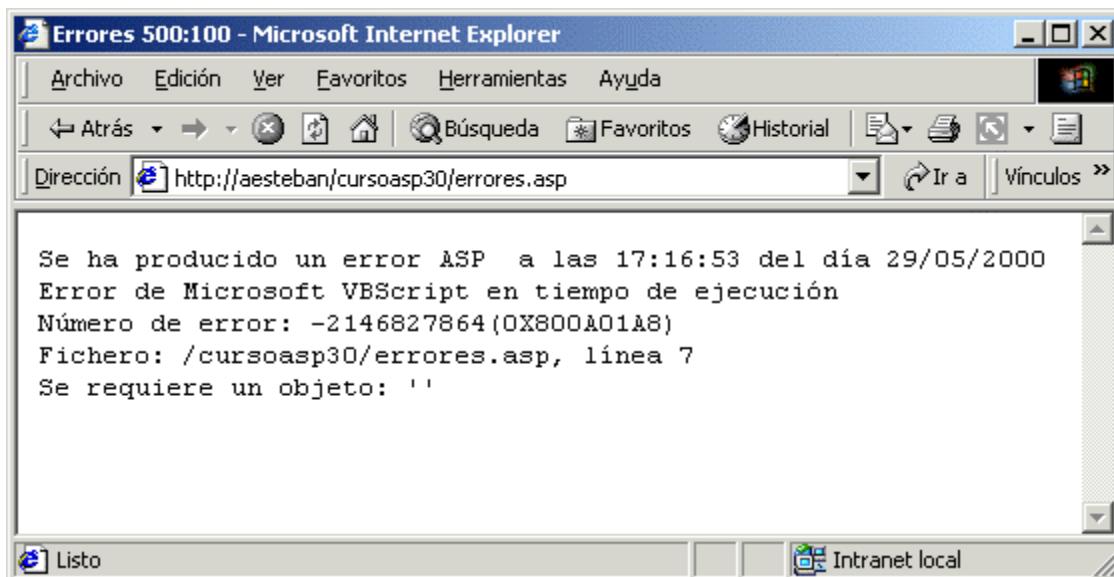


Figura 45. Ejemplo de error

Otro error producido por la sentencia `Response.Write "hola"` se puede ver en la Figura 46.

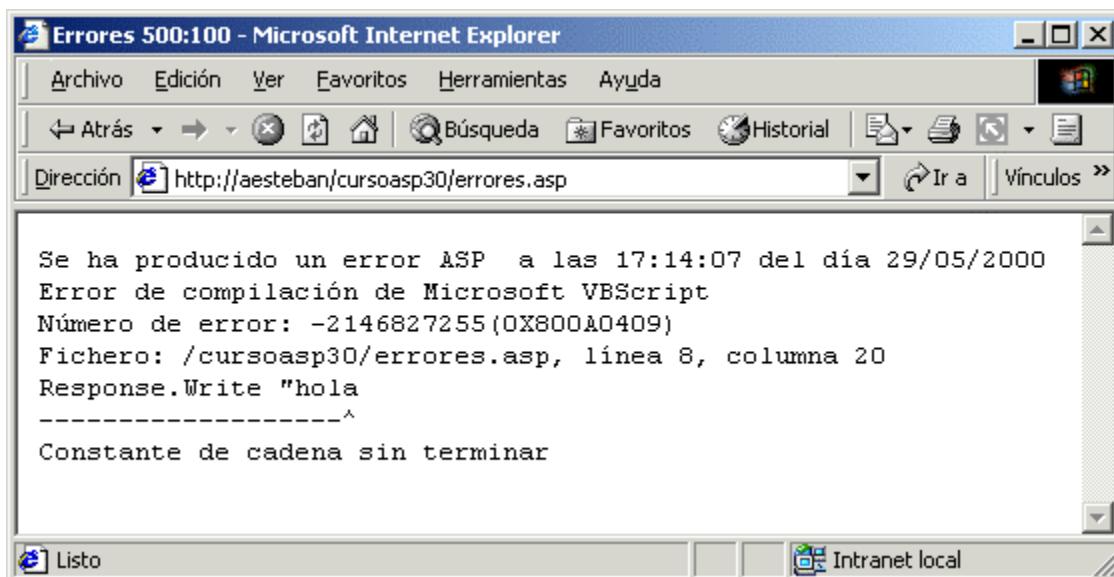


Figura 46. Otro ejemplo de error

Si el archivo de procesamiento de errores, en nuestro caso la página ErrorASP.asp, presenta algún error se informará del error como si no existiera ninguna página de error personalizado para los errores 500:100.



# Componentes de VBScript

---

## Componentes de VBScript

En el primer tema de este curso ya habíamos visto que podíamos distinguir varios tipos de objetos dentro de ASP: los que pertenecen al modelo de objeto de ASP llamados objetos integrados (Request, Response, Application, Session, etc.) que ya hemos tratado en temas anteriores, los componentes de servidor que se ofrecen con ASP (Content rotator, componente para funciones del explorador, Tools, registro de IIS, etc.), y por último los componentes desarrollados por nosotros mismos o por terceras partes y que podemos registrar en el servidor Web, que también veremos en su momento.

Pero existe un cuarto tipo o grupo de objetos que son los componentes de la librería del intérprete de comandos, es decir, componentes de VBScript. Estos componentes los encontramos junto con el intérprete de comandos de VBScript y son los que se comentan a continuación:

- Dictionary: este objeto ofrece un sistema de almacenamiento de valores, los valores se pueden acceder referenciados por su nombre. Lo podríamos comparar con las colecciones de Visual Basic. Es un array bidimensional que contiene valores referenciados por sus claves.
- FileSystemObject: permite acceder al sistema de archivos del servidor Web, nos permite manipular ficheros de texto, carpetas y unidades de disco. Los siguientes objetos que vamos a comentar dependen de este objeto y se utilizan en colaboración con el mismo.
- Drive: permite acceder a las unidades de disco del servidor Web. Un objeto Drive, al igual que todos los que vienen a continuación, se crea a partir de un objeto FileSystemObject.

- Folder: ofrece acceso a las carpetas del sistema de archivos de servidor Web. El objeto Folder nos va a permitir acceder a todas las propiedades de una carpeta determinada y, a través de su métodos, copiar, mover y borrar carpetas.
- File: permite acceder a los ficheros de una carpeta en el sistema de archivos del servidor Web. El objeto File nos va a permitir acceder a todas las propiedades de un fichero y, por medio de su métodos, copiar, mover o borrar un fichero determinado.
- TextStream: ofrece acceso al contenido de los ficheros de texto almacenados en el servidor Web. Mediante este objeto podremos modificar y crear ficheros de texto.

En este tema vamos a comentar cada uno de estos componentes con distintos ejemplos. Y en el próximo capítulo trataremos los componentes incluidos con ASP.

## El objeto Dictionary

Algunos lenguajes tales como Visual Basic ofrecen una estructura de datos denominada colección. Una colección es un tipo de array que ofrece una serie de funciones para el almacenamiento de elementos y su recuperación, y podemos acceder a sus elementos a través de una clave que será una cadena única o bien a través de su índice.

A lo largo de los distintos capítulos ya hemos utilizado colecciones en algunos de los objetos del modelo de objetos de ASP, por ejemplo la colección Form del objeto Request.

VBScript ofrece un objeto llamado Dictionary que ofrece las funcionalidades de una colección. Funciona como un array bidimensional contiendo una clave y los datos relacionados con la misma. Para manipular los elementos contenidos en un objeto Dictionary debemos utilizar las propiedades y métodos que ofrece este objeto, que veremos inmediatamente en este apartado.

Para instanciar un objeto Dictionary debemos utilizar el método CreateObject del objeto Server como indica el Código fuente 153.

```
Set objDic=Server.CreateObject ("Scripting.Dictionary")
```

Código fuente 153

O también podemos crear objetos Dictionary mediante la etiqueta de HTML <OBJECT>

```
<OBJECT RUNAT="SERVER" SCOPE="Session|Application|Page" ID="objDic"
PROGID="Scripting.Dictionary">
</OBJECT>
```

Código fuente 154

Debemos recordar que en una página ASP la etiqueta <OBJECT> únicamente se puede utilizar con el ámbito de página, es decir, SCOPE="PAGE". Si deseamos crear un objeto con la etiqueta <OBJECT> y que tenga ámbito de sesión o de aplicación debemos hacerlo en el fichero especial de la aplicación GLOBAL.ASA.

Las propiedades que ofrece el objeto Dictionary son las siguientes:

- CompareMode: propiedad de lectura/escritura que devuelve o establece el modo de comparación para las claves del objeto Dictionary. Existen dos tipos de comparación entre claves, binaria(0) , produce una comparación que distingue entre mayúsculas y minúsculas (case-sensitive) y textual (1), no distingue entre mayúsculas y minúsculas. Por defecto el modo de comparación es binario.
- Count: propiedad de sólo lectura que devuelve el número de elementos, pares clave/valor.
- Item(clave): devuelve o establece el valor de un elemento que se corresponde con la clave especificada por parámetro. Si al establecer el valor al elemento, la clave especificada no existe se creará esa nueva clave con el valor del elemento correspondiente. Algo bastante curioso es que si intentamos recuperar el valor de un elemento cuya clave no existe, se creará un nuevo elemento vacío con la clave indicada.
- Key: propiedad de sólo escritura que modifica el valor de una clave existente. SI la clave no existe se producirá un error.

Y los métodos de este objeto son:

- Add(clave,valor): añade un nuevo elemento con la clave y valor indicados. Si se intenta añadir un elemento nuevo con una clave que ya existe en el objeto Dictionary se producirá un error.
- Exists(clave): este método devuelve verdadero si la clave especificada existe. Si no existe devuelve falso.
- Items(): devuelve un array con los valores de todos los elementos contenidos dentro del objeto Dictionary.
- Keys: devuelve un array con las claves.
- Remove(clave): elimina el par clave/valor indicado por la clave. Si intentamos borrar un elemento cuya clave no existe se producirá un error.
- RemoveAll(): elimina todos los elementos del objeto Dictionary.

En el Código fuente 155 se va a mostrar un ejemplo en el que se utilizan los métodos y propiedades del objeto Dictionary.

La página ASP del ejemplo se trata de una página que permite listar los contenidos de un objeto Dictionary, añadir nuevos elementos, modificar valores y claves existentes, eliminar todos los elementos o elementos determinados.

El objeto Dictionary lo vamos a crear en el fichero GLOBAL.ASA para que tenga ámbito de sesión y en el evento Session\_OnStart se va a inicializar el objeto con unos elementos iniciales.

```
<SCRIPT LANGUAGE="VBScript" RUNAT="Server">
Sub Session_OnStart
    objDic.Add "uno", "Valor uno"
    objDic.Add "dos", "Valor dos"
    objDic.Add "tres", "Valor tres"
End Sub
</SCRIPT>
```

```
<object id="objDic" runat="server" scope="session" progid="Scripting.Dictionary">
</object>
```

Código fuente 155

La página se compone de un formulario con varios campos que permiten realizar las siguientes operaciones sobre el objeto Dictionary que hemos creado a nivel de sesión:

- Modificar una clave.
- Modificar el valor de un elemento.
- Añadir un nuevo elemento.
- Eliminar un elemento.
- Eliminar todos los elementos.

Cada formulario posee un campo de tipo submit, así según el botón que se pulse se realizará una operación u otra, se debe recordar que los cuando un campo posee varios campos submit sólo se envía la información referente al que se ha pulsado. Veamos el Código fuente 156.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%'Se comprueba que botón se ha pulsado para realizar la operación correspondiente
If Request.Form("modificaClave")<>"" Then
    'se verifica que no exista la nueva clave
    If Not objDic.Exists(Cstr(Request.Form("valorClave")))) Then
        objDic.Key(Cstr(Request.Form("claves1")))=Cstr(Request.Form("valorClave"))
    Else
        Response.Write "ERROR:El valor de la clave ya existe<br>"
    End if
ElseIf Request.Form("modificaValor")<>"" Then
    objDic.Item(Cstr(Request.Form("claves2")))=Request.Form("valorElemento")
ElseIf Request.Form("nuevoElemento")<>"" Then
    If Not objDic.Exists(Cstr(Request.Form("nuevaClave")))) Then
        objDic.Add Cstr(Request.Form("nuevaClave")),Request.Form("nuevoValor")
    Else
        Response.Write "ERROR:El valor de la clave ya existe<br>"
    End if
ElseIf Request.Form("eliminaElemento")<>"" Then
    objDic.Remove(Cstr(Request.Form("claves3")))
Elseif Request.Form("eliminaTodo")<>"" Then
    objDic.RemoveAll
End if%>
<b>Contenidos del objeto Dictionary</b><br>
<%'se obtienen el array de valores y el de elementos del objeto Dictionary
arrayElementos=objDic.Items()
arrayClaves=objDic.Keys()
For i=0 To objDic.Count-1
    Response.Write arrayClaves(i) &" = "& arrayElementos(i)&"<br>
Next%>
<hr>
<form action="dic.asp" method="post" id=form1 name=form1>
```

```

<b>Modificación de claves y valores</b><br>
<%If objDic.Count>0 Then%>
    <input type="submit" name="modificaClave" value="Modifica clave">
    Dictionary.Key("<select name='claves1'>
        <%'se muestra una lista con las claves
        For i=0 to Ubound(arrayClaves)%>
            <option value="<%=arrayClaves(i)%>"><%=arrayClaves(i)%>
        <%Next%>
    </select>")=<input type="text" name="valorClave" size="10">"<br>
    <input type="submit" name="modificaValor" value="Modifica valor">
    Dictionary.Item("<select name='claves2'>
        <%'se muestra una lista con las claves
        For i=0 to Ubound(arrayClaves)%>
            <option value="<%=arrayClaves(i)%>"><%=arrayClaves(i)%>
        <%Next%>
    </select>")=<input type="text" name="valorElemento" size="10">"<br>
<%Else%>
    El objeto no tiene elementos
<%End if%>
<hr>
<b>Añadiendo y borrando elementos</b><br>
<input type="submit" name="nuevoElemento" value="Añade elemento">
Dictionary.Add("<input type='text' name='nuevaClave' size='10'>",
    "<input type='text' name='nuevoValor' size='10'>")<br>
<%If objDic.Count>0 Then%>
    <input type="submit" name="eliminaElemento" value="Elimina elemento">
    Dictionary.Remove("<select name='claves3'>
        <%For i=0 to Ubound(arrayClaves)%>
            <option value="<%=arrayClaves(i)%>"><%=arrayClaves(i)%>
        <%Next%>
    </select>")
<%End if%>
<br>
<input type="submit" name="eliminaTodo" value="Elimina todo">
Dictionary.RemoveAll()
</form>
</BODY>
</HTML>

```

Código fuente 156

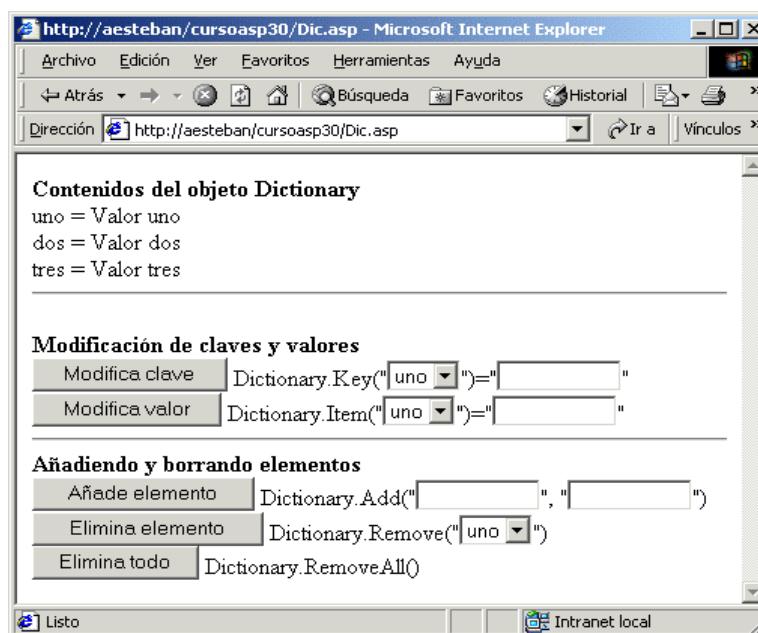


Figura 47. Utilizando el objeto Dictionary

Como se puede observar se hacen una serie de comprobaciones para evitar la repetición de claves y también para evitar realizar borrados o modificaciones cuando no existe ningún elemento. Al declarar el objeto Dictionary en el fichero GLOBAL.ASA a nivel de sesión accedemos directamente al mismo a través de su valor indicado en el parámetro PROGID. Un ejemplo de ejecución de la página ASP es el que muestra la Figura 47.

La página completa se puede obtener [aquí](#).

## Objeto FileSystemObject

La función de este componente es permitir el acceso al sistema de archivos del servidor. Gracias al objeto FileSystemObject podremos manipular ficheros de texto, directorios, ficheros genéricos y unidades de disco desde nuestras páginas ASP.

Para instanciar un objeto FileSystemObject lo haremos de la misma forma que hacíamos con el objeto Dictionary o con cualquier objeto.

```
Set objFSO=Server.CreateObject ("Scripting.FileSystemObject")
<OBJECT RUNAT="SERVER" SCOPE="PAGE" ID="objFSO"
PROGID="Scripting.FileSystemObject">
</OBJECT>
```

Código fuente 157

Al utilizar el objeto FileSystemObject puede ser interesante añadir una referencia a su librería de tipos para poder hacer uso de las constantes definidas en la misma.

```
<!--METADATA TYPE="typelib" FILE="C:\Winnt\system32\scrrun.dll"-->
```

Código fuente 158

Muy relacionados con este objeto se encuentran los objetos Drive, File, Folder y TextStream, ya que a partir de un objeto FileSystemObject vamos a obtener instancias de estos otros componentes de VBScript. Por ejemplo, a través del método GetDrive() del objeto FileSystemObject se obtiene un objeto Drive.

Además este objeto presenta una única propiedad llamada Drives, que es una colección de objetos Drive que representa todas las unidades existentes en el servidor Web, incluyendo las unidades de red.

Por el contrario el objeto FileSystemObject ofrece un gran número de métodos, a continuación vamos a comentar todos ellos y los vamos a agrupar atendiendo a las funciones que ofrecen los mismos.

## Métodos para trabajar con unidades de disco

- **DriveExists(unidadDisco):** devuelve verdadero si la unidad de disco indicada existe, devuelve falso en caso contrario. El parámetro que identifica a la unidad puede tener el formato c, c:, c\, o \\maquina\nombreCompartido, o el camino completo a un fichero o directorio.

- GetDrive(unidadDisco): devuelve un objeto Drive que se corresponde con la unidad especificada. A través del objeto Drive podremos obtener información detallada de la unidad a través de las propiedades que ofrece este objeto y que veremos en el apartado correspondiente.
- GetDriveName(unidadDisco): devuelve una cadena que representa al nombre de la unidad indicada.

Vamos a realizar un sencillo ejemplo que comprueba las unidades de disco que existen en el servidor, mostrando también su nombre completo. Para ello se utiliza un bucle con los códigos ANSI desde la A a la Z, si la unidad existe se muestra junto con su nombre completo. Veamos el Código fuente 159.

```
<%Set objFSO=Server.CreateObject ("Scripting.FileSystemObject")
For i=65 To 90 'códigos ANSI de la A a la Z
    letra=Chr(i)&":"
    If objFSO.DriveExists(letra) Then
        Response.Write "Unidad encontrada, nombre:
"&objFSO.GetDriveName(letra)&"<br>"
    End if
Next%>
```

Código fuente 159

Un ejemplo concreto de la ejecución de la página anterior se puede ver en la Figura 48.

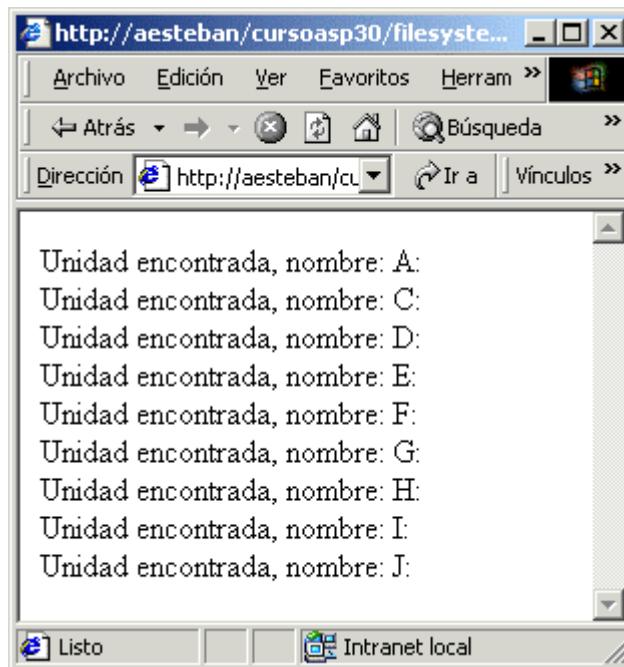


Figura 48. Utilizando los métodos relacionados con unidades

## Métodos para trabajar con carpetas

- BuildPath(ruta, nombre): devuelve una cadena que resulta de añadir el fichero o directorio indicado en nombre a la ruta especificada, añadiendo un carácter \ si es necesario.

- CopyFolder(origen, destino, sobreescribir): copia la carpeta o carpetas indicadas en origen, pudiéndose utilizar caracteres comodín, a la carpeta de destino especificada, incluyendo todo el contenido de la carpeta origen, es decir, se trata de una copia recursiva. En sobreescribir indicaremos True o False según proceda, el valor por defecto es True. Se producirá un error si la carpeta de destino existe y el valor de sobreescribir es False. Para poder copiar una carpeta, y en definitiva para poder realizar cualquier operación de escritura, es necesario que el usuario anónimo de Internet (IUSR\_NombreMaquina), posea los permisos necesarios para realizar tales operaciones. Esto es aplicable a todos los objetos del sistema de archivos. Tanto origen como destino deben ser rutas físicas. Si destino finaliza en \ se copiará el origen en la carpeta destino, en caso contrario se creará la carpeta destino.
- CreateFolder(nombreCarpeta): crea el directorio con la ruta y nombre especificados. Se produce un error si la carpeta ya existe.
- DeleteFolder(carpeta, forzar): elimina la carpeta especificada junto con todo su contenido, permitiendo utilizar caracteres comodín. Si el parámetro forzar tiene el valor True se eliminará la carpeta aunque sólo sea de lectura, el valor por defecto de este parámetro es False.
- FolderExists(carpeta): devuelve verdadero si la carpeta indicada existe, y falso en caso contrario.
- GetAbsolutePathName(ruta): devuelve la ruta completa correspondiente a la que indicamos por parámetro. Desde la ejecución de una página ASP este valor siempre será c:\winnt\system32.
- GetFolder(carpeta): devuelve un objeto Folder que se corresponde con la carpeta indicada. A través del objeto Folder se podrá obtener información detallada de la carpeta y también se podrá manipular a través de los métodos que ofrece el objeto Folder.
- GetParentFolderName(ruta): devuelve el nombre de la carpeta padre de la ruta especificada. No comprueba si la ruta existe.
- GetSpecialFolder(carpeta): devuelve un objeto Folder que se corresponde con la carpeta especial indicada, podemos indicarlo mediante las constantes definidas en la librería de tipos del objeto FileSystemObject, estas constantes son: WindowsFolder(0), SystemFolder(1) y TemporaryFolder(2).
- MoveFolder(origen, destino): mueve la carpeta o carpetas indicadas en el origen al destino especificado, en el parámetro origen se pueden utilizar comodines. Se producirá un error si la carpeta de destino ya existe. Si destino finaliza en \ se asume que es un directorio en el que se moverá el directorio de origen, en caso contrario se asumirá que es el nombre del nuevo directorio.

El Código fuente 160 es un pequeño resumen que muestra la utilización de este grupo de métodos del objeto FileSystemObject.

```
<%Set objFSO=Server.CreateObject ("Scripting.FileSystemObject")%>
BuildPath devuelve: <=%=objFSO.BuildPath("c:\winnt\temp", "fichero.txt")%><br>
<%objFSO.CopyFolder "c:\asfroot", "c:\temp"
Response.Write "carpeta copiada<br>">
objFSO.CreateFolder "c:\temp\Carpeta Nueva"
Response.Write "carpeta creada<br>"%>
FolderExists devuelve: <=%=objFSO.FolderExists("c:\temp\Carpeta Nueva")%><br>
<%objFSO.DeleteFolder "c:\temp\*.*"
```

```

Response.Write "carpeta borrada<br>"%
AbsolutePathName devuelve: <%=objFSO.GetAbsolutePathName("origen")%><br>
GetParentFolderName devuelve: <%=objFSO.GetParentFolderName("c:\tmp\Carpet
Nueva")%><br>
<%objFSO.MoveFolder "c:\tmp","c:\asfroot\Nueva Carpeta"
Response.Write "La carpeta se ha movido<br>"%

```

Código fuente 160

Y su resultado es el mostrado en la Figura 49.

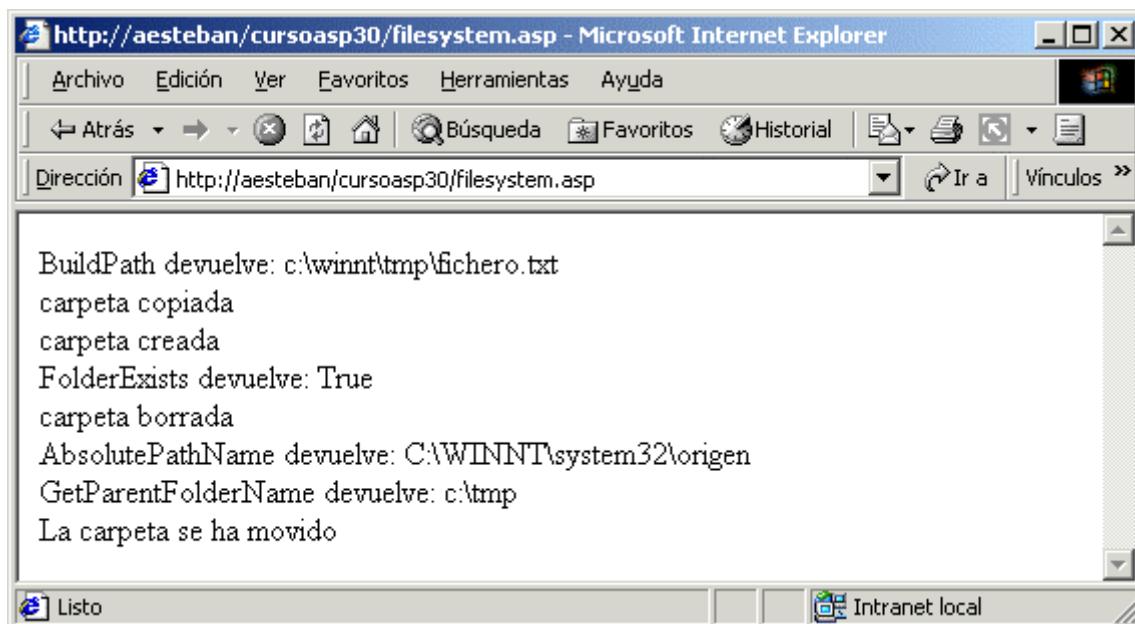


Figura 49. Utilizando los métodos relacionados con carpetas

## Métodos para trabajar con ficheros

- **CopyFile(origen, destino, sobreescibir):** copia el fichero o ficheros especificados en origen al destino indicado. Si destino finaliza con \ entonces se asume que destino es una carpeta, en caso contrario se asume que es el nombre del fichero de destino. Se producirá un error si el fichero de destino ya existe y sobreescibir tiene el valor falso.
- **CreateTextFile(nombreFichero, sobreescibir, unicode):** crea un nuevo fichero de texto con el nombre de fichero especificado y devuelve un objeto TextStream que representa al fichero de texto. Si el parámetro sobreescibir, que es opcional, tiene el valor True el fichero si existe se sobreescibirá, por defecto tiene valor False. El parámetro unicode también es opcional y si tiene el valor True se creará el contenido del fichero en texto en formato unicode, el valor por defecto de este parámetro es False.
- **DeleteFile(nombreFichero, forzar):** borra el fichero o ficheros especificados, ya que permite utilizar comodines. Para borrar ficheros de sólo lectura se debe indicar True en el parámetro forzar, su valor por defecto es False. Si el fichero no existe se producirá un error.
- **FileExists(nombreFichero):** devuelve verdadero si el fichero indicado existe, devolverá falso en caso contrario.

- GetBaseName(nombreFichero): devuelve únicamente el nombre del fichero indicado, sin ruta y sin extensión.
- GetExtensionName(nombreFichero): devuelve únicamente la extensión del fichero indicado.
- GetFile(nombreFichero): devuelve un objeto File que va a representar al fichero indicado en el parámetro. Mediante las propiedades y métodos de File podremos obtener información detallada del fichero y realizar operaciones con el.
- GetFileName(nombreFichero): devuelve el nombre del fichero indicado junto con su extensión.
- GetTempName(): devuelve un nombre de fichero generado aleatoriamente, que se puede utilizar para crear ficheros temporales.
- MoveFile(origen, destino): mueve el fichero o ficheros especificados en origen a la carpeta indicada en destino. Si destino finaliza con \ se asume que es el directorio en el que se va a mover el fichero, en caso contrario destino será la ruta y el nombre del nuevo fichero. Si el fichero de destino existe se producirá un error.
- OpenTextFile(nombreFichero, modo, crear, formato): crea un fichero de texto con el nombre indicado y devuelve un objeto TextStream que permite manipularlo, el resto de los parámetros son opcionales. El modo puede ser una de las siguientes constantes ForReading(1), es decir, de sólo lectura, ForWriting(2), para escritura eliminando todos los contenidos, y ForAppending(8), de escritura para añadir contenidos sin eliminar los existentes, por defecto el valor de este parámetro es ForReading. El fichero si se abre para escritura y no existe y el parámetro crear tiene el valor True se creará un nuevo fichero, por defecto el valor de este parámetro es False. Los formatos válidos del fichero son TristateFalse(0) para abrirlo como texto ASCII, TristateTrue(-1) para abrir el fichero como unicode y TristateUseDefault(-2) para abrir el fichero utilizando el formato por defecto del sistema. Por defecto el parámetro formato tiene el valor TristateFalse.

Al igual que hicimos con el grupo de métodos anteriores, vamos a mostrar un ejemplo resumen de utilización de todos estos métodos (Código fuente 161).

```
<%Set objFSO=Server.CreateObject ("Scripting.FileSystemObject")
objFSO.CopyFile "c:\Work\image5.gif","c:\basura\",True
Response.Write "Fichero copiado<br>
objFSO.CreateTextFile "c:\work\fichero.txt"
Response.Write "Fichero creado<br>
objFSO.DeleteFile "c:\tmp\*.gif"
Response.Write "Ficheros borrados<br>%
FileExists devuelve: <%=objFSO.FileExists("c:\Work\image5.gif")%><br>
GetBaseName devuelve: <%=objFSO.GetBaseName("c:\Work\image5.gif")%><br>
GetExtensionName devuelve: <%=objFSO.GetExtensionName("c:\Work\image5.gif")%><br>
GetFileName devuelve: <%=objFSO.GetFileName("c:\Work\image5.gif")%><br>
GetTempName devuelve: <%=objFSO.GetTempName()%><br>
<%objFSO.MoveFile "c:\basura\image5.gif","c:\tmp\"%
Response.Write "El fichero se ha movido<br>
objFSO.OpenTextFile "c:\work\fichero.txt"
Response.Write "Fichero abierto<br>%>
```

Código fuente 161

El resultado del código anterior se puede ver en la Figura 50.

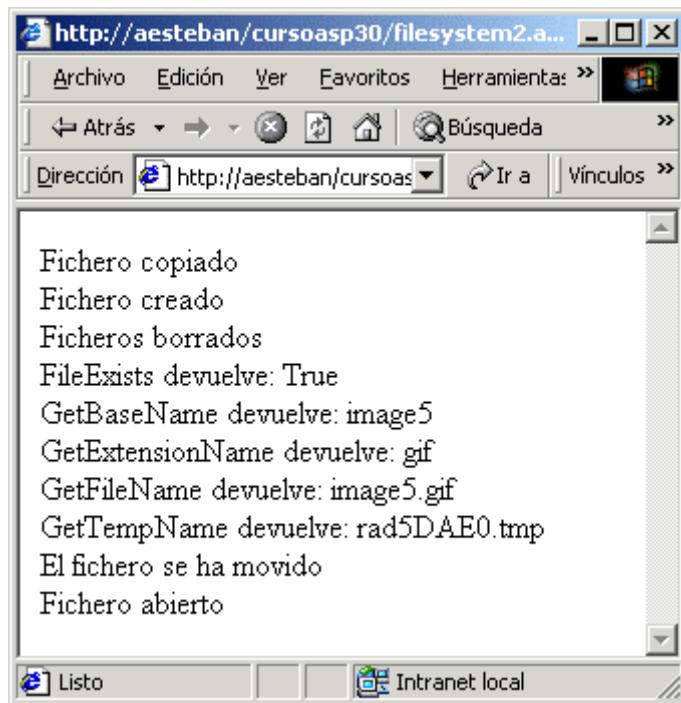


Figura 50. Utilizando los métodos relacionados con ficheros

Como se ha podido comprobar el objeto FileSystemObject es bastante complejo ya que ofrece una gran funcionalidad, ahora vamos a pasar a tratar cada uno de los objetos relacionados con FileSystemObject, es decir, los objetos Drive, que representa a una unidad, Folder, que representa a una carpeta, File, que representa a un fichero y TextStream que representa a un fichero de texto

## El objeto Drive

Ya hemos comentado en varios puntos la utilidad del objeto Drive. El objeto Drive nos proporciona el acceso a las propiedades de una unidad de disco o recurso compartido de la red. Estas propiedades, que son todas de lectura menos una, se muestran a continuación:

- AvailableSpace: devuelve la cantidad de espacio disponible en la unidad. La unidades se encuentran en bytes.
- DriveLetter: devuelve la letra de la unidad.
- DriveType: devuelve un entero que identifica el tipo de unidad, el tipo de unidad son constantes de la librería de tipos que incluimos con el fichero FileSystemObject. Estas constantes son Unknown(0) para unidades de tipo desconocido, Removable(0) unidades extraíbles, Fixed(2) unidades fijas, Remote(3) unidades de red, CDRom(4) unidades de discos compactos y RamDisk(5) unidades de bloques de RAM.
- FileSystem: devuelve una cadena con el tipo de sistema de archivos presente en la unidad.
- FreeSpace: devuelve la cantidad de espacio libre en la unidad, en bytes.
- IsReady: devuelve True si la unidad está lista.

- Path: devuelve la letra de la unidad seguida de dos puntos (C:).
- RootFolder: devuelve un objeto Folder que representa la carpeta raíz de la unidad.
- SerialNumber: el número de serie de la unidad.
- ShareName: nombre de la unidad cuando se trata de una unidad remota.
- TotalSize: tamaño total de la unidad en bytes.
- VolumeName: establece y devuelve el nombre del volumen de la unidad, es la única propiedad de lectura/escritura.

El objeto Drive no presenta métodos.

Debemos recordar que un objeto Drive lo obtenemos de la propiedad Drives del objeto FileSystemObject y a través del método GetDrive() de FileSystemObject, es decir, un objeto Drive siempre se debe crear a partir de un objeto FileSystemObject existente.

Veamos un ejemplo de utilización del objeto Drive. Se trata de recorrer todas las unidades existentes en el servidor Web y mostrar información acerca de ellas. Para ello utilizaremos la propiedad Drives del objeto FileSystemObject, ya que nos devuelve una colección de objetos Drive, que utilizaremos para mostrar las propiedades de las unidades. El código es el que muestra el Código fuente 162.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%'función para el tipo de unidad
Function TipoUnidad(tipo)
    Select Case tipo
        Case 0: TipoUnidad="Unknown"
        Case 1: TipoUnidad="Removable"
        Case 2: TipoUnidad="Fixed"
        Case 3: TipoUnidad="Remote"
        Case 4: TipoUnidad="CDRom"
        Case 5: TipoUnidad="RamDisk"
    End Select
End Function
Set objFSO = Server.CreateObject("Scripting.FileSystemObject")
'se obtiene la colección de unidades
Set colDrives = objFSO.Drives
'se recorren todas las unidades
For Each objDrive in colDrives
    Response.Write "Letra de unidad: <B>" & objDrive.DriveLetter & "</B><br>"
    Response.Write "Tipo de unidad: <B>" & TipoUnidad(objDrive.DriveType) &
"</B><br>"
    If objDrive.DriveType = 3 Then
        If objDrive.IsReady Then
            Response.Write "Unidad de red: <B>" & objDrive.ShareName &
"</B>"
        Else
            Response.Write "<B>La unidad de red no está disponible</B>"
        End If
    ElseIf objDrive.IsReady then
        Response.Write "Sistema de archivos: <B>" & objDrive.FileSystem &
"</B> &nbsp;&nbsp;"
```

```

        Response.Write "Número de serie: <B>" & objDrive.SerialNumber &
                      "</B>&nbsp;&nbsp;" 
        Response.Write "Nombre volumen: <B>" & objDrive.VolumeName &
                      "</B>&nbsp;&nbsp;" 
        Response.Write "Espacio disponible: <B>" &
                      FormatNumber(objDrive.AvailableSpace / 1024, 0) & "</B>
KB&nbsp;&nbsp;" 
        Response.Write "Espacio libre: <B>" &
                      FormatNumber(objDrive.FreeSpace / 1024, 0) & "</B>
KB&nbsp;&nbsp;" 
        Response.Write "Tamaño: <B>" & FormatNumber(objDrive.TotalSize / 1024,
0) &
                      "</B> KB"
    End if
    Response.Write "<hr>"
Next%>
</BODY>
</HTML>

```

Código fuente 162

En el caso de mi servidor Web obtengo el resultado siguiente.

Letra de unidad: **A**

Tipo de unidad: **Removable**

Sistema de archivos: **FAT** Número de serie: **1029443311** Nombre  
volumen: **PKBACK# 001** Espacio disponible: **516 KB** Espacio libre: **516 KB**  
Tamaño: **1.424 KB**

---

Letra de unidad: **C**

Tipo de unidad: **Fixed**

Sistema de archivos: **NTFS** Número de serie: **-459248756** Nombre  
volumen: **Mi disco** Espacio disponible: **1.073.744 KB** Espacio libre:  
**1.073.744 KB** Tamaño: **3.161.056 KB**

---

Letra de unidad: **D**

Tipo de unidad: **CDRom**

Sistema de archivos: **CDFS** Número de serie: **18442234** Nombre  
volumen: **Audio CD** Espacio disponible: **0 KB** Espacio libre: **0 KB**  
Tamaño: **0 KB**

---

Letra de unidad: **E**

Tipo de unidad: **Fixed**

Sistema de archivos: **FAT** Número de serie: **-195361032** Nombre  
volumen: **EIDOS** Espacio disponible: **646.400 KB** Espacio libre:  
**646.400 KB** Tamaño: **1.249.696 KB**

---

Letra de unidad: **F**

Tipo de unidad: **Remote**

**La unidad de red no está disponible**

---

Letra de unidad: **G**

Tipo de unidad: **Remote**

**La unidad de red no está disponible**

---

Letra de unidad: **H**  
Tipo de unidad: **Remote**  
**La unidad de red no está disponible**

---

Letra de unidad: **I**  
Tipo de unidad: **Remote**  
**La unidad de red no está disponible**

---

Letra de unidad: **J**  
Tipo de unidad: **Remote**  
**La unidad de red no está disponible**

---

Para obtener información detallada de las unidades A: o de CDRom es necesario que tengan un disco en su interior, para que la propiedad IsReady nos devuelva verdadero.

La página ASP del ejemplo se puede obtener [aquí](#).

## El objeto Folder

Este es otro objeto relacionado con el objeto FileSystemObject. El objeto Folder representa una carpeta del servidor Web, y ofrece una serie de propiedades y métodos para obtener información de las carpetas y manipularlas.

Un objeto Folder lo podemos obtener a partir de un objeto FileSystemObject mediante sus métodos GetFolder() o GetSpecialFolder(), también se puede obtener a partir de un objeto Drive mediante la propiedad RootFolder.

El objeto Folder ofrece una serie de propiedades para obtener información de una carpeta determinada. Todas las propiedades son de lectura menos un par de ellas. Pasamos a continuación a comentar cada una de ellas:

- Attributes: devuelve los atributos de la carpeta. Esta propiedad es de sólo lectura y de lectura/escritura, dependiendo de los atributos. Estos atributos son una combinación de los siguientes valores: Normal(0) no hay atributos asignados, ReadOnly(1) atributo de lectura/escritura, Hidden(2) atributo de lectura/escritura, System(4) atributo de lectura/escritura, Directory(16) atributo de sólo lectura, Archive(32) atributo de lectura/escritura, Alias(1024) atributo de sólo lectura y Compressed(2048) atributo de sólo lectura. Según lo descrito, por ejemplo, una carpeta oculta de sólo lectura tendrá el valor 3.
- DateCreated: devuelve la fecha y hora de creación de la carpeta.
- DateLastAccessed: devuelve la fecha y la hora de la última vez que se accedió a la carpeta.
- DateLastModified: devuelve la fecha y hora en la que se modificó la carpeta por última vez.
- Drive: devuelve la letra de la unidad en la que se encuentra la carpeta.
- Files: devuelve una colección Files que contiene objetos File que representan los ficheros contenidos en la carpeta.
- IsRootFolder: devuelve verdadero o falso para indicar si la carpeta es la carpeta raíz de la unidad a la que pertenece.

- Name: devuelve el nombre de la carpeta.
- ParentFolder: devuelve un objeto Folder que se corresponde con la carpeta padre de la carpeta actual.
- Path: devuelve la ruta relativa de la carpeta.
- ShortName: devuelve el nombre corto de la carpeta de acuerdo con la nomenclatura DOS 8.3.
- ShortPath: devuelve la ruta corta de la carpeta de acuerdo con la nomenclatura DOS 8.3.
- Size: devuelve el tamaño en bytes de todos los ficheros y subcarpetas contenidos en la carpeta.
- SubFolders: devuelve una colección Folders que contiene objetos Folder que representa cada una de las subcarpetas contenidas en la carpeta.
- Type: devuelve una descripción del tipo de carpeta, tal como "Papelera de reciclaje".

Vamos a ver un ejemplo que consiste en mostrar la información de todas las subcarpetas de la carpeta raíz de una unidad determinada del servidor Web. Para obtener el objeto Folder que representa a la carpeta raíz de la unidad tenemos que utilizar la propiedad RootFolder del objeto Drive. El objeto Drive lo obtenemos a partir del método GetDrive() del objeto FileSystemObject.

Para recorrer todas las carpetas utilizamos la colección Folders a través de la propiedad SubFolders del objeto Folder.

La página ofrece un formulario con una lista que muestra todos las unidades disponibles en el servidor, al seleccionar una de las unidades se muestran todas las carpetas incluidas en la carpeta raíz de la unidad.

Veamos el Código fuente 163.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<script language="javascript">
    function enviaUnidad(control)
    {
        document.formulario.submit();
    }
</script>
<BODY>
<%'Se crea un objeto FileSystemObject
Set objFSO = Server.CreateObject("Scripting.FileSystemObject")
'se obtiene la colección de unidades
Set colDrives = objFSO.Drives
'se crea la lista de unidades%>
<form name="formulario" action="folder.asp" method="post">
    <b>Unidades disponibles:</b><select name="unidad" onchange="enviaUnidad()">
        <option value="">
            <%For Each objDrive in colDrives
                If objDrive.IsReady Then%>
                    <option
value="<%=objDrive.DriveLetter%>:"><%=objDrive.DriveLetter%>:
                <%End if%>
            </option>
        <%Next%>
    </select>
</form>
<%'Se cierra el objeto FileSystemObject
Set objFSO = Nothing
%>
```

```

Next%>
</select>
</form>
<%If Request.Form("unidad")<>"" Then%>
    <big><b>Unidad seleccionada: <%=Request.Form("unidad")%></b></big><br>
    <%'Se obtiene la unidad
    Set objDrive = objFSO.GetDrive(Request.Form("unidad"))
    'Se obtiene la carpeta raíz de la unidad
    Set objRoot = objDrive.RootFolder
    'Se obtiene la colección Folders
    Set colFolders = objRoot.SubFolders
    'Se recorren todas las subcarpetas
    If colFolders.Count=0 Then%>
        <b>No hay carpetas en la unidad <%=Request.Form("unidad")%>
    <%Else
        For Each objFolder in colFolders
            'se muestra la información de cada carpeta
            Response.Write "Nombre: " & objFolder.Name & "  &nbsp;"&nbsp;
            Response.Write "Nombre corto: " & objFolder.ShortName & "  &nbsp;
&nbsp;"&nbsp;
            Response.Write "Tipo: " & objFolder.Type & "<BR>"
            Response.Write "Ruta: " & objFolder.Path & "  &nbsp;"&nbsp;
            Response.Write "Ruta corta: " & objFolder.ShortPath & "<BR>"
            Response.Write "Creado: " & objFolder.DateCreated & "  &nbsp;"&nbsp;
            Response.Write "Modificado: " & objFolder.DateLastModified & "<P>"&nbsp;
        Next
    End if
End if%>
</BODY>
</HTML>

```

Código fuente 163

Como se puede observar hacemos uso de una pequeña función en JavaScript que detecta cuando cambia de valor la lista de unidades, y en ese momento envía el formulario con la unidad seleccionada.

En la Figura 51 podemos ver un ejemplo de ejecución de esta página ASP.

- **Copy(destino, sobreescribir):** copia la carpeta y todo su contenido a la carpeta especificada en destino. Si el destino finaliza en \ se asume que es una carpeta en la que se va a copiar la carpeta de origen. En caso contrario se considera que es la ruta completa de una nueva carpeta que se creará con los contenidos de la carpeta de origen. Se producirá un error si el parámetro sobreescribir tiene el valor False y la carpeta de destino existe.
- **Delete(forzar):** elimina la carpeta y todos sus contenidos. Si el parámetro opcional forzar tiene el valor True se borrarán las carpetas y ficheros de sólo lectura. Su valor por defecto es False.
- **Move(destino):** mueve la carpeta y todos sus contenidos al destino especificado. Si el destino finaliza en \ se considera que es el directorio de destino al que se va a mover la carpeta, y en caso contrario se considera que es el nuevo nombre de la carpeta.
- **CreateTextFile(nombreFichero, sobreescribir, unicode):** crea un nuevo fichero de texto en la carpeta. El método tiene el mismo significado que el visto en objeto FileSystemObject.

La página se encuentra disponible en este [enlace](#).

Ahora vamos a ver los distintos métodos que ofrece el objeto Folder para la manipulación de carpetas.

En el Código fuente 164 se ofrece un ejemplo que resume la utilización de los métodos del objeto Folder.

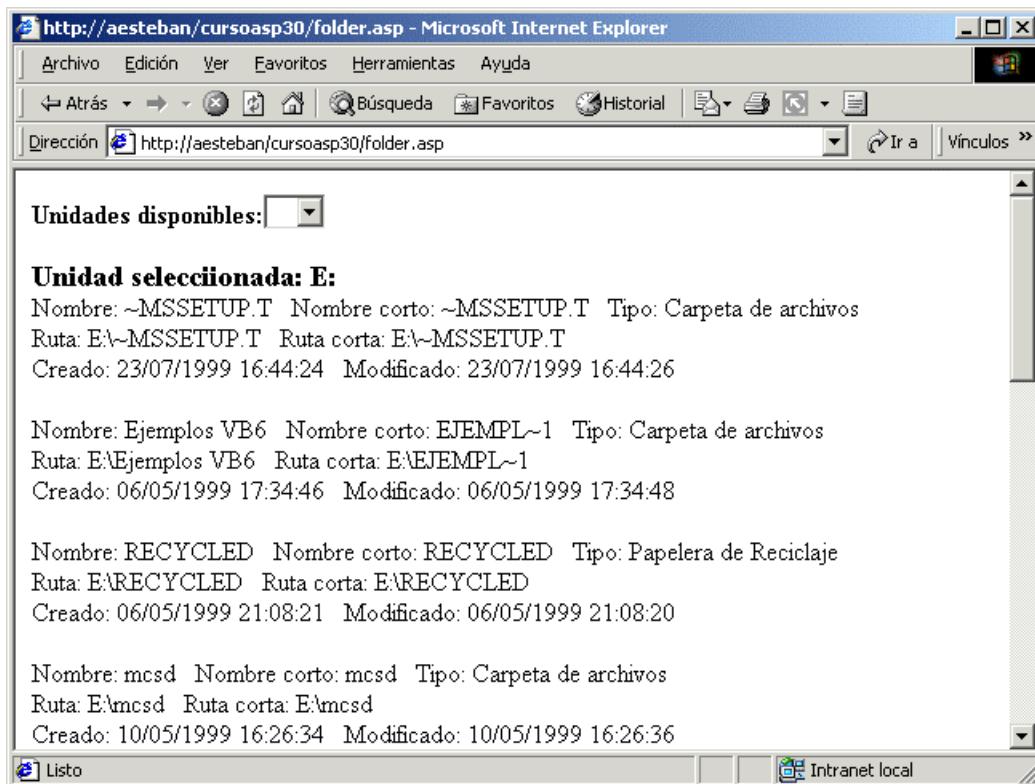


Figura 51. Utilizando las propiedades del objeto Folder

```
<%Set objFSO = Server.CreateObject("Scripting.FileSystemObject")
Set objFolder=objFSO.GetFolder("c:\tmp")
objFolder.Copy "c:\basura\" 
objFolder.Delete(true)
Set objFolder=objFSO.GetFolder("c:\basura")
objFolder.Move "c:\sp1\
objFolder.CreateTextFile "fichero.txt"%>
```

Código fuente 164

## El objeto File

El objeto File nos va a permitir acceder a todas las propiedades de un fichero y, por medio de su métodos, copiar, mover o borrar un fichero determinado.

Para obtener un objeto File tendremos que recurrir al método GetFile del objeto FileSystemObject que, como ya sabemos, devuelve un objeto File que corresponde al fichero de la ruta especificada, también podemos hacerlo a través de la propiedad Files del objeto Folder, que devuelve una colección de objetos File.

El Objeto File ofrece una serie de propiedades muy similares a las del objeto Folder, y al igual que ocurría con el objeto Folder, todas las propiedades son de lectura menos Attributes y Name. A continuación se comentan las propiedades.

- Attributes: devuelve los atributos del fichero. Esta propiedad es de sólo lectura y de lectura/escritura, dependiendo de los atributos. Estos atributos son una combinación de los siguientes valores: Normal(0) no hay atributos asignados, ReadOnly(1) atributo de lectura/escritura, Hidden(2) atributo de lectura/escritura, System(4) atributo de lectura/escritura, Directory(16) atributo de sólo lectura, Archive(32) atributo de lectura/escritura, Alias(1024) atributo de sólo lectura y Compressed(2048) atributo de sólo lectura.
- DateCreated: fecha y hora de creación del fichero.
- DateLastAccessed: fecha y hora de la última vez que se accedió al fichero.
- DateLastModified: fecha y hora de la última vez que se modificó el fichero.
- Drive: devuelve la letra que representa a la unidad en la que se encuentra el fichero.
- Name: modifica o devuelve el nombre del fichero.
- ParentFolder: devuelve el objeto Folder que representa a la carpeta padre del fichero.
- Path: ruta del fichero.
- ShortName: nombre corto del fichero en el formato DOS 8.3.
- ShortName: ruta corta del fichero en el formato DOS 8.3.
- Size: tamaño del fichero en bytes.
- Type: cadena que representa una descripción del tipo de fichero.

En el siguiente ejemplo se ilustra la utilización de las propiedades del objeto File. La página ASP consta de un formulario en el que se indica un directorio, y a partir del directorio indicado vamos a mostrar todos los ficheros que contiene junto con las propiedades de los mismos, para ello utilizaremos la propiedad Folders del objeto Folder. El objeto Folder se obtiene mediante el método GetFolder del objeto FileSystemObject. Veamos el Código fuente 165.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<form name="formulario" action="file.asp" method="post">
    <b>Carpeta:</b><input type="text" name="carpeta" value="" size="20">
    <input type="submit" name="enviar" value="Mostrar ficheros">
</form>
<%If Request.Form("carpeta")>< Then
    'Se crea un objeto FileSystemObject
    Set objFSO = Server.CreateObject("Scripting.FileSystemObject")
    If objFSO.FolderExists(Request.Form("carpeta")) Then
        Set objFolder = objFSO.GetFolder(Request.Form("carpeta"))
        'Se obtiene la colección Files
        Set colFiles = objFolder.Files
        If colFiles.Count=0 Then%>
            <b>No hay ficheros en la carpeta
<%=Request.Form("carpeta")%><br>
```

```

<%Else%>
    <b>Ficheros de la carpeta <%=Request.Form("carpeta")%></b><br>
    <%For Each objFile in colFiles
        'se muestra la información de cada fichero
        Response.Write "Nombre: " & objFile.Name & "&nbsp;
&nbsp;" 
        Response.Write "Nombre corto: " & objFile.ShortName &_
                        "&nbsp; &nbsp;" 
        Response.Write "Tamaño: " &
FormatNumber(objFile.Size/1024,0)&_
                        "KB <br>"
        Response.Write "Tipo: " & objFile.Type & "<BR>"
        Response.Write "Ruta: " & objFile.Path & "&nbsp; &nbsp;" 
        Response.Write "Ruta corta: " & objFile.ShortPath &
"<BR>" 
        Response.Write "Creado: " & objFile.DateCreated &_
                        "&nbsp; &nbsp;" 
        Response.Write "Modificado: " &
objFile.DateLastModified&"<P>" 
        Next
    End if
Else%>
    <b>La carpeta <%=Request.Form("carpeta")%> no existe</b>
<%End if
End if%>
</BODY>
</HTML>

```

Código fuente 165

En la figura 6 se puede observar un ejemplo de ejecución.

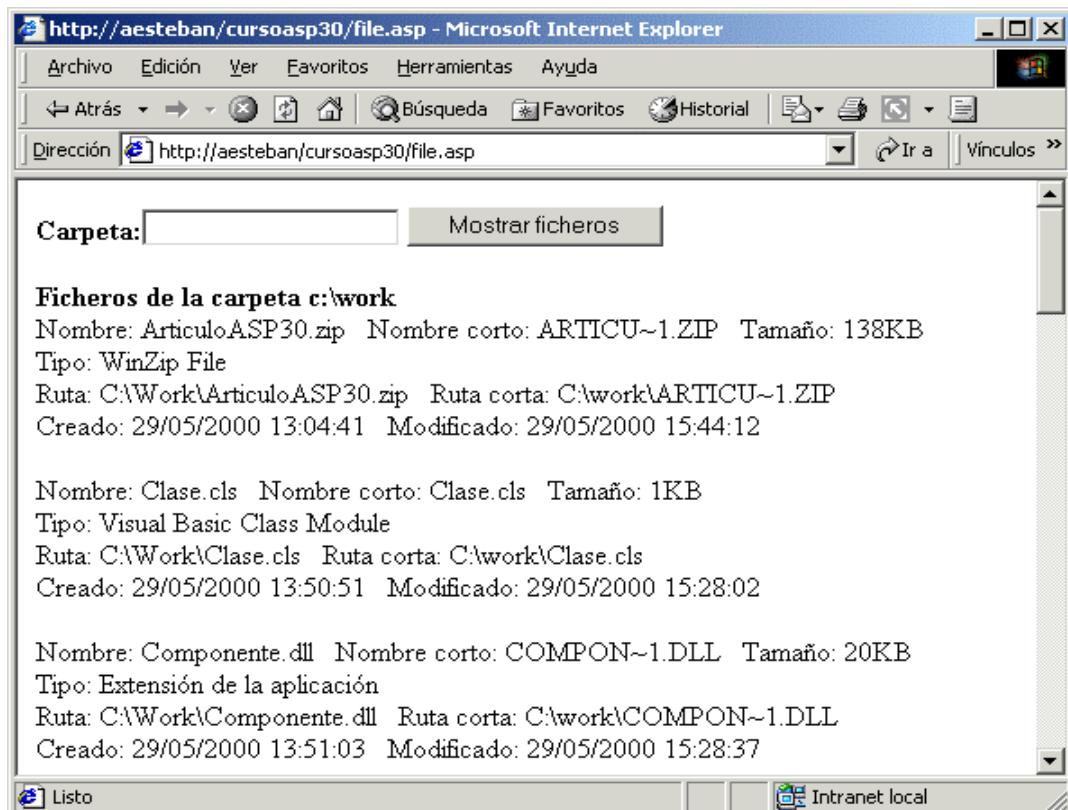


Figura 52. Mostrando información de ficheros

La página ASP está disponible en este [enlace](#).

El objeto File ofrece también los siguientes métodos:

- Copy(destino, sobreescibir): copia el fichero al destino especificado. Si destino finaliza en \ se considera que se trata de la carpeta a la que se va a copiar el fichero, en caso contrario destino será el nuevo nombre del fichero. Se producirá un error si el destino existe y el parámetro opcional sobreescibir tiene el valor False, su valor por defecto es True.
- Delete(forzar): elimina el fichero. Si forzar tiene el valor True se eliminará aunque sea de sólo lectura, por defecto el valor de este parámetro opcional es False.
- CreateTextFile(nombreFichero, sobreescibir, unicode): crea un fichero de texto con el nombre indicado. Tiene el mismo uso que el método del objeto FileSystemObject.
- OpenAsTextStream(modo, formato): abre el fichero devolviendo un objeto TextStream para manipularlo. El modo puede ser una de las siguientes constantes ForReading(1), es decir, de sólo lectura, ForWriting(2), para escritura eliminando todos los contenidos, y ForAppending(8), de escritura para añadir contenidos sin eliminar los existentes, por defecto el valor de este parámetro es ForReading. Los formatos válidos son TristateFalse(0) para abrirlo como texto ASCII, TristateTrue(-1) para abrir el fichero como unicode y TristateUseDefault(-2) para abrir el fichero utilizando el formato por defecto del sistema. Por defecto el parámetro formato tiene el valor TristateFalse.

## Objeto Textstream

Este es el último de los objetos pertenecientes al motor de secuencias de comandos de VBScript. Mediante el objeto TextStream se podrá leer y modificar la información de ficheros de texto almacenados en el servidor.

Para obtener un objeto TextStream que nos permita manipular el contenido de un fichero de texto, tenemos varias opciones:

- A partir de los métodos CreateTextFile() y OpenTextFile() del objeto FileSystemObject.
- A partir del método CreateTextFile() del objeto Folder.
- A partir de los métodos CreateTextFile() y OpenAsTextStream() del objeto File.

El método CreateTextFile permite crear un fichero de texto en el servidor Web, y tiene la siguiente sintaxis, que es igual en todos los objetos:

```
objeto.CreateTextFile(nombrefichero[, sobreescibir[, unicode]])
```

Donde objeto puede ser una instancia de un objeto FileSystemObject, Folder o File, nombrefichero es el nombre del fichero que se desea crear, sobreescibir tendrá los valores True o False según se desee o no sobreescibir un fichero existente y unicode también poseerá los valores True o False para indicar en qué formato se creará el fichero: en unicode o ASCII.

Al crear un fichero de texto, si se quiere manipular, se deberá guardar el resultado devuelto por el método CreateTextFile() en un objeto TextStream. En el siguiente ejemplo se muestra cómo se crea un fichero vacío llamado vacio.txt, como se puede observar el fichero se crea a partir de un objeto

FileSystemObject y se guarda dentro de un objeto TextStream para permitir la manipulación del fichero de texto, en este caso simplemente se utiliza para cerrar el fichero. En el ejemplo no se indica el parámetro unicode, por lo tanto se tomará el valor por defecto que es False, es decir se creará un fichero ASCII.

```
<%Set objFSO = Server.CreateObject("Scripting.FileSystemObject")
Set objTextStream=objFSO.CreateTextFile("c:\Tmp\vacio.txt",True)
objTextStream.Close%>
```

Código fuente 166

El método OpenTextFile permite abrir un fichero de texto en el servidor para lectura o escritura. Su sintaxis es la siguiente:

```
objeto.OpenTextFile(nombrefichero[, modo[, crear[, formato]]])
```

Donde objeto es una instancia de un objeto FileSystemObject, nombrefichero es le nombre de fichero a abrir. El modo puede ser una de las siguientes constantes ForReading(1), es decir, de sólo lectura, ForWriting(2), para escritura eliminando todos los contenidos, y ForAppending(8), de escritura para añadir contenidos sin eliminar los existentes, por defecto el valor de este parámetro es ForReading. El fichero si no existe y el parámetro crear tiene el valor True se creará un nuevo fichero, por defecto el valor de este parámetro es False. Los formatos válidos son TristateFalse(0) para abrirlo como texto ASCII, TristateTrue(-1) para abrir el fichero como unicode y TristateUseDefault(-2) para abrir el fichero utilizando el formato por defecto del sistema. Por defecto el parámetro formato tiene el valor TristateFalse.

Se debe señalar que la opción de crear el fichero tendrá el valor True sólo cuando el fichero se haya abierto para escritura, si está abierto para lectura se producirá un error.

Al igual que cuando se creaba un fichero, en el caso de abrirlo también se requiere un objeto TextStream para manipularlo. En el Código fuente 167 se muestra como se abriría un fichero para lectura y como se mostraría su contenido en el navegador.

```
<%@ Language=VBScript %>
<!--METADATA TYPE="typelib" FILE="C:\Winnt\system32\scrrun.dll"-->
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set objFSO = Server.CreateObject("Scripting.FileSystemObject")
Set objTextStream=objFSO.OpenTextFile("c:\tmp\texto.txt",ForReading,False)
While Not objTextStream.AtEndOfStream
    Response.write(objTextStream.ReadLine & "<br>")
Wend
objTextStream.Close%>
</BODY>
</HTML>
```

Código fuente 167

Como se puede observar el fichero TEXTO.TXT se abre para lectura. Mientras no se llegue al final del fichero se va leyendo cada fila con el método ReadLine del objeto TextStream, y mostrando su

contenido en el navegador mediante Response.Write. La propiedad que nos indica si hemos llegado al final del fichero es la propiedad AtEndOfStream del objeto TextStream. Por último se cierra el fichero con el método Close. De todas formas, un poco más adelante comentaremos todas las propiedades y métodos que nos ofrece el objeto TextStream.

No debemos olvidar realizar la referencia a la librería de tipos que contiene las constantes definidas para el acceso al sistema de ficheros, en nuestro caso hemos utilizado la constante ForReading.

En el Código fuente 168 se muestra también como se abriría un fichero para escritura, la función que realiza este código es el de copiar el contenido de un fichero en otro:

```
<%Set objFSO = Server.CreateObject("Scripting.FileSystemObject")
Set objTextStream=objFSO.OpenTextFile("c:\tmp\texto.txt",ForReading)
Set copia=objFSO.OpenTextFile("c:\tmp\copia.txt",ForWriting,True)
While Not objTextStream.AtEndOfStream
    copia.WriteLine(objTextStream.ReadLine)
Wend
objTextStream.Close
copia.Close%>
```

Código fuente 168

Si llamamos al método OpenTextFile indicándole que cree el fichero si no existe, en el caso de que no exista el efecto será el mismo si llamamos al método CreateTextFile.

Como se ha podido observar en los ejemplos anteriores, el objeto TextStream nos ofrece una serie de métodos y propiedades para tratar los ficheros de texto. Las propiedades que nos ofrece el objeto TextStream son las siguientes:

- AtEndOfStream: propiedad se sólo lectura que indica se ha llegado al final del fichero. Si el puntero de archivo se encuentra al final del fichero esta propiedad tendrá el valor True, y en caso contrario tendrá el valor False.
- AtEndOfLine: indica si se ha llegado al fin de línea dentro de un fichero. Esta propiedad de sólo lectura se puede utilizar cuando necesitemos leer una línea de un fichero carácter a carácter.
- Column: esta propiedad contiene el número de columna del carácter actual dentro del fichero empezando por 1.
- Line: número de línea actual del fichero empezando por 1.

Y los métodos que proporciona el objeto TextStream son:

- Close(): cierra un fichero abierto.
- Read(numeroCaracteres): lee y devuelve en una cadena el número de caracteres que se le pasa como parámetro.
- ReadAll(): lee un fichero completo y devuelve su contenido en una cadena. En ficheros de gran tamaño, utilizar el método ReadAll() consume muchos recursos de memoria. Se deberían utilizar otras técnicas para utilizar un fichero abierto para lectura, como leerlo línea a línea.

- ReadLine(): lee y devuelve una línea completa del fichero en una cadena.
- Skip(numeroCaracteres): salta el número de caracteres que se le indique al leer de un fichero.
- SkipLine(): salta a la siguiente línea en la lectura de un fichero. Saltar una línea significa descartar todos los caracteres de la línea actual y pasar a la siguiente.
- Write(cadena): Escribe en el fichero la cadena que se le indique.
- WriteLine(): escribe en un fichero la cadena que se le pase como parámetro y un carácter de nueva línea. Si no se le indica ninguna cadena escribirá solamente un salto de línea.
- WriteBlankLines(numeroLineas): escribe el número de saltos de línea que se le indique.

Se debe tener en cuenta que algunos de los métodos son apropiados cuando el fichero se ha abierto para lectura (Read, ReadAll, ReadLine, Skip, SkipLine) y otros para cuando se ha abierto para escritura (Write, WriteLine, WriteBlankLines). Si se utiliza un método de forma inadecuada, por ejemplo lanzar el método Skip cuando el fichero se ha abierto para escritura, se producirá un error.

También se debe indicar que para escribir ficheros debemos tener los permisos adecuados, en este caso el usuario anónimo de Internet debe tener permisos de escritura para poder escribir en un fichero, lo mismo ocurre con todos los objetos del sistema de archivos.

Se va a mostrar un ejemplo que presenta un formulario que permite manipular el contenido de un fichero desde un formulario. Las opciones que permite son modificar el contenido con los cambios realizados, añadir nuevo contenido al existente y eliminar el contenido del fichero. Estas operaciones se realizan a través de un área de texto que muestra el contenido actual del fichero, según el botón que pulsemos se realizará una acción u otra.

Con actualizar indicamos que el contenido del área de texto va a ser el del fichero, al pulsar añadir añadimos el texto del área de texto al contenido ya existente del fichero, y al borrar lo vaciamos el fichero de contenido. Veamos el Código fuente 169.

```
<%@LANGUAGE="VBScript"%>
<HTML>
<HEAD>
<!-- METADATA TYPE="typelib" FILE="c:\WinNT\System32\scrrun.dll" -->
</HEAD>
<BODY>
<%nombreFichero=Server.MapPath("fichero.txt")>
'crea una instancia de un objeto FileSystemObject
Set objFSO=Server.CreateObject("Scripting.FileSystemObject")
'se comprueba el botón que se ha pulsado
If Request.Form("actualizar")<>"" Then
    'se obtiene el nuevo texto y se divide en un array de líneas
    lineas=Split(Request.Form("texto"),vbCrLf)
    Set objTextStream=objFSO.OpenTextFile(nombreFichero,ForWriting)
    'se escribe el nuevo texto eliminado el existente
    For i=0 To UBound(lineas)
        objTextStream.WriteLine lineas(i)
    Next
    objTextStream.Close
ElseIf Len(Request.Form("anyadir")) Then
    lineas=Split(Request.Form("texto"),vbCrLf)
    Set objTextStream = objFSO.OpenTextFile(nombreFichero,ForAppending,False)
    For i=0 To UBound(lineas)
        objTextStream.WriteLine lineas(i)
```

```

Next
    objTextStream.Close
End If
If Len(Request.Form("borrar")) Then
    'se borra el contenido del fichero
    Set objTextStream=objFSO.OpenTextFile(nombreFichero,ForWriting)
    objTextStream.Close
End If
%>
<FORM ACTION="<%="Request.ServerVariables("SCRIPT_NAME")%>" METHOD="POST">
Los contenidos del fichero <B><%=nombreFichero%></B> son:<P>
<TEXTAREA NAME="texto" ROWS="10" COLS="50">
<%
'abrimos el fichero para mostrarlo en el área de texto
'si no existe se crea vacío
If Not objFSO.FileExists(nombreFichero) Then
    objFSO.CreateTextFile(nombreFichero)
End if
Set objTextStream=objFSO.OpenTextFile(nombreFichero,ForReading,True)
While Not objTextStream.AtEndOfStream
    Response.Write objTextStream.ReadLine&vbCrLf
Wend
objTextStream.Close%></TEXTAREA><br>
<INPUT TYPE="SUBMIT" NAME="actualizar" VALUE="Actualizar">
<INPUT TYPE="SUBMIT" NAME="anyadir" VALUE="Añadir">
<INPUT TYPE="SUBMIT" NAME="borrar" VALUE="Borrar">
</FORM>
</BODY>
</HTML>

```

Código fuente 169

Y en la Figura 53 se puede observar un ejemplo de su ejecución.

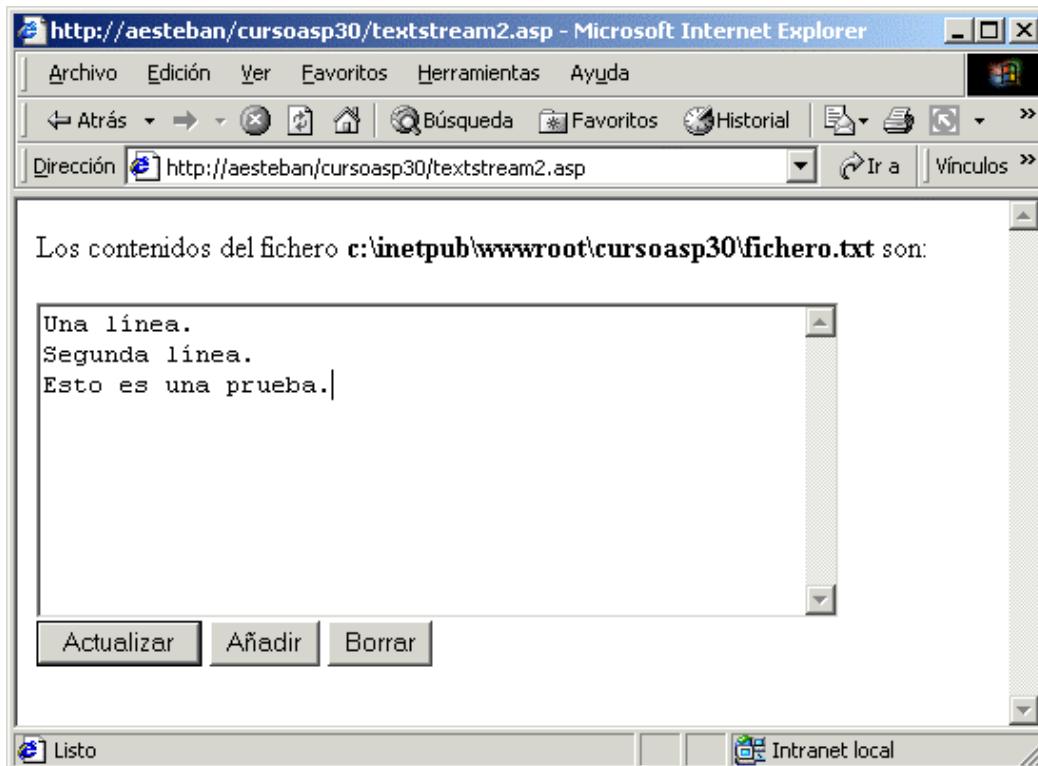


Figura 53. Utilizando el objeto TextStream

El código de este ejemplo se encuentra disponible [aquí](#).

Aquí finaliza este capítulo que trataba el conjunto de componentes de VBScript, en el siguiente capítulo se tratarán otros componentes que ya no pertenecen a VBScript pero si al servidor Web IIS5, y son los que se instalan junto con ASP de forma predeterminada.

Pero antes de dar por concluido este tema vamos a mostrar un esquema que muestra la relación entre los objetos FileSystemObject, Drive, Folder, File y TextStream, ya que como hemos comentado en varias ocasiones, un objeto de un tipo crea a partir de otro, y además de distintas maneras.

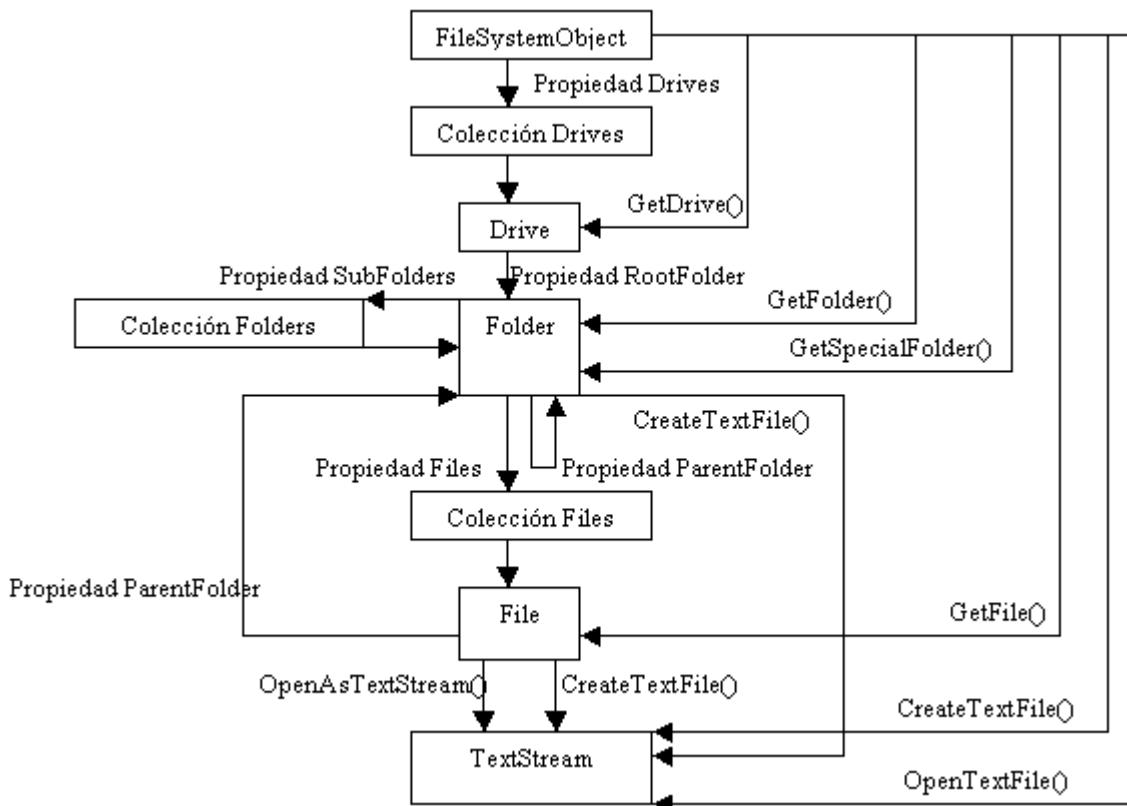


Figura 54. Relación entre los objetos del sistema de ficheros



# Componentes de servidor

---

## Introducción

En capítulos anteriores hemos visto los componentes integrados dentro de ASP y que forman parte del modelo de objetos de ASP y los componentes que ofrece el motor de secuencias de comandos de ASP contiene una serie de componentes que ofrecen una serie de funciones que liberan al programador de implementarlas, estas funciones pueden ser tareas comunes que se deban realizar dentro de un desarrollo en el entorno Web. Estos componentes están diseñados para ejecutarse en el servidor Web como parte de una aplicación basada en ASP.

Estos componentes se denominan componentes ActiveX Server o componentes de servidor, y anteriormente eran conocidos como servidores de automatización. Los componentes ActiveX Server se invocan desde páginas ASP (en nuestro caso), pero también pueden ser llamados desde otros orígenes, como por ejemplo aplicaciones ISAPI, desde otro componente de servidor o desde otros lenguajes OLE compatibles.

Los componentes de servidor son ficheros DLL que se ejecutan en el mismo espacio de direcciones de memoria que las páginas ASP y el servidor Web Internet Information Server.

Los componentes ActiveX Server que se encuentran incluidos dentro de ASP en su versión 3.0 y que se van a comentar en este tema son:

- Componente de acceso a bases de datos, ADO (ActiveX Data Objects). A través de la utilización de este componente se puede ofrecer acceso a bases de datos desde una página ASP, así por ejemplo, se puede mostrar el contenido de una tabla, permitir que los usuarios realicen consultas y otras operaciones sobre una base de datos.

- Componente Ad Rotator. Este componente permite mostrar una serie de imágenes alternativas con un vínculo a otra dirección desde la imagen presentada. Este componente se suele utilizar para mostrar diferentes anuncios de forma alternativa dentro de una página ASP.
- Componente Funciones del explorador. A través de este componentes podemos recuperar datos acerca del tipo de navegador del cliente y que capacidades o funciones tiene.
- Componente vínculo de contenidos. Facilita el desplazamiento lógico entre las diferentes páginas ASP de una aplicación ASP.
- Componente Content Rotator (rotador de contenidos). Este componente permite hacer rotaciones de cadenas de contenido HTML en una página.
- Componente Page Counter (contador de páginas). Permite llevar una cuenta del número de veces que se ha accedido a una página determinada dentro de nuestro sitio Web.
- Componente Counters. A través de este componente podremos almacenar, crear, incrementar y consultar cualquier contador.
- Componente MyInfo. Nos permite almacenar información personal que será ofrecida por el administrador del sitio Web.
- Componente Tools. Es el denominado componente de utilidades. Ofrece una serie de funciones diversas, como la generación de números aleatorios o la comprobación de la existencia de un fichero en el servidor.
- Componente Permission Checker. A través de este componente podremos determinar si a un usuario se le ha dado permisos para acceder a un fichero determinado.
- Componente Status. Este componente, de momento, únicamente está disponible para el servidor Personal Web Server en plataformas Macintosh, resulta extraño pero es así. Nos ofrece una información variada acerca del estado del servidor Web.
- Componente de registro de IIS. Mediante este componente tenemos acceso a la información y manipulación de los ficheros de registro (log) generados por el servidor Web IIS 5.0.

Para poder utilizar un componente de servidor dentro de una página ASP debemos instanciarlo, para ello deberemos utilizar el método `CreateObject` del objeto `Server` de la misma forma que lo hacíamos con los componentes de VBScript.

También se puede crear una instancia de un componente de servidor a través de la etiqueta `<OBJECT>` de HTML, como ya vimos en el tema anterior

La diferencia entre los dos mecanismos es desde el punto de vista de la eficiencia. Cuando se instancia un objeto con el método `Server.CreateObject` se crea inmediatamente el objeto, aunque no lo lleguemos a utilizar, sin embargo, si instanciamos el objeto con la etiqueta `<OBJECT>` el objeto sólo se creará en el momento que lo vayamos a utilizar por primera vez.

A lo largo de los siguientes apartados del presente capítulo vamos a ir comentando los distintos componentes de servidor que se ofrecen junto con la instalación de ASP.

## Componente Ad Rotator

Este componente de servidor se encarga de rotar una serie de anuncios (imágenes) dentro de una página ASP. Cada vez que un usuario vuelva a cargar la página el componente AdRotator mostrará una imagen con una URL asociada, atendiendo a un fichero especial denominado fichero de planificación. Un ejemplo de funcionamiento de este componente de servidor lo podemos ver en la Figura 55, la imagen central junto con su URL asociada se construyen mediante el Ad Rotator.

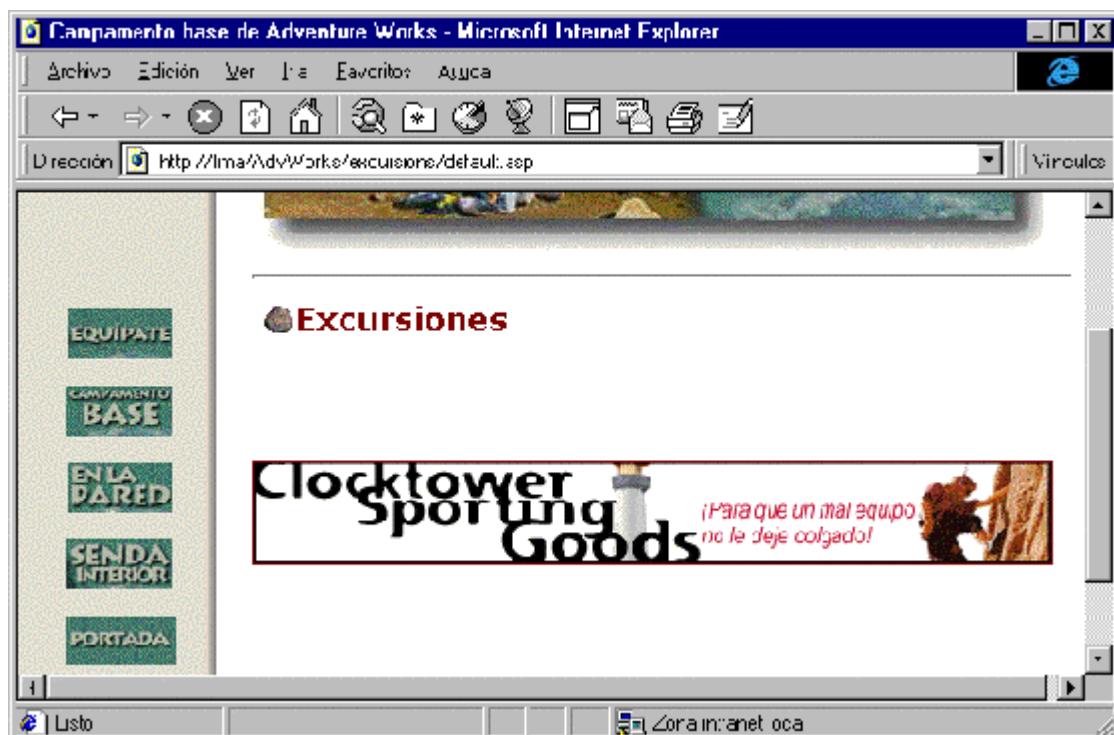


Figura 55. Ejemplo Componente AdRotator

Una de las características de HTML es que permite la creación de imágenes con vínculos. La etiqueta HTML que permite visualizar una imagen en una página es la etiqueta <IMG>. El componente AdRotator genera una etiqueta <IMG> a la que le asigna también una URL, al pulsar sobre la imagen se redireccionará al usuario hacia la URL asociada a esa imagen. Para llevar a cabo este proceso el componente AdRotator se sirve de dos ficheros: el fichero de redirección y el de planificación.

El fichero de redirección es una página ASP que debemos crear si queremos utilizar este componente. Este fichero suele incluir secuencias de comandos para analizar la cadena de consulta enviada por el componente AdRotator y para redirigir al usuario a la dirección URL asociada con el anuncio sobre el que se ha pulsado. En el Código fuente 170 el fichero de redirección simplemente redirige al usuario a la página del anunciante.

```
<%Response.Redirect(Request.QueryString("url"))%>
```

Código fuente 170

Dentro de la colección QueryString, en este caso, podemos encontrar dos variables: url, que indica la URL que tiene asociada la imagen e image, que indica el nombre de la imagen sobre la que se ha pulsado.

Como ya se había comentado con anterioridad, dentro del fichero de redireccionamiento también podemos realizar otras tareas además de redireccionar al navegador a una dirección determinada. Con el Código fuente 171 se consigue saber el número de usuarios que pulsan sobre un anuncio determinado. En este código aparece un par de componentes de VBScript ya conocidos por todos como son el objeto FileSystemObject y TextStream, que nos permiten leer y escribir en un fichero del servidor en el que vamos a almacenar el número de usuarios que han pulsado sobre el anuncio número dos.

```
<!--METADATA TYPE="typelib" FILE="C:\Winnt\system32\scrrun.dll"-->
<%'Cuenta el número de usuarios que han pulsado sobre un anuncio determinado
'el fichero de pulsaciones está en este caso en el mismo directorio
'que la página ASP de redirección
nombreFichero=Server.MapPath("pulsaciones.txt")
if Request.QueryString("image")="/cursoASP30/images/Anuncio2.gif" Then
    Set objFSO= Server.CreateObject("Scripting.FileSystemObject")
    If objFSO.FileExists(nombreFichero) Then
        'Se abre el fichero para su lectura
        Set objTextStream=objFSO.OpenTextFile(nombreFichero)
        pulsaciones=CIInt (objTextStream.ReadLine)
        pulsaciones=pulsaciones+1
        objTextStream.Close
    Else
        pulsaciones=1
    End If
    Set objTextStream=objFSO.OpenTextFile(nombreFichero,ForWriting,True)
    objTextStream.WriteLine pulsaciones
    objTextStream.Close
    Set objFSO=Nothing
End if
Response.Redirect(Request.QueryString("url"))%>
```

Código fuente 171

En la sentencia if...Then... se consulta la variable image de la colección QueryString del objeto Request, para comprobar si contiene el nombre del fichero de imagen del segundo anuncio. Si es el anuncio número dos, se lee del fichero el número actual de usuarios que lo han pulsado y se actualiza este número incrementándolo en uno.

Este es un ejemplo muy básico, por que si dos usuarios pulsan a la vez sobre el anuncio número dos uno de ellos no se contará. Para evitar este problema se debería guardar el objeto que representa al fichero dentro de una variable del objeto Application, y al ir a utilizar esta variable lanzar sobre el objeto Application los métodos Lock y UnLock.

El fichero de planificación contiene información utilizada por el componente AdRotator para administrar y mostrar las diversas imágenes de los anuncios. En él se especifican los detalles de los anuncios como puede ser tamaño, imagen a utilizar, URL asociada, frecuencia de aparición de cada imagen.

Este fichero tiene un formato determinado que se pasa a comentar a continuación. Se encuentra dividido en dos secciones mediante un asterisco (\*). La primera sección establece los parámetros aplicables a todas las imágenes de los anuncios existentes, y la segunda sección especifica la información de cada imagen individual y el porcentaje de tiempo de presentación de cada anuncio.

En la primera sección hay cuatro parámetros opcionales y globales, si no se especifica ninguno de estos parámetros la primera línea del archivo debería empezar por un asterisco. El formato general del fichero de planificación es el siguiente:

```
REDIRECT url
WIDTH ancho
HEIGHT alto
BORDER borde
*
Fichero que contiene la imagen
Url asociada
Texto alternativo
Número de impactos
```

La primera línea, REDIRECT url, indica el fichero (página ASP) que implementa el redireccionamiento, es decir, indica cual es el fichero de redireccionamiento. Las tres líneas siguientes, pertenecientes a la primera sección del fichero de planificación, hacen referencia al aspecto de la imagen: dimensiones y borde.

En la segunda sección tenemos la ubicación del fichero gráfico que representa la imagen a mostrar, a continuación está la URL a la que será enviado el navegador cuando el usuario pulse sobre la imagen, esta URL será la dirección del anunciante, por ejemplo; si el anuncio no tiene ningún vínculo asociado se deberá escribir un guión (-).

La siguiente línea contiene el texto alternativo que aparecerá cuando el navegador no esté habilitado para cargar imágenes o bien el texto que aparecerá cuando el cursor del ratón se sitúe sobre la imagen, el equivalente a la propiedad ALT de la etiqueta <IMG>. Por último se debe indicar el número relativo de apariciones del anuncio.

En el Código fuente 172 se va a mostrar y comentar un ejemplo de un fichero de planificación para un componente Ad Rotator:

```
Redirect redirecciona.asp
BORDER      0
*
/cursoASP30/images/Anuncio1.gif
http://www.almagesto.com
Campus Virtual Almagesto
40
/cursoASP30/images/Anuncio2.gif
http://www.eidos.es
Sitio Web del Grupo EIDOS
30
/cursoASP30/images/Anuncio3.gif
http://www.eidos.es/goritmo
Revista Algoritmo
30
```

Código fuente 172

En este ejemplo el fichero encargado de redireccionar al navegador a la dirección asociada a la imagen que se ha pulsado es la página ASP redirecciona.asp. Al no especificarse las dimensiones de la imagen, se tomarán los valores por defecto: WIDTH=440 y HEIGHT=60. Se han utilizado tres imágenes, cuya frecuencia de aparición será: para la primera un 40% de las veces que se cargue la página, para la segunda un 30% y para la tercera otro 30%.

El componente AdRotator además de estos dos ficheros especiales que ya se han comentado, posee también tres propiedades y un método. Sus propiedades son las siguientes:

- Border: indica el tamaño del borde que rodea a la imagen del anuncio, cumple la misma función que la línea BORDER del fichero de planificación.
- Clickable: indica si el anuncio es un hipervínculo o no, tendrá los valores True o False, respectivamente.
- TargetFrame: esta propiedad especifica el marco de destino en el que se debe cargar el vínculo. Esta propiedad realiza la misma función que el parámetro TARGET de una instrucción HTML de creación de vínculos.

El único método que posee este componente ActiveX de servidor es el método GetAdvertisement. A este método se le debe pasar como parámetro el nombre del fichero de planificación, y la función de este método es la de obtener las especificaciones, a partir del fichero de planificación, para el siguiente anuncio y le da formato de código HTML.

Para crear un componente AdRotator cuyo fichero de planificación se llama planificacion.txt se deberá escribir el Código fuente 173.

```
<%Set anuncios=Server.CreateObject("MSWC.AdRotator")%>
<%=anuncios.GetAdvertisement("planificacion.txt")%>
```

Código fuente 173

Con el método GetAdvertisement siempre debemos utilizar los delimitadores <%=%> o bien Response.Write, para enviar al navegador el código HTML encargado de mostrar la imagen correspondiente con su vínculo asociado.

Si tenemos en cuenta el código anterior y su fichero de planificación, una posible generación de código HTML por parte del método GetAdvertisement, podría ser la que muestra el Código fuente 174.

```
<A
HREF="redirecciona.asp?url=http://www.eidos.es&image=/cursoASP30/images/Anuncio2.gif"
>
<IMG SRC="/cursoASP30/images/Anuncio2.gif" ALT="Sitio Web del Grupo EIDOS"
WIDTH=440
HEIGHT=60 BORDER=0></A>
```

Código fuente 174

En el siguiente [enlace](#) se encuentra un ejemplo que utiliza este componente, en este ejemplo se encuentran también los ficheros de redirección y de planificación.

En la Figura 56 se puede observar como se relaciona el componente AdRotator con todos los ficheros que utiliza.

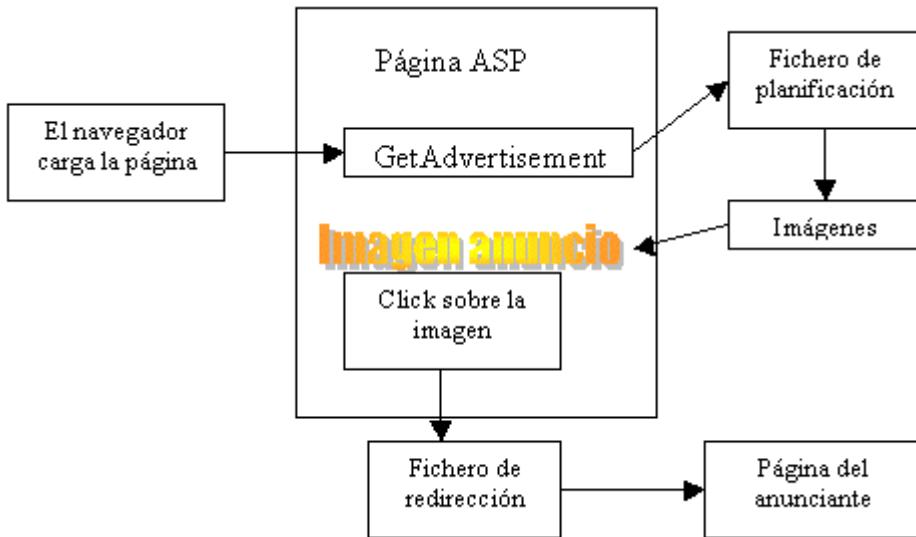


Figura 56. Utilización del componente AdRotator

## Componente funciones del navegador

El componente de funciones del navegador (Browser Capabilities) permite consultar las funciones que posee el navegador Web que ha cargado la página ASP. Cuando un navegador se conecta a un servidor Web, le envía automáticamente un encabezado HTTP llamado User Agent. Este encabezado es una cadena que identifica el explorador y su número de versión.

El componente encargado de comprobar las funciones del navegador compara el encabezado User Agent con las entradas de un fichero especial llamado Browscap.ini. Para crear una instancia de este componente deberemos escribir lo que muestra el Código fuente 175.

```
<%Set navegador=Server.CreateObject ("MSWC.BrowserType") %>
```

Código fuente 175

El fichero Browscap.ini es un fichero de texto que asigna las propiedades del navegador atendiendo al encabezado HTTP User Agent, este fichero se debe encontrar en el mismo directorio que la librería que contiene al componente, es decir, el fichero Browscap.dll.

Si se encuentra coincidencia entre el encabezado User Agent y una entrada del fichero Browscap.ini, el componente asumirá las propiedades del navegador de la lista que coincide con el encabezado User Agent.

Si el componente no encuentra una coincidencia entre el encabezado User Agent y una entrada del fichero BROWSCAP.INI, toma las propiedades predeterminadas del navegador Web. Pero si dentro de Browscap.ini no se han especificado estas propiedades predeterminadas se establecerá la cadena UNKNOWN para las propiedades del navegador.

Dentro del fichero BROWSCAP.INI se pueden realizar definiciones de propiedades para diferentes navegadores, también se pueden establecer valores predeterminados para utilizarlos si el navegador no pertenece a ninguna de las definiciones existentes.

Para cada definición de navegador se debe proporcionar un valor del encabezado User Agent y las propiedades y valores que se desea asociar a ese encabezado. El formato general de este fichero es el siguiente:

```
;comentarios  
[EncabezadoHTTPUserAgent]  
parent = DefiniciónExplorador  
propiedad1 = valor1  
  
propiedadN = valorN  
[Default Browser Capability Settings]  
PropiedadPredeterminada1 = ValorPredeterminado1  
  
PropiedadPredeterminadaN = ValorPredeterminadoN
```

Dentro de este fichero los comentarios se indicarán con punto y coma. El encabezado User Agent puede contener comodines utilizando el carácter \* para reemplazar cero o más caracteres. Por ejemplo si escribimos el siguiente valor para el encabezado User Agent:

```
[Mozilla/2.0 (compatible; MSIE 3.0;* Windows 95)]
```

Coincidiría con los siguientes encabezados User Agent:

```
[Mozilla/2.0 (compatible; MSIE 3.0; Windows 95)]  
[Mozilla/2.0 (compatible; MSIE 3.0; AK; Windows 95)]  
[Mozilla/2.0 (compatible; MSIE 3.0; SK; Windows 95)]  
[Mozilla/2.0 (compatible; MSIE 3.0; AOL; Windows 95)]
```

Si se dan varias coincidencias, el componente devolverá las propiedades de la primera definición coincidente.

Mediante la propiedad parent, podemos utilizar dentro de una definición de un navegador todos los valores de las propiedades de otro navegador cuyo nombre se indica en DefinicionExplorador. La definición del navegador actual heredará todas las propiedades declaradas en la definición del navegador principal. Esto es útil para definir propiedades de una nueva versión de un navegador, ya que las nuevas versiones suelen conservar la mayoría de las propiedades de la versión anterior. Estos valores de propiedades heredados se pueden sobreescribir estableciendo un nuevo valor para esa propiedad.

El número de propiedades a definir puede ser cualquiera, las únicas restricciones es que el nombre de la propiedad debe comenzar por un carácter alfabético y no puede tener más de 255 caracteres. Normalmente para cada tipo de navegador solamente se definirán las propiedades que vayan a ser utilizadas. El valor que se le asignará a la propiedad por defecto es una cadena, si se quiere asignar un entero se deberá utilizar el signo de número (#) y para especificar un valor booleano se utilizará TRUE o FALSE.

Con el valor de User Agent [Default Browser Capability Settings] se especifican las propiedades por defecto de un navegador cuya cabecera User Agent no ha coincidido con ninguna de las especificadas en el fichero BROWSCAP.INI.

Se debe realizar un mantenimiento del fichero Browscap.ini para tenerlo actualizado con las nuevas versiones de navegadores Web que van apareciendo. Existen un par de direcciones que ofrecen la última versión de este fichero: <http://www.cyscape.com/browscap> y <http://www.asptracker.com>.

En el Código fuente 176 se crea un componente defunciones del navegador y se muestran por pantalla algunas de las características del navegador del cliente.

```
<%Response.Write("La Cabecera USER_AGENT enviada por el navegador es: "&_
Request.ServerVariables("HTTP_USER_AGENT") &"<br>")
Set navegador=Server.CreateObject("MSWC.BrowserType")
if navegador.vbscript=TRUE Then Response.Write("Soporta VBScript<br>")
if navegador.cookies=TRUE Then Response.Write("Soporta Cookies<br>")
if navegador.beta=FALSE Then Response.Write("No es una beta<br>")
if navegador.javaapplets=TRUE Then Response.Write("Soporta Java<br>")
Response.Write("Plataforma: "&navegador.platform"<br>")
Response.Write("Navegador: "&navegador.browser&navegador.version"<br>")%>
```

Código fuente 176

Si en el servidor Web tenemos el siguiente fichero BROWSCAP.INI.

```
;Internet Explorer 4.0
[IE 4.0]
browser=IE
Version=4.0
cookies=TRUE
vbscript=TRUE
javascript=TRUE
javaapplets=TRUE
ActiveXControls=TRUE
Win16=False
beta=False

[Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)]
parent=IE 4.0
Version=4.01
platform=Win95
```

Código fuente 177

Si el navegador que se conecta es el Internet Explorer y envía la cabecera User Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows 95), el resultado de la ejecución del Código fuente 177 sería:

```
La Cabecera USER_AGENT enviada por el navegador es:
Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)
Soporta VBScript
Soporta Cookies
No es una beta
Soporta Java
Plataforma: Win95
Navegador: IE4.01
```

Pero si el navegador que se conecta envía otra cabecera diferente, el navegador no será reconocido y por lo tanto todas las propiedades se establecerán como UNKNOWN, y la salida que devolverá la ejecución del ejemplo anterior sería:

```
La Cabecera USER_AGENT enviada por el navegador es: Mozilla/4.03
[es] (Win95; I)
```

Plataforma: Unknown  
Navegador: UnknownUnknown

## Componente Nextlink

Este componente llamado también componente de vinculación de contenido, tiene como misión gestionar un conjunto de direcciones URL para poder utilizarlas dentro de páginas ASP de forma sencilla, permite tratar las páginas como las de un libro, es decir, ofrece la gestión de una navegación entre las diferentes páginas ASP de una forma sencilla.

Se debe tener en cuenta que el componente de Vinculación de Contenido trata un sitio Web como si fuera un libro que debe ser leído desde el principio al final. Sin embargo este no es el comportamiento natural de un sitio Web, ya que su distribución no es lineal, por lo tanto esta herramienta puede ser necesaria para guiar al usuario a través de un sitio Web, es decir es una herramienta lineal para un espacio no lineal.

El componente Nextlink hace referencia a un fichero especial llamado lista de vínculos, este fichero contiene una lista de las direcciones de las páginas vinculadas, este fichero se debe encontrar en el servidor Web.

El fichero de lista de vínculos contiene una línea de texto por cada dirección URL de la lista. Cada línea termina con un retorno de carro y cada elemento de una línea se separa mediante un tabulador. Los vínculos a las páginas que aparece en este fichero deben estar en el orden deseado, como si fuera un libro. El formato de este fichero es el siguiente:

URL - Página - Web	descripción
--------------------	-------------

No se admiten URLs absolutas, es decir, las que comienzan con "http:", "//" o "\\. Las URLs utilizadas deben ser relativas con el formato nombreFichero o directorio/nombreFichero.

Cuando se debe modificar el orden o el número de las páginas de contenido, sólo es necesario actualizar la lista de URLs que se encuentra dentro del fichero lista de vínculos y no se tiene que actualizar los vínculos de exploración de cada página. Una ejemplo de un fichero de lista de vínculos podría ser el siguiente:

intro.asp	Introducción a ASP
cap1.asp	VBScript
cap2.asp	Objetos Integrados
cap3.asp	Componentes ActiveX Server

Para tener acceso a las URLs que aparecen especificadas dentro del fichero lista de vínculos, el objeto Nextlink ofrece una serie de métodos:

- GetListCount(fichero): devuelve el número de vínculos del fichero lista de vínculos.
- GetNextURL(fichero): obtiene la siguiente URL teniendo en cuenta la página actual. Si la página actual no está especificada en dicho fichero, GetNextURL devuelve la dirección URL de la última página de la lista.
- GetPreviousDescription(fichero): este método devuelve la descripción del elemento anterior del fichero lista de vínculos, atendiendo a la página actual. Si la página actual no está en el fichero, se devolverá la descripción del primer elemento de la lista.

- GetListIndex: devuelve el índice que corresponde a la página actual dentro del fichero lista de vínculos. La primera página tendrá el índice 1. Si la página actual no existe dentro de la lista de vínculos se devolverá 0.
- GetNthDescription(fichero, indice): devuelve la descripción correspondiente al elemento de la lista de vínculos cuyo índice coincide con el índice que se le pasa como parámetro a este método.
- GetPreviousURL(fichero): obtiene la URL anterior teniendo en cuenta la página actual. Si la página actual no está especificada en dicho fichero, GetPreviousURL devuelve la dirección URL de la primera página de la lista de vínculos.
- GetNextDescription(fichero): este método devuelve la descripción del siguiente elemento del fichero lista de vínculos, atendiendo a la página actual. Si la página actual no está en el fichero, se devolverá la descripción del último elemento de la lista de vínculos.
- GetNthURL(fichero, indice): devuelve la dirección URL correspondiente al elemento de la lista de vínculos cuyo índice coincide con el índice que se le pasa como parámetro a este método.

Todos estos métodos poseen un parámetro común que indica la ubicación del fichero lista de vínculos con el que debe trabajar un componente de vinculación de contenido.

En el Código fuente 178 se muestra como se utilizaría este componente para indicar en un página un enlace a la página siguiente, anterior y para volver a la primera página.

```
<div align="center">
<hr>
<%Set vinculos=Server.CreateObject ("MSWC.NextLink")
   listaVinculos="vinculos.txt"
If (vinculos.GetListIndex(listaVinculos) > 1) Then %>
   <a href="<%=vinculos.GetPreviousURL(listaVinculos)%">>Página
anterior</a>&nbsp;
<%End If%>
<%If (vinculos.GetListIndex(listaVinculos)<vinculos.GetListCount(listaVinculos))
Then%>
   <a href="<%=vinculos.GetNextURL(listaVinculos)%">>Página siguiente</a>&nbsp;
<%End If%>
<a href="<%=vinculos.GetNthURL(listaVinculos,1)%">>Volver Inicio</a>
</div>
```

Código fuente 178

Para comprobar si es necesario crear un enlace a la página anterior se verifica que la página actual no es la primera, esta comprobación se realiza en el primer If...Then..., y en el segundo If se comprueba si es necesario crear un enlace a la página siguiente, para ello se comprueba si la página actual es la última o no comparando el valor devuelto por el método GetListIndex con el devuelto por GetListCount.

Para realizar el enlace a la página anterior se utilizará el método GetPreviousURL, para crear el enlace a la página siguiente se utiliza GetNextURL y para el enlace de la página de inicio se utiliza el método GetNthURL pasándole como parámetro el índice con valor 1.

Este código lo podemos escribir en una página llamada PIE.ASP y realizar un INCLUDE en las páginas correspondientes para que muestren en la parte inferior esta sencilla barra de navegación. El aspecto sería el de la Figura 57.

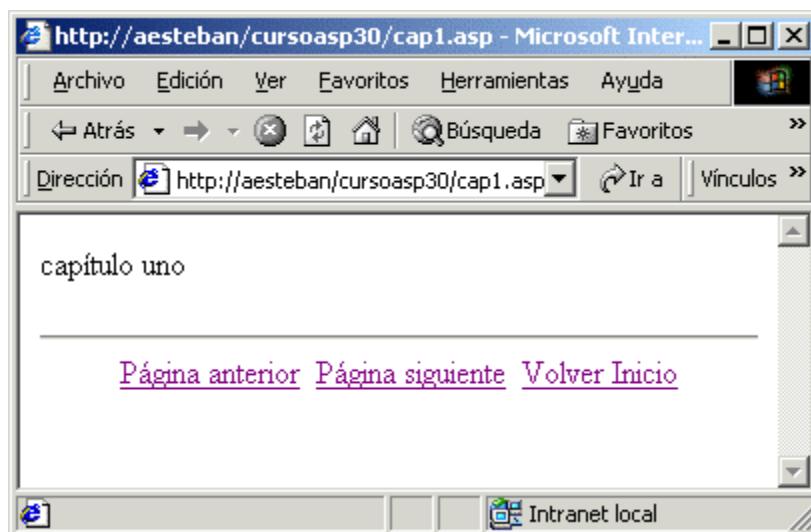


Figura 57. Mostrando una barra de navegación

En el capítulo de introducción de la lista de vínculos del ejemplo nos puede interesar mostrar un índice de todo el curso. En el Código fuente 179 se muestra como se crearía una tabla de contenidos a partir del fichero lista de vínculos de un componente Nextlink.

```
<%Set vinculos=Server.CreateObject ("MSWC.NextLink")
listaVinculos="vinculos.txt"
num=vinculos.GetListCount (listaVinculos)
i=1
%>
Índice:
<ul>
<%While (i<=num) %>
    <li>
        <a href="<%>=vinculos.GetNthURL(listaVinculos,i)%">">
            <%>=vinculos.GetNthDescription(listaVinculos,i)%></a>
    </li>
    <%i=i+1%>
<%Wend%>
```

Código fuente 179

Para recorrer el fichero lista de vínculos es necesario obtener el número de URLs que existen en el fichero, es decir, el número de enlaces, para ello utilizamos el método GetListCount, para acceder a cada elemento del fichero deberemos utilizar un bucle While. Dentro de este bucle se lanza el método GetNthURL, para obtener la URL cuyo índice se corresponde con el valor del contador del bucle, también se lanza el método GetNthDescription para obtener la descripción correspondiente.

El aspecto que se correspondería con el ejemplo de fichero de vínculos anterior sería el de la Figura 58.

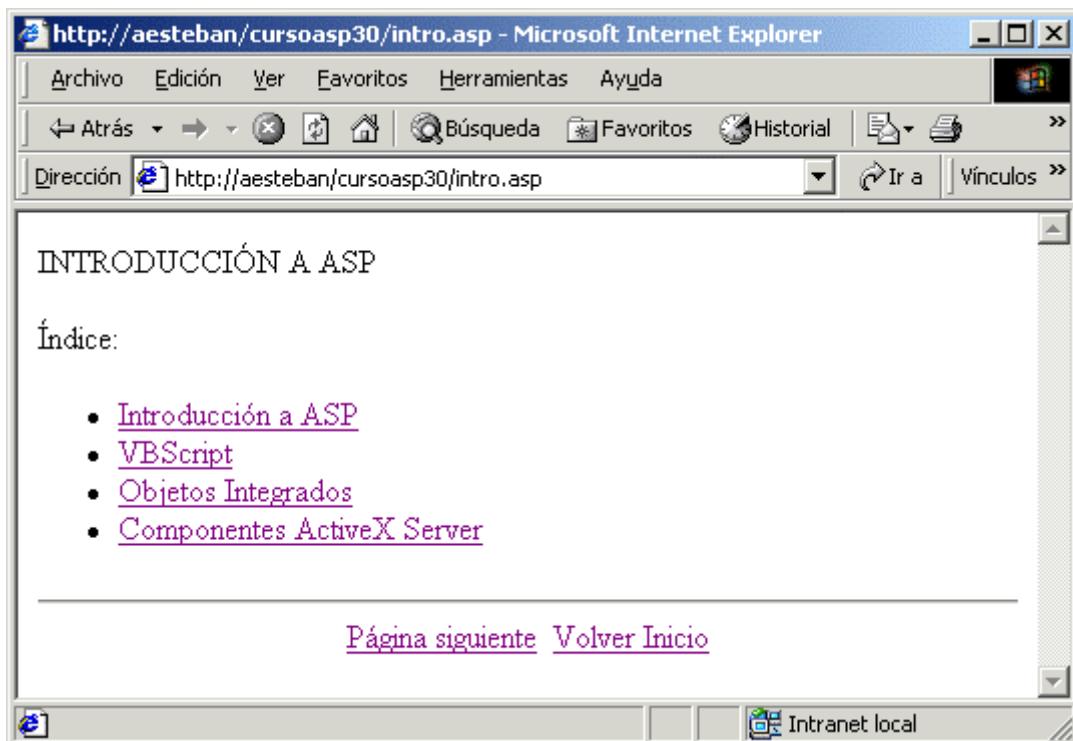


Figura 58. Mostrando una tabla de contenidos

Al igual que se hizo con el componente AdRotator, pongo a disposición de los lectores en el siguiente [enlace](#) una serie de páginas ASP y un fichero de vínculos que hacen uso del componente NextLink.

## Componente Content Rotator

Este es uno de los componentes de servidor que se incluyó con la versión 2.0 de ASP. Su finalidad es similar al componente Ad Rotator (rotador de anuncios) que ya habíamos comentado en este capítulo. Pero este nuevo componente no permite sólo alternar diferentes imágenes con enlaces atendiendo a una determinada frecuencia, sino que permite mostrar cualquier contenido HTML que irá apareciendo atendiendo a la frecuencia establecida en el fichero de planificación correspondiente, al igual que ocurría con la frecuencia asignada a las imágenes del componente AdRotator.

El fichero de planificación para este componente es un fichero de texto que debe estar disponible en una ruta virtual del servidor Web y puede incluir cualquier número de entradas de cadenas de contenido HTML, por ello también se le denomina fichero de contenido.

Cada entrada se compone de dos partes: una línea que empieza por dos signos de porcentaje (%%) seguida, de manera opcional, por un número entre 0 y 65.535 que indica el peso relativo de la cadena de contenido HTML y unos comentarios acerca de la entrada. La segunda parte de la entrada contiene la cadena de contenido HTML propiamente dicha (texto, imágenes, hipervínculos...).

La sintaxis de estas entradas sería:

```
%% [#Peso] [/Comentarios]
CadenaContenido
```

En el Código fuente 180 se ofrece un ejemplo de fichero de planificación para el componente Content Rotator.

```
%% 2 // Esta es la primera línea del fichero de contenidos.  
<A HREF = "http://www.eidos.es"></A>  
%% 4 // Segunda Línea.  
<B>Cursos - Internet</B>  
<UL>  
    <LI>Programación de aplicaciones para Internet con ASP.  
    <LI>Lenguaje HTML.  
    <LI>La Especificación CGI.  
    <LI>JavaScript.  
</UL>
```

Código fuente 180

La probabilidad de que el objeto presente una determinada cadena de contenido se expresa como el peso asociado a dicha cadena dividido por la suma de todos los pesos de las distintas entradas del fichero de contenido. Así, para el fichero de ejemplo, el componente Content Rotator presentará la primera cadena de contenido una tercera parte del tiempo, y la segunda las dos terceras partes del tiempo.

Un peso 0 hará que se pase por alto la entrada, y si no se especifica peso para una entrada determinada se asume que su peso es 1.

El componente Content Rotator, para recuperar las cadenas de contenido del fichero de planificación, dispone de dos métodos:

- ChooseContent(fichero): recupera una cadena de contenido HTML del archivo de contenidos y la presenta en la página actual. Este método recuperará una nueva cadena de contenido cada vez que se cargue la página.
- GetAllContent(fichero): recupera todas las cadenas de contenido del archivo de contenidos y las escribe directamente en la página como una lista con una etiqueta <HR> después de cada entrada.

En el Código fuente 181 a ver un ejemplo de cómo se recuperaría el contenido del fichero CONTENIDO.TXT, haciendo uso del método ChooseContent().

```
<%Set objContenido = Server.CreateObject("MSWC.ContentRotator")%>  
<%=objContenido.ChooseContent("contenido.txt")%>
```

Código fuente 181

Cada vez que el usuario ejecute la secuencia de comandos anterior, se visualizará una de las dos cadenas de contenido HTML incluidas en el fichero de planificación CONTENIDO.TXT

Vamos ahora el Código fuente 182 haciendo uso del método GetAllContent para representar todas las cadenas de contenido incluidas en el fichero CONTENIDO.TXT:

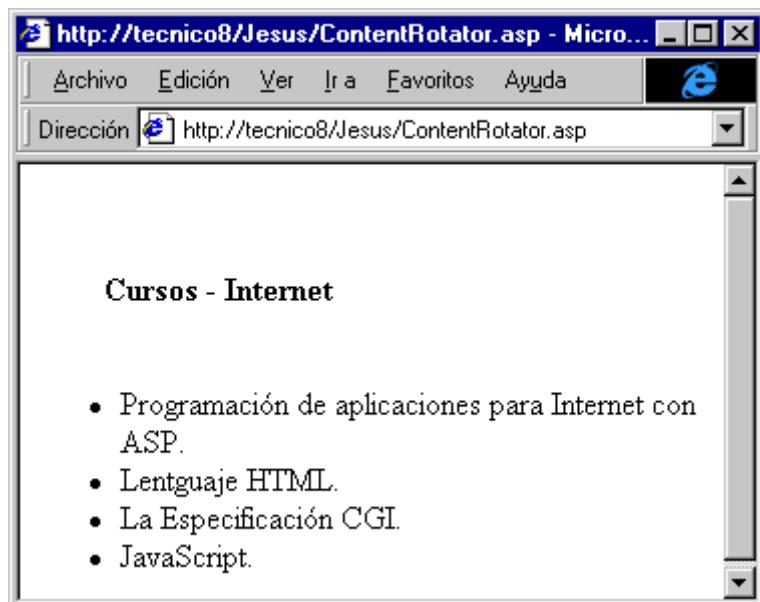


Figura 59. Utilizando el método ChooseContent()

```
<B>Cadenas de contenido HTML incluidas en "contenido.txt":</B>
<%Set objContenido = Server.CreateObject("MSWC.ContentRotator")%>
<%=objContenido.GetAllContent("contenido.txt")%>
```

Código fuente 182

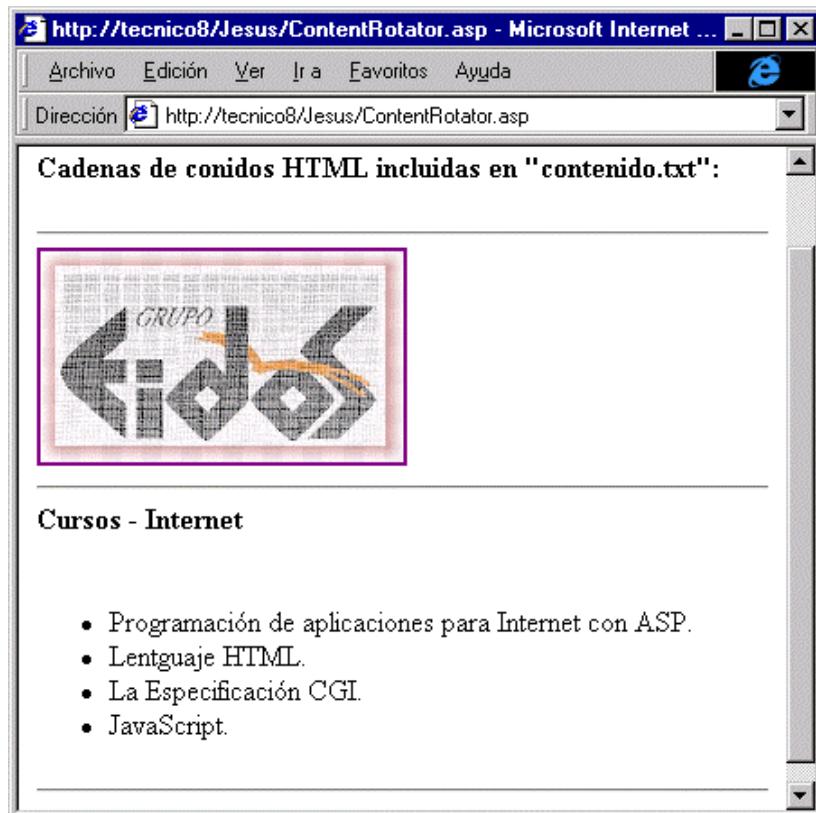


Figura 60. Utilizando el método GetAllContent()

El resultado de la ejecución del Código fuente 182 sería lo que muestra la Figura 60.

## Componente Pagecounter

Este componente nos permite llevar la cuenta del número de veces que una página Web ha sido abierta. El componente guarda el número de visitas que se han realizado en una página determinada y lo almacena periódicamente en un archivo de texto.

Al guardarse la cuenta de visitas por página en un fichero de texto, si por ejemplo se resetea el servidor los datos no se perderán. Este fichero de texto se llama HITCNT.CNT y se encuentra en el directorio c:\winnt\system32\inetsrv\data, un ejemplo de contenido de este fichero es:

```
21 /cursoasp30/pagecounter.asp
5 /cursoasp30/adrot.asp
0 adrot.asp
12 /cursoasp30/page/visitas.asp
3 /cursoASP30/page/adrot.asp
2 /cursoASP30/page/pagecounter.asp
0 /cursoASP30/page/Rec.asp
```

El fichero HITCNT.CNT se creará o actualizará cuando se ejecute el evento Application\_OnEnd del fichero GOLBAL.ASA de la aplicación ASP correspondiente.

Este componente también fue incluido con la versión 2.0 de ASP.

El componente contador de páginas se instancia como muestra el Código fuente 183.

```
<%Set ObjetoContador = Server.CreateObject("MSWC.PageCounter")%>
```

Código fuente 183

Los métodos de los que dispone el objeto Page Counter son los que se comentan a continuación:

- Hits(rutapagina): devuelve un valor LONG representado el número de veces que se ha abierto una página Web determinada. Este método puede recibir como parámetro la ruta de la página a la que se le va a llevar la cuenta. Si se omite este parámetro el método devolverá el número de visitas de la página actual.
- PageHit(): Incrementa el número de visitas a la página actual. Sintaxis.
- Reset(rutapagina): Pone a 0 el número de visitas de la página cuya ruta se le pasa como parámetro. Si no se especifica ninguna página en el parámetro, se pondrá a 0 el número de visitas de la página actual.

Veamos un ejemplo del uso de este componente en el que nos vamos a fabricar un contador de visitas para la página actual. Guardaremos la página como CONTADOR.ASP:

```
<% Set Contador = Server.CreateObject("MSWC.PageCounter")%>
<b>Número de visitas:<%=Contador.Hits%>
```

```
<%Contador.PageHit 'incrementamos el contador en uno cada vez que se cargue la
página%>
```

Código fuente 184

En cualquier momento, desde otra página podremos conocer el número de visitas de la página anterior. Suponiendo que dicha página se encuentra en la ruta virtual "/CursoASP30/pagecounter.asp", visualizaríamos su número de visitas como indica el Código fuente 185.

```
<b>El número de visitas de la página Contador.asp es:
<%=Contador.Hits("/CursoASP30/pagecounter.asp")%>
```

Código fuente 185

La ruta que se indica en los métodos Hits() y Reset() del objeto PageCounter debe iniciarse siempre con /, es decir, debe partir desde la raíz del sitio Web.

Si queremos registrar las visitas realizadas a todas las páginas ASP mediante este componente, podemos construirnos la página ASP que muestra el Código fuente 186 que se utilizaría como un INCLUDE en todas las páginas.

```
<%Set Contador=Server.CreateObject("MSWC.PageCounter")
Contador.PageHit
Set Contador=Nothing%>
```

Código fuente 186

Siguiendo este ejemplo vemos que sería muy fácil realizar una página ASP que contenga una tabla informativa de las visitas que han recibido todas las páginas del Web con contadores de un sitio Web determinado.

En este nuevo ejemplo del objeto Page Counter vamos a mostrar el número de visitas que han tenido las páginas contenidas en un directorio determinado de nuestro sitio Web. Para ello vamos hacer uso de los ya conocidos componentes de acceso al sistema de ficheros. Veamos el Código fuente 187 de esta página.

```
<%strDirectorio="/cursoASP30/page"
' creamos un objeto PageCounter
Set Contador = Server.CreateObject("MSWC.PageCounter")
'en esta misma página también se registran sus visitas
Contador.PageHit
Set objFSO=Server.CreateObject("Scripting.FileSystemObject")
'obtnemos una refencia a la carpeta dónde están las páginas
Set objFolder=objFSO.GetFolder(Server.MapPath(strDirectorio))
'obtenemos los ficheros de la carpeta
Set colFiles=objFolder.Files
'se recorren los ficheros
For each objFile in colFiles
    'se recupera el nombre para obtener después el nº de visitas
    strNombreFichero=objFile.Name
    Response.Write strNombreFichero&" --número de visitas: "&_
```

```
Contador.Hits(strDirectorio&"/"&strNombreFichero) &"<br>"  
Next%>
```

Código fuente 187

Se supone que todas las páginas mostradas deben registrar mediante el método PageHit() que un usuario a cargado la página, sino parecerá siempre cero en su número de visitas. En la Figura 61 se puede observar la ejecución del Código fuente 187.

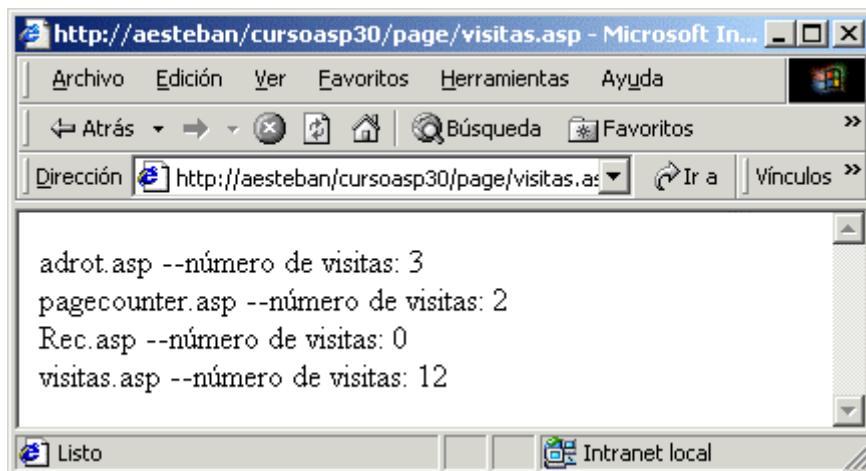


Figura 61. El componente PageCounter en acción

## Componente Counters

Este componente nos permite crear, manipular y almacenar cualquier número de contadores individuales a través de una instancia del mismo.

Sólo necesitamos crear un único objeto Counters en el sitio Web, a través del cual podremos crear los contadores individuales que queramos. Para ello debemos crear el objeto Counters en el archivo especial de aplicación ASP llamado GLOBAL.ASA y guardarlo en una variable de aplicación. Con lo cual dicho objeto persistirá durante toda la vida de la aplicación.

```
Sub Application_OnStart  
    Set ObjContador = Server.CreateObject("MSWC.Counters")  
    Set Application("Contador") = ObjContador  
End Sub
```

Código fuente 188

También podemos crearlo con la etiqueta OBJECT como muestra el Código fuente 189.

```
<object id="objContador" runat="server" scope="application"  
progid="MSWC.Counters">
```

Código fuente 189

Los contadores que creemos a partir de este objeto Counters no se incrementan automáticamente al cargar la página asp en la que han sido incluidos, sino que debemos hacerlo con el método correspondiente, que en este caso es Increment.

Todos los contadores se almacenan en un archivo de texto del servidor Web y se utilizan con valores enteros. Este fichero de texto se llama Counters.txt y se encuentra en el directorio c:\winnt\system32\inetsrv\Data, un ejemplo de contenido de este fichero es:

```
contadorUno:16
contadorDos:0
Contador1:31
Contador2:1
```

Podremos manipular los contadores con los siguientes métodos que ofrece el objeto Counters:

- Get: Devuelve el valor actual del contador cuyo nombre se le pasa como parámetro. Si el contador no existe, el método lo crea y lo pone a 0.
- Increment: Incrementa en un 1 el valor actual del contador cuyo nombre se le pasa como parámetro, y devuelve el nuevo valor del contador. Si el contador no existe, el método lo crea y lo pone a 1.
- Set: Recibe dos parámetros, el nombre del contador y un entero, asignando al contador el valor entero y devolviendo el nuevo valor. Si el contador no existe, el método lo crea y le asigna el valor entero.
- Remove: Elimina de la colección Counters y el contador cuyo nombre se le pasa como parámetro.

Veamos un ejemplo del uso de este componente en el que vamos a crear dos contadores "Contador1" y "Contador2" a partir del objeto de aplicación Counters que anteriormente hemos creado en el archivo GLOBAL.ASA mediante la etiqueta <OBJECT>. Ambos contadores se incrementarán cada vez que se actualice la página. En el momento que el segundo contador alcance el valor 10 lo volvemos a poner a 0.

```
<b>"Contador1": <%= objCounter.Increment ("Contador1") %>
<br><b>"Contador2":<%=objCounter.Increment ("Contador2") %>
<%If CInt(objCounter.Get ("Contador2"))=10 Then
    objCounter.Set "Contador2",0
End If%>
```

Código fuente 190

En cualquier momento, desde otra página ASP podremos conocer el valor de ambos contadores como indica el Código fuente 191.

```
<b>Valor del "Contador1": <%=objCounter.Get ("Contador1") %>
<br><b>Valor del "Contador2":<%=objCounter.Get ("Contador2") %>
```

Código fuente 191

## Componente MyInfo

Este componente, que también fue novedad en la versión 2.0 de ASP, nos permite almacenar información personal que será ofrecida por el administrador del sitio Web.

Esta información se cargará en las propiedades correspondientes del objeto MyInfo y podrá ser recogida a través de secuencias de script de la forma: <%=MyInfo.Propiedad%>. Todas las propiedades del objeto MyInfo devuelven una cadena.

Con el componente MyInfo al igual que ocurría con Counters, únicamente se suele crear una instancia a nivel de aplicación en el fichero GLOBAL.ASA. Como vemos en el Código fuente 192, o también como muestra el Código fuente 193.

```
Sub Application_OnStart
    Set objMyInfo=Server.CreateObject("MSWC.MyInfo")
    Set Application("objMyInfo")=objMyInfo
End Sub
```

Código fuente 192

```
<object id="objMyInfo" runat="server" scope="application" progid="MSWC.MyInfo">
```

Código fuente 193

En cualquier momento podremos crear nuevas propiedades para el objeto MyInfo y asignarles un valor. Por ejemplo, suponiendo que hemos creado el objeto con la etiqueta OBJECT, podríamos utilizar el Código fuente 194.

```
<%objMyInfo.Nombre = "Angel"
objMyInfo.Apellidos = "Esteban Núñez"
objMyInfo.Correo = "aesteban@eidos.es"%>
Nombre: <%=objMyInfo.Nombre%><br>
Apellidos: <%=objMyInfo.Apellidos%><br>
Correo: <%=objMyInfo.Correo%>
```

Código fuente 194

Las tres nuevas propiedades que acabamos de crear se almacenarán de forma persistente junto con las demás propiedades de MyInfo. Por lo que podemos hacer uso de las propiedades de este objeto para almacenar valores que permanezcan constantes en todo el sitio Web. Este objeto no tiene métodos, y las propiedades son las que le indiquemos nosotros.

La información que contiene el objeto MyInfo almacena en un fichero XML llamado myinfo.xml y se encuentra en el directorio c:\winnt\system32\inetsrv\data. Un ejemplo de contenido de este fichero podría ser el Código fuente 195.

```
<XML>
<Nombre>Angel</>
```

```
<Apellidos>Esteban Núñez</>
<Correo>aesteban@eidos.es</>
</XML>
```

Código fuente 195

## Componente Tools

Este componente, componente de herramientas, proporciona una serie de utilidades a través de sus métodos. Estos métodos son: FileExists, ProcessForm, Random, Owner y PluginExists. Hay que tener en cuenta que los dos últimos métodos (Owner y PluginExists) únicamente han sido implementados en Personal Web Server para Macintosh.

Este componente, también fue novedad en la versión 2.0 de ASP.

Vamos a describir los métodos del componente Tools:

- FileExists: Este método permite comprobar si el archivo cuya URL se le pasa como parámetro está publicado en el sitio Web. Por tanto, las URLs que recibe como parámetro son relativas y no absolutas. Si la URL existe dentro del directorio de publicación devolverá True, en caso contrario devolverá False.
- ProcessForm: Este método es el más complejo del objeto Tools y procesa el contenido de un formulario enviado por un visitante del sitio Web. Su sintaxis es:
 

```
ObjetoTools.ProcessForm(URL_Archivo_Resultados,
URL_Archivo_Plantilla, [PuntoInserción])
```
- Donde: URL\_Archivo\_Resultados: es la URL del archivo de salida en el que se han escrito los datos procesados; URL\_Archivo\_Plantilla: es la URL del archivo que contiene las instrucciones para procesar los datos y PuntoInserción: es un parámetro opcional que indicará en qué lugar del archivo de resultados se han de insertar los datos procesados. El fichero de salida puede ser una página ASP que se ejecutará en un navegador, el fichero de plantilla puede ser también un página ASP pero no se ejecutará, sino que es copiado su código ASP al fichero de salida, pero si utilizamos la sintaxis <%% %%>, al realizar la copia al fichero se saldrá si se ejecutará el código ASP correspondiente.
- Random: Este método devuelve un número entero aleatorio entre -32768 y 32767.
- Owner: Este método devuelve True si el nombre y la contraseña enviados en el encabezado de la petición coinciden con los del Administrador establecidos en la interfaz de Personal Web Server. En caso contrario devuelve False. Hay que tener en cuenta que este método únicamente han sido implementado en Personal Web Server para Macintosh.
- PluginExists: Comprueba la existencia de un complemento de servidor. Devuelve True si el complemento está registrado y False en caso contrario. Igual que el método anterior, sólo ha sido implementado en Personal Web Server para Macintosh.

Los métodos Random() y FileExists() son muy sencillos y los podemos observar en el Código fuente 196.

```
<%Set objTools=Server.CreateObject ("MSWC.Tools")%>
```

```
<%=objTools.FileExists("versiones.asp")%><br>
<%=objTools.Random()%>
```

Código fuente 196

El método ProcessForm es algo más complicado, si tenemos el fichero de plantilla que muestra el Código fuente 197,

```
Nombre: <%=Request.Form("nombre")%><br>
Apellidos: <%=Request.Form("apellidos")%><br>
Edad: <%=Request.Form("edad")%><br>
Generado en: <%=Now%>
```

Código fuente 197

al pasarle los datos de un formulario generará este otro fichero de salida:

```
Nombre: Angel<br>
Apellidos: Esteban Núñez<br>
Edad: 25<br>
Generado en: 05/06/2000 17:18:01
```

Veámoslo con un sencillo ejemplo. Al pulsar el botón del formulario se ejecutará el método ProcessForm y escribirá los datos en el fichero de salida atendiendo al fichero que funciona como plantilla. Además, en esta página a través de un objeto TextStream se muestra el contenido del fichero de salida una vez ejecutado el método ProcessForm. El código es el Código fuente 198.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set objTools=Server.CreateObject("MSWC.Tools")%>
<form method="post" action="tools.asp">
    Nombre:<input type="text" name="nombre" value="" size="20"><br>
    Apellidos:<input type="text" name="apellidos" value="" size="40"><br>
    Edad:<input type="text" name="edad" value="" size="2"><br>
    <input type="submit" name="enviar" value="Enviar"><br>
</form>
<%If Request.Form("enviar")<>"" Then
    objTools.ProcessForm "ficheroSalida.asp", "plantilla.asp"%>
    Se ha generado el siguiente fichero de salida:<br>
    <%Set objFSO=Server.CreateObject("Scripting.FileSystemObject")%
    Set objTextStream=objFSO.OpenTextFile(Server.MapPath("ficheroSalida.asp"))
    Response.Write(objTextStream.ReadAll)
    objTextStream.Close
End if%>
</BODY>
</HTML>
```

Código fuente 198

Y en la Figura 62 se puede ver un ejemplo de ejecución.

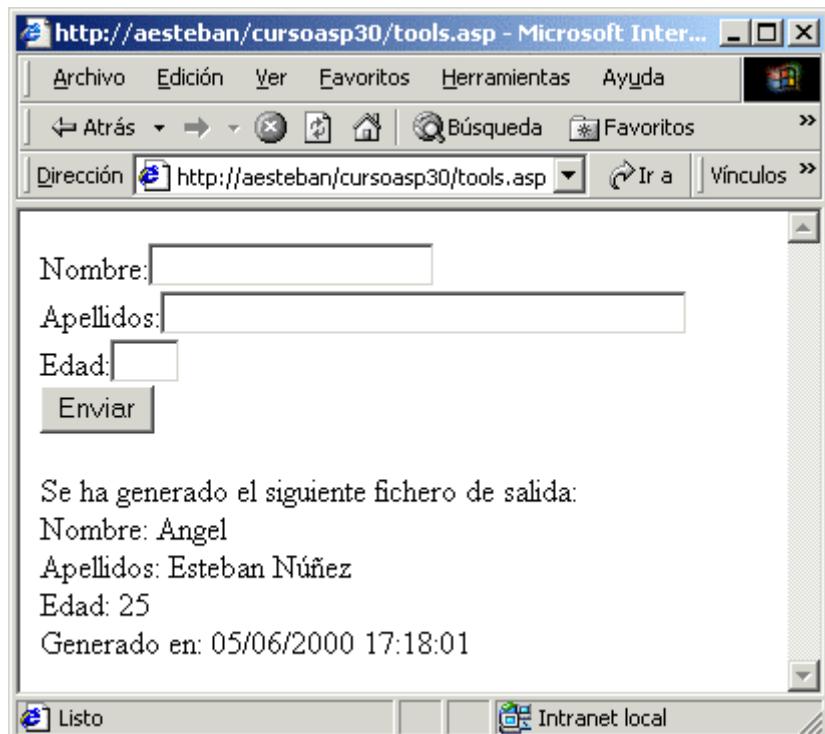


Figura 62. Utilizando el método ProcessForm()

Si especificamos el parámetro punto de inserción se buscará la cadena especificada en el fichero de plantilla y añadirá los datos a partir de esa coincidencia.

## Componente PermissionChecker

Este componente nos va a permitir determinar si a un usuario se le ha dado permisos para acceder a un fichero determinado. Este componente se basa en los mecanismos de autenticación de usuario de IIS, es decir, usuario anónimo, autenticación básica y autenticación desafío/respuesta de Windows NT.

En el caso de que se encuentre activada la autenticación anónima, todos los usuarios iniciarán sus sesiones con la cuenta de usuario anónimo de IIS. En este caso, al compartir todos los usuarios la misma cuenta, El componente PermissionChecker no podrá autenticar usuarios de forma individual.

En el caso de aplicaciones en las que todos los usuarios tengan cuentas individuales, como en los sitios Web dentro de una Intranet, es recomendable la desactivación de la autenticación anónima para que PermissionChecker pueda identificar a cada usuario de forma individual.

Para denegar el acceso anónimo a una página Web, podemos usar uno de estos tres métodos:

- En la lista de control de acceso al archivo, excluir la cuenta del usuario anónimo a nivel de NTFS.
- Desde el Administrador de servicios de Internet, en la hoja de propiedades de la seguridad de directorio se desactiva el acceso anónimo. En el capítulo siguiente veremos en detenimiento la configuración y administración del servidor Web Internet Information Server.
- Mediante el siguiente script de servidor.

```
<%If Request.ServerVariables("LOGON_USER") = "" Then
    Response.Redirect ("NoAutorizado.htm")
End If%>
```

Código fuente 199

En este script comprobamos si la variable del servidor LOGON\_USER de la colección ServerVariables está vacía, en tal caso se tratará de la cuenta de usuario anónimo y le redireccionamos a una página que le indica al usuario que no está autorizado para el acceso.

En el caso de sitios Web mixtos de Internet e Intranet, es recomendable activar la autenticación anónima y al menos uno de los dos métodos de autenticación por contraseña, autenticación básica o autenticación desafío/respuesta de Windows NT. De esta forma, si se deniega a un usuario el acceso anónimo a una página, el servidor intentará autenticar al usuario mediante alguno de dos métodos antes comentados.

A través de este componente podremos, incluso, controlar la navegación entre páginas. Es decir, para un hiperenlace determinado podríamos comprobar si el usuario tiene permisos de acceso a la página direccionada, y en caso negativo desactivar el hiperenlace. La sintaxis de creación de este componente es:

```
Set Objeto = Server.CreateObject ("MSWC.PermissionChecker")
```

Este componente dispone de un solo método, llamado HasAccess, a través del cual vamos a poder realizar el chequeo del acceso al un archivo especificado. Su sintaxis es:

```
objPermissionChecker.HasAccess ("rutaArchivo")
```

Donde rutaArchivo podrá ser una ruta física o virtual a un archivo determinado. HasAccess() devolverá un valor booleano indicando si el usuario Web tiene acceso al archivo. En el caso de que el archivo no exista, devolverá False.

En el Código fuente 200 vemos un ejemplo de utilización del componente, en el que chequeamos el acceso del usuario a un archivo del Web y en caso afirmativo le proporcionamos un enlace para que acceda al mismo.

```
<%Set objPermission=Server.CreateObject ("MSWC.PermissionChecker")
If objPermission.HasAccess ("/MiWeb/Privado.htm") Then%>
    <A HREF= "/MiWeb/Privado.html"> Area Privada </A>
<%Else%>
    Enlace no disponible.
<%End If%>
```

Código fuente 200

Si el usuario no se ha autenticado para acceder a la página ASP del ejemplo, nunca tendrá acceso al enlace, aunque el usuario anónimo de Internet esté autorizado el método HasAccess() devolverá False.

## Componente Status

Este componente únicamente está disponible para el servidor Personal Web Server en plataformas Macintosh. Posee una serie de propiedades que nos van a proporcionar información variada sobre el estado del servidor Web. Su sintaxis es:

```
Set InfoEstado = Server.CreateObject("MSWC.Status")
```

Hemos decidido tratar este componente en este capítulo como una curiosidad.

Estas son las propiedades de Status, hasta el momento implementadas únicamente en sistemas Macintosh.

Propiedad	Descripción
VisitorsSinceStart	Número de visitantes distintos desde el inicio del servidor.
RequestsSinceStart	Número de peticiones desde el inicio del servidor.
ActiveHTTPSessions	Número actual de conexiones http.
HighHTTPSessions	Número máximo de conexiones http simultáneas desde el inicio del servidor.
ServerVersion	Versión de Personal Web Server.
StartTime	Hora de inicio del servidor.
StartDate	Fecha de inicio del servidor.
FreeMem	Cantidad de memoria no utilizada disponible en el servidor.
FreeLowMem	Valor mínimo de la cantidad de memoria no utilizada disponible desde el inicio del servidor.
VisitorsToday	Número de visitantes distintos, desde la medianoche.
RequestsToday	Número de peticiones realizadas desde la medianoche.
BusyConnections	Número total de peticiones rechazadas al estar el servidor utilizando todas las conexiones que puede aceptar.
RefusedConnections	Número total de peticiones rechazadas por fallos de autenticación.
TimedoutConnections	Número total de conexiones cerradas sin haber recibido ninguna petición.
Ktransferred	Número total de kilobytes enviados por el servidor desde su inicio.
TotalRequests	Número total de peticiones recibidas desde que se restablecieron los contadores mediante la herramienta de administración.

CurrentThreads	Suma del número de conexiones http activas y el número de subprocesos del conjunto de subprocesos de conexión que no están administrando actualmente ninguna conexión.
AvailableThreads	Número de subprocesos del conjunto de subprocesos de conexión que no están administrando actualmente ninguna conexión.
RecentVisitors	Una tabla HTML que enumera los 32 últimos visitantes.
PupularPages	Una tabla HTML que enumera las 32 páginas visitadas más recientemente.

Tabla 16

## Componente de registro de IIS

Este nuevo componente de servidor es denominado componente de registro o programa de registro de IIS. Mediante este componente tenemos acceso a la información y manipulación de los ficheros de registro (log) generados por el servidor Web IIS 5.0.

Este componente, al igual que todos los existentes en ASP 3.0, se instala conjuntamente con el servidor Web Internet Information Server 5.0. El fichero DLL que contiene a este nuevo componente es logscript.dll.

Para instanciar un componente de registro debemos utilizar la sentencia que muestra el Código fuente 201.

```
Set objRegistro=Server.CreateObject("MSWC.IISLog")
```

Código fuente 201

Es importante señalar que el usuario que tiene acceso a la secuencia de comandos ASP que crea la instancia del componente de registro debe autenticarse como Administrador u Operador en el servidor donde se ejecuta IIS, si es un usuario anónimo, el componente de registro de IIS no funcionará correctamente.

Para manipular los ficheros de registro de IIS el componente IISLog ofrece una serie de métodos que se muestran a continuación.

- AtEndOfFile(): devuelve un valor booleano que indica si se leyeron o no todos los registros del archivo de log.
- CloseLogFile(modo): cierra los ficheros de registro atendiendo al parámetro modo. Si modo tiene el valor ForReading(1) cierra los ficheros que se han abierto para lectura, ForWriting(2) los que se han abierto para escritura y AllOpenFiles(32) cierra todos los ficheros abiertos.
- OpenLogFile(nombreFichero, modo, nombreServicio, instanciaServicio, formatoSalida): abre el fichero de registro indicado para lectura o escritura, según el parámetro modo, que por defecto tiene valor ForReading(1). El parámetro opcional nombreServicio es una cadena que indica que los registros que se obtengan deben coincidir con el servicio especificado, el

servicio Web se identifica mediante la cadena "W3SVC". El parámetro instanciaServicio, también opcional, indica que se deben obtener los registros que coincidan con esta instancia de servicio. El último parámetro también opcional, indica el formato del fichero de registro cuando se abre para escritura.

- ReadFilter(inicio, fin): Este método permite leer los registros que se encuentran en un intervalo de tiempo (fecha y hora) determinado. Filtra los registros del archivo según la fecha y la hora. Los dos parámetros que indican el rango de fechas son opcionales y si se omite inicio se comienza por el primer registro del fichero, y si se omite fin se devolverá hasta la última línea el fichero.
- ReadLogRecord(): lee el siguiente registro disponible del archivo de registro actual.
- WriteLogRecord(objIISLog): Este método permite escribir nuevos registros, leídos de otro objeto IISLog. Escribe los registros desde un fichero de log abierto para lectura a otro fichero que ha sido para escritura. Como parámetro se le pasa una instancia del componente de registro de IIS que representa el objeto que contiene los datos fuente.

Para obtener la información del registro actual el componente IISLog ofrece veinte propiedades de sólo lectura que se corresponden con los distintos campos de un registro de un archivo de registro. En la Tabla 17 se muestra un resumen de todas las propiedades del componente IISLog.

Propiedad	Descripción
BytesReceived	Número de bytes recibidos del navegador como una petición.
BytesSent	Número de bytes enviados al navegador como una respuesta.
ClientIP	Dirección IP del cliente.
Cookie	Indica los contenidos de cualquier cookie enviada en la petición.
CustomFields	Un vector de cabeceras personalizadas que se añadieron a la petición.
DateTime	La fecha y hora de la petición en formato GMT.
Method	El tipo de operación, tal como puede ser GET o POST.
ProtocolStatus	El mensaje de estado devuelto al cliente, por ejemplo 200 OK.
ProtocolVersion	Una cadena con la versión del protocolo utilizado, por ejemplo HTTP/1.1.
Referer	La URL de la página que contiene el enlace que inició la petición, si está disponible.
ServerIP	La dirección IP del servidor Web.
ServerName	El nombre del servidor Web.
ServerPort	El número de puerto por el que se recibió la petición.
ServiceName	Nombre del servicio, como puede ser el servicio FTP (MSFTPSVC) o Web (W3SVC).

TimeTaken	El tiempo de procesamiento total para devolver y crear la página devuelta.
URIQuery	Cualquier parámetro añadido a la cadena de consulta (QueryString) de la URL en la petición.
URIStem	La URL que demandó el cliente.
UserAgent	La cadena de agente de usuario (tipo de navegador) enviada por el cliente.
UserName	Nombre de inicio de sesión del usuario si no ha accedido de forma anónima.
Win32Status	Código de estado Win32 después de haber procesado la petición.

Tabla 17

Se puede configurar el tipo de registro que queremos en nuestro servidor a través de IIS 5.0, de esta forma podremos añadir o eliminar de nuestro fichero de registro los campos descritos anteriormente. Para ello acudiremos a las propiedades del sitio Web y en la pestaña sitio Web pulsaremos el botón Propiedades contenido en el epígrafe de Habilitar registro. Se debe seleccionar uno de los formatos de registro que se corresponden con un fichero de registro, por lo tanto la opción registro ODBC no sería válida.

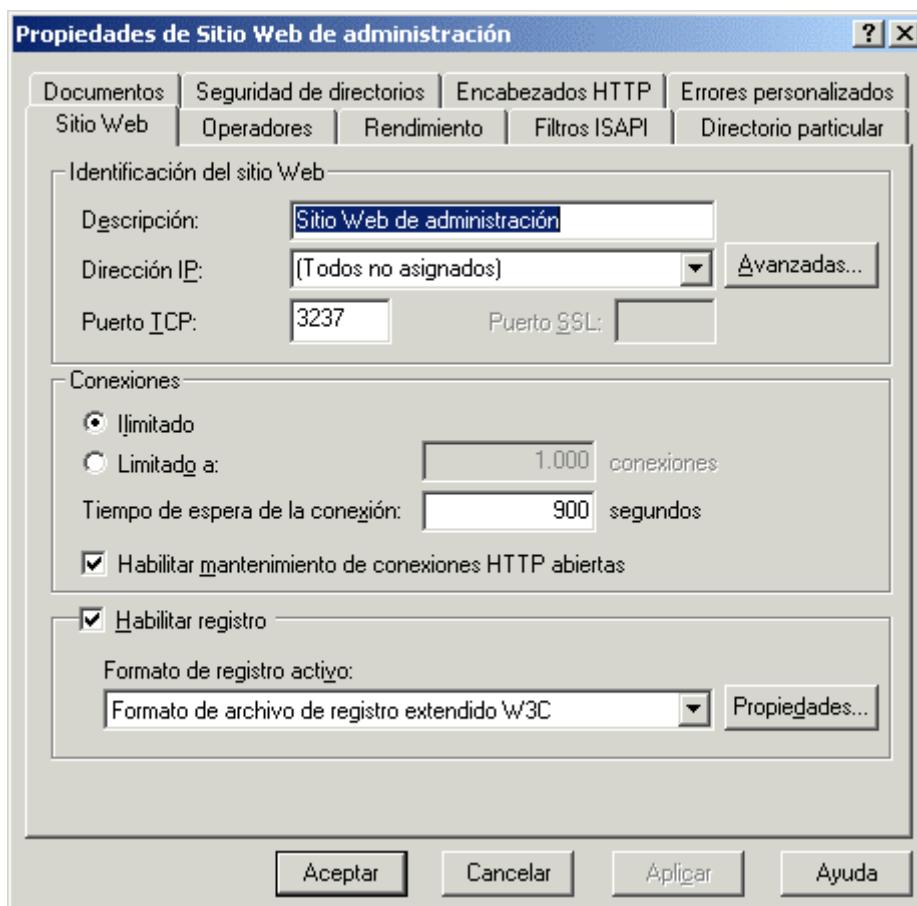


Figura 63. Habilitando el registro de IIS

En la Figura 64 se puede ver como podemos configurar la forma en que IIS realiza el registro de la actividad del sitio Web. En la pestaña de Propiedades generales podemos indicar el periodo de tiempo para la creación fichero de registro, por ejemplo, si seleccionamos Diario se creará un fichero de registro para cada día, o también podemos indicar que se cree un nuevo fichero cada vez que el fichero de registro alcance un tamaño determinado, según la selección realizada el nombre del fichero de registro tendrá un formato determinado. Otro parámetro a indicar es la ruta en la que se va a almacenar el fichero de registro.

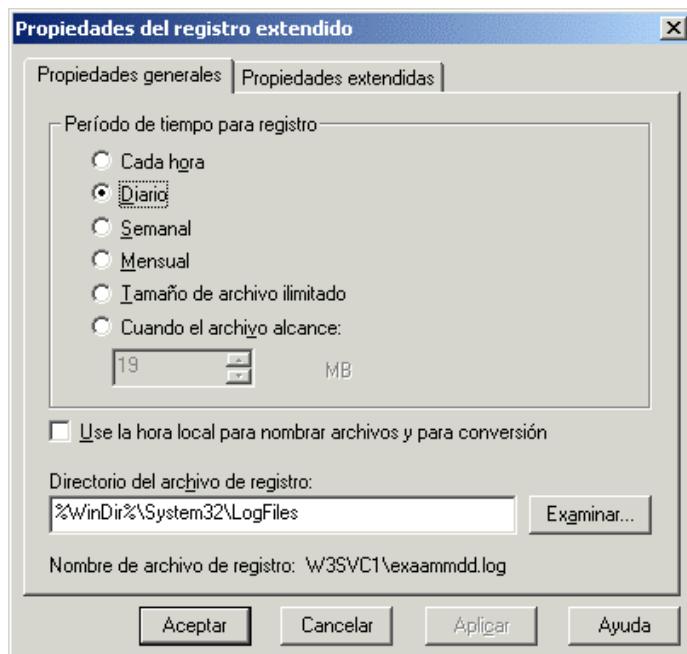


Figura 64. Propiedades generales del fichero de registro

En la pestaña de Propiedades extendidas () podemos indicar exactamente que información es necesaria almacenar en el proceso de registro.

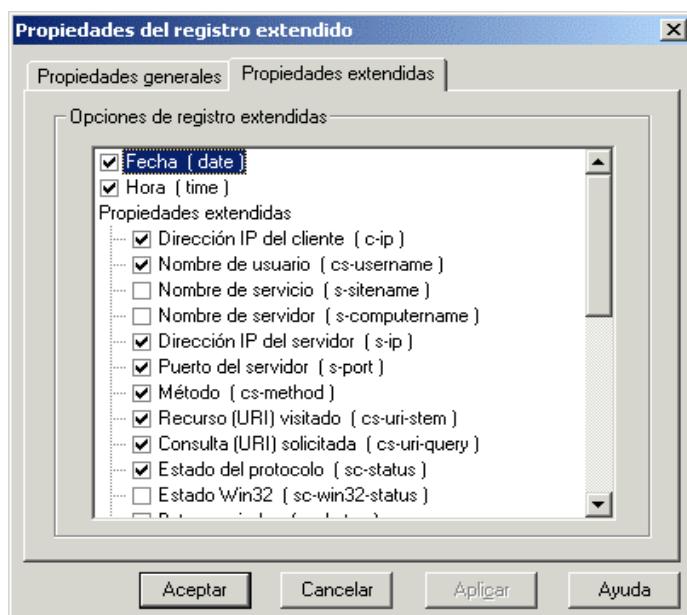


Figura 65. Selección de la información necesaria en el fichero de registro

En el Código fuente 202 se muestra la utilización de este nuevo objeto ActiveX de servidor. En este sencillo código se utiliza el componente de registro para mostrar algunos de los campos contenidos en el fichero de registro.

```
<html>
<head>
<!--METADATA TYPE="TypeLib" FILE="c:\winnt\system32\inetsrv\logscript.dll"-->
</head>
<body>
<%Set objRegistro=Server.CreateObject("MSWC.IISLog")
objRegistro.OpenLogFile "c:\winnt\system32\logfiles\w3svc1\ex000517.log"_
,ForReading,"W3SVC",1,0
objRegistro.ReadFilter DateAdd("d",-1,Now),Now%>
<table align="center" border="0" cellspacing="2" cellpadding="5">
<tr>
    <th>Fecha/Hora</th>
    <th>IP del cliente</th>
    <th>Método</th>
    <th>URL</th>
</tr>
<%While Not objRegistro.AtEndOfLog
    objRegistro.ReadLogRecord%>
    <tr>
        <td><%=objRegistro.DateTime%></td>
        <td><%=objRegistro.ClientIP%></td>
        <td><%=objRegistro.Method%></td>
        <td><%=objRegistro.URIStem%></td>
    </tr>
<%wend
objRegistro.CloseLogFile(ForReading)%>
</body>
</html>
```

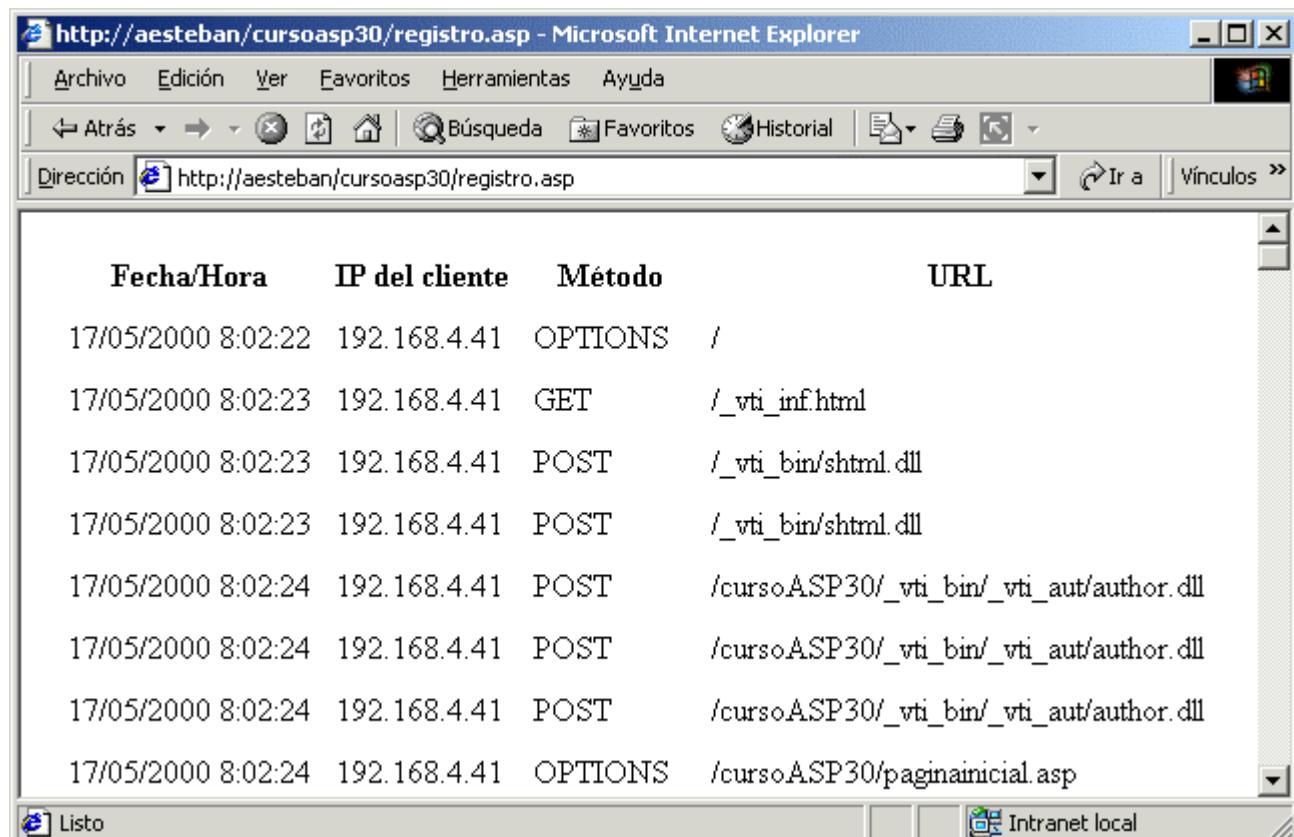
Código fuente 202

Se ha utilizado un filtro para recuperar la información del fichero de registro referente al servicio Web y únicamente de las últimas 24 horas. También se puede observar que se utiliza una directiva METADATA para incluir las constantes definidas en la librería que contiene al componente de registro.

La información que se va a mostrar del fichero de registro va a ser la fecha y hora de la petición, la dirección IP del cliente que ha realizado la petición, el método que se ha utilizado y la URL correspondiente. En la Figura 66 se puede ver un ejemplo de ejecución de la página anterior.

El nombre del fichero de registro variará según sea nuestra configuración del registro en el sitio Web correspondiente, la ubicación de estos ficheros de registro suele ser el directorio c:\winnt\system32\logfiles\w3svc1 para el servicio Web.

Esta página ASP que utiliza el componente de registro se puede utilizar únicamente restringiendo el acceso anónimo a la propia página o al directorio que la contiene a nivel de permisos de NTFS, en caso contrario no podremos acceder al fichero de registro, ya sea para leer o escribir datos. Esta recomendación aparece en la propia documentación de ASP que se ofrece con IIS5 sin embargo, he conseguido que este ejemplo funcione ejecutando la página ASP como otra cualquiera, es decir, directamente con el usuario anónimo de Internet.



The screenshot shows a Microsoft Internet Explorer window with the title bar "http://aesteban/cursosoasp30/registro.asp - Microsoft Internet Explorer". The menu bar includes Archivo, Edición, Ver, Favoritos, Herramientas, and Ayuda. The toolbar includes Back, Forward, Stop, Home, Search, Favorites, History, and other navigation icons. The address bar shows the URL "http://aesteban/cursosoasp30/registro.asp". The main content area displays a table of log entries:

Fecha/Hora	IP del cliente	Método	URL
17/05/2000 8:02:22	192.168.4.41	OPTIONS	/
17/05/2000 8:02:23	192.168.4.41	GET	/_vti_inf.html
17/05/2000 8:02:23	192.168.4.41	POST	/_vti_bin/shtml.dll
17/05/2000 8:02:23	192.168.4.41	POST	/_vti_bin/shtml.dll
17/05/2000 8:02:24	192.168.4.41	POST	/cursoASP30/_vti_bin/_vti_aut/author.dll
17/05/2000 8:02:24	192.168.4.41	POST	/cursoASP30/_vti_bin/_vti_aut/author.dll
17/05/2000 8:02:24	192.168.4.41	POST	/cursoASP30/_vti_bin/_vti_aut/author.dll
17/05/2000 8:02:24	192.168.4.41	OPTIONS	/cursoASP30/paginainicial.asp

Figura 66. Utilizando el componente IISLog

## Componentes de terceras partes

Además de los componentes de servidor que vienen integrados con ASP, existen empresas que se dedican a construir componentes ActiveX Server. Una de estas empresas es ServerObjects Inc., si se visita su sitio Web, <http://www.serverobjects.com/>, se puede observar un listado de todos los componentes para ASP que ofrece, algunos gratuitos y otros de pago.

Uno de los componentes que se ofrecen es ASPMail, este componente nos permitirá enviar correo desde nuestra aplicación ASP. Para ello utiliza el protocolo SMTP (Simple Mail Transfer Protocol).

Para poder utilizar el componente ASPMail, primero debemos descargar desde ServerObjects Inc. el fichero .DLL en el que se encuentra este componente. Una vez que ya tenemos el fichero .DLL (smtpsvg.dll) debemos moverlo al subdirectorio \winnt\system32 dentro de la máquina que va a funcionar como servidor Web. Para registrar el nuevo componente en el servidor debemos usar la utilidad regsvr32 y escribir la instrucción que muestra el Código fuente 203 en la línea de comandos.

```
regsvr32 smtpsvg.dll
```

Código fuente 203

Lo que viene a continuación no pretender ser una detallada referencia del componente ASPMail, simplemente se va a mostrar como se utilizaría un componente de servidor de terceros a través de algunos ejemplos del componente ASPMail comentando las propiedades y métodos más importantes.

En el Código fuente 204 se muestra como se enviaría un mensaje de correo con un documento adjunto:

```
<%Set correo=Server.CreateObject("SMTPsvg.Mailer")
correo.RemoteHost="mail.correo.es"
correo.FromName="Pepe Trola"
correo.FromAddress="Pepe@correo.es"
correo.AddRecipient "Angel Esteban", "aesteban@eidos.es"
correo.Subject="Prueba correo ASP"
correo.BodyText="Buenos días, que tal..."
correo.AddAttachment "C:\tmp\documento.doc"
If correo.SendMail Then
    Response.Write "Correo enviado"
Else
    Response.Write correo.Response
End if%>
```

Código fuente 204

Como se puede observar una vez instanciado el componente se utiliza la propiedad RemoteHost para indicar el servidor de correo que se va a utilizar, se pueden indicar hasta tres servidores de correo separadas sus direcciones por punto y coma (;), esto se puede utilizar para indicar un servidor SMTP primario y otro secundario, si el servidor de correo primario se encuentra fuera de servicio se utilizará el secundario.

A continuación se indica el nombre y la dirección de la persona que va a enviar el correo, para ello se utilizan las propiedades FromName y FromAddress, respectivamente. En la siguiente línea se indica el nombre y la dirección del destinatario del mensaje mediante el método AddRecipient. En la llamada a este método se indicará el nombre del destinatario y su dirección de correo.

El asunto del mensaje se indica dentro de la propiedad Subject, y el texto del mensaje, es decir, el mensaje en sí, se indica dentro de la propiedad BodyText.

En este ejemplo además de enviar un simple mensaje, se adjunta un fichero como anexo al mensaje. Para adjuntar un fichero a un mensaje de correo se debe utilizar el método AddAttachment, al que se le pasa como parámetro la ruta del fichero que se quiere adjuntar.

Una vez asignado el contenido del mensaje a la propiedad BodyText y adjuntado un fichero con el método AddAttachment se procede a enviar el mensaje mediante el método SendMail.

El método SendMail devolverá True si el mensaje se envió correctamente y False si se dieron errores. Los errores que se han producido se recogen dentro de la propiedad Response. De las líneas 10 a la 14 se muestra como se suele tratar el envío de correo y como se muestran los errores. En una sentencia If...Then...Else se consulta el valor devuelto por el método SendMail, si es igual a False se muestran los errores mediante la propiedad Response.

En el siguiente ejemplo se puede observar que los datos del mensaje se obtienen de un formulario. Este formulario se encuentra dentro de una página ASP y posee el botón de tipo "submit" que enviará el contenido del formulario a la propia página ASP, y a partir de la información del formulario se construirá el mensaje y se enviará. El aspecto que va a tener el formulario es el que muestra la Figura 67.

El código encargado de crear el formulario, y una vez pulsado el botón etiquetado como "Enviar Mensaje", enviar el mensaje a su destinatario correspondiente es el que aparece en el Código fuente 205.

**Envía Correo**

<b>Nombre Remitente:</b>	<b>Dirección Remitente:</b>
<input type="text"/>	<input type="text"/>
<b>Nombre Destinatario:</b>	<b>Dirección Destinatario:</b>
<input type="text"/>	<input type="text"/>
<b>Asunto:</b>	
<input type="text"/>	
<b>Mensaje:</b>	
<div style="border: 1px solid black; height: 150px; width: 100%;"></div>	
<b>Enviar Mensaje</b>	

Figura 67. Formulario para el envío de correo a través del componente ASPMail.

```
<HTML>
<HEAD>
<TITLE>Correo</TITLE>
</HEAD>
<BODY>
<%If Request.Form("Enviar")="" Then%>
    <form action="aspmail.asp" method="POST" id=form1 name=form1>
        <table border="0" align="center">
            <tr>
                <th colspan="2" align="center">
                    <big>Envía Correo</big>
                </th>
            </tr>
            <tr>
                <td align="center">
                    <strong>Nombre Remitente:</strong><br>
                    <input type="Text" name="nombreRemitente">
                </td>
                <td align="center">
                    <strong>Dirección Remitente:</strong><br>
                    <input type="Text" name="dirRemitente">
                </td>
            </tr>
            <tr>
                <td align="center">
                    <strong>Nombre Destinatario:</strong><br>
                    <input type="Text" name="nombreDestinatario">
                </td>
            </tr>
        </table>
    </form>
<%End If%>
```

```

<td align="center">
    <strong>Dirección Destinatario:</strong><br>
    <input type="Text" name="dirDestinatario">
</td>
</tr>
<tr>
    <td colspan="2" align="center">
        <strong>Asunto:</strong><br>
        <input type="Text" name="asunto">
    </td>
</tr>
<tr>
    <td colspan="2" align="center">
        <strong>Mensaje:</strong><br>
        <textarea name="mensaje" cols="50" rows="10"></textarea><br>
        <input type="Submit" name="Enviar" value="Enviar Mensaje">
    </td>
</tr>
</table>
</form>

<%Else
    Set correo=Server.CreateObject("SMTPsvg.Mailer")
    correo.RemoteHost="servidor.correo.es"
    correo.FromName=Request.Form("nombreRemitente")
    correo.FromAddress=Request.Form("dirRemitente")
    correo.AddRecipient
    Request.Form("nombreDestinatario"),Request.Form("dirDestinatario")
    correo.Subject=Request.Form("asunto")
    correo.BodyText=Request.Form("mensaje")
    if correo.SendMail Then
        Response.write("OK, correo enviado con éxito a: "&
                      Request.Form("dirDestinatario"))
    Else
        Response.Write("Se han producido errores:"&correo.Response)
    End If
End If%>
</BODY>
</HTML>

```

Código fuente 205

Como se puede observar en el código, la ejecución de la página ASP tiene dos fases. La primera fase muestra el formulario cuando se carga la página ASP, y la segunda fase, que se produce cuando se ha pulsado el botón de enviar del formulario, recupera la información del formulario construye el mensaje y lo envía. El código fuente de este ejemplo se puede encontrar en el siguiente enlace.

Como se puede observar la información del formulario se recupera con la colección Form del objeto integrado Request. En la sentencia If...Then... se distingue si se proviene de la pulsación del botón de formulario o simplemente se está cargando la página ASP.

En el siguiente [enlace](#) se pueden encontrar el código del ejemplo. Otra utilidad que se le puede dar a este componente es la de enviar el correo de forma automatizada a diferentes destinatarios. Estos destinatarios podrían estar almacenados en una tabla de una base de datos, y dentro de un bucle While recorreríamos toda la tabla y enviaríamos un mensaje de correo a cada destinatario a partir de los datos obtenidos de la tabla. En el Código fuente 206 se muestra ese mecanismo.

```

<%Set correo=Server.CreateObject("SMTPsvg.Mailer")
correo.RemoteHost = "servidorCorreo.net"
correo.FromName = "Administrador"
correo.FromAddress ="admin@host.net"

```

```
correo.Subject = "Anuncio"
correo.BodyText = "Me dirijo a usted con el motivo..."
correo.Timeout = 120
RS.MoveFirst
Response.Write " Enviando mensajes...<p>"
WHILE not RS.Eof
    correo.AddRecipient RS("nombre"), RS("email")
    if not correo.SendMail then
        Response.Write(correo.Response & "...<p>")
    else
        Response.Write(" Todo OK...<p>")
    end if
    correo.ClearRecipients
    RS.MoveNext
    Response.Flush
WEND
correo.ClearBodyText
RS.Close
Set RS=Nothing%>
```

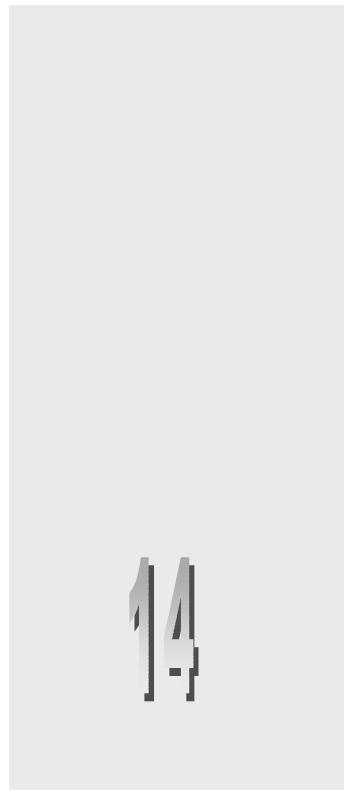
Código fuente 206

Como se puede observar aparece un objeto llamado RS, este objeto es un objeto perteneciente a ADO, y representa un objeto Recordset, es decir, contiene el resultado de realizar una consulta sobre una base de datos.

En este caso el objeto RS contendrá los nombres y direcciones de correo electrónico de las personas a las que queramos enviar el mensaje. El objeto Recordset se verá en profundidad en un tema monográfico de este objeto de ADO, simplemente se debe saber que para desplazarnos por las líneas del Recordset debemos lanzar el método MoveNext, y mientras no se llegue al final del mismo, su propiedad Eof será igual a False.

En este código se puede ver otra propiedad de un objeto ASPMail, la propiedad Timeout. Esta propiedad indica el tiempo máximo que el componente ASPMail esperará una respuesta del servidor de correo. El valor predeterminado es de 30 segundos. También se utilizan otros dos nuevos métodos: el método ClearRecipients y el método ClearBodyText. El primero de ellos borra los datos del destinatario y el segundo el texto del mensaje.





# ASP e Internet Information Server 5.0

---

## Introducción

Active Server Pages se basa en una serie de tecnologías diferentes, es decir, debajo de ASP existen una serie de piezas que son necesarias para el correcto funcionamiento de ASP, estas piezas son:

- El sistema operativo Windows 2000.
- El protocolo de comunicaciones TCP/IP.
- Un servidor Web que soporte Active Server Pages, en este caso Internet Information Server.
- Un servidor de bases de datos, como puede ser Microsoft SQL Server.

Internet Information Server (IIS) es un servidor Web que permite publicar información en una intranet de la organización o en Internet. Internet Information Server transmite la información mediante el protocolo de transferencia de hipertexto (HTTP).

Internet Information Server puede configurarse para proporcionar servicios de Protocolo de transferencia de archivos (FTP), y a partir de la versión 4.0, puede proporcionar también servicio de correo electrónico basado en el protocolo SMTP (Simple Mail Transfer Protocol) y servicio de noticias basado en el protocolo NNTP (Network News Transfer Protocol).

Active Server Pages forman parte de Internet Information Server desde la versión 3.0 del servidor Web. Al instalar IIS se instala por defecto ASP.

Cuando llega una petición al servidor Web, éste la procesa, desde páginas HTML a vídeo, el proceso sigue los métodos convencionales de un servidor Web, tal como enviar el fichero al navegador que ha realizado la petición.

A diferencia de los métodos convencionales de servir la información, cuando se realiza una petición de una página ASP y llega al servidor Web desde un navegador, el servidor invoca el componente DLL que se encarga de tratar la petición ASP y ejecutar la página correspondiente. El cliente que realiza la petición debe tener permisos para ejecutar la página ASP y para realizar las acciones que implican ejecutar el código de la página ASP en el servidor. Como resultado el servidor Web (IIS) devolverá HTML estándar.

En este capítulo vamos a tratar los temas de configuración y administración de IIS 5.0 que tienen que ver más o menos directamente con ASP, no se trata de un capítulo que intente explicar la administración y configuración de IIS 5.0, ya que esto supondría un curso completo.

## El servidor web Internet Information Server 5.0

Internet Information Server 5.0 es un servidor Web para plataformas Windows 2000 completamente integrado con el sistema operativo. IIS 5.0 forma parte de la instalación de Windows 2000 y permite disponer de un servidor Web tanto en el entorno de Internet como en el entorno de Intranet.

IIS 5.0 se encuentra en todas las versiones de Windows 2000: Professional, Server, y Advanced Server, pero para implementar un servidor Web es más adecuado elegir la versión Server o Advanced Server, aunque para pruebas o desarrollo puede ser completamente válida la versión Professional.

IIS 5.0 ofrece una administración muy sencilla que se realizará mediante el Administrador de servicios de Internet.

La versión 5.0 de IIS permite que el desarrollo de aplicaciones Web sea mucho más robusto y la creación de sitios Web sea más configurable y completa. Ofrece un entorno escalable basado en los componentes cliente/servidor que se pueden integrar dentro de las aplicaciones Web.

Internet Information Server 5.0 es el servidor Web más rápido y recomendable para la plataforma Windows 2000, ya que se encuentra integrado completamente con el Servicio de Directories de Windows 2000, esta combinación del servicio Web con los servicios del sistema operativo permite desarrollar aplicaciones basadas en la Web fiables y escalables.

## Instalando IIS 5.0

Al instalar Windows 2000, en cualquiera de sus versiones, deberemos indicar que deseamos instalar también el componente de Windows 2000 llamado Servicios de Internet Information Server (IIS). Este componente va a convertir nuestro equipo en un servidor Web que soporte aplicaciones ASP.

Para averiguar si tenemos instalado IIS 5.0 en nuestra máquina podemos realizar una sencilla prueba que consiste en escribir la siguiente URL <http://nombreEquipo> en el navegador Web. Si IIS 5.0 está instalado aparece una ventana similar a la Figura 68.

Otra forma de comprobarlo es acudiendo al menú de inicio de Windows y en el grupo de programas Herramientas administrativas, debemos tener el Administrador de servicios de Internet.



Figura 68. IIS 5.0 está instalado

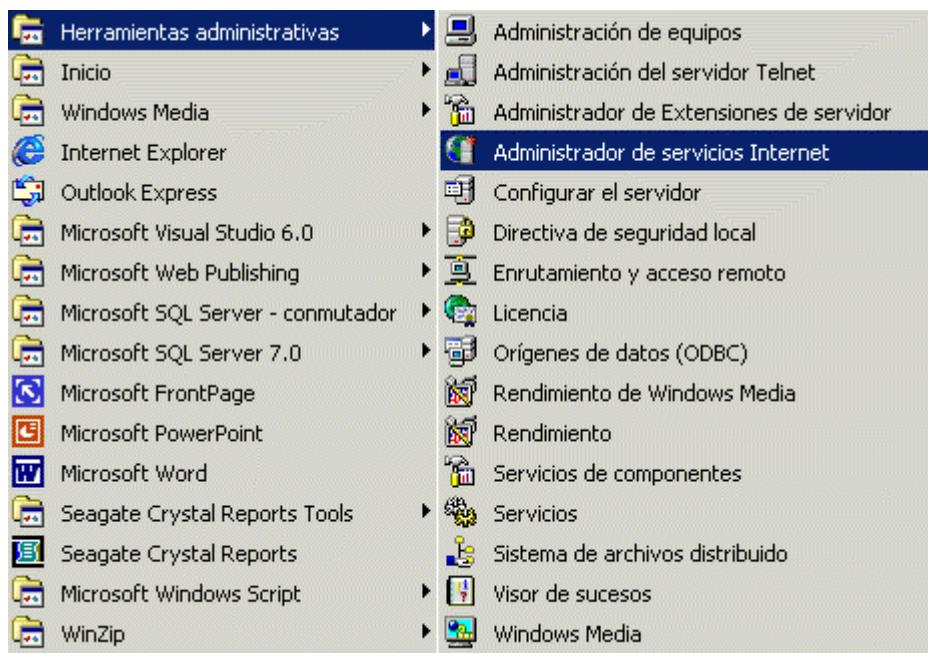


Figura 69. El Administrador de servicios de Internet en el menú de Inicio

Si observamos que IIS 5.0 no se encuentra instalado en nuestra máquina deberemos ir al Panel de Control y en la opción Agregar/quitar programas seleccionar componentes de Windows. Una vez hecho esto marcaremos el componente Servicios de Internet Information Server y procederemos a su instalación.

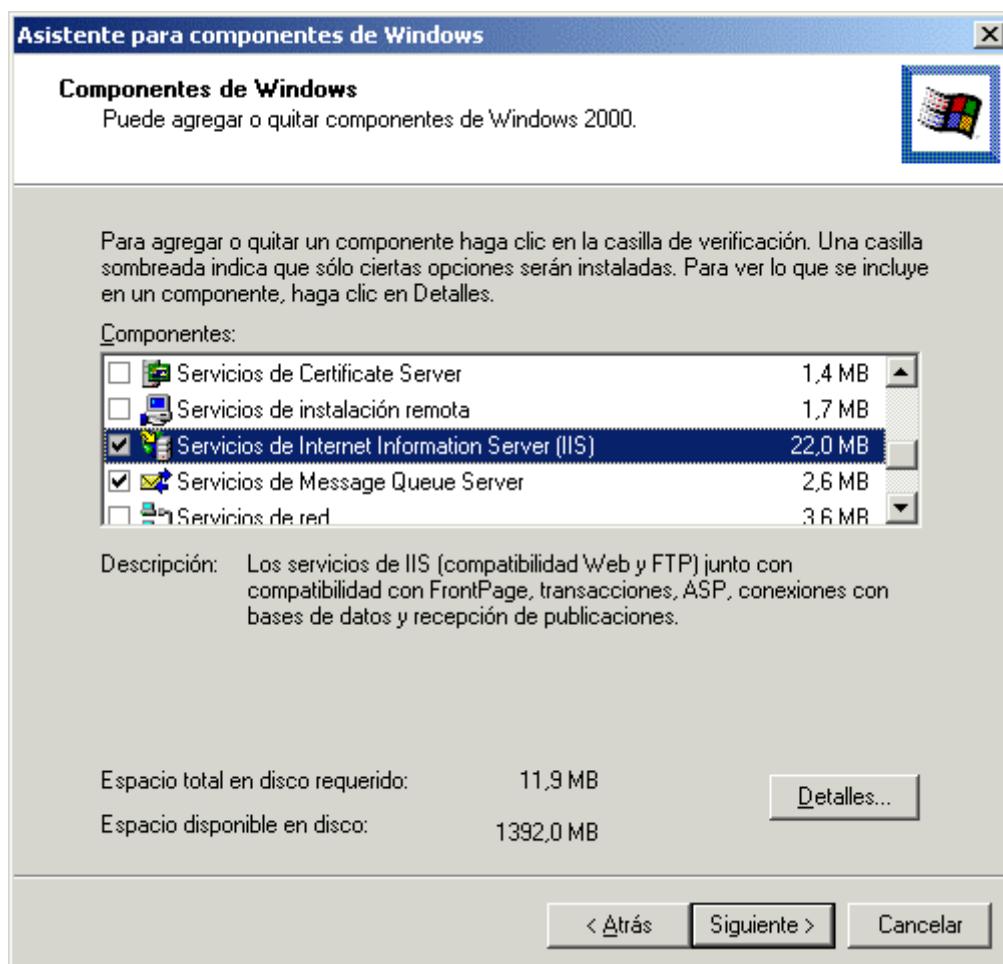


Figura 70. Instalando IIS 5.0

La instalación de IIS 5.0 es muy sencilla, lo que nos pedirá será la ruta del directorio de publicación en Internet y poco más, por defecto esta ruta es c:\Inetpub\wwwroot.

## Novedades de IIS 5.0

Realmente no existen grandes diferencias ni novedades dramáticas respecto a la versión anterior de IIS. En nuestro caso sólo vamos a comentar las novedades que nos interesen desde nuestro punto de vista, es decir, desde ASP.

Un adelanto importante fue el que ofreció IIS 3.0 ya que fue el primer servidor Web que soportaba páginas ASP y el que introdujo esta tecnología. IIS 3.0 era poco robusto y su configuración y administración se encontraba bastante restringida a unas pocas opciones.

La versión 4.0 de IIS supuso un gran paso dentro de la política de Microsoft respecto a Internet, ya que IIS 4.0 era mucho más completo que su predecesor, permitiendo una administración y configuración bastante detallada. También resulta mucho más robusto y escalable.

IIS 4.0 implementaba el concepto de aplicación Web, permitiendo aislar las mismas y ofrecía también distintas formas de ejecución para una aplicación ASP. Las aplicaciones ASP se podían configurar desde IIS 4.0, cosa que no ocurría con IIS 3.0.

En la versión 4.0 de IIS se eliminó el anticuado servicio Gopher y se incluyó la versión 2.0 de las páginas activas de servidor. Gracias a la integración de IIS 4.0 con el servidor de transacciones MTS 2.0 (Microsoft Transaction Server) se podían crear páginas ASP transaccionales, fue en este momento cuando apareció el objeto integrado de ASP ObjectContext, que ya hemos comentado en su capítulo correspondiente, y que volveremos a comentar más adelante.

Y por fin llegamos a la versión actual de IIS, la versión 5.0. Esta versión al igual que las predecesoras ofrece una nueva versión de ASP, la versión 3.0 que es precisamente el tema principal de este curso.

Como ya adelantábamos IIS 5.0 no supone un gran cambio respecto a IIS 4.0, vamos a comentar alguna de las novedades que aporta la nueva versión del servidor Web de Microsoft. Con IIS 5.0 se permite una nueva forma de definir las aplicaciones ASP que consiste en definir aplicaciones ASP para que se ejecuten de forma aislada al servicio Web pero agrupadas, de todas formas en el apartado dedicado a la configuración de aplicaciones ASP desde IIS 5.0 se comentará esta característica.

Se ha eliminado el servidor de transacciones MTS 2.0, ahora ha pasado a formar parte de lo que se denomina Servicios de componentes. IIS 5.0 también se encuentra perfectamente integrado con Servicios de componentes aunque no se encuentran en la misma herramienta de administración.

IIS 5.0 es más robusto y escalable que su predecesor, pudiéndose incluso recuperar de fallos de forma automática.

Las extensiones de FrontPage, necesarias para la creación de proyectos Web desde Visual InterDev en el servidor, se encuentran integradas de forma completa en la herramienta administrativa de IIS 5.0 que se denomina Administrador de servicios de Internet, y que posee el mismo aspecto que la consola MMC de la versión 4.0. Ya no es posible desde el menú de Inicio tener acceso al sitio Web de administración de IIS 5.0, sin embargo este Web de administración sigue existiendo y lo veremos en próximos apartados.

## El administrador de servicios de internet

Internet Information Server 5.0 ofrece dos herramientas de administración, por un lado podemos utilizar una herramienta denominada Administrador de servicios de Internet, que es similar a lo que ya existía en IIS 4.0, es decir, se trata de un complemento de la consola MMC (Microsoft Management Console), y por otro lado podemos administrar IIS 5.0 a través de una versión del administrador de servicios Internet basada en la administración a través de Web, utilizando lo que se denomina sitio Web de administración.

La administración remota a través de Web es posible gracias al sitio Web de administración que ofrece IIS 5.0 como parte de su instalación. Por defecto este sitio Web se encuentra restringido a la máquina local, es decir, a la dirección IP 127.0.0.1 o al nombre de equipo localhost (máquina local). Esta restricción es lógica, ya que no desearemos que la administración de nuestro servidor Web pueda ser realizada por cualquier usuario anónimo de Internet.

La administración remota puede ser muy útil cuando se produzca algún problema en el servidor y el administrador del mismo no se encuentre físicamente en el mismo lugar, el problema lo podrá resolver cómodamente desde el lugar en el que se encuentre.

Se debe tener en cuenta que la dirección IP de la máquina en la que se encuentre el administrador debe tener el acceso concedido al sitio Web de administración y debe pertenecer a los operadores de sitios Web.

El sitio Web de administración poseerá el mismo nombre que el sitio Web predeterminado del servidor en el que esté instalado IIS 5.0, pero se diferenciará en el número de puerto asignado al sitio Web de administración. Al instalar IIS 5.0 y crear el sitio Web de administración se le asigna un número de puerto aleatorio. Un poco más adelante veremos como averiguar este número de puerto y así poder conocer la dirección del sitio Web de administración y utilizarlo para administrar IIS 5.0 de forma remota.

Aunque la administración a través de este sitio Web ofrece muchas de las funciones recogidas por el Administrador de servicios de Internet, hay una serie de facilidades de administración que no se encuentran disponibles, ya que requieren coordinación con el sistema operativo.

Vamos a comentar la herramienta de Administración de servicios de Internet. La ejecutamos desde del menú Inicio|Herramientas administrativas|Administrador de servicios Internet. Esta herramienta de administración, nos va a permitir administrar y configurar en un mismo entorno todos los servicios del servidor IIS 5.0.el uso de esta herramienta es bastante sencillo e intuitivo.

La consola que ofrece el Administrador de servicios de Internet se divide en dos paneles, el panel de la izquierda se denomina panel de alcance (*scope pane*), y el de la derecha panel de resultados. El panel de alcance presenta una jerarquía en forma de árbol con los componentes que se pueden administrar en la consola.

En el panel de alcance se representan los servidores que se pueden administrar mediante un icono de un ordenador acompañado del nombre del servidor.

Cada uno de los nodos del árbol que aparece en el panel de alcance son instancias de servicios individuales y pueden contener a su vez un conjunto de subobjetos administrables. Podemos abrir las páginas de propiedades de cualquier nodo pulsando el botón derecho sobre le mismo y seleccionando la opción Propiedades del menú desplegable.

El segundo panel que aparece en la consola es el de resultados, este panel visualiza los contenidos o resultados referentes al nodo seleccionado en el panel de alcance, de manera que se puede acceder a todo tipo de información relacionada con cada nodo. En la Figura 71 se puede ver el aspecto que muestra el Administrador de servicios de Internet.

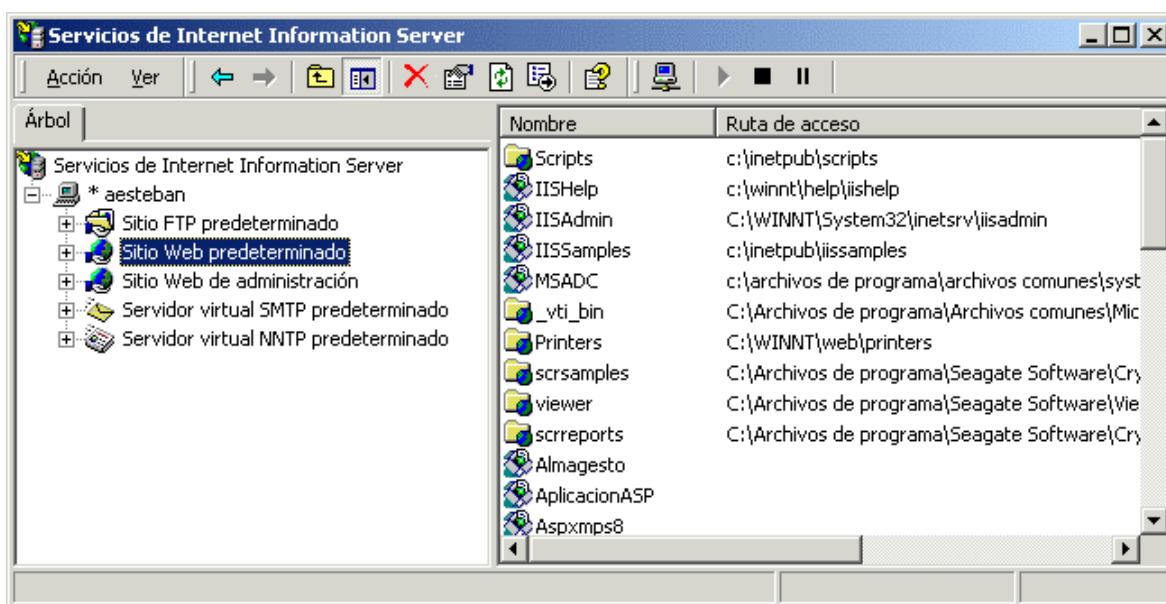


Figura 71. Administración de los servicios de IIS

Desde la consola no sólo podremos realizar la administración local de un servidor, sino que podremos realizar la administración remota de otros servidores IIS 5.0.

Para administrar un servidor remoto deberemos pulsar con el botón derecho sobre el nombre del servidor y seleccionar la opción conectar o bien sobre el icono de la barra de herramientas que representa un equipo.

A continuación aparecerá una ventana en la que se nos pide el nombre del equipo al que nos queremos conectar, una vez indicado el nombre y pulsado el botón de aceptar, de forma automática se añadirá otro elemento servidor a la consola identificado con su nombre de equipo correspondiente.

En la Figura 72 se puede ver el aspecto que muestra un servidor local y dos servidores remotos, para distinguirlos el servidor local se señala con un asterisco (\*) y la apariencia del icono que lo representa es diferente.

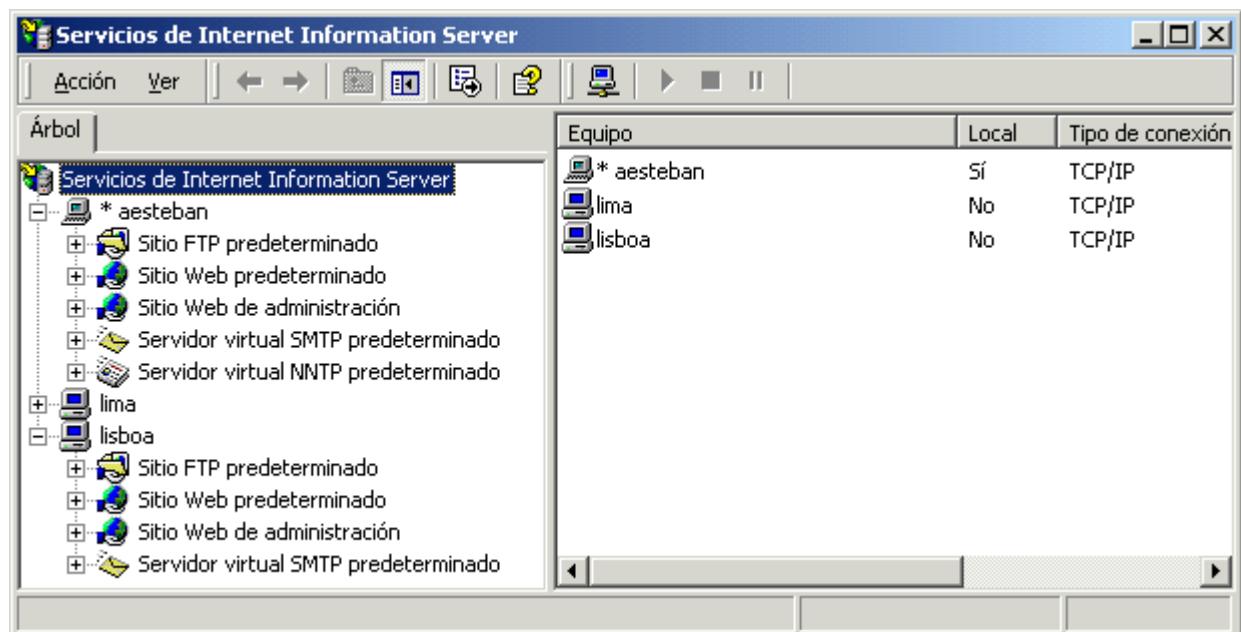


Figura 72. Administración remota de servidores IIS 5.0

Como ya se ha comentado, la utilización de la consola es muy sencilla lo único que hace falta es realizar una serie de pruebas para familiarizarse con ella.

La configuración de Internet Information Server se basa en la utilización de hojas de propiedades, un ejemplo se puede ver en la figura 6. A estas hojas de propiedades (*Property Sheets*) accederemos pulsando con el botón derecho del ratón, sobre el elemento a configurar, y seleccionando la opción Propiedades que aparece en el menú contextual.

Cada hoja de propiedades mostrará una serie de parámetros para configurar, dependiendo del elemento seleccionado (directorio, sitio Web o fichero). Las hojas de propiedades tendrán también una serie de pestañas o fichas que intentan facilitarnos la administración de los servicios agrupando en cada ficha parámetros configurables relacionados. Se pueden definir propiedades para sitios Web, directorios e incluso ficheros. En la Figura 74 se puede apreciar el aspecto que presentaría la hoja de propiedades correspondiente a un fichero.

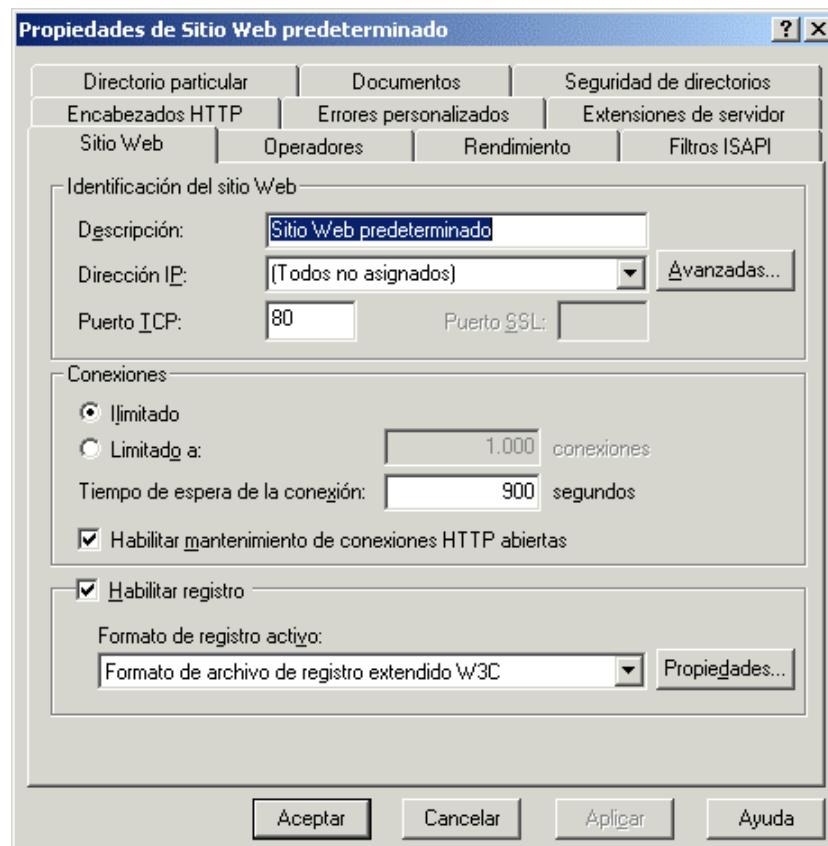


Figura 73. Hoja de propiedades de un sitio Web

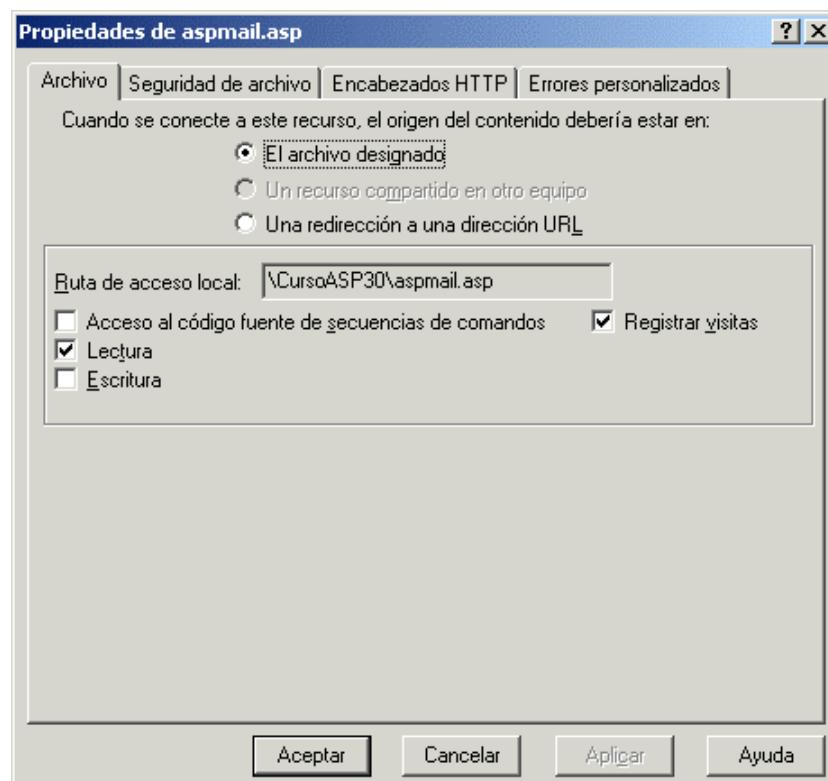


Figura 74. Hoja de propiedades de un fichero

## Elementos de IIS 5.0

Dentro del Administrador de servicios de Internet se pueden distinguir varios nodos, dependiendo también de la instalación que se haya realizado de IIS 5.0. En la instalación más completa vemos lo siguiente:

- Sitio FTP predeterminado: es el sitio FTP del servidor, desde aquí configuraremos el servicio FTP que ofrece IIS 5.0.
- Sitio Web predeterminado: es el elemento que más nos va a interesar, desde aquí se configura el servicio Web que ofrece IIS 5.0, incluyendo la configuración de las aplicaciones ASP.
- Sitio Web de administración: nos ofrece la posibilidad de administrar IIS 5.0 desde un sitio Web, como ya hemos comentado anteriormente.
- Servidor virtual SMTP predeterminado: representa el servicio de correo de IIS 5.0, se trata del servicio de correo saliente SMTP (Simple Mail Transfer Protocol). En el siguiente capítulo veremos como interactuar con este servicio desde ASP a través de una serie de componentes denominados CDONTS (Collaboration Data Object for NT Server).
- Servidor virtual NNTP predeterminado: representa el servicio de noticias de IIS 5.0, se trata del servicio NNTP (Network News Transport Protocol).

En este capítulo nos vamos a centrar en el sitio Web predeterminado, que representa el sitio Web por defecto del servidor IIS 5.0 y que se corresponde con <http://nombreMaquina>.

Para averiguar en qué puerto se encuentra el sitio Web de administración y así poder utilizarlo de forma remota, pulsaremos con el botón derecho del ratón sobre este elemento y seleccionaremos la opción Propiedades. En la hoja de propiedades del sitio Web de administración parece el número de puerto (puerto TCP) que tiene asignado. Y para acceder desde el navegador al Web de administración simplemente escribiremos: <http://localhost:numPuerto> o si queremos administrar IIS desde otra máquina escribiremos: <http://nombreServidor:numPuerto>.

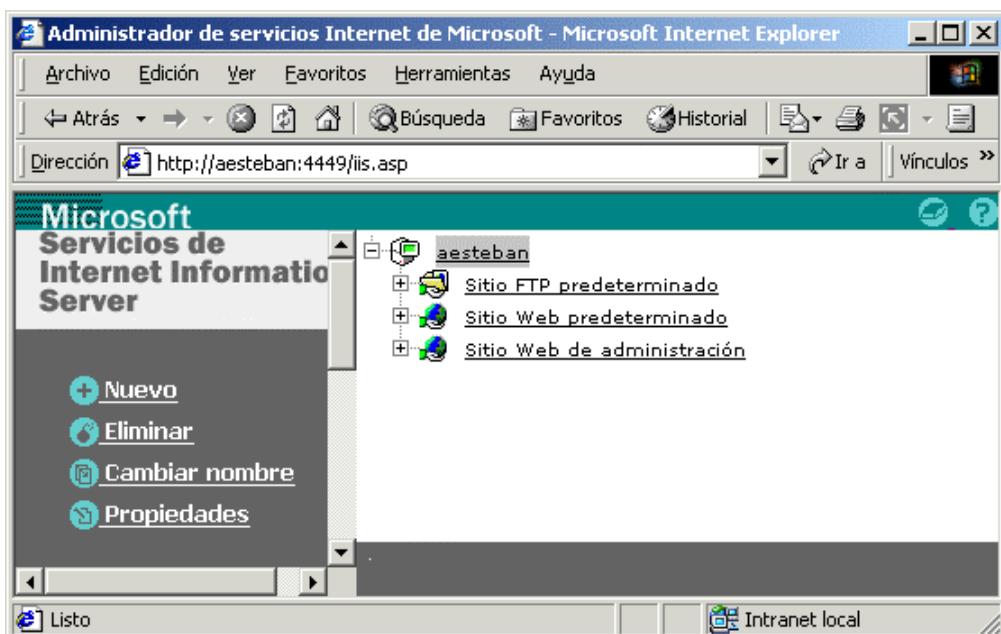


Figura 75. El sitio Web de administración

## Administración del sitio web

Cuando creamos un sitio Web lo hacemos con la intención de hacer disponible una serie de información más o menos compleja (desde páginas HTML estáticas hasta aplicaciones Web basadas en páginas ASP) para una serie de usuarios, ya sea para nuestra propia empresa u organización (Intranet) o para un grupo de usuarios más numeroso y general (Internet). A través del sitio Web ofrecemos la información y aplicaciones Web que se encuentren dentro de un directorio determinado en el servidor Web, es decir, en el servidor en el que tenemos instalado IIS 5.0.

Este directorio a partir del cual se publican los documentos que se encuentran en él en el entorno de Internet o Intranet se denomina directorio de publicación o particular (*home*), o también lo podemos encontrar como directorio raíz.

Para publicar documentos en el sitio Web, si este sitio Web únicamente consta de archivos ubicados en un disco duro del equipo (servidor IIS 5.0), simplemente deberá copiar los ficheros correspondientes al directorio de publicación del sitio Web. En el caso del sitio Web predeterminado, si se han respetado los valores por defecto en la instalación de IIS, su directorio de publicación (a partir de ahora vamos a utilizar siempre este término en lugar de directorio raíz o particular) será c:\Inetpub\wwwroot.

Cada sitio Web es necesario que tenga un directorio de publicación, en este directorio se suele ubicar la página que aparece por defecto cuando indicamos el nombre del sitio Web.

Cada sitio Web debe tener asignados una dirección IP y un número de puerto que servirán para identificar el sitio Web, junto con el nombre de encabezado de host, entre los diferentes sitios Web que tengamos hospedados en nuestro servidor.

Dentro del sitio Web podremos definir una serie de propiedades cuyos valores se aplicarán a todos los elementos contenidos en el sitio Web (ficheros o subdirectorios) o no, es decir, si modificamos una propiedad en el sitio Web tenemos la posibilidad de elegir si deseamos que el valor se propague al resto de los elementos o no.

A continuación vamos a proceder a comentar las propiedades más relevantes que podemos encontrar en la hoja de propiedades de un sitio Web. No se va a tratar de una descripción detallada, ya que no es el objetivo del curso, aunque en el apartado correspondiente si que vamos a ver en detalle la configuración de las aplicaciones ASP y su creación.

## Sitio Web

Si seleccionamos el sitio Web predeterminado y pulsamos sobre él con el botón derecho del ratón podremos seleccionar la opción Propiedades del menú que aparece. De esta forma podremos observar las diferentes propiedades que podemos configurar a nivel de sitio Web.

La primera hoja que podemos observar dentro de las hojas de propiedades de un sitio Web es la identificada mediante Sitio Web. En esta ficha, podremos configurar una serie de parámetros relacionados con la identificación del sitio Web, las conexiones al sitio Web y la forma de registrar los accesos a nuestro sitio Web.

En cuanto a la identificación del sitio Web tenemos cuatro parámetros: Descripción, Dirección IP, Puerto TCP y Puerto SSL. En Descripción indicaremos el nombre con el que identificaremos nuestro sitio dentro de la consola, no se debe confundir con el nombre de dominio o URL que utilizamos para

acceder al sitio Web a través de Internet. De esta forma podremos cambiar la denominación de sitio Web Predeterminado por la que deseemos.

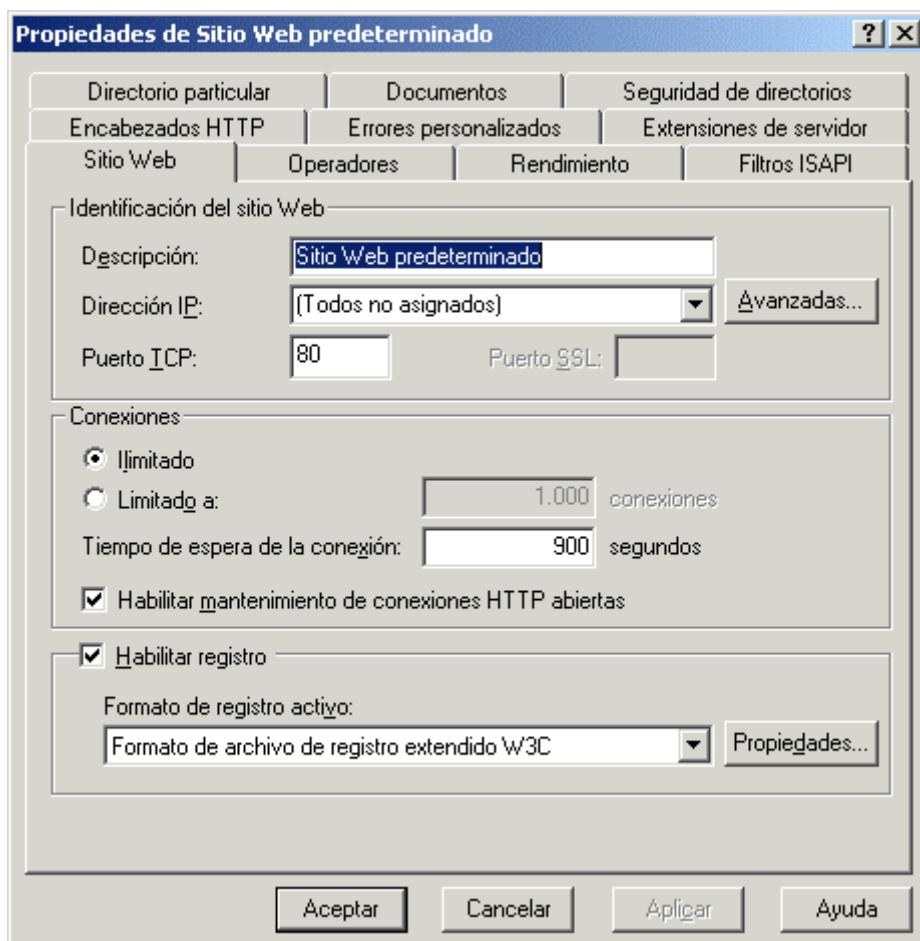


Figura 76. Propiedades del sitio Web

El siguiente parámetro es la dirección IP que tiene asociada el sitio Web, podemos elegir una de las que tenga asignadas el servidor o bien elegir la opción Todos no asignados, si elegimos esta última opción estaremos asignando al sitio Web todas las direcciones IP del servidor que se encuentren libres. De esta forma pasa a ser el sitio Web predeterminado.

El número de puerto normalmente no será necesario modificarlo, por defecto el servicio Web se encuentra en el puerto 80.

El puerto SSL (Secure Sockets Layer) sólo se debe utilizar para conexiones seguras, basados en certificados de cliente y servidor.

En la conexión a nuestro sitio Web podremos limitar el número de conexiones de clientes o bien indicar que aceptamos un número ilimitado de ellas. También podremos indicar el tiempo de espera máximo (time-out) utilizado para establecer una conexión, esto asegurará que se cerrarán todas las conexiones si el protocolo HTTP no puede establecer una conexión desde el cliente en el tiempo especificado, por defecto son 900 segundos.

Podremos activar el registro de nuestro sitio Web seleccionando la opción correspondiente. Tenemos diferentes formatos para guardar la información relativa los accesos a nuestro sitio Web. Cada uno de

estos formatos definen que datos se van a almacenar y que estructura va a tener el fichero o tabla en la que van a ser almacenados.

## Directorio particular

Esta hoja de propiedades es utilizada para configurar el directorio de publicación del sitio Web y se puede observar en la Figura 77.

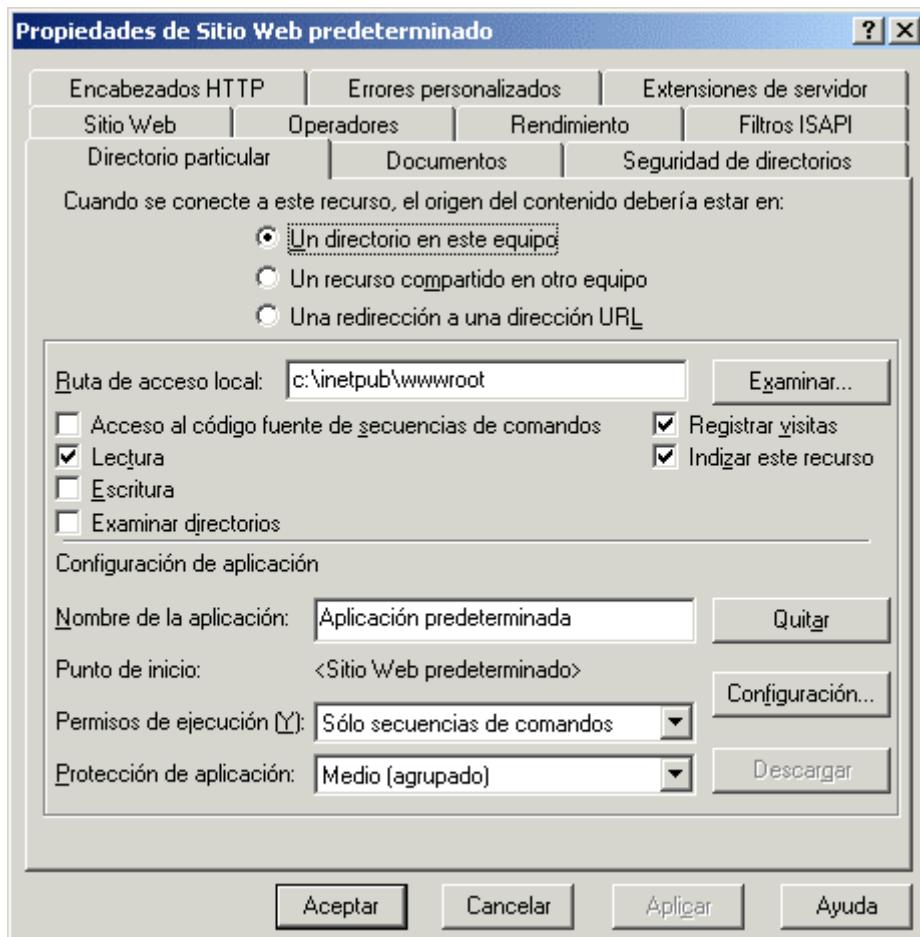


Figura 77. Propiedades del directorio de publicación en Internet

En esta hoja podremos modificar la ubicación del directorio de publicación del sitio Web, por defecto el directorio de publicación es c:\Inetpub\wwwroot. Se tomará el directorio de publicación indicado si está seleccionada la opción de Un directorio en este equipo, también podremos indicar que el directorio de publicación se encuentra en un recurso compartido de red (Un directorio ubicado en otro equipo) o bien podremos redirigir a todos nuestros clientes a una URL determinada.

También se pueden configurar una serie de parámetros para indicar si queremos registrar la actividad del sitio Web, indizar su contenido, y el tipo de acceso que va a tener el usuario.

Para que el usuario pueda ver el listado de directorios que se corresponde con nuestro sitio Web deberemos activar la casilla Permitir examinar directorios, de esta forma si el usuario escribe el nombre de dominio en el que se encuentra nuestro sitio Web sin especificar ninguna página y si el documento predeterminado no está activado en el directorio especificado, aparecerá el listado de

directorios en forma de una lista de vínculos de hipertexto correspondientes a los archivos y subdirectorios del directorio virtual actual. Esta opción suele estar desactivada, ya que de esta forma revelaríamos a nuestros usuarios la estructura exacta de nuestro sitio Web, algo que puede llegar a ser peligroso desde el punto de vista de la seguridad.

Si queremos registrar el acceso al directorio raíz debemos activar la casilla Registrar visitas, se utilizará el formato de log que se encuentre definido para el sitio Web.

Podemos observar, también, que tenemos una opción llamada Permisos de ejecución, pero estos permisos son relativos a la ejecución de aplicaciones ASP. Para no permitir la ejecución de programas ni secuencias de comandos en el directorio seleccionaremos la opción Ninguno. Si queremos que se ejecuten secuencias de scripts asignadas a un motor de secuencias de comandos en este directorio seleccionaremos la opción Sólo secuencias de comandos, este sería el caso para las páginas ASP. Y si queremos ejecutar cualquier aplicación dentro de este directorio seleccionaremos la opción Sec. comandos y ejecutables.

La configuración de la aplicación la veremos en detalle en el apartado correspondiente.

## Documentos

En esta hoja podremos indicar la página por defecto del sitio Web. Cuando un usuario no especifique en el navegador ningún fichero en concreto de nuestro sitio Web se le mostrará esta página por defecto. Podemos indicar diferentes documentos predeterminados dentro de la lista que se puede apreciar en la Figura 78, el orden en el que se indican los documentos es significativo, el servidor Web devolverá el primero que encuentre. El orden de prioridad de los documentos lo podemos modificar utilizando las flechas. El documento predeterminado podrá ser tanto una página HTML como una página ASP.

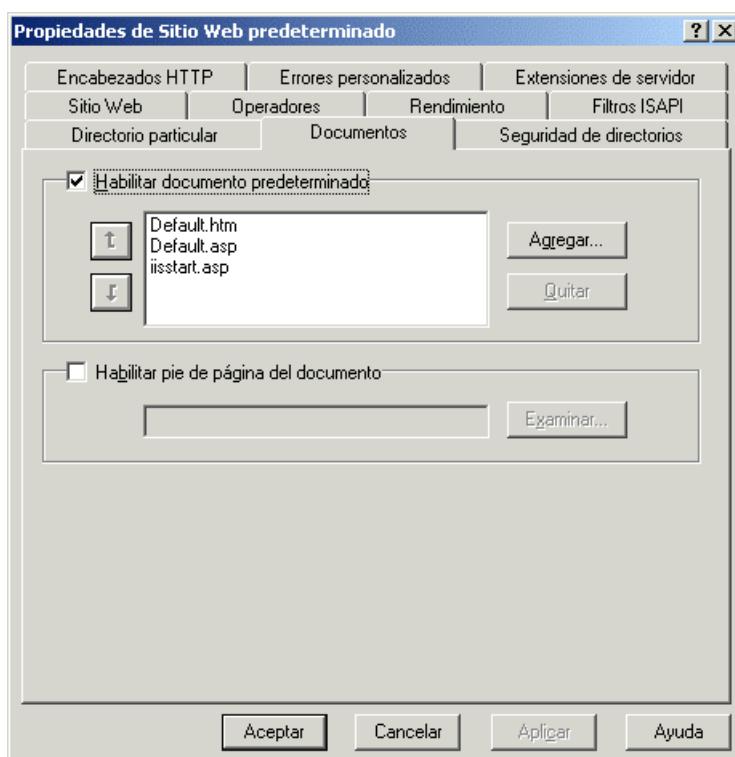


Figura 78. Hoja de propiedades Documentos

Mediante la opción Habilitar pie de página del documento podremos configurar el servidor Web para que inserte de manera automática un fichero en formato HTML en la parte inferior de todos los documentos del sitio Web.

## Operadores

En esta hoja podemos definir qué cuentas de usuarios del dominio que pertenecen a la figura especial del operador Web. Un operador es una cuenta de usuario de Windows 2000 que tiene permisos para alterar la dinámica del servicio Web, es decir, iniciar, detener o pausar el sitio Web.

Para añadir una cuenta para que actúe de operador del sitio Web pulsaremos únicamente el botón Agregar y seleccionaremos la cuenta de usuario de Windows correspondiente, podremos indicar tanto cuentas como grupos de cuentas de usuarios. Para eliminar una cuenta de los operadores se selecciona de la lista y se pulsa el botón Quitar. La hoja de propiedades Operadores la podemos observar en la Figura 79.

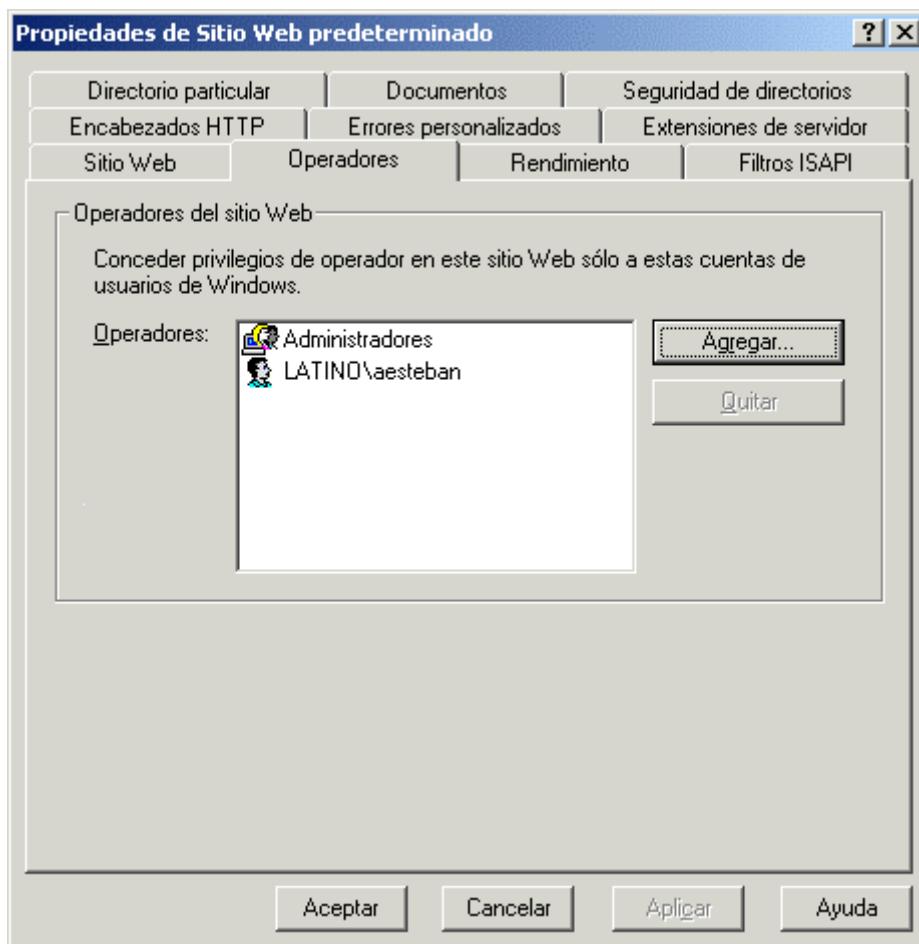


Figura 79. Operadores del sitio Web

Por defecto el grupo Administradores de dominio de Windows se encuentra dentro de los operadores del sitio Web.

## Errores personalizados

IIS 5.0 permite personalizar los mensajes de error del protocolo HTTP que se envían a los clientes. A través de la hoja de propiedades Errores personalizados podemos indicar los diferentes mensajes de error que se enviarán al cliente. Estos mensajes de error personalizados pueden tener forma de fichero o de dirección URL. Esto ya lo vimos al utilizar el nuevo objeto integrado del modelo de objetos de ASP, el objeto ASPError.

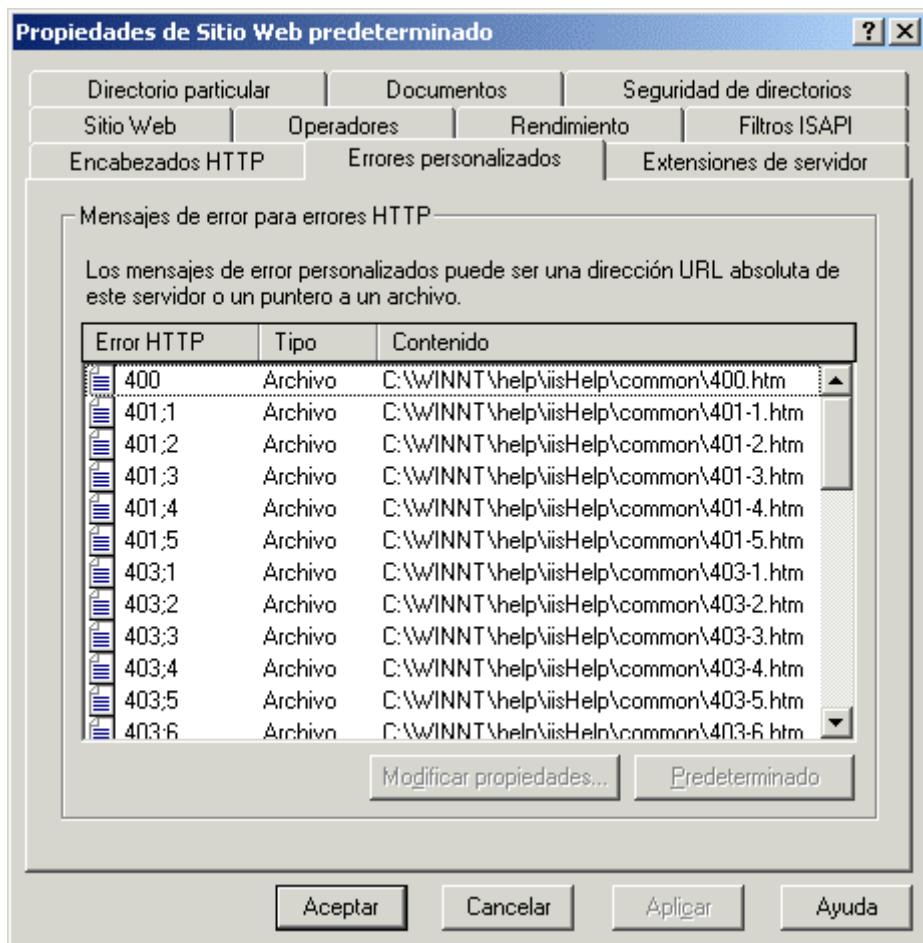


Figura 80. Errores personalizados

## Rendimiento

Para ajustar el rendimiento de nuestro servidor Web podemos utilizar la hoja de propiedades Rendimiento. En esta hoja podemos configurar el ajuste del rendimiento de nuestro servidor atendiendo al número de accesos que se espera por día, disponemos de tres opciones: menos de 10.000, menos de 100.000 y más de 100.000. De esta forma el servidor podrá ajustar los recursos de una manera más apropiada.

También podemos limitar el ancho de banda a los KB/s que deseemos. Esta opción nos permite racionalizar el ancho de banda y repartirlo entre diversos servicios de Internet (sitios Web, FTP...).

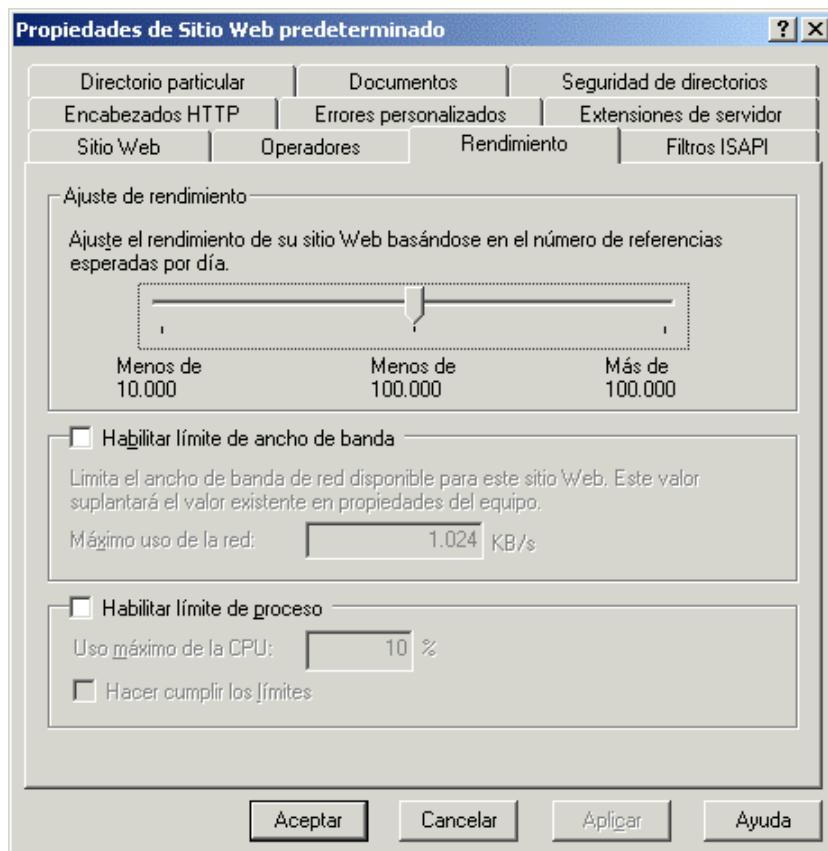


Figura 81. Rendimiento del sitio Web

Y como novedad de IIS 5.0 también se puede asignar un límite de proceso de uso de la CPU expresado en tanto por ciento, también se puede forzar a cumplir los límites establecidos. El límite de proceso se aplica a las aplicaciones ASP que se han definido como de ejecución aislada.

## Seguridad de directorios

En la hoja de propiedades etiquetada como Seguridad de directorios, configuraremos la seguridad de nuestro sitio Web. Se pueden definir distintos niveles de seguridad atendiendo a diferentes criterios, también se puede tener una combinación de la seguridad ofrecida por IIS 5.0 y Windows 2000 a través de NTFS, no debemos olvidar que IIS se encuentra integrado con el sistema operativo. Esta hoja se divide en tres partes, pasamos a comentar cada una de ellas.

En la opción Control de autenticación y acceso anónimo si pulsamos el botón Modificar podremos configurar las características de acceso anónimo y autenticación del servidor Web. Para que el servidor pueda autenticar los usuarios, primero se deben crear cuentas de usuario válidas de Windows 2000 y luego asignar permisos a los ficheros a través de NTFS.

A través de la opción Restricciones de nombre de dominio y dirección IP, podremos permitir o denegar el acceso a diferentes equipos o grupos de equipos atendiendo a sus direcciones IP.

Y en la última opción Comunicaciones seguras crearemos un par de claves SSL (Secure Sockets Layer) y la petición de un certificado de servidor.

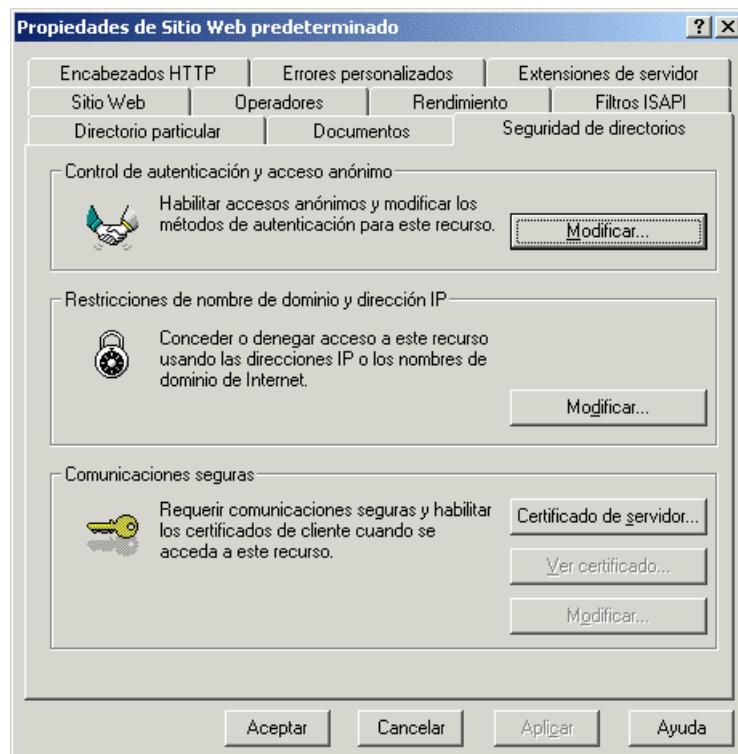


Figura 82. Seguridad de directorios

## Filtros ISAPI

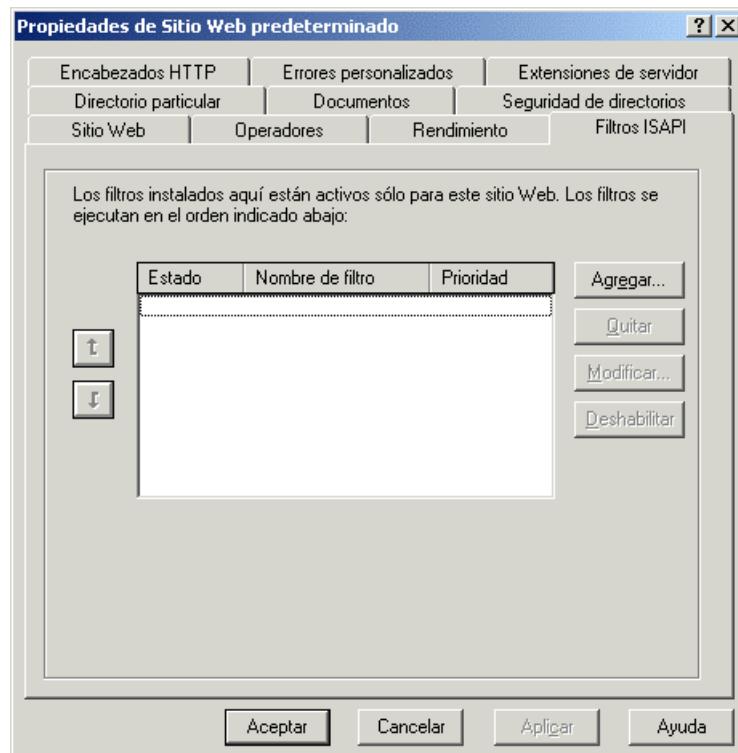


Figura 83. Filtros ISAPI

Para instalar filtros ISAPI (Internet Server Application Program Interface) utilizaremos la hoja de propiedades Filtros ISAPI. Un filtro ISAPI es un programa que responde a sucesos durante el procesamiento de una petición HTTP y está cargado siempre en la memoria del sitio Web. Físicamente son ficheros .DLL. Por defecto no hay ningún filtro ISAPI instalado.

## Encabezados HTTP

En esta hoja de propiedades vamos a poder establecer los valores de las cabeceras HTTP que se envíen a nuestros clientes (navegadores Web).

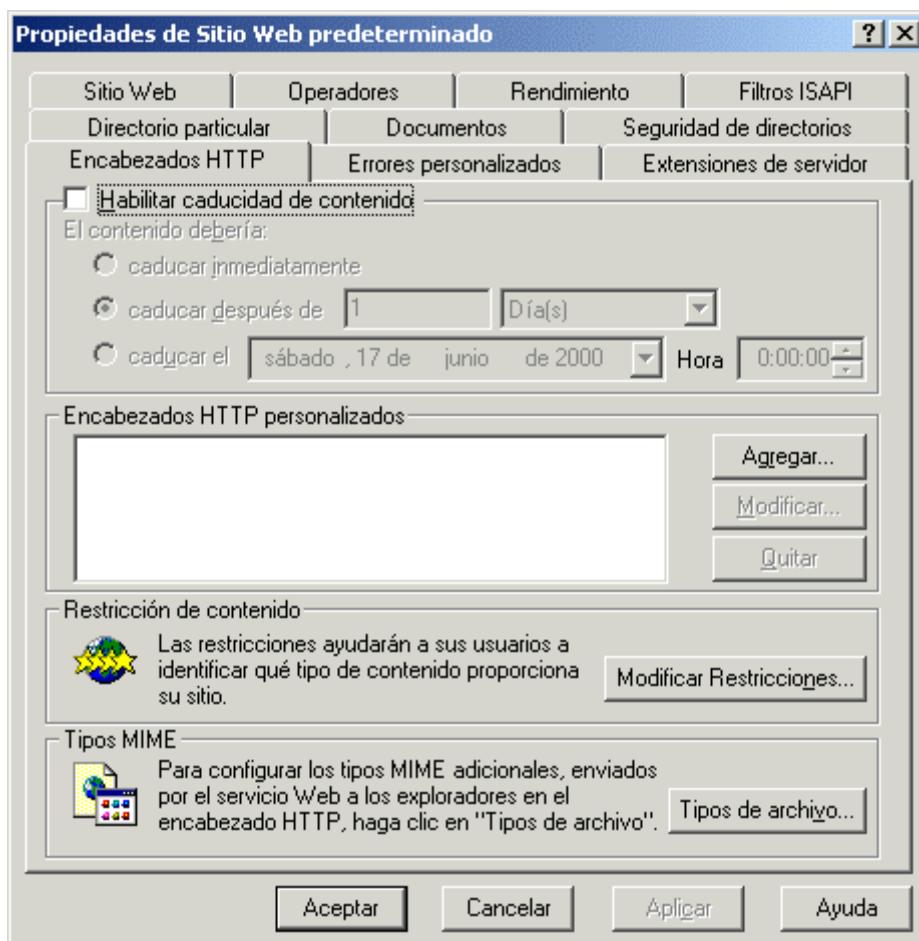


Figura 84. Encabezados http

Si activamos la casilla de verificación Habilitar caducidad de contenido nos permitirá establecer el momento en el que la información de la página se considera no válida u obsoleta (ha caducado), el navegador Web comparará la fecha del sistema con la de la página que le llega y determinará si se visualiza la página almacenada en la caché o bien pedirá una actualización de la misma.

En la propiedad *Encabezados HTTP personalizados* podremos enviar un encabezado HTTP personalizado del servidor Web al navegador del equipo cliente.

Las páginas Web pueden incorporar cabeceras con información que identifique sus contenidos desde un punto de vista moral, la configuración de estas cabeceras la realizaremos a través del apartado Restricción de contenido. De esta manera los usuarios pueden filtrar las páginas en función de ese

contenido. IIS utiliza un método desarrollado por el organismo RSAC (Recreational Software Advisory Council) que clasifica los contenidos atendiendo a diversos grados de violencia, pornografía y lenguaje ofensivo. Para establecer estas restricciones de contenido se debe pulsar el botón Modificar Restricciones.

En la opción Tipos MIME (Multipurpose Internet Mail Extensions) podemos determinar los tipos de archivos que se enviarán al cliente Web.

## Extensiones de servidor

Desde esta hoja de propiedades, que es novedad en IIS 5.0, podemos configurar algunos de los parámetros relativos a las extensiones de servidor de FrontPage.

Las extensiones de FrontPage nos van a servir para conectarnos al sitio Web con clientes como el Explorador de FrontPage, el Editor de FrontPage y la herramienta de desarrollo Visual InterDev, para desarrollar páginas HTML o páginas ASP, y manipular los sitios Web.

En la versión 5.0 de IIS estas extensiones se encuentran completamente integradas con el Administrador de servicios de Internet. Las extensiones de servidor de FrontPage se instalan conjuntamente con IIS 5.0.

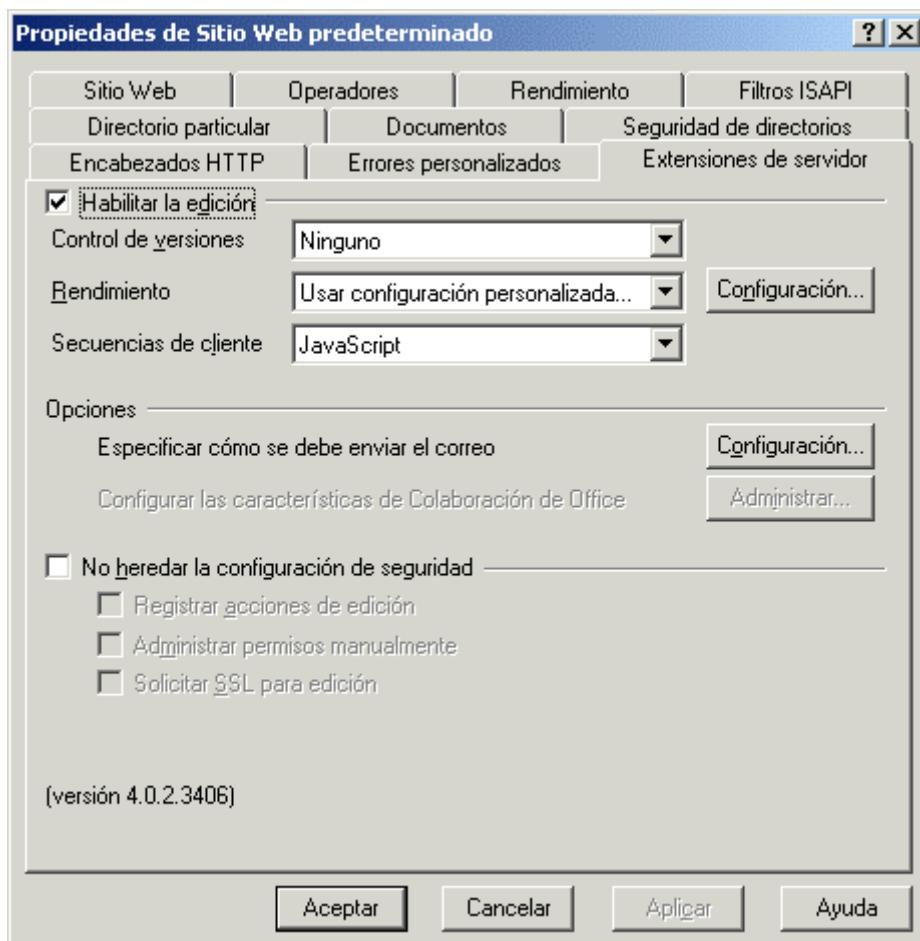


Figura 85. Extensiones de servidor de FrontPage

En el caso de no estar instaladas en el sitio Web predeterminado no parecerá esta hoja de propiedades. Para instalar las extensiones de servidor acudiremos al sitio Web predeterminado y pulsando con el botón derecho del ratón seleccionaremos la opción de menú Todas las tareas| Configurar las Extensiones de servidor. Esta opción lanzará un wizard que nos guiará en el proceso de instalación que resulta sencillo.

En la figura 18 podemos ver la hoja de propiedades que nos permite configurar parte de las extensiones de servidor de FrontPage. Para permitir que los autores modifiquen contenidos del sitio Web deberemos activar la opción Habilitar la edición, es recomendable desactivar esta opción una vez que el sitio Web se haya finalizado y ya se encuentre en producción. También podemos indicar que se lleve un control de versiones sobre las diferentes modificaciones realizadas sobre las páginas por los autores, el tipo de script de cliente por defecto y también opciones relativas al rendimiento del sitio Web.

Otros parámetros de esta hoja de propiedades permiten especificar la configuración del correo del sitio Web, que se utilizará para algunos formularios, para indicar un dirección de contacto etc. Otro parámetro nos permite especificar si un subWeb va a heredar o no la configuración de seguridad del Web padre. Dentro de estas opciones de seguridad podemos especificar si queremos registrar las acciones realizadas por los autores, si es necesario o no la conexión segura a través de SSL para proteger los datos enviados y si se va a permitir o no administrar manualmente los permisos a través de las listas de acceso (ACL).

Además de configurar las extensiones de FrontPage desde esta hoja de propiedades podemos manipularlas pulsando son el botón derecho del ratón sobre el directorio o sitio Web deseado y en la opción Todas las tareas vemos que aparecen acciones relacionadas con la administración de las extensiones de FrontPage (Configurar extensiones de servidor, Comprobar extensiones de servidor, etc.), también aparecen algunas opciones relacionadas con las extensiones dentro del menú Nuevo (Administrador de extensiones de servidor, Web de extensiones de servidor, etc.).

## La aplicación web

Aquí volvemos a retomar el término de aplicación Web, que ya hemos tratado en capítulos anteriores. En este apartado veremos como configurar una aplicación ASP desde IIS 5.0.

Podemos definir una aplicación Web como un conjunto de páginas Web en combinación con scripts de servidor que permiten realizar una serie de tareas como si de una aplicación convencional cliente/servidor se tratase.

Desde el punto de vista físico y de administración del servidor Web, una aplicación es un conjunto de ficheros que se ejecutan dentro de un conjunto definido de directorios del sitio Web, es decir, una aplicación se corresponde con una estructura de directorios del sitio Web, y desde la consola de administración de IIS 5.0 deberemos indicar el punto de arranque de la aplicación y hasta dónde se extiende.

Se puede decir que una aplicación ASP es un tipo de aplicación Web determinado.

Para distinguir los directorios de arranque de una aplicación el Administrador de servicios de Internet los muestra con un icono especial, se representan como una pequeña bola con un aspa metida en una caja, esto se puede observar en la Figura 86, no se deben confundir con los directorios virtuales, que se representan con una pequeña bola azul.

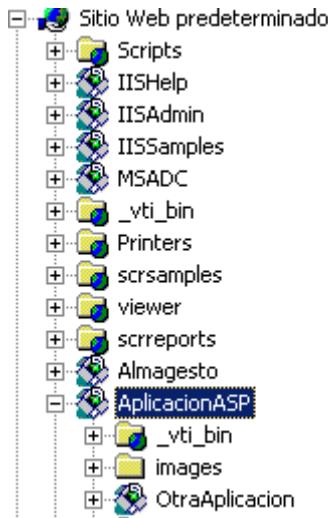


Figura 86. Aplicaciones Web

Se puede observar en la figura anterior que existen varias aplicaciones Web diferentes, de esta forma todos los ficheros del directorio AplicacionASP forman parte de una misma aplicación menos los ficheros que se encuentran en el directorio OtraAplicacion que forman parte de una aplicación Web distinta.

Podemos anidar tantas aplicaciones Web como deseemos teniendo en cuenta que debe existir una lógica a la hora de crear distintas aplicaciones Web, aunque no es recomendable anidar aplicaciones Web.

Las aplicaciones ASP ofrecen un estado de aplicación, de esta forma una aplicación puede compartir información entre los diferentes ficheros que se encuentren incluidos en la aplicación.

Desde el punto de vista de las páginas ASP podemos distinguir dos tipos de contextos en una aplicación ASP, lo que se denomina estado de la aplicación y estado de la sesión. No es posible intercambiar información entre aplicaciones distintas, cada aplicación tiene un estado de aplicación único con todos sus estados de sesión que se corresponden con los clientes que en ese momento se encuentran conectados a la aplicación Web. Tampoco es posible intercambiar información a nivel de sesión aunque las sesiones se encuentren en la misma aplicación, el intercambio de información se encuentra restringido a nivel del estado de la aplicación y dentro de una misma aplicación.

Para crear el punto de inicio de una aplicación Web, o crear una aplicación ASP, desde nuestro punto de vista es lo mismo, debemos seleccionar en el Administrador de servicios de Internet el directorio que va a hacer las veces de punto de inicio, y en la ficha de propiedades llamada Directorio pulsaremos el botón Crear dentro del apartado Configuración de aplicación. De forma automática se creará la aplicación y aparecerán dos nuevos botones Quitar y Configurar. El primero de ellos se utilizará para eliminar la aplicación creada y el segundo para configurarla. También podremos indicar un nombre para la aplicación, aunque este nombre es únicamente descriptivo.

En IIS 4.0 ya podíamos definir aplicaciones ASP y su directorio de inicio, también podíamos indicar a través de la configuración de los directorios de un sitio Web si la aplicación se ejecutaba en otro espacio de memoria como un proceso aislado.

Con IIS 5.0 el concepto de aplicación ASP no ha variado, es decir, una aplicación es un conjunto de páginas ASP que se ejecutan en un conjunto de directorios definidos dentro de un sitio Web, tampoco ha variado excesivamente la forma de configurar las aplicaciones ASP. Lo más destacable que ofrece

IIS 5.0 con respecto a las aplicaciones ASP es la posibilidad de definir tres niveles de protección distintos para dichas aplicaciones.

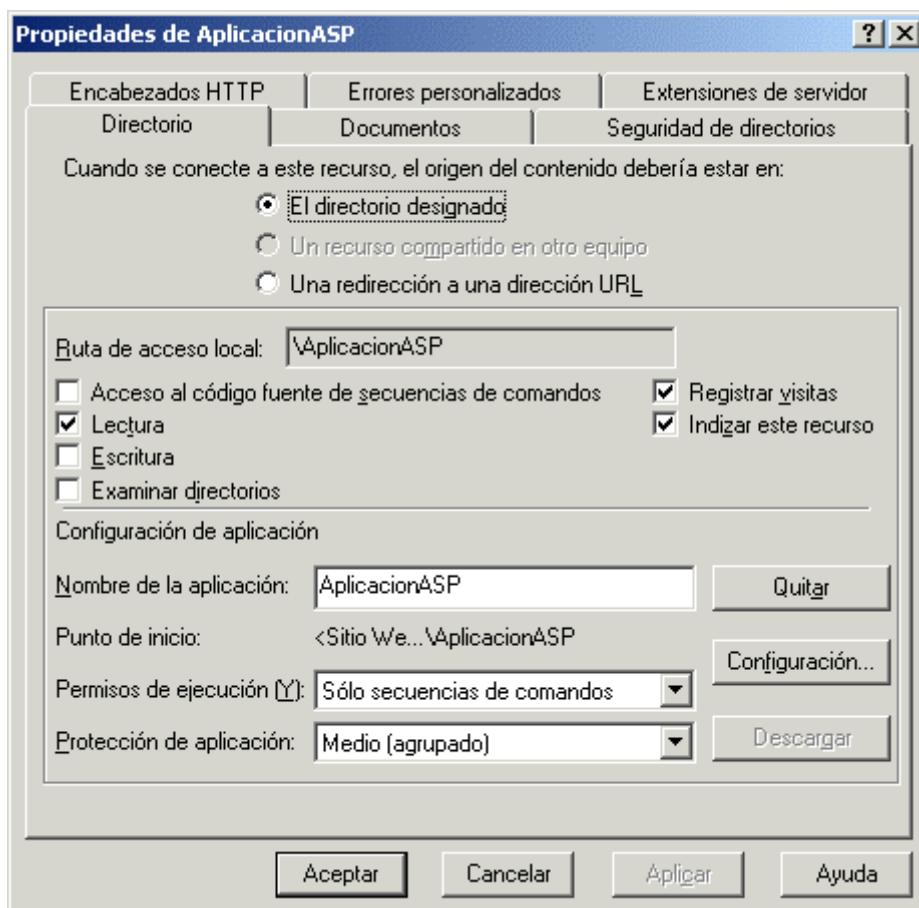


Figura 87. Creando una aplicación

En IIS 4.0 podíamos indicar que nuestra aplicación se ejecutara en el mismo espacio de memoria que el servidor Web o que se ejecutara en un proceso aislado, pero con IIS 5.0 tenemos otra posibilidad intermedia, que consiste en que la aplicación se ejecuta en un proceso agrupado con el resto de las aplicaciones ASP.

Los tres grados de protección que ofrece IIS 5.0 para las aplicaciones ASP se denominan bajo, medio y alto, veámoslo en la Tabla 18.

Protección	Descripción
Baja (proceso IIS)	Las aplicaciones ASP se ejecutan todas en el mismo espacio de memoria que el servidor Web IIS 5.0. Si una aplicación ASP falla afectará a todo el servidor Web, poniendo en peligro la ejecución de la aplicación InetInfo.exe, que es el ejecutable del servidor Web. Ofrece la ejecución más rápida y eficiente de las aplicaciones ASP, pero también la que ofrece más riesgos.

Media (agrupado)	Esta es la protección por defecto, todas las aplicaciones ASP se ejecutan agrupadas en un espacio de memoria distinto que el del servidor Web, en este caso todas las aplicaciones ASP del servidor Web utilizan una instancia compartida del ejecutable DLLHost.exe. De esta forma se protege al ejecutable InetInfo.exe de los posibles fallos de las aplicaciones ASP. Si se produce un fallo en una aplicación ASP no afecta al servidor Web, pero sí a resto de las aplicaciones ASP.
Alta (aislado)	Cada aplicación ASP se ejecuta en un espacio de memoria distinto, es decir, cada aplicación se ejecuta en una instancia distinta y exclusiva del ejecutable DLLHost.exe. De esta forma si una aplicación falla no afectará al resto de las aplicaciones ASP ni tampoco al servidor Web, ya que se ejecuta en su propio espacio de memoria. Microsoft recomienda que por cada servidor Web no se definan más de diez aplicaciones aisladas. Este tipo de protección es recomendable para aplicaciones ASP de alto riesgo o críticas.

Tabla 18

Por defecto el sitio Web predeterminado se define como una aplicación ASP agrupada o con grado de protección medio, este es el modo de protección de la aplicación más usual y recomendable, ya que ofrece una buena relación en lo que a rendimiento y seguridad se refiere, con el grado de protección alto comprometemos el rendimiento y con el grado de protección bajo se compromete la seguridad del funcionamiento del servidor Web.

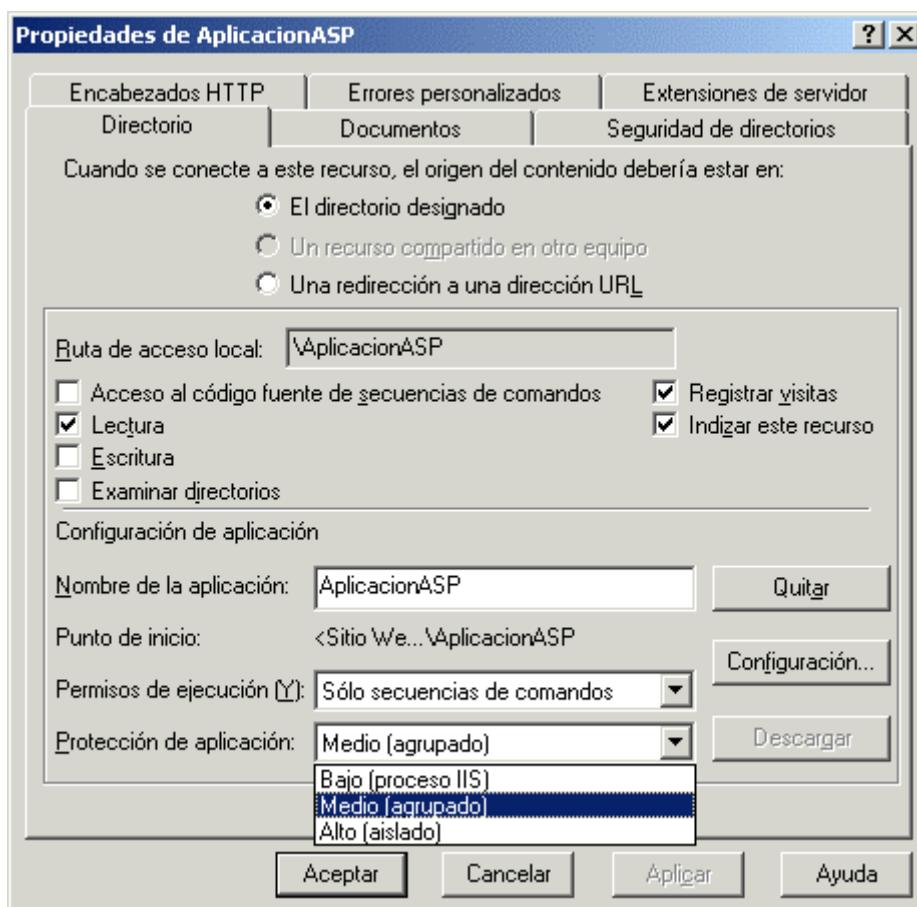


Figura 88. Indicando el modo de protección de la aplicación

Como resumen de los distintos grados de protección con los que podemos configurar nuestras aplicaciones ASP diremos que lo recomendable es ejecutar el servidor Web (InetInfo.exe) en su propio proceso, ejecutar aplicaciones decisivas en sus propios procesos y ejecutar el resto de aplicaciones en un proceso agrupado y compartido.

Una vez creada la aplicación Web, si pulsamos sobre el botón Configuración aparecerán tres hojas de propiedades que nos permitirán configurar nuestra aplicación Web.

La primera de estas hojas de propiedades es la ficha llamada *Asignaciones para la aplicación* (figura 22). Desde esta hoja de propiedades podremos asociar extensiones de ficheros con programas ISAPI o CGI, de esta forma si hemos asociado la extensión ASP al fichero ejecutable c:\winnt\system32\inetsrv\asp.dll, cuando un usuario realice una petición de una página con la extensión ASP se ejecutará el programa ISAPI indicado anteriormente, que en este caso se corresponde con el intérprete de páginas ASP o motor de secuencias de comandos. En las asignaciones de la aplicación se pueden indicar tanto librerías DLL como ficheros .EXE ejecutables.

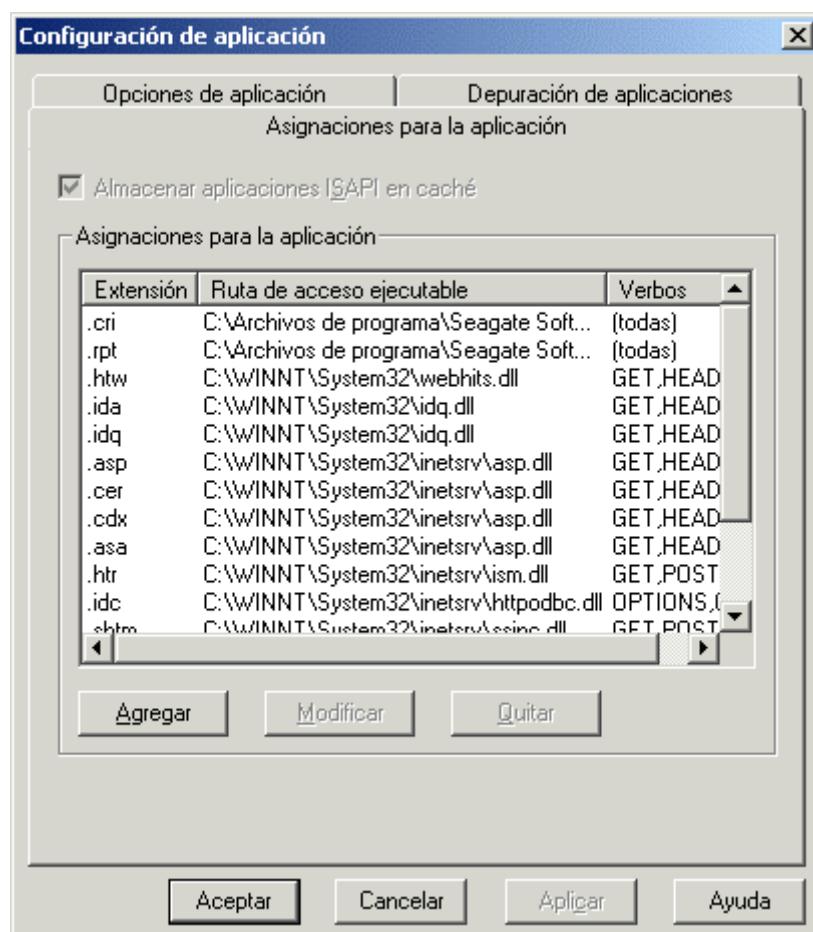


Figura 89. Asignación de aplicaciones

Para agregar una nueva asignación de una extensión a un ejecutable en el servidor Web debemos pulsar el botón Agregar, aparecerá un nuevo diálogo en el que deberemos indicar varios parámetros. También podemos quitar y modificar asignaciones existentes, en la Figura 90 se puede ver editada una asignación de los ficheros ASP.

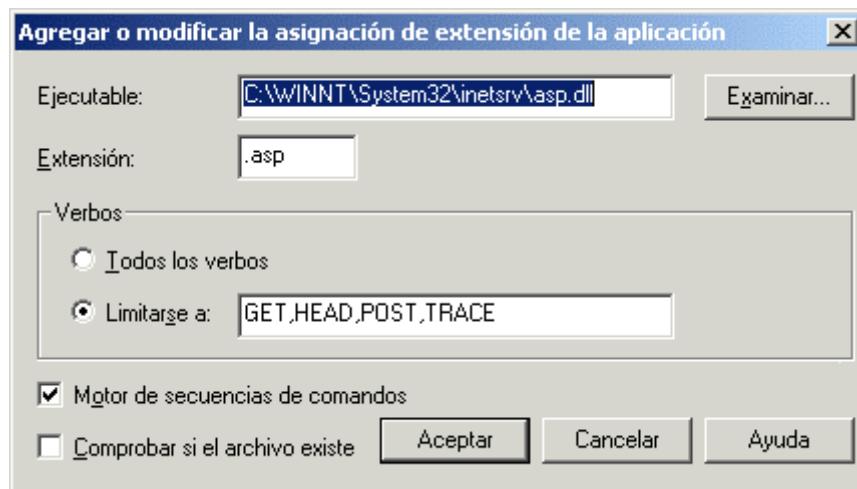


Figura 90. Agregando una asignación de extensión

En la caja de texto Ejecutable debemos indicar la ruta de acceso al programa que procesará el fichero con la extensión correspondiente, en el ejemplo que aparece en la imagen indicamos una DLL (ASP.dll) que se corresponde con el intérprete de ASP y que nos va a permitir ejecutar secuencias de comandos ASP. En la caja de texto Extensión indicaremos la extensión que queremos asignar al ejecutable, en este caso es la extensión .asp.

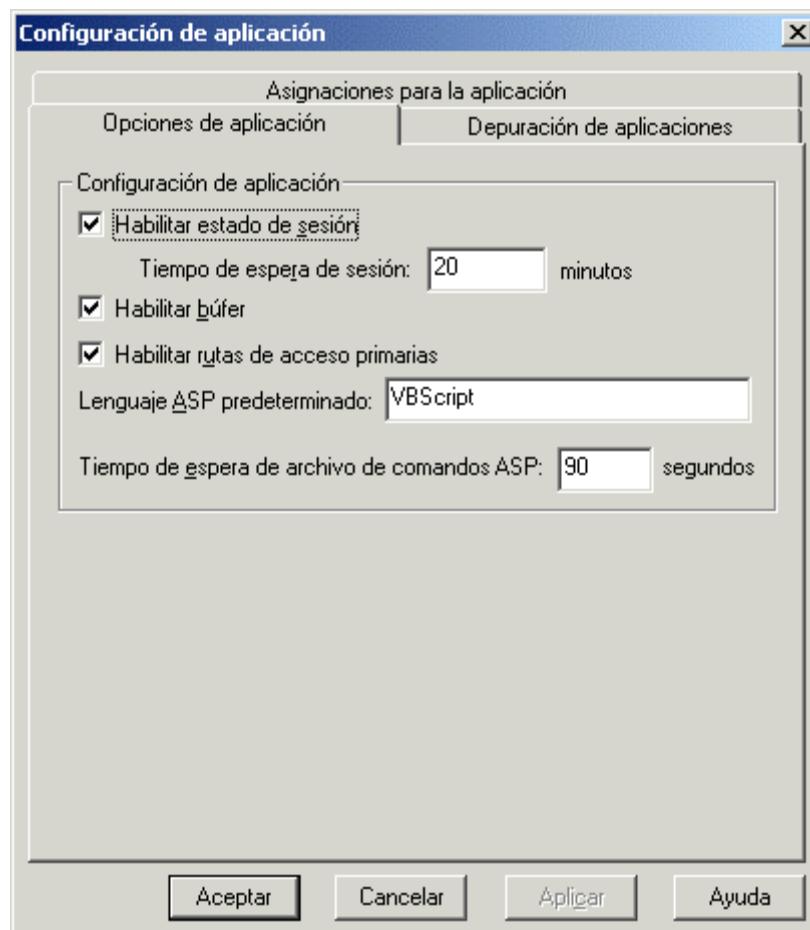


Figura 91. Opciones de la aplicación

En el epígrafe verbos podemos indicar los métodos del protocolo HTTP que no deben transmitirse a una aplicación debemos indicarlo en la caja de texto Limitarse a, por ejemplo, podremos prohibir los métodos PUT y DELETE del protocolo HTTP, al indicar varios verbos debemos separarlos mediante comas.

Para indicar al servidor Web que compruebe la existencia del fichero solicitado por el cliente y asegurarse de que el usuario que ha realizado la petición tiene permiso de acceso a ese fichero se debe activar la casilla de verificación Comprobar si el archivo existe.

Para permitir el procesamiento de ficheros con la extensión indicada el directorio debe tener permisos que permitan la ejecución de secuencias de comandos y se debe activar la casilla de verificación Motor de secuencias de comandos.

Volviendo a la hoja de propiedades Asignaciones para la aplicación, podemos observar que hay una casilla de verificación llamada Almacenar aplicaciones ISAPI en caché, si activamos esta casilla, la DLL que invoquemos se cargará y almacenará en memoria caché para que las peticiones posteriores puedan procesarse sin tener que invocarlas de nuevo, de lo contrario las DLLs se descargarán después de su ejecución afectando negativamente al rendimiento del servidor Web.

La siguiente hoja de propiedades que el Administrador de servicios de Internet pone a nuestra disposición para configurar nuestras aplicaciones Web es la hoja de propiedades llamada Opciones de aplicación. Desde esta hoja de propiedades podremos configurar una serie de parámetros que afectarán a las páginas ASP que se encuentren contenidas en la aplicación Web actual, y que se corresponden con distintas propiedades de objetos integrados en ASP y con directivas del lenguaje de secuencias de comandos.

Los parámetros que podemos configurar son los siguientes:

- Estado de sesión: en la casilla correspondiente podremos habilitar y deshabilitar el estado de la sesión para las páginas ASP. Si está habilitado, se creará una sesión por cada usuario conectado a la aplicación. Se corresponde con la directiva @ENABLESESSIONSTATE, y esta directiva puede sobreescribir el valor de este parámetro, al igual que ocurre con todos los parámetros aquí definidos, si tienen un equivalente, su equivalente los puede sobreescribir. Por defecto este parámetro se encuentra activado.
- Tiempo límite de la sesión: si hemos habilitado el estado de sesión podremos indicar el tiempo límite que puede permanecer inactiva una sesión (time-out de la sesión). Corresponde a modificar el valor de la propiedad Timeout del objeto Session de los objetos integrados de ASP. Por defecto, el valor de este parámetro es de 20 minutos.
- Habilitar búfer: si habilitamos esta opción el resultado de la ejecución de las páginas ASP no se enviará directamente al navegador del cliente, sino que pasará previamente por un búfer intermedio, esto nos permite tener un mayor control sobre el procesamiento de las páginas ASP. Es equivalente a modificar la propiedad Buffer del objeto Response de ASP. Por defecto este parámetro está activado.
- Habilitar rutas de acceso primarias: al habilitar esta casilla permitimos que las páginas ASP puedan hacer uso de rutas de acceso relativas al directorio primario del directorio actual, es decir, podemos ir hacia atrás en la estructura de directorios con la sintaxis de los dos puntos (...). Por defecto este parámetro está activado y no tiene correspondencia dentro de ASP.
- Lenguaje ASP predeterminado: en esta caja de texto indicamos el lenguaje de secuencias de comandos que vamos a utilizar en nuestras páginas ASP. Se puede especificar el nombre de

cualquier lenguaje de script para el que tengamos instalado en el servidor el intérprete correspondiente. Por defecto el lenguaje de script utilizado es Visual Basic Script (VBScript). Es equivalente a utilizar la directiva @LANGUAGE de las páginas ASP.

- Tiempo límite de secuencia de comandos ASP: en esta caja de texto indicamos en segundos el tiempo máximo de espera para la ejecución de una página ASP. Si el script de la página ASP no ha terminado su ejecución cuando ha pasado el tiempo especificado se producirá un error y se detendrá la ejecución de la secuencia de comandos de la página ASP. Es equivalente a la propiedad ScriptTimeout del objeto Server de ASP. El valor por defecto de este parámetro es de 90 segundos.

En la hoja de propiedades Depuración de aplicaciones (Figura 92) podremos indicar si deseamos depurar nuestras páginas ASP. La depuración de páginas ASP sólo se debería habilitar en un servidor Web de desarrollo nunca en un servidor Web en producción.

Podremos indicar si permitimos la depuración de páginas ASP en el cliente, en el servidor o en ambos.

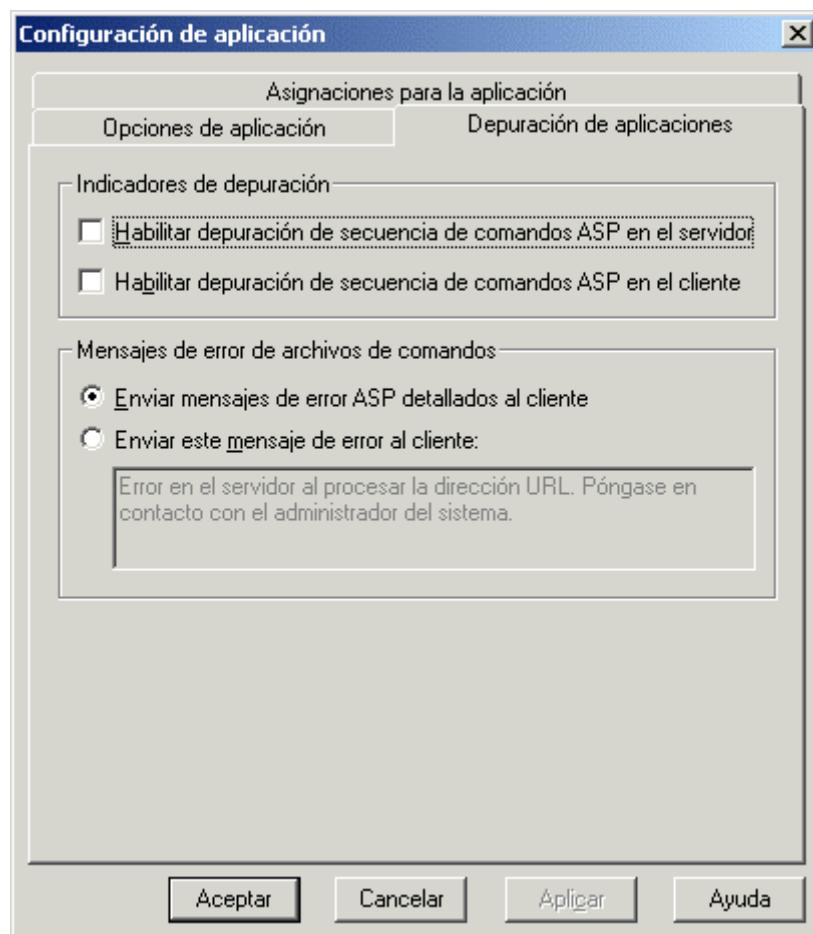


Figura 92. Depuración de aplicaciones

En esta hoja de propiedades también tenemos la opción de elegir si queremos enviar los mensajes de error de las páginas ASP al cliente, o por el contrario le enviamos un mensaje que podremos elegir nosotros para indicar que se ponga en contacto con el webmaster o administrador del sistema. Para un servidor Web en producción sería recomendable elegir la segunda opción, ya que hace peor efecto que el cliente observe un error ASP indicando la línea en la que se ha producido, el código de error, etc., es

mucho más claro si ponemos un texto genérico indicando que se ha producido un error y que lo indique a una dirección determinada.

Si hemos indicado que nuestra aplicación Web se va a ejecutar en un espacio de proceso aislado, es decir, con un nivel de protección alto, aparecerá una nueva hoja de propiedades específica para este tipo de aplicaciones. Esta nueva hoja de propiedades es llamada Opciones de proceso (Figura 93) y en ella podremos configurar el proceso correspondiente a la aplicación actual y también tenemos una serie de opciones que afectan al proceso del servidor Web.

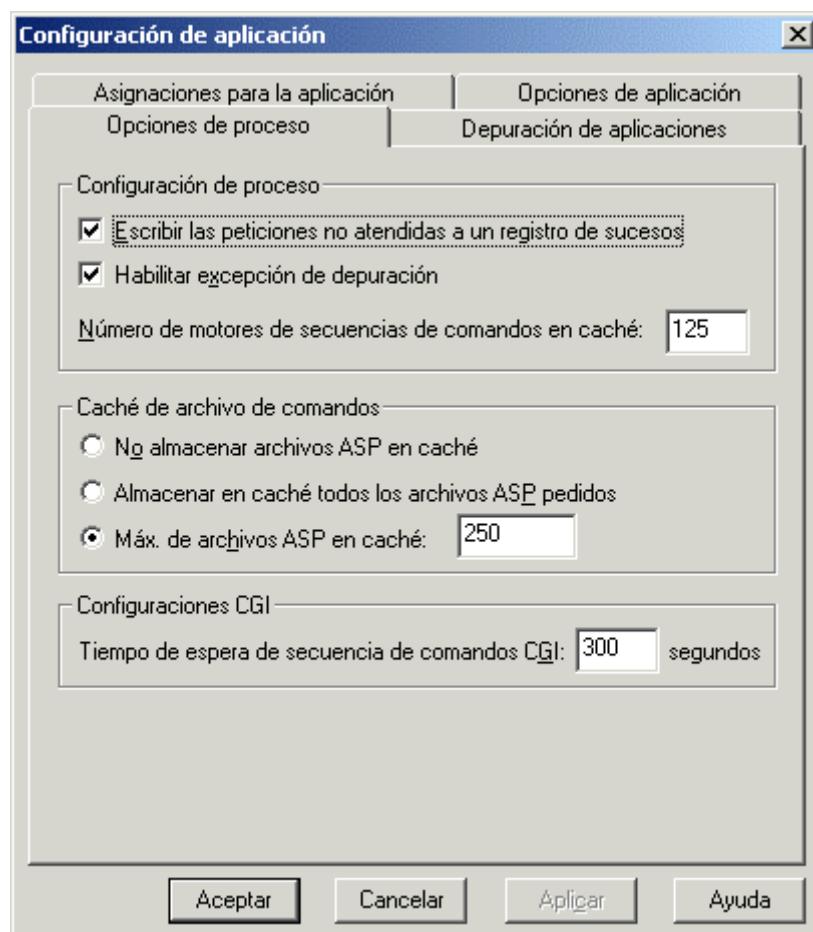


Figura 93. Opciones de proceso

Si activamos la casilla de verificación Escribir las peticiones no atendidas a un registro de sucesos, cuando una petición de un cliente no se haya atendido se registrará en el registro de sucesos de Windows.

Deabajo de esta opción tenemos otra que nos permite habilitar la excepción de depuración, es decir, si esta opción está habilitada, cuando una página ASP no pueda crear un objeto proporcionado por un componente ActiveX de servidor, al cliente se le enviará un mensaje de error genérico, en lugar del mensaje de error específico de ASP. Cuando estemos depurando un componente será recomendable desactivar esta opción, sin embargo en el servidor de producción es mejor tener esta opción inhabilitada. Por defecto estos dos parámetros se encuentran habilitados. Por defecto se almacena en caché un máximo de 250 páginas ASP.

En la caja de texto correspondiente dentro de esta hoja de propiedades podremos indicar el número de motores de secuencias de comandos o intérpretes de scripts (ficheros DLL) que deseamos mantener en memoria caché, por defecto se permiten 125.

También se nos ofrece la posibilidad de almacenar en memoria caché páginas ASP. Podemos elegir el número de páginas que se van a almacenar en caché o bien no especificar ningún límite, toda esta configuración la realizaremos a través de las opciones englobadas bajo el epígrafe de Caché de archivo de comandos.

El último parámetro que podemos configurar dentro de esta hoja tiene que ver con los scripts CGI, concretamente, podremos indicar el tiempo máximo de espera para la ejecución de un script CGI, es decir, el time-out del procesamiento de los scripts CGI. Por defecto el tiempo máximo de espera es de 300 segundos.



# CDONTS y ASP

---

## Introducción

En capítulos anteriores hemos visto distintos tipos de componentes y objetos que se pueden utilizar en ASP:

- Objetos pertenecientes al modelo de objetos de ASP (Request, Server, Application, etc.).
- Componentes incluidos en el intérprete de secuencias de comandos de VBScript (Dictionary, FileSystemObject, TextStream, etc.).
- Componentes incluidos con la instalación de ASP (AdRotator, MyInfo, ContentRotator, Tools, ADO, etc.).

En este capítulo vamos a ver un conjunto de componentes agrupados todos ellos bajo el nombre de CDONTS (Collaboration Data Objects for NT Server). CDONTS es una librería de Microsoft que ofrece la posibilidad de utilizar sistemas de mensajería, basándose en servidores de correo como Microsoft Exchange Server o el servicio SMTP incluido en IIS 5.0.

Como el propio nombre de estos componentes indica, son componentes para NT Server, en nuestro caso para 2000 Server.

CDONTS se instala como parte del servicio de correo SMTP. Podremos acceder de igual forma a Microsoft Exchange Server 5.5 o superior y al servicio de correo SMTP incluido en IIS 5.0.

En el presente capítulo los ejemplos realizados con CDONTS han sido probados con el servicio de correo de IIS 5.0, para averiguar si tenemos instalado este servicio de IIS 5.0 debemos utilizar el

Administrador de servicios de Internet y comprobar si existe el nodo llamado Servidor virtual SMTP predeterminado.

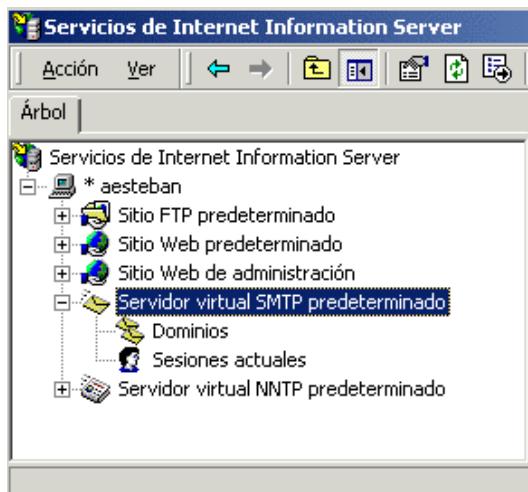


Figura 94. El servicio SMTP dentro de IIS 5.0

Si no se encuentra instalado este servicio acudiremos a la opción Agregar o quitar programas del Panel de control. Dentro de esta opción seleccionaremos Agregar o quitar componentes de Windows y de los componentes seleccionaremos Servicios de Internet Information Server (IIS). Si pulsamos el botón etiquetado como Detalles veremos los subcomponentes de IIS y seleccionaremos el componente Servicio SMTP.

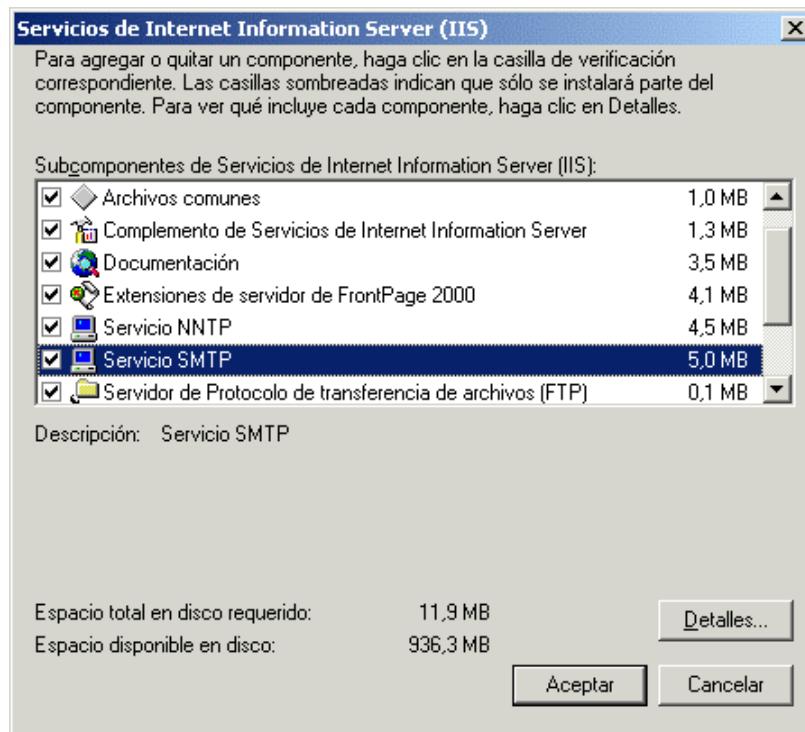


Figura 95. Instalando el servicio SMTP

A lo largo de este capítulo comentaremos cuando sea necesario algunos puntos relacionados con la configuración del servicio SMTP de IIS 5.0.

Antes de seguir con los distintos apartados que exponen el modelo de objetos de CDONTS, vamos a ver el Código fuente 207 que utiliza un objeto de CDONTS llamado NewMail y que nos permite de manera muy sencilla enviar un mensaje de correo electrónico.

```
<%Set objMail=Server.CreateObject("CDONTS.NewMail")
objMail.Send "pepe@trola.com","aesteban@mail.angel.es","Asunto","Hola que tal te
va...%">
```

Código fuente 207

Con este código ya podríamos enviar un mensaje de correo a través de nuestro servicio SMTP. En siguientes apartados comentaremos detenidamente de nuevo este código.

## Modelo de objetos de CDONTS

La librería CDONTS se encuentra formada por un modelo de objetos que vamos a comentar es este capítulo. Veamos gráficamente este modelo con la Figura 96.

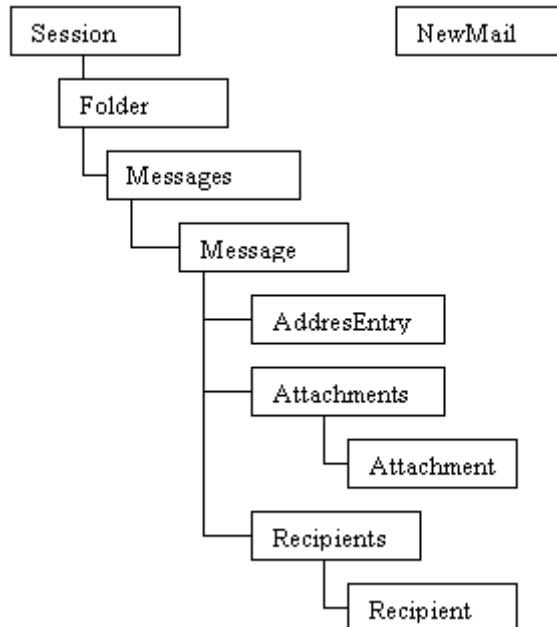


Figura 96. Modelo de objetos de CDONTS

Como se puede observar todos los objetos se encuentran relacionados a excepción del objeto NewMail que aparece desconectado del resto. A continuación se ofrece una breve descripción de los objetos incluidos en CDONTS.

- NewMail: mediante este objeto podremos enviar nuevos mensajes de correo.

- Session: este objeto contiene toda la información necesaria para manipular mensajes y también las carpetas de entrada (inbox) y de salida de mensajes (outbox), es decir, podremos leer los mensajes recibidos y enviar nuevos mensajes. Mediante este objeto podremos acceder a los mensajes almacenados de un usuario determinado. El resto de los objetos de CDONTS tienen como referencia el objeto Session.
- Folder: representa una carpeta o contenedor que contiene los mensajes almacenados correspondientes a la sesión actual. Vamos a poder acceder a los mensajes recibidos y a los mensajes pendientes de enviar.
- Message: se corresponde con un mensaje de un objeto Folder, el objeto Folder ofrece una propiedad llamada Messages que contiene una colección de objetos Message.
- AddressEntry: este objeto contiene información relativa al remitente de un mensaje determinado, se obtiene a partir de la propiedad Sender del objeto Message.
- Attachments: colección perteneciente al objeto Message que contiene objetos Attachment, y que representa todos los adjuntos de un mensaje de correo determinado.
- Attachment: objeto que representa un fichero adjunto de un mensaje.
- Recipients: colección perteneciente al objeto Message que contiene objetos Recipient, y que representa todos los destinatarios de un mensaje determinado.
- Recipient: objeto que representa un destinatario de un mensaje.

En los distintos apartados de este tema veremos cada uno de los objetos del modelo de objetos de CDONTS.

## El objeto Newmail

Como ya hemos dicho anteriormente, este objeto que se encuentra fuera de la jerarquía del objeto Session nos va a permitir enviar mensajes de correo electrónico.

Los métodos que ofrece este objeto son:

- Send(remitente, destinatario, asunto, mensaje, importancia): mediante este método enviaremos un mensaje al destinatario que indiquemos. Todos los parámetros son opcionales, pudiendo aplicar el método Send() sin parámetros, ya que el objeto NewMail ofrece una serie de propiedades que permiten configurar el mensaje antes de enviarlo. En el parámetro destinatario podemos expresar distintos destinatarios separados con comas. Tanto destinatario como remitente son direcciones de correo electrónico. En el asunto especificaremos el asunto del mensaje y en mensaje el cuerpo del mensaje, es decir, el mensaje en sí. Y mediante el parámetro importancia indicamos si es alta (cdohigh), normal (cdonormal) o baja (cdolow), por defecto este parámetro tiene el valor cdonormal. Todos estos parámetros se corresponden con propiedades del objeto NewMail que veremos a continuación.
- AttachFile(fuente, nombreFichero, codificación): permite adjuntar el fichero especificado en el parámetro fuente a un mensaje de correo, se debe indicar una ruta física completa, el resto de los parámetros sonopcionales. En nombreFichero indicamos el nombre con el que aparece el fichero adjunto en el cliente de correo, y en codificación indicamos el método de

codificación empleado para enviar el fichero adjunto, esta parámetro tiene dos posibles valores cdoEncodingUUencode y cdoEncodingBase64, por defecto presenta el primero de ellos.

En el ejemplo del Código fuente 208 se trata de enviar un correo a aesteban@mail.angel.es, que es de prioridad baja y que posee dos ficheros adjuntos.

```
<!--METADATA TYPE="typelib" FILE="C:\Winnt\system32\cdonts.dll"-->
<%Set objMail=Server.CreateObject("CDONTS.NewMail")%
objMail.AttachFile "c:\tmp\imagen.png","Figura"
objMail.AttachFile "c:\tmp\tema30.html","Tema en HTML"
objMail.Send "pepe@trola.com","aesteban@mail.angel.es"_
,"Asunto","Hola que tal te va...",cdoLow%>
```

Código fuente 208

Como se puede observar se ha realizado una referencia a la librería de CDONTS, para así poder utilizar las constantes definidas en ella. En este caso se ha utilizado la constante para establecer la prioridad el mensaje.

Si todo ha funcionado correctamente el mensaje habrá llegado a su destinatario. Si el destinatario abre el mensaje de correo con el cliente de correo Microsoft Outlook Express tendrá el aspecto que muestra la Figura 97.

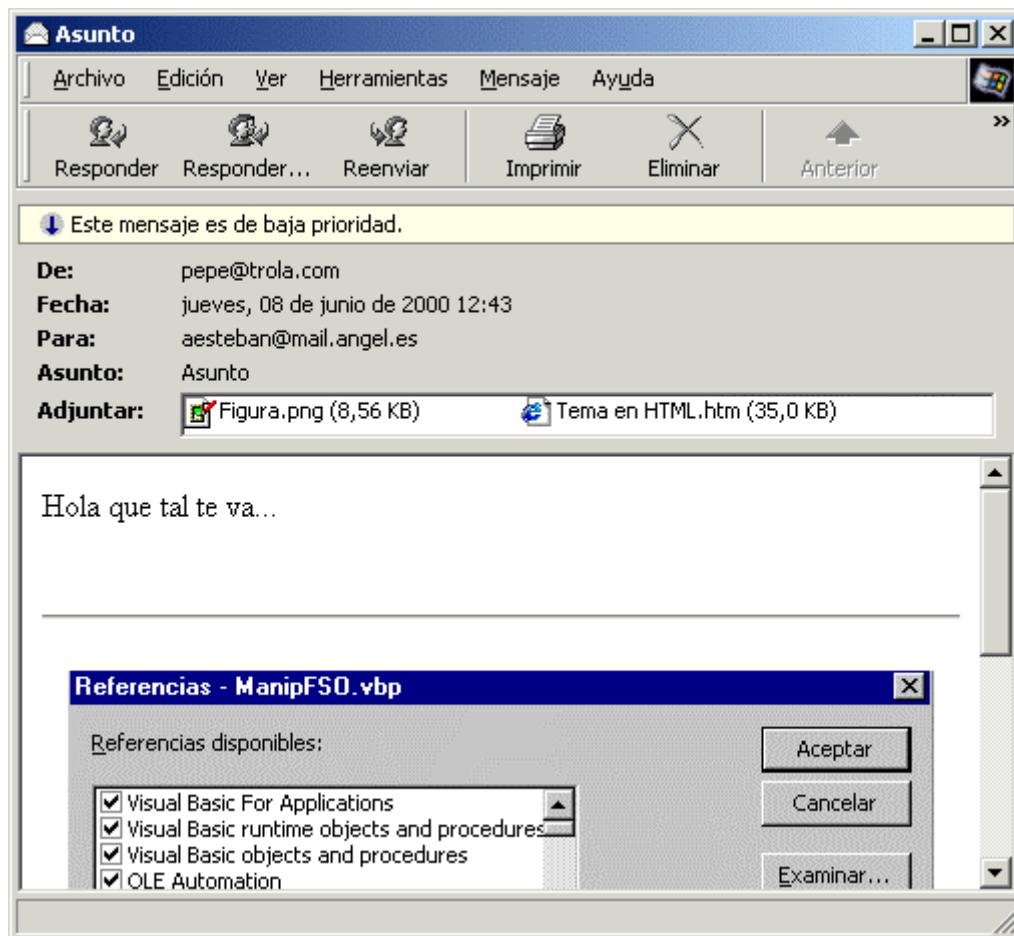


Figura 97. El mensaje en Outlook Express

Las propiedades que posee el objeto NewMail son las siguientes, y como ya hemos dicho algunas de ellas se corresponden con los parámetros del método Send():

- From: en esta propiedad indicamos el remitente del mensaje de correo. Se asigna una cadena que representa una dirección de correo, no es posible utilizar nombres.
- To: destinatario del mensaje, si existen varios destinatarios se deben separar mediante comas.
- CC: indicamos las direcciones de correo a las que deseamos enviar copia del mensaje.
- BCC: tiene la misma función que el caso anterior, pero en este caso se oculta la lista de destinatarios que reciben copia.
- Importance: es la importancia o prioridad del mensaje, se le puede asignar los valores cdoHigh(alta), cdoNormal(normal) o cdoLow(baja). Por defecto tiene el valor de prioridad normal.
- Body: representa el cuerpo del mensaje, es decir, el texto del mensaje. Puede contener texto plano o en HTML.
- BodyFormat: mediante este propiedad indicamos el formato del cuerpo del mensaje, puede aceptar los valores cdoBodyFormatHTML, para indicar que el formato del cuerpo del mensaje es HTML o cdoBodyFormatText para indicar que el formato del cuerpo del mensaje es texto plano, por defecto el valor de esta propiedad es cdoBodyFormatText. Para poder utilizar el valor cdoBodyFormatHTML se debe utilizar el formato de correo MIME, indicándolo en la propiedad MailFormat.
- MailFormat: indica el formato con el que se crean los mensajes de correo, puede tener dos valores posibles, formato de texto plano (cdоМailFormatText), que es el valor por defecto, y formato MIME (cdоМailFormatMIME). Con el formato MIME podemos enviar contenidos más elaborados en nuestros mensajes de correo.
- Version: esta propiedad de sólo lectura devuelve la versión de la librería CDONTS que estamos utilizando, la versión actual es la 1.2.
- ContentLocation: permite especificar un camino relativo o absoluto para todas las URLs contenidas en el cuerpo del mensaje, esta propiedad tiene sentido cuando estamos enviando mensajes en formato HTML.
- ContentBase: propiedad relacionada con la anterior, ya que nos permite especificar la ruta base para todas URLs contenidas en el cuerpo del mensaje. Los valores de ContentLocation se consideran relativos a los valores de ContentBase. Así si la propiedad ContentBase tiene el valor "http://www.almagesto.com" y ContentLocation tiene el valor "images/", para las URLs presentes en el cuerpo del mensaje se construirá la ruta http://www.almagesto.com/images/. Esto sería equivalente a asignar a la propiedad ContentBase el valor "http://www.almagesto.com/images/".

Veamos un ejemplo que muestra la utilización de las propiedades del objeto NewMail. En este caso se definen algunas de las propiedades y luego se envía el correo mediante el método Send() sin parámetros.

```
<!--METADATA TYPE="typelib" FILE="C:\Winnt\system32\cdonts.dll"-->
```

```
<%Set objMail=Server.CreateObject("CDONTS.NewMail")
objMail.From="pepe@trola.com"
objMail.To="aesteban@mail.angel.es"
objMail.Importance=cdoLow
objMail.MailFormat=cdoMailFormatMIME
objMail.Subject="Asunto"
objMail.BodyFormat=cdoBodyFormatHTML
objMail.ContentBase="http://aesteban/images/"
strTexto=<html><body><b><i>Hola que tal te va...</i></b><br>
strTexto=strTexto&"<div align='center'><img src='almagesto.gif'></div>"
strTexto=strTexto&"</body></html>"
objMail.Body=strTexto
objMail.Send%>
```

Código fuente 209

En este ejemplo se ha dado el formato MIME al mensaje y al cuerpo el formato HTML, el resultado de visualizar este correo mediante el cliente de correo Outlook Express es el que muestra la Figura 98.

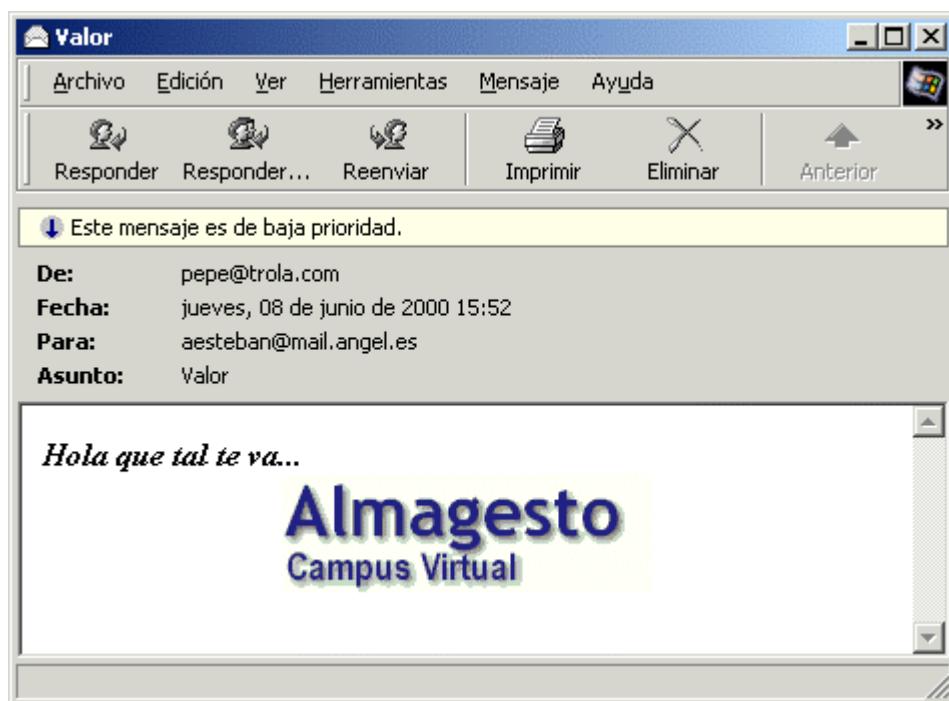


Figura 98. Mensaje de correo con formato

A través del objeto NewMail también podemos tener acceso y manipular las cabeceras del mensaje de correo, que normalmente en los clientes de correo permanecen ocultas. Parte de la información contenida en las cabeceras se genera a partir de las propiedades del objeto NewMail y otra parte las genera el servidor de correo de forma automática al enviar el mensaje.

Estas cabeceras las podemos ver con el cliente de correo Outlook Express si en el menú de el mensaje acudimos a la opción Archivo|Propiedades, y luego elegimos la pestaña Detalles.

Mediante la colección Value del objeto NewMail podemos añadir nuevas cabeceras a los mensajes o modificar las existentes.

En el Código fuente 210 se añade una nueva cabecera y se modifica la cabecera Subject.

```
<!--METADATA TYPE="typelib" FILE="C:\Winnt\system32\cdonts.dll"-->
<%Set objMail=Server.CreateObject("CDONTS.NewMail")%
objMail.From="pepe@trola.com"
objMail.To="aesteban@mail.angel.es"
objMail.Importance=cdoLow
objMail.MailFormat=cdoMailFormatMIME
objMail.Subject="Asunto"
objMail.BodyFormat=cdoBodyFormatHTML
objMail.ContentBase="http://aesteban/images/"
strTexto=<html><body><b><i>Hola que tal te va...</i></b><br>
strTexto=strTexto&"<div align='center'><img src='imagen.gif'></div>"%
strTexto=strTexto&"</body></html>"%
objMail.Body=strTexto
objMail.Value("Subject")="Modifico el asunto"
objMail.Value("NuevaCabecera")="Nuevo valor"
objMail.Send%>
```

Código fuente 210

Los detalles del mensaje en Outlook Express tienen el aspecto que nos muestra la

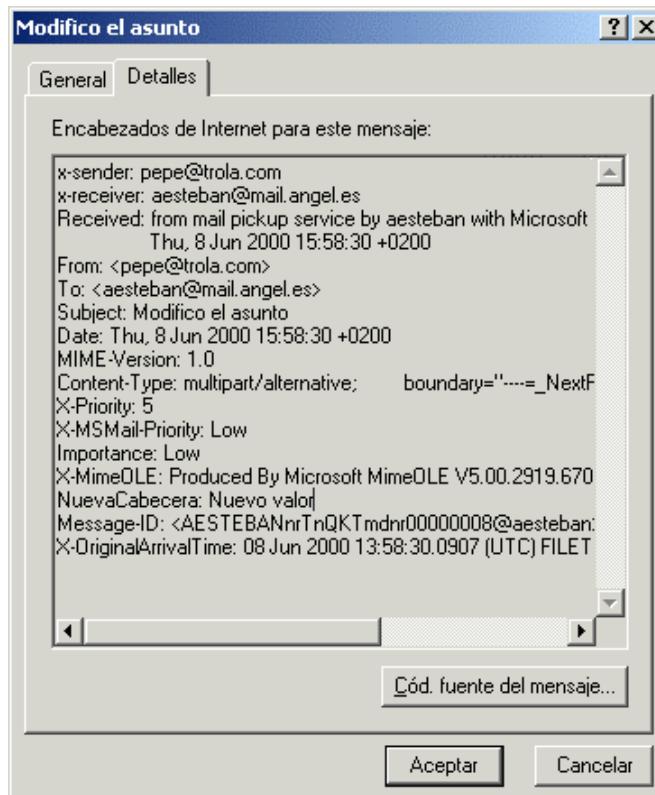


Figura 99. Cabeceras del mensaje

Hemos visto lo sencillo que resulta enviar un mensaje mediante el objeto NewMail de la libreía CDONTS, ahora vamos a seguir viendo los distintos objetos que proporciona esta librería. Los siguientes objetos permiten acceder a los mensajes que se encuentran almacenados en el servidor de correo.

## El objeto Session

Del objeto Session dependen el resto de los objetos de la librería CDONTS, a través del objeto Session iniciamos una sesión con el servidor de correo mediante un usuario determinado.

El servidor con el que iniciamos la sesión será el servidor de correo que se encuentre instalado en el servidor Web en el que se ejecutan las páginas ASP, como ya hemos dicho este servidor puede ser el servicio SMTP incluido en IIS 5.0 o Exchange Server 5.5 o superior.

Para realizar las pruebas con los distintos objetos de CDONTS se ha utilizado el servicio SMTP de IIS 5.0. Para indicar una dirección de correo dentro de este servidor utilizaremos la siguiente sintaxis: cuenta@nombreServidor, siendo nombreServidor el nombre del dominio por defecto del servidor SMTP. Aunque desde el Administrador de servicios de Internet podemos definir alias para el dominio por defecto. Para ello acudiremos dentro del Administrador de servicios de Internet al nodo llamado Servidor virtual SMTP predeterminado, y en la rama Dominios pulsando con el botón derecho del ratón seleccionamos la opción Nuevo|Dominio, en ese momento se lanza el asistente para la creación de nuevos dominios.

Dentro del asistente seleccionamos la opción de Alias y pulsamos el botón Siguiente. A continuación se debe indicar el nombre que vamos a dar al alias de nuestro dominio, yo en mi caso le he llamado mail.angel.es, siendo mi dominio por defecto aesteban, es decir, el nombre de mi servidor. Pulsamos el botón Finalizar. Si todo ha sido correcto veremos una pantalla similar a la de la Figura 100.

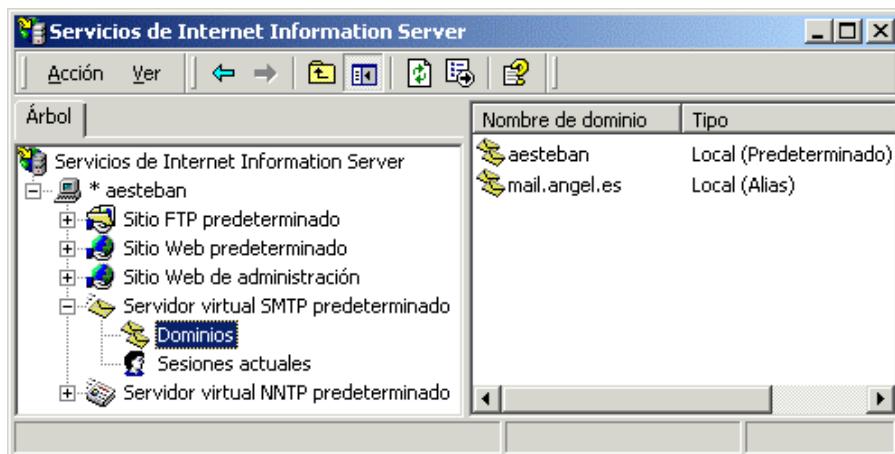


Figura 100. Creando dominios alias

Después de este pequeño comentario acerca de la creación de dominios, volvamos al tema principal que nos ocupa dentro de este apartado.

El objeto Session presenta los siguientes métodos:

- LogonSMTP(nombre, dirección): mediante este método se inicia una sesión con el servidor de correo. En el parámetro nombre se indica el nombre que se desea que se muestre al usuario, y en dirección la dirección completa del usuario que se desea conectar al servidor de correo. Este método no realiza ningún tipo de validación, si la conexión no se ha podido establecer no recibimos ningún mensaje de error. Cada usuario conectado tendrá acceso a su propio buzón, pudiendo leer el correo de su "bandeja de entrada" (inbox) y enviar correo a través de su "bandeja de salida" (outbox).

- LogOff(): este método finalizará la sesión actual con el servidor de correo. Podremos reutilizar el objeto Session si deseamos realizar una nueva sesión llamando de nuevo al método LogonSMTP.
- GetDefaultFolder(tipoCarpeta): mediante este método se obtiene un objeto Folder que representa la bandeja o carpeta de entrada o la de salida del usuario conectado. El parámetro de este método puede tener dos valores cdoDefaultFolderInbox(1) para obtener la bandeja de entrada y poder leer los mensajes recibidos o cdoDefaultFolderOutbox(2) para obtener la bandeja de salida y poder enviar mensajes.
- SetLocaleID(codigo): este método permite establecer un código de localización que afectará al entorno del lenguaje, formatos horarios, fechas, valores monetarios, etc. El parámetro tiene los mismos valores que vimos para la propiedad LCID del objeto integrado Session de ASP. Este método si se utiliza se debe lanzar antes de establecer la sesión con el servidor de correo mediante el método LogonSMTP().

Ahora vamos a pasar a comentar las propiedades del objeto Session:

- Inbox: propiedad de sólo lectura que devuelve un objeto Folder que representa la bandeja de entrada de la sesión actual.
- Outbox: propiedad de sólo lectura que devuelve un objeto Folder que representa la bandeja de salida de la sesión actual.
- MessageFormat: propiedad de lectura y escritura que permite obtener e indicar el formato de codificación de los mensajes. Tiene dos valores posibles CDOMime, para mensajes de tipo MIME, y CDOText para mensajes de tipo texto plano.
- Name: propiedad de sólo lectura que contiene el nombre del usuario conectado y que se ha especificado en el método LogonSMTP().
- Version: propiedad de sólo lectura que indica la versión de la librería CDONTS.

El objeto Session además de estas propiedades ofrece otras tres que son comunes a todos los objetos de su jerarquía, y son las siguientes:

- Session: esta propiedad devuelve el objeto Session que representa la raíz de la jerarquía de un objeto determinado.
- Class: esta propiedad devuelve un entero que nos indica el tipo de objeto: Session(0), Folder(2), Message(3), Recipient(4), Attachment(5), AddressEntry(8), Messages(16), Recipients(17) y Attachments(18).
- Parent: devuelve un objeto que representa el padre del objeto especificado. La propiedad Parent del objeto Session devuelve Nothing y la del objeto AddressEntry devuelve un objeto Message.

Una vez comentados los métodos y propiedades del objeto Session vamos a mostrar un ejemplo que hace uso de algunos de ellos.

Este ejemplo consiste en un formulario que nos permite conectarnos al servicio SMTP de IIS 5.0. El formulario permite especificar el nombre del usuario y la cuenta de correo del usuario que se va a conectar. Una vez que se ha establecido la conexión con el servidor SMTP se muestran algunos de los valores de las propiedades de la sesión actual. También existe la posibilidad de desconectarnos para

iniciar una nueva sesión, la sesión se debe almacenar en una variable a nivel de sesión, ya que más tarde deberemos utilizarla para realizar la desconexión. Veamos el Código fuente 211. Y en la Figura 101 se puede ver un ejemplo de la ejecución del código anterior.

```
<!--METADATA TYPE="typelib" FILE="C:\Winnt\system32\cdonts.dll"-->
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%If Request.Form("Desconectar")<>"" Then
    'se realiza la desconexión
    Set objSession=Session("oSession")
    objSession.LogOff
End if
If Request.Form("Conectar")="" Then%
    <form action=<%=Request.ServerVariables("SCRIPT_NAME")%> method="post">
        Nombre:<input type="text" name="nombre" size="20" value=""><br>
        Cuenta de correo:<input type="text" name="cuenta" size="20"
value=""><br>
        <input type="submit" name="conectar" value="Conectar">
    </form>
<%Else
    'se realiza la conexión
    Set objSession=Server.CreateObject ("CDONTS.Session")
    objSession.LogonSMTP Request.Form("nombre"),Request.Form("cuenta")
    Set Session("oSession")=objSession%
    Usuario conectado: <%=objSession.Name%><br>
    <%If objSession.MessageFormat=CDOMime Then
        strFormato="CDOMime"
    Else
        strFormato="CDOTText"
    End if%>
    Formato de los mensajes: <%=strFormato%><br>
    Versión de CDONTS: <%=objSession.Version%><br>
    <form action=<%=Request.ServerVariables("SCRIPT_NAME")%> method="post">
        <input type="submit" name="desconectar" value="Desconectar">
    </form>
<%End if%>
</BODY>
</HTML>
```

Código fuente 211

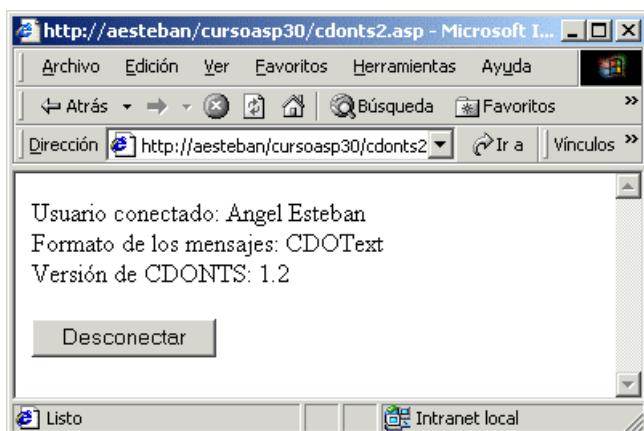


Figura 101. Utilizando el objeto Session de CDONTS

## El objeto Folder

Este objeto representa una carpeta o contenedor que se va a corresponder con las bandejas de entrada y de salida de los mensajes de correo de la sesión actual. Un objeto Folder lo obtenemos a partir de la propiedades Inbox y Outbox de un objeto Session que representa una sesión ya establecida con el servicio SMTP. También lo podemos obtener a través del método GetDefaultFolder().

El objeto Folder suele representar las siguientes carpetas estándar:

- Bandeja de entrada (inbox): es la localización que tiene el correo recibido.
- Bandeja de salida (outbox): es la localización temporal que tiene el correo antes de proceder a su envío.

Mediante el objeto Folder vamos a tener acceso a los contenidos de la carpeta indicada. Aunque tenemos acceso a los mensajes este acceso es algo restrictivo, podremos eliminar mensajes existentes, leer mensajes existentes y enviar mensajes nuevos, sin embargo no podemos cambiar las propiedades de los mensajes ya existentes.

En este punto se debe hacer una aclaración acerca del servidor SMTP de IIS y los objetos de la librería CDONTS.

El servidor de correo SMTP de IIS 5.0 a diferencia del servidor Exchange Server, no posee buzones individuales, es decir, no existe una bandeja para cada usuario sino que todos los mensajes recibidos se almacenan en una misma localización y todos los mensajes enviados en otra.

Si hemos aceptado las configuraciones por defecto al instalar el servicio SMTP de IIS, la bandeja de entrada se corresponde con el directorio c:\inetpub\mailroot\Drop y la de salida con el directorio c:\inetpub\mailroot\Pickup.

Sin embargo aunque los buzones sean globales, cuando iniciamos la sesión con el servidor SMTP, a través de los objetos de CDONTS únicamente tendremos acceso a los mensajes que se corresponden con los del cliente que se conectado, de esta forma se intenta simular los buzones personales.

Debido a esto para utilizar el servicio SMTP no es necesario configurar buzones ni definir cuentas de correo de usuarios, de todas formas en los siguientes ejemplos todo esto se verá más claro.

Hecha esta aclaración volvemos a tratar el objeto Folder, empezando ahora a comentar las propiedades que nos ofrece.

- Name: propiedad de sólo lectura que contiene el nombre de la carpeta a la que representa el objeto Folder. Normalmente tiene los valores "Inbox" y "Outbox".
- Messages: esta propiedad devuelve una colección de objetos Message, que representará los mensajes presentes en la carpeta.

Veamos el tratamiento que tiene la colección Messages.

La colección Messages posee las siguientes propiedades:

- Count: número de objetos Message contenidos en la misma.
- Item(indice): devuelve un objeto Message de la colección cuyo índice coincide con el parámetro indicado. El primer índice es 1.

Y los siguientes métodos:

- Add(asunto, texto, importancia): permite añadir un nuevo mensaje a la colección. Todos los parámetros son opcionales, y cuando se ejecuta este método se devuelve un objeto Message para poder manipularlo. Este método únicamente se puede aplicar a una colección Messages que pertenece a la bandeja de salida.
- Delete(): elimina todos los mensajes de la colección. Para borrar un mensaje en concreto debemos utilizar el método Delete del objeto Message, pero eso lo veremos en el siguiente apartado.
- GetFirst(): método que devuelve el primer objeto Message de la colección Messages. Si el objeto no existe devuelve Nothing.
- GetLast(): método que devuelve el último objeto Message de la colección Messages. Si el objeto no existe devuelve Nothing.
- GetNext(): método que devuelve el siguiente objeto Message de la colección Messages. Si el objeto no existe devuelve Nothing.
- GetPrevious(): método que devuelve el anterior objeto Message de la colección Messages. Si el objeto no existe devuelve Nothing.

Vamos a ampliar y modificar el ejemplo del apartado anterior que utilizaba el objeto Session de CDONTS para establecer una conexión, para mostrar la utilización práctica del objeto Folder.

En este nuevo ejemplo vamos a establecer la sesión con el servicio SMTP con el mismo formulario y vamos a indicar el número de mensajes que posee el usuario conectado en ese momento en cada una de sus bandejas.

También aparece un nuevo botón que permite eliminar al usuario todos los mensajes de su bandeja de entrada. El Código fuente 212.

```
<!--METADATA TYPE="typelib" FILE="C:\Winnt\system32\cdonts.dll"-->
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%If Request.Form("Desconectar")<>"" AND Session("oSession")<>"" Then
    'se realiza la desconexión
    Set objSession=Session("oSession")
    objSession.LogOff
    Session("oSession")=""
Elseif Request.Form("EliminarInbox")<>"" Then
    'se eliminan todos los mensajes de Inbox
    Set objSession=Session("oSession")
    Set objFolderInbox=objSession.Inbox
    objFolderInbox.Messages.Delete
End if
If Request.Form("Conectar")="" AND Session("oSession")="" Then%
    <form action="<%=?Request.ServerVariables("SCRIPT_NAME")%>" method="post">
        Nombre:<input type="text" name="nombre" size="20" value=""><br>
        Cuenta de correo:<input type="text" name="cuenta" size="20"
value=""><br>
        <input type="submit" name="conectar" value="Conectar">
    </form>
```

```

<%Elseif Request.Form("Conectar")<>"" Then
    'se realiza la conexión
    Set objSession=Server.CreateObject ("CDONTS.Session")
    objSession.LogonSMTP Request.Form("nombre"),Request.Form("cuenta")
    'se guarda a nivel se sesión la referencia a la sesión establecida
    'con el servicio SMTP
    Set Session("oSession")=objSession
End if
If Session("oSession")<>"" Then
    'se recupera la sesión con SMTP
    Set objSession=Session("oSession")%
    Usuario conectado: <b><%=objSession.Name%></b><br>
    <form action=<%=Request.ServerVariables("SCRIPT_NAME")%>" method="post">
        <input type="submit" name="desconectar" value="Desconectar">
        <input type="submit" name="EliminarInbox" value="Vaciar Inbox">
    </form>
    <%Set objFolderInbox=objSession.GetDefaultFolder(cdoDefaultFolderInbox)%>
    El número de mensajes en <%=objFolderInbox.Name%> es
    <%=objFolderInbox.Messages.Count%><br>
    <%Set objFolderOutbox=objSession.Outbox%>
    El número de mensajes en <%=objFolderOutbox.Name%> es
    <%=objFolderOutbox.Messages.Count%>
<%End if%>
</BODY>
</HTML>

```

Código fuente 212

Y en la siguiente figura se muestra un ejemplo de un usuario conectado:

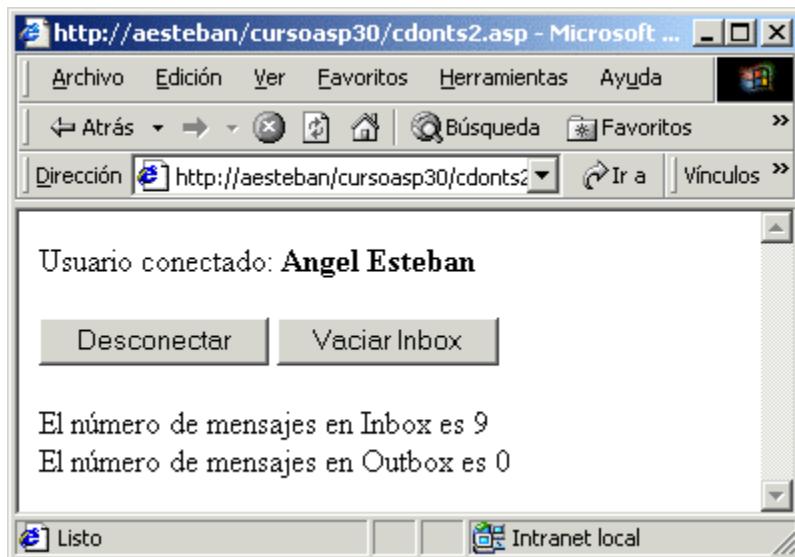


Figura 102. Utilizando el objeto Folder

En el siguiente apartado veremos como mostrar los datos de los mensajes contenidos en las bandejas y también como eliminar mensajes determinados y enviar nuevos.

## El objeto Message

Este objeto representa un mensaje contenido en un objeto Folder, es decir, en una carpeta. Una referencia a un objeto Message la obtenemos a través de la colección Messages del objeto Folder correspondiente.

En este apartado además de comentar el objeto Message vamos a tratar las dos colecciones que contiene: Recipients y Attachments, y también veremos el objeto AddressEntry, que permite especificar los datos del remitente.

El objeto Message lo utilizaremos para manipular los mensajes de correo, y para ello este objeto ofrece las siguientes propiedades:

- Attachments: una colección de objetos Attachment que contiene los ficheros adjuntos de un mensaje.
- ContentBase: URL base para los contenidos del mensaje.
- ContentID: identificador del tipo de contenido MIME del mensaje.
- ContentLocation: URL relativa para los contenidos del mensaje.
- HTMLText: cuerpo del mensaje en formato HTML.
- Importance: prioridad del mensaje.
- MessageFormat: formato de codificación del mensaje.
- Recipients: colección de objetos Recipient que representa a cada uno de los destinatarios del mensaje.
- Sender: devuelve un objeto AddressEntry que representa al remitente del mensaje.
- Size: tamaño en bytes del mensaje.
- Subject: asunto del mensaje.
- Text: cuerpo del mensaje en formato de texto plano.
- TimeReceived: fecha y hora de recepción del mensaje.
- TimeSent: fecha y hora de envío del mensaje.

Muchas de estas propiedades son comunes a las del objeto NewMail.

El objeto Message ofrece los siguientes métodos:

- Send(): este método envía el mensaje a los destinatarios indicados.
- Delete(): elimina el mensaje de forma permanente.

Vamos a seguir utilizando el mismo ejemplo que el de los apartados anteriores para mostrar la utilización del objeto Message. En este caso se trata de mostrar el contenido de la bandeja de entrada

del usuario que se ha conectado. La información que se va a mostrar de cada mensaje es la prioridad, remitente, asunto y fecha de recepción.

Para mostrar el contenido de la bandeja de entrada simplemente recorreremos la colección Messages de objeto Folder que se corresponde con inbox. Y utilizaremos las propiedades correspondientes de cada objeto Message para mostrar la información deseada.

El código que se debe añadir al ejemplo anterior, después de haber obtenido una referencia a la carpeta Inbox, es el Código fuente 213.

```
<%If objFolderInbox.Messages.Count>0 Then%>
    <table border="0">
        <tr><th>Prioridad</th>
        <th>De</th>
        <th>Asunto</th>
        <th>Recibido</th>
    </tr>
    <%For Each objMessage In objFolderInbox.Messages%>
        <tr>
            <td><%=Prioridad(objMessage.Importance)%></td>
            <td><%=objMessage.Sender%></td>
            <td><%=objMessage.Subject%></td>
            <td><%=objMessage.TimeSent%></td>
        </tr>
    <%Next%>
</table>
<%End if%>
```

Código fuente 213

Se utiliza una función llamada Prioridad, cuya función es la de devolver una cadena con el texto que representa a la prioridad correspondiente, su código es el Código fuente 214.

```
<%Function Prioridad(prio)
    Select Case prio
        Case cdoLow: Prioridad="Baja"
        Case cdoNormal: Prioridad="Normal"
        Case cdoHigh: Prioridad="Alta"
    End Select
End Function%>
```

Código fuente 214

Y un ejemplo de la ejecución del Código fuente 214 lo tenemos en la Figura 103.

Como ya hemos comentado al principio el objeto Message ofrece dos colecciones a través de sus propiedades Recipients y Attachments.

La colección Recipients contiene objetos Recipient que representan a cada uno de los destinatarios de un mensaje. Para añadir un nuevo destinatario utilizamos el método Add() sobre la colección Recipients, que además nos devolverá un objeto Recipient por si deseamos manipularlo. La sintaxis del método Add() es la siguiente:

```
Set objRecipient=colRecipients.Add(nombre, dirección, tipo)
```

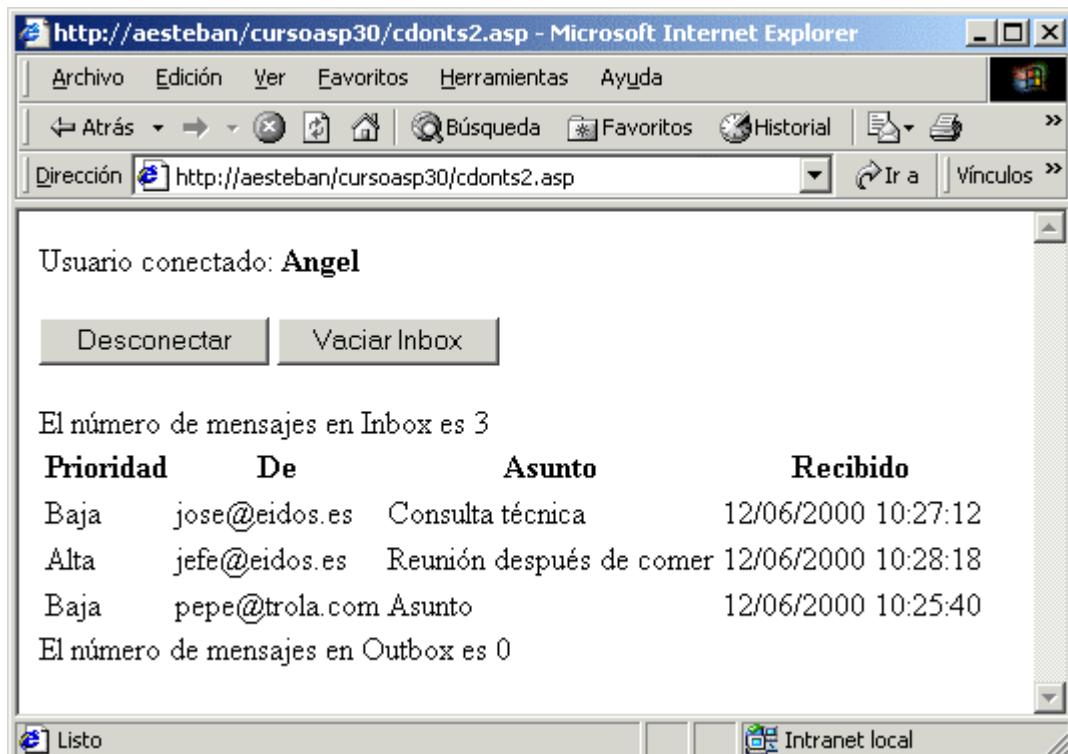


Figura 103. Mensajes de la bandeja de entrada

Todos los parámetros de este método son opcionales. En nombre indicamos el nombre del destinatario que se va a mostrar, en dirección se indica la dirección de correo del destinatario y en tipo una constante que indica si es un destinatario de copia o no. Los valores de tipo pueden ser CdoTO (es el destinatario sin copia), CdoCC (se le envía copia) o CdoBCC (se le envía copia pero no aparece en la lista de los demás destinatarios).

Los mensajes contenidos en la bandeja de entra no se pueden modificar, sólo son de lectura, por lo tanto si lanzamos el método Add() sobre una colección Recipients de un objeto Message que pertenezca al objeto Folder Inbox, se producirá un error.

El método Delete de la colección Recipients elimina todos los objetos Recipient de la colección.

Los parámetros que aparecían en el método Add() de la colección Recipients se corresponden con las propiedades del objeto Recipient, y son las siguientes:

- Name: nombre del destinatario. Es el nombre que se va a mostrar.
- Address: dirección de correo del destinatario.
- Type: tipo de destinatario, puede tener uno de los siguientes valores: CdoTO, CdoCC, CdoBCC.

La otra colección que podemos encontrar en el objeto Message es la colección Attachments que se encuentra formada por objetos Attachment y que representa los ficheros adjuntos de un mensaje determinado.

Mediante el método Delete() de la colección Attachments eliminamos todos los ficheros adjuntos de un mensaje, y mediante el método Add() se añade un nuevo fichero adjunto.

Al igual que ocurría con la colección Recipients, al método Add() devuelve un objeto Attachment para poder manipularlo, la sintaxis de este método es:

Set objAttachment=colAttachments.Add(nombre, tipo, fuente, localización, base)

Todos los parámetros de este método son opcionales. El parámetro nombre indica el nombre con el que se va a mostrar el adjunto, el tipo de adjunto puede ser cdoFileData(1) para indicar un fichero y cdoEmbeddedMessage(4) para indicar que el adjunto es otro mensaje. En fuente se indica la ruta completa del fichero adjunto o bien un objeto Message en el caso de ser el adjunto de tipo cdoEmbeddedMessage. Localización y base son dos parámetros que especifican cabeceras para ficheros adjuntos de tipo MIME.

El método Add() no se puede utilizar con mensajes que se encuentren en la bandeja de entrada, ya que estos mensajes son únicamente de lectura.

El objeto Attachment ofrece una serie de propiedades que se corresponden con los parámetros del método Add() de la colección Attachments, y son las siguientes:

- ContentBase: cabecera para un adjunto MIME que especifica la dirección URL base.
- ContentID: identificador único para un adjunto MIME.
- ContentLocation: URL relativa para un adjunto MIME.
- Name: nombre con el que se va a mostrar el fichero adjunto.
- Source: ruta o localización del adjunto.
- Type: tipo de adjunto, puede presentar las siguientes constantes cdoFileData o cdoEmbeddedMessage.

Además el objeto Attachment ofrece los siguientes métodos:

- Delete(): para eliminar un adjunto determinado.
- ReadFromFile(nombreFichero): carga el adjunto desde el fichero indicado por parámetro. Se debe indicar la ruta completa del fichero.
- WriteToFile(nombreFichero): escribe en el fichero indicado el contenido del adjunto.

Relacionado con el objeto Message también encontramos el objeto AddressEntry. El objeto AddressEntry se obtiene a través de la propiedad Sender del objeto Message y representa al remitente del mensaje.

Este objeto ofrece las siguientes propiedades:

- Name: nombre del remitente o alias, es el nombre que se muestra.
- Address: dirección de correo electrónico del remitente.
- Type: tipo de dirección. Para CDONTS siempre es "SMTP".

Ahora vamos a mostrar un ejemplo que utiliza todos estos nuevos objetos. Para ello vamos a retomar el ejemplo visto con el objeto NewMail, y lo que vamos a hacer es la misma operación, es decir,

enviar un correo nuevo, pero utilizando algunos de los objetos vistos hasta ahora que dependen del objeto Session inicial de la librería CDONTS.

```
<!--METADATA TYPE="typelib" FILE="C:\Winnt\system32\cdonts.dll"-->
<%'se realiza la conexión
Set objSession=Server.CreateObject("CDONTS.Session")
objSession.LogonSMTP "Angel", "aesteban@mail.angel.es"
'se obtiene la bandeja de salida
Set objFolderOutbox=objSession.Outbox
'se obtiene la colección de mensajes
Set colMensajes=objFolderOutbox.Messages
strTexto="Mensaje enviado con los objetos de CDONTS"
'se añade un mensaje nuevo y se obtiene la referencia al mismo
Set objMensaje=colMensajes.Add("Prueba de CDONTS",strTexto,cdoNormal)
'añadimos un adjunto al mensaje
objMensaje.Attachments.Add "Fichero Adjunto",cdoFileData,"c:\tmp\imagen.png"
'añadimos un destinatario al mensaje
objMensaje.Recipients.Add "Jose María", "chema@mail.angel.es", cdoTO
'añadimos un destinatario al que se envía una copia
objMensaje.Recipients.Add "Pepe", "pepe@mail.angel.es", cdoCC
'se envía el mensaje
objMensaje.Send%>
```

Código fuente 215

Como podemos comprobar es mucho más sencillo y recomendable utilizar el objeto NewMail a la hora de enviar correo. El mensaje se ha enviado con el remitente que corresponde a la sesión actual, en este caso aesteban@mail.angel.es. Si abrimos el mensaje que hemos creado con el cliente de correo Outlook Express, vemos que tiene el aspecto que nos muestra la Figura 104.

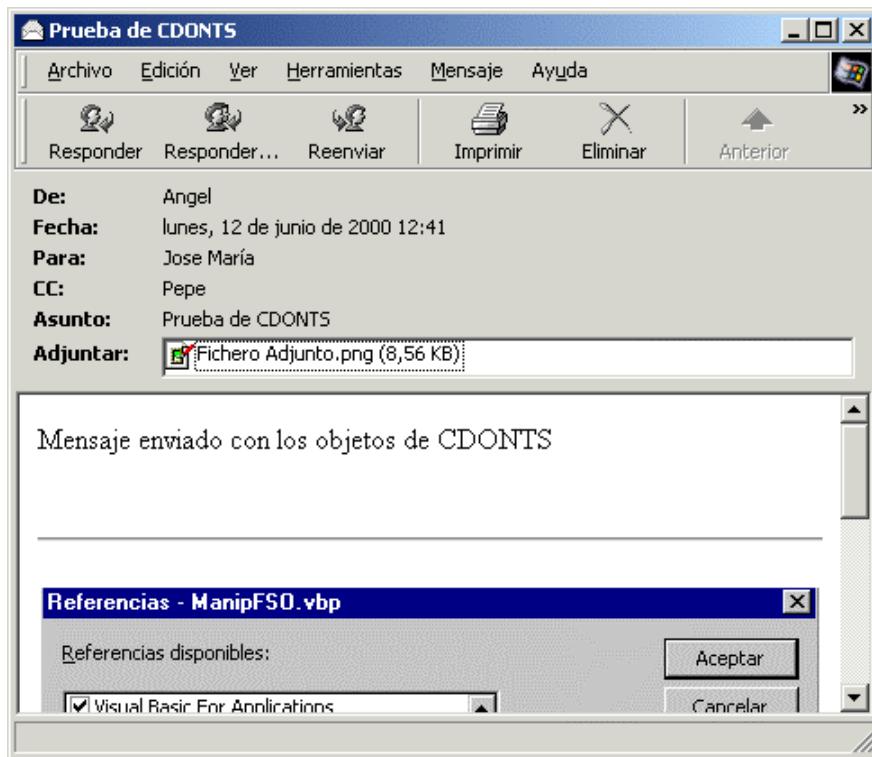


Figura 104. Mensaje leído con Outlook Express

Y si queremos podemos ver los detalles del mismo(Figura 105).

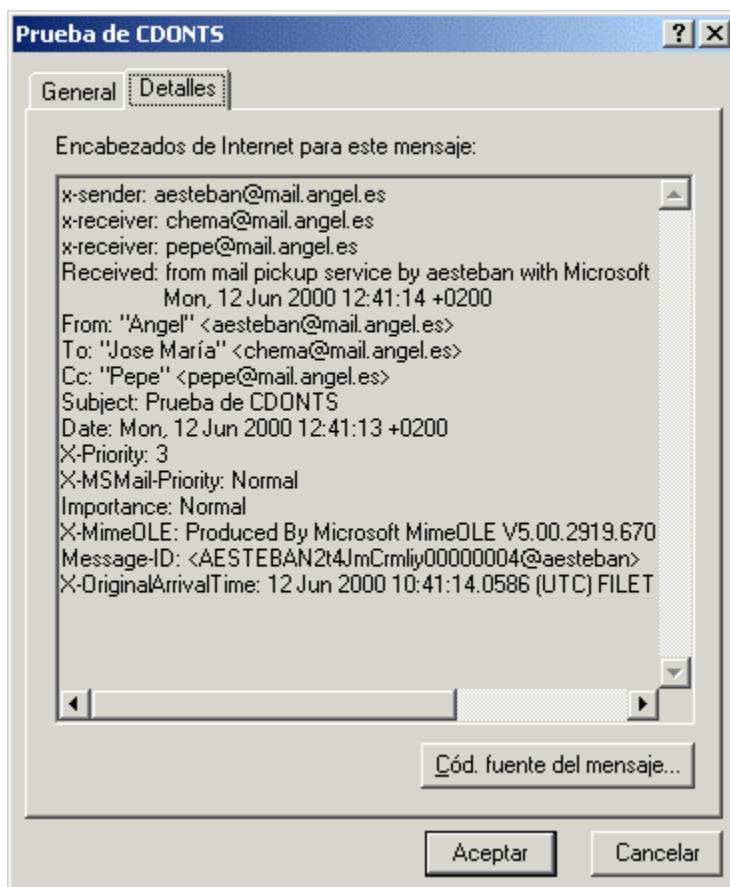


Figura 105. Detalles del mensaje

Antes de terminar este tema vamos a añadir una mayor funcionalidad al ejemplo en el que mostrábamos el contenido de la bandeja de entrada de un usuario conectado a una sesión de nuestro servidor SMTP.

La nueva funcionalidad consiste en poder visualizar el contenido de cada uno de los mensajes que se encuentran en la bandeja de entrada del usuario conectado actualmente. Para ello vamos a situar en un enlace los asuntos de cada uno de los mensajes y para identificar cada uno de ellos utilizar una variable del `QueryString` que va a contener un entero para cada mensaje.

En la página de lectura del mensaje además vamos a permitir la opción de borrar el mensaje que estamos leyendo.

Veamos el Código fuente 216 que debemos modificar en la página que muestra el contenido de la bandeja de entrada. Se debe añadir la creación de los enlaces para cada asunto en el bucle `For Each` que recorre todos los mensajes.

```

<% i=0
For Each objMessage In objFolderInbox.Messages
    i=i+1%>
    <tr>
        <td><%=Prioridad(objMessage.Importance)%></td>
        <td><%=objMessage.Sender%></td>

```

```

<td><a href="LeerMensaje.asp?idmensaje=<%=i%>"><%=objMessage.Subject%></a></td>
<td><%=objMessage.TimeSent%></td>
</tr>
<%Next%>

```

Código fuente 216

Como se puede comprobar en los enlaces se hace referencia a una página llamada LeerMensaje.asp, es en esta página dónde se encuentra el código necesario para mostrar el contenido del mensaje solicitado.

La página de lectura de mensajes va a tener un área de texto para mostrar el contenido del mensaje, y también dos botones: uno para volver a la página anterior, y otro para eliminar el mensaje que estamos leyendo. La página encargada de borrar el mensaje no es la de lectura de los mimos, sino que el mensaje será borrado en la página que muestra el contenido de al bandeja de entrada.

Antes de nada veamos el Código fuente 217 de la página de lectura de mensajes.

```

<%'recuperamos la sesión con el servidor SMTP
Set objSession=Session("oSession")
Set colMensajes=objSession.Inbox.Messages
'recuperamos el identificador del mensaje
idmensaje=Request.QueryString("idmensaje")
'se recupera el mensaje correspondiente de la colección
Set objMensaje=colMensajes.Item(idmensaje)%>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%'se obtiene el objeto que representa al remitente
Set objAddressEntry=objMensaje.Sender%>
<b>De:</b> <%=objAddressEntry.Name%> (<%=objAddressEntry.Address%>)<br>
<b>Asunto:</b> <%=objMensaje.Subject%><br>
<form action="ListadoMensajes.asp" method="post">
<textarea rows="10" cols="50">
<%=objMensaje.Text%>
</textarea><br>
<input type="submit" name="Volver" value="Volver">
<input type="hidden" name="idmensaje" value="<%=idmensaje%>">
<input type="submit" name="Eliminar" value="Eliminar">
</form>
<%'si existen adjuntos en el mensaje se muestran enlaces a los mismos
If objMensaje.Attachments.Count>0 then%
    <b>Adjuntos:</b><br>
    <ul>
        <%For Each objAdjunto in objMensaje.Attachments
            'se almacena el adjunto en un directorio del servidor Web
            objAdjunto.WriteToFile(Server.MapPath("/cursoasp30") &"\" &objAdjunto.Name)
            'se construye el enlace al adjunto%>
            <li><a href="<%=objAdjunto.Name%>"><%=objAdjunto.Name%></a><br></li>
        <%Next%>
    </ul>
<%End if%>
</BODY>
</HTML>

```

Código fuente 217

Como se puede ver también se crean enlaces para los adjuntos del mensaje si los tiene. Los ficheros adjuntos se graban en el servidor Web con el método WriteToFile() del objeto Attachment, para que los pueda recuperar el destinatario del mensaje. La forma de grabar el adjunto en una misma carpeta y aprovechando el valor de su propiedad Name, no demasiado adecuado, ya que si existe un fichero adjunto con ese mismo nombre se sobreescibirá.

Veamos, en la Figura 106, el aspecto que ofrece esta nueva página.

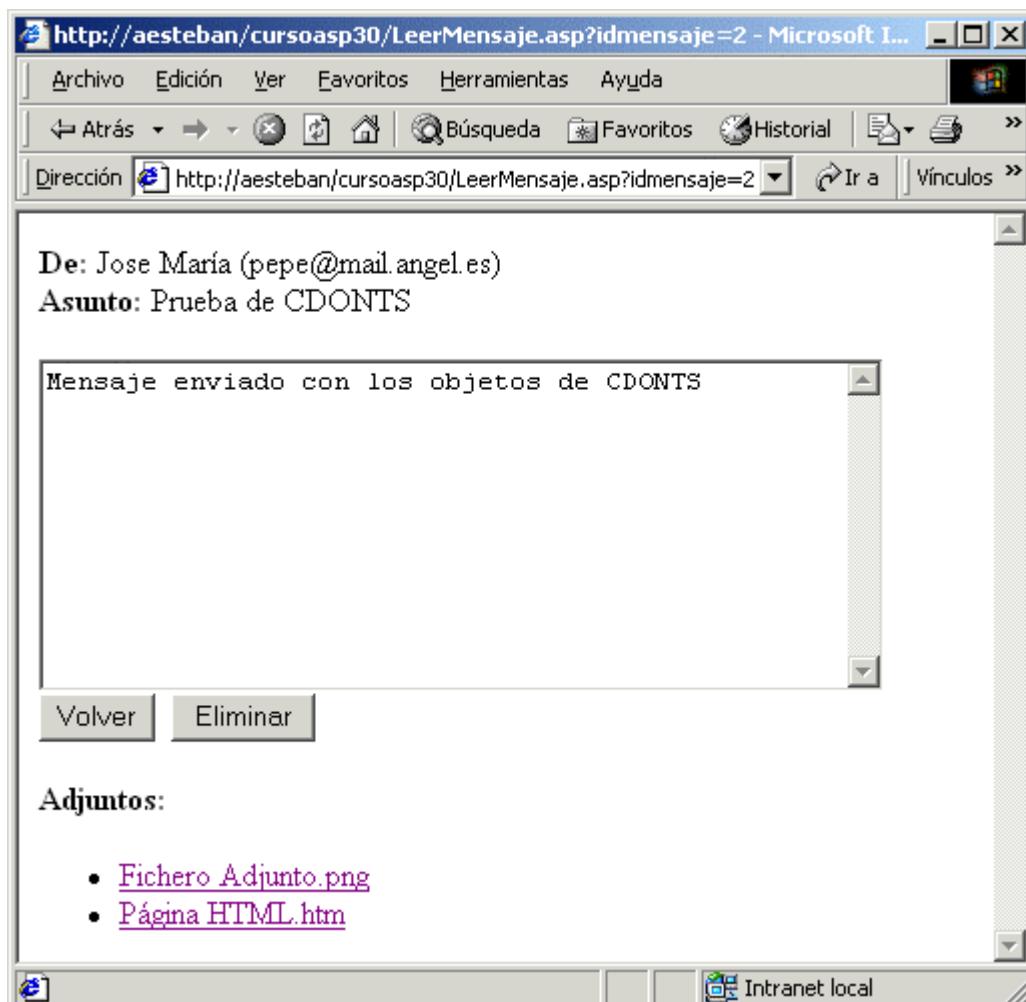


Figura 106. Página de lectura del mensaje

La página LISTADOMENSAJES.ASP le pasa a LEERMENSAJE.ASP a través del QueryString un parámetro para identificar el mensaje que deseamos leer. Igualmente la página LEERMENSAJE.ASP le devuelve a la página LISTADOMENSAJES.ASP este mismo identificador de mensaje para que cuando sea demandando la página LISTADOMENSAJES.ASP pueda eliminar el mensaje deseado. Para que la página LISTADOMENSAJES.ASP pueda eliminar el mensaje solicitado, deberemos añadir el Código fuente 218 justo antes de empezar a recorrer la colección de mensajes presentes en la bandeja de entrada..

```
<%If Request.Form("Eliminar")<>"" Then
    'se obtiene el mensaje que se desea borrar
    Response.Write "paso: "& Request.Form("idmensaje")
```

```
Set objMensaje=objFolderInbox.Messages.Item(Request.Form("idmensaje"))
'se borra
objMensaje.Delete
End if%>
```

Código fuente 218

Simplemente se obtiene el objeto Message correspondiente de la colección Messages y se le lanza el método Delete().

En el siguiente [enlace](#) se encuentra el código completo de este ejemplo.



# Introducción a ActiveX data Objects (ADO)

---

## Introducción

En este tema vamos a realizar una introducción al componente de servidor incluido en la instalación de ASP que nos faltaba por explicar. A este componente de servidor, denominado componente de acceso a bases de datos, se le dedica varios temas debido a su importancia y complejidad.

El componente de acceso a bases de datos, es en realidad un conjunto de objetos que se agrupan dentro de la denominación ActiveX Data Objects (ADO). Mediante el conjunto de objetos ActiveX Data Objects vamos a tener acceso a bases de datos, es decir, nos va a permitir realizar conexiones a bases de datos, ejecutar sobre ellas sentencias SQL, procedimientos almacenados, obtener resultados, etc. Es decir, nos va a ofrecer desde nuestras páginas ASP todas las operaciones que se suelen realizar sobre una base de datos.

Como ya se ha comentado, ADO está formado por diferentes objetos que nos van a permitir realizar operaciones con bases de datos, estos objetos se encuentran relacionados entre sí dentro de una jerarquía, pero se tiene la ventaja que estos objetos pueden ser creados fuera de esta jerarquía, es decir, los objetos ADO existen más allá de la jerarquía de su modelo de objetos.

## OLE DB

Antes de entrar en los detalles de ADO vamos a comentar el API (Application Program Interface) en el que se basa ADO. ActiveX Data Objects (ADO) nos permite desarrollar aplicaciones ASP para

acceder y manipular bases de datos a través de un proveedor OLE DB (Object Linking and Embedding para bases de datos). ADO es la primera tecnología de Microsoft basada en OLE DB.

OLE DB es una especificación basada en un API construido con C++, por lo tanto se encuentra orientado a objetos. OLE DB consiste en consumidores de datos y proveedores de datos. Los consumidores toman los datos desde interfaces OLE DB, los proveedores ofrecen estos interfaces OLE DB. En algunos casos OLE DB puede acceder a los datos de forma más rápida que DAO y RDO, esto es así debido a que DAO y RDO deben pasar a través de la capa ODBC y OLE DB se puede conectar directamente a fuentes de datos relacionales con el proveedor correspondiente.

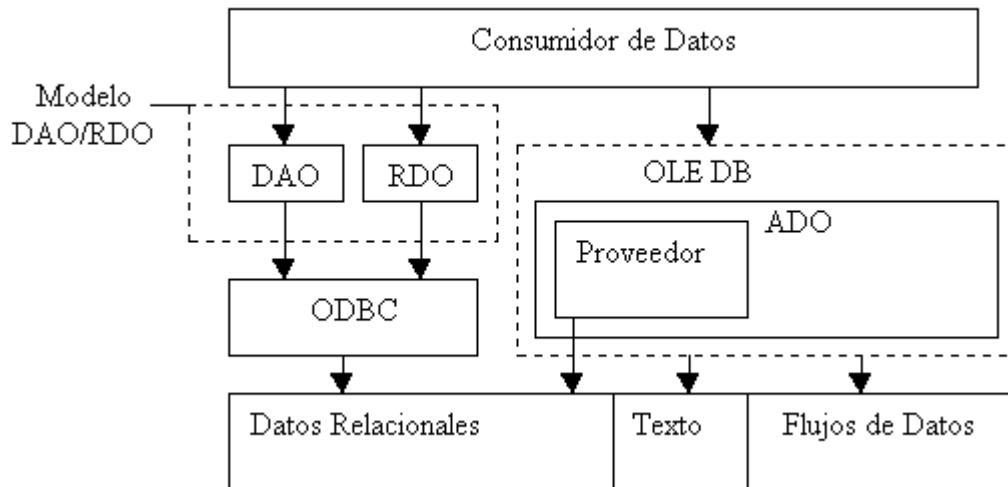


Figura 107. ADO y OLE DB

OLE DB puede ser utilizado para extender la funcionalidad de proveedores de datos sencillos. Estos objetos más especializados y sofisticados se denominan proveedores de servicio, permiten abstraer al consumidor de datos del tipo de datos al que se accede y su localización. Los proveedores de servicio son consumidores y proveedores, es decir, un proveedor de servicio puede consumir interfaces OLE DB y construir una tabla sencilla para ese entrada, y entonces puede ofrecer interfaces OLE DB para un aplicación cliente para que construya su salida en HTML, este ejemplo sería nuestro caso, es decir una página ASP que usa ADO.

Debemos tener en cuenta que OLE DB es una especificación de bajo nivel. ADO encapsula el API de OLE DB en un API de más alto nivel que proporciona dentro de su tecnología de componentes un lenguaje neutral, que permite aprovechar todas las ventajas y características de OLE DB sin realizar una programación a bajo nivel.

Ya hemos introducido el concepto de proveedor y consumidor en el que se basa OLE DB, el consumidor es simplemente el que consume (utiliza) los datos y el proveedor el que ofrece esos datos. Para acceder a una base de datos determinada debemos tener el proveedor OLE DB correspondiente. De esta forma el acceso a una base de datos determinada viene condicionado por la existencia del proveedor OLE DB correspondiente.

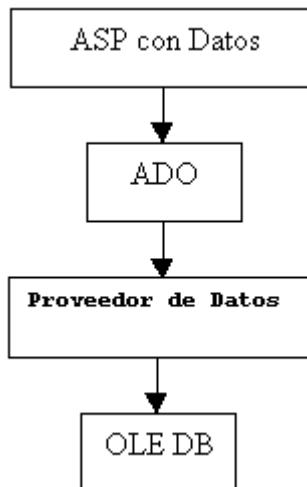


Figura 108. ASP y ADO

Para las bases de datos que no existan o no dispongamos de su proveedor OLE DB podemos utilizar el proveedor OLE DB para ODBC, ya que ODBC se encuentra más difundido y es posible que exista el driver ODBC correspondiente. Es importante que no confundamos los proveedores con los drivers, en la Figura 109 se muestra un esquema dónde se pueden distinguir el lugar que ocupan los drivers y los proveedores OLE DB.

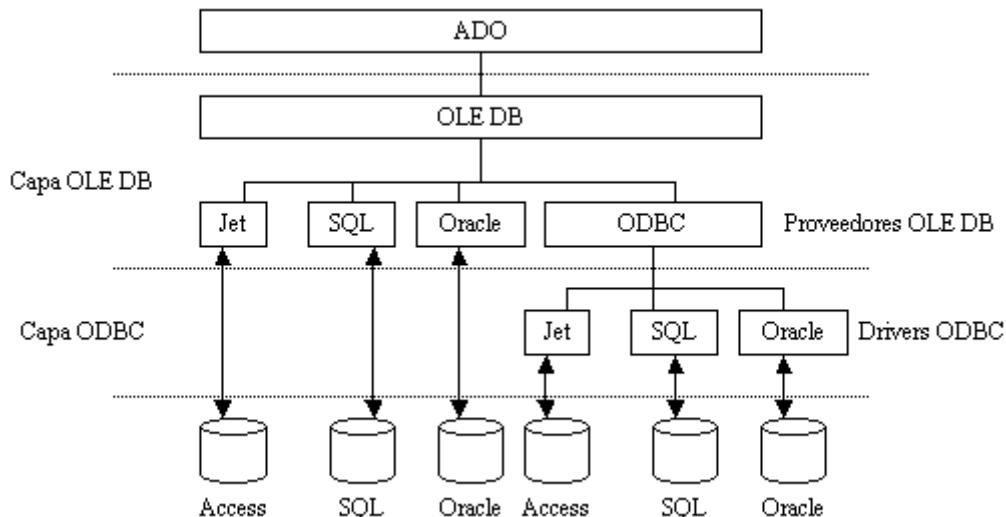


Figura 109. Proveedores y drivers

Como se puede observar los proveedores se encuentran en la capa OLE DB y los drivers en la capa ODBC. ADO 2.5 por defecto ofrece los siguientes proveedores OLE DB.

- Jet OLE DB 4.0: para acceder a bases de datos de Microsoft Access.
- DTS Packages: para los servicios de transformación de datos de SQL Server, DTS (Data Transformation Services).
- Internet Publishing: para acceder a servidores Web.

- Indexing Service: para acceder a catálogos de índices.
- Site Server Search: para acceder al catálogo de búsqueda de Site Server.
- ODBC Drivers: para acceder a fuentes de datos ODBC.
- OLAP Services: para el servidor OLAP de Microsoft.
- Oracle: para acceder a servidores de bases de datos Oracle.
- SQL Server: para acceder a servidores de datos de Microsoft SQL Server.
- Simple Provider: para ficheros de texto simples.
- MSDataShape: para datos jerárquicos.
- Microsoft Directpry Services: para acceder a los servicios de directorios de Windows 2000.
- DTS Flat File: para el gestor de ficheros planos de los servicios de transformación de datos (DTS) de MS SQL Server.

De todas formas existen más proveedores OLE DB que son ofrecidos por otras empresas.

En este curso se van a utilizar principalmente los proveedores OLE DB de ODBC y de SQL Server.

## Características generales de ADO

ADO permite crear aplicaciones ASP para acceder y manipular datos en una base de datos a través de un proveedor OLE DB. Las principales ventajas de ADO son su velocidad, facilidad de uso, baja carga de memoria y requieren poco espacio en el disco duro.

Una ventaja de ADO es que si ya lo hemos utilizado en Visual Basic 6, veremos que será prácticamente igual utilizarlo en ASP, aunque eso sí, teniendo siempre en cuenta el entorno tan particular en el que nos encontramos.

Las características que posee ADO para construir aplicaciones cliente/servidor y aplicaciones ASP son las siguientes:

- Sus objetos se pueden crear de forma independiente fuera de su jerarquía o modelo de objetos. No es necesario navegar a través de una jerarquía para crear un objeto, la mayoría de los objetos ADO se pueden instanciar de forma independiente. Esto nos permite crear únicamente los objetos que necesitamos.
- Ofrece soporte para realizar llamadas a procedimientos almacenados con parámetros de entrada/salida.
- Diferentes tipos de cursos.
- Soporta resultados múltiples devueltos desde procedimientos almacenados.

Se debe tener en cuenta que aunque ADO aporte todas estas características, los proveedores y drivers a los que llama ADO pueden no contemplar estas características. Por lo tanto antes debemos consultar la

documentación de los proveedores OLE DB que vamos a utilizar para informarnos de las características que ofrecen.

El entorno de utilización de ADO se encuentra enmarcado dentro del modelo de tres capas. En el modelo de tres capas (*three-tier*), los comandos del cliente son enviados a una capa intermedia (*middle-tier*) de servicios, la cual enviará sentencias SQL a la base de datos. La base de datos procesa las sentencias y devuelve los resultados a la capa intermedia que se los enviará al usuario. En nuestro caso el cliente será el navegador que ha cargado la página ASP y la capa intermedia serán los componentes ActiveX Data Objects (ADO). En la Figura 110 se muestra esta situación:

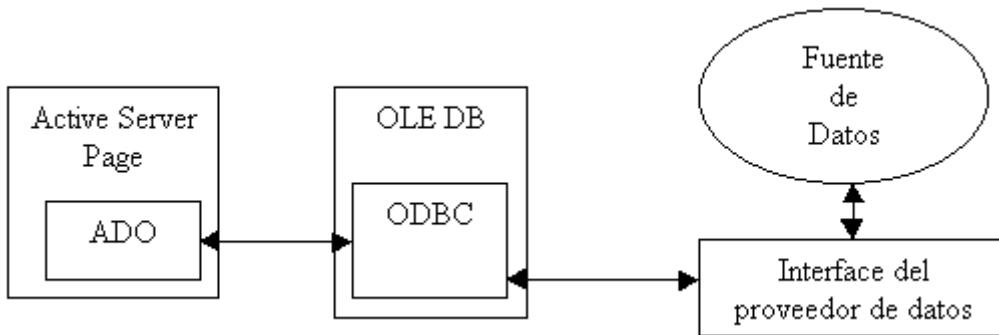


Figura 110. Modelo en tres capas.

ADO utiliza la característica pooling de conexiones de ODBC 3.0 para efectuar el acceso a las bases de datos de forma más eficiente. Esta característica consiste en mantener abiertas conexiones de bases de datos y administrar la compartición de conexiones entre diferentes solicitudes de usuarios para mantener el rendimiento y reducir el número de conexiones inactivas.

En cada solicitud de conexión se determina si del grupo de conexiones hay una conexión inactiva, si la hay, el grupo de conexiones devuelve esa conexión en lugar de efectuar una nueva conexión a la base de datos.

Por defecto el pooling de conexiones está activado en ASP. Para desactivar esta característica se debe establecer a cero la entrada de registro llamada StartConnectionPool.

Esta característica es bastante útil para ASP debido al entorno en el que nos encontramos. Las aplicaciones tradicionales de bases de datos suelen crear una única conexión con la base de datos que se utiliza durante toda la duración de la aplicación. Sin embargo, debido a que en el entorno Web no se puede mantener la información entre diferentes peticiones de páginas, una aplicación de bases de datos en el entorno Web debe abrir y cerrar una conexión en cada página.

Además no es recomendable almacenar referencias a objetos de ADO a nivel de variables del objeto Session o Application.

## Modelo de objetos de ADO

Como se ha dicho anteriormente ADO es más complejo que el resto de los componentes ActiveX Server que se incluyen con ASP. ADO presenta un completo modelo de objetos que se muestra en la Figura 111.

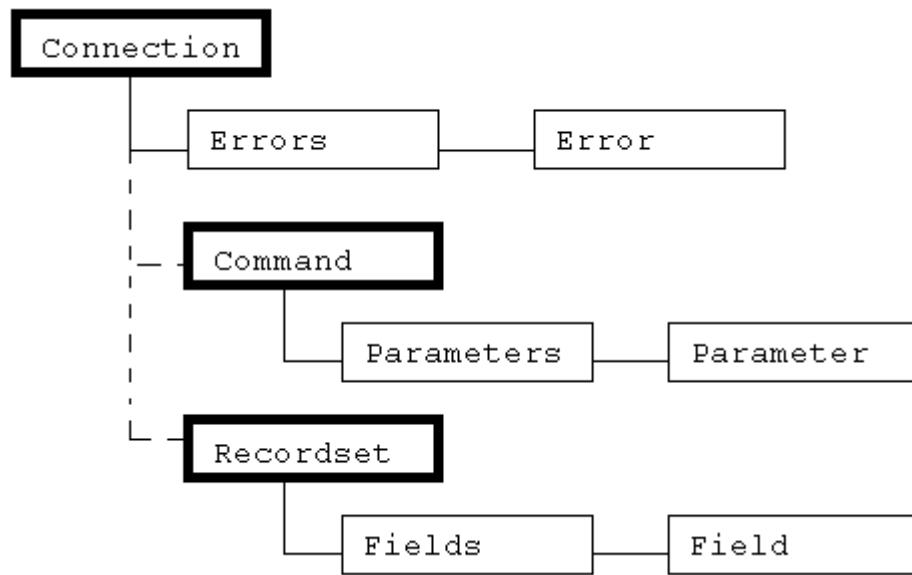


Figura 111. Modelo de Objetos de ADO

Como se puede observar en la figura hay tres objetos que se diferencian del resto, es decir, los objetos: Connection, Command y Recordset. Estos objetos son los principales dentro de ADO, y se pueden crear fuera de la jerarquía de la figura, y se pueden manejar sin tener que interactuar con un intermediario. Estos tres objetos se crearán mediante el método `CreateObject` del objeto Server, al igual que hacíamos con el resto de los componentes de servidor, el resto de los objetos de ADO se crean a partir de estos tres.

Desde el punto de vista práctico el objeto Recordset es el más importante, ya que es el que va a permitir acceder a los registros de la base de datos, pero los objetos Connection y Command permiten la creación de un objeto Recordset.

ASP 3.0 incluye la última versión de ADO, la versión 2.5, que incluye dos nuevos objetos: el objeto Record y el objeto Stream. De todas formas ya adelantamos que desde ASP 3.0 todavía no se encuentra implementada de forma satisfactoria estos dos objetos.

## Principales objetos de ADO

Podemos decir que los principales objetos de ADO son tres: Connection, Recordset y Command. En este apartado se ofrece una breve descripción y una primera toma de contacto con estos objetos.

Trataremos en profundidad cada uno de los objetos en los capítulos correspondientes. En este apartado se trata de realizar una visión general y una primera aproximación.

El objeto Connection de ADO representa una conexión con una base de datos. La función del objeto Connection es la de recoger toda la información del proveedor de datos del que se podrá crear si es necesario un objeto Recordset. La base de datos a la que se quiere realizar la conexión se debe indicar mediante la cadena de conexión correspondiente.

Para establecer la conexión con la base de datos a través del objeto Connection se debe utilizar el método `Open` que recibirá como parámetro la cadena de conexión que podrá contener la siguiente información: el proveedor que se desea utilizar, el servidor de la base de datos, la base de datos a la

que nos queremos conectar, el usuario y la contraseña de la misma. Esta información varía según el proveedor OLE DB que se utilice.

A través de un objeto Connection podremos también realizar transacciones utilizando los métodos BeginTrans, CommitTrans y RollbackTrans.

Además de abrir una conexión, mediante el objeto Connection podremos ejecutar sentencias SQL, para ello lanzaremos el método Execute. El método Execute tomará como parámetro un comando SQL y lo ejecutará sobre la base de datos a la que corresponde la conexión, y devolverá un objeto Recordset, para recuperar este objeto deberemos escribir lo que indica el Código fuente 219.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "DSN=FuenteDatos;UID=pepe;PWD=xxx"
Set objRecordset=objConexion.Execute ("SELECT * FROM Usuarios") %>
```

Código fuente 219

En el Código fuente 219, además de recuperar un objeto Recordset a partir de la ejecución de una sentencia SQL sobre una base de datos, también se puede observar como se realiza una conexión a una fuente de datos llamada FuenteDatos, utilizando el usuario pepe que posee la contraseña xxx. En este caso se está utilizando el proveedor OLE DB para ODBC, aunque no lo hemos indicado en la cadena de conexión, pero al indicar una DSN ADO hace uso de este proveedor de forma automática.

Para poder utilizar una DSN la debemos definir en el servidor Web y para ello se debe utilizar el gestor de drivers de ODBC que podemos encontrar en el panel de control de Windows. Se pueden crear diferentes tipos de DSN: DSN de usuario, de sistema y de fichero. En este caso se debe crear una DSN de sistema dentro del servidor Web en el que se va a encontrar la aplicación ASP.

Una DSN de sistema estará disponible para todos los usuarios y servicios del sistema en el servidor. A una DSN de sistema se puede acceder de forma sencilla a través de su nombre. Para crear la DSN de sistema en el servidor Web se debe utilizar la utilidad Orígenes de datos (ODBC) incluida en la opción Herramientas administrativas del panel de control. Dentro del gestor de fuentes de datos se debe pulsar sobre el botón etiquetado como Agregar, elegir el driver adecuado, seleccionar la base de datos con los parámetros que sean necesarios y pulsar el botón Finalizar. Esta misma conexión la podemos establecer utilizando otro proveedor OLE DB, como se puede ver en el Código fuente 220.

```
<% Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;User ID=pepe;Password=xxx;Initial
Catalog=BD;" &
                "Data Source=servidor"
Set objRecordset=objConexion.Execute ("SELECT * FROM Usuarios") %>
```

Código fuente 220

En este caso se ha utilizado el proveedor OLE DB de SQL Server, también se indica que el servidor en el que se encuentra la base de datos. Este proveedor no necesita de ninguna configuración en el servidor Web, por lo que es más flexible de utilizar que el proveedor OLE DB de SQL Server que el proveedor de ODBC, además si utilizamos el proveedor ODBC tenemos un paso intermedio más, que es el que existe entre la fuente de datos ODBC y el driver ODBC correspondiente.

Pasamos ahora a otro objeto de ADO, el objeto Recordset. El objeto Recordset generado de esta forma (ejemplo anterior) tiene una serie de propiedades por defecto que hacen de este Recordset el cursor menos potente, pero que puede ser muy útil para algunas ocasiones. Representa un cursor estático de sólo lectura, en el que únicamente se puede avanzar hacia delante.

Un objeto Recordset contendrá un conjunto de registros de la base de datos, cada fila de un objeto Recordset es un registro.

La segunda forma de obtener un objeto Recordset es a través del objeto Command. Un objeto Command representa una llamada a un procedimiento almacenado de una base de datos, a través de este objeto podremos pasar parámetros a los procedimientos almacenados. Aunque también podemos utilizar el objeto Command para ejecutar sentencias SQL precompiladas.

La llamada al método Execute de un objeto Command originará un objeto Recordset con las mismas características que cuando se llamaba al método Execute del objeto Connection. En el Código fuente 221 se muestra como se ejecutaría un objeto Command y guardaría su resultado en un objeto Recordset creado por defecto.

```
<%Set objComando=Server.CreateObject ("ADODB.Command")
objComando.ActiveConnection=objConexion
objComando.CommandText="devuelveNombres"
Set objRecordset=objComando.Execute%>
```

Código fuente 221

Como se puede observar en el código anterior, se le debe indicar al objeto Command sobre que conexión se va a realizar la ejecución del procedimiento almacenado, una de las formas de hacerlo es asignándole a la propiedad ActiveConnection el objeto que representa la conexión.

Cuando queremos tener un control total sobre el tipo de cursor que se va a crear deberemos crear explícitamente un objeto Recordset, de esta forma podremos manipular sus propiedades directamente.

Para que el objeto Recordset se "llene" de datos procedentes de la base de datos deberemos utilizar su método Open, y sus propiedades Source, para indicar la fuente de los datos (objeto Command, sentencia SQL, nombre de la tabla o procedimiento almacenado), y ActiveConnection para indicar la conexión sobre la que se realizarán las operaciones. En el Código fuente 222 se muestra un ejemplo, cuya fuente de datos es una tabla llamada teléfono.

```
<%Set objRecordset=Server.CreateObject ("ADODB.Recordset")
objRecordset.Source="telefono"
objRecordset.ActiveConnection="DSN=telefonos;UID=sa;PWD=xxxx"
objRecordset.Open%>
```

Código fuente 222

En este caso, a la propiedad ActiveConnection se le asigna, en lugar del objeto Connection correspondiente, la cadena de conexión que representa la conexión con una DSN determinada.

Como se puede observar a través de los ejemplos que hemos mostrado, dentro del entorno de el acceso a datos con ADO existen muchas formas de hacer una misma cosa, por lo tanto deberemos utilizar la forma más adecuada atendiendo a la situación en la que nos encontramos.

## UDA

Tanto ADO como OLE DB se encuentran englobados en una estrategia de Microsoft denominada UDA (Universal Data Access). UDA representa la posibilidad de acceder a diferentes tipos de datos utilizando la misma técnica.

Esta no es una idea nueva, ya que hace mucho tiempo que ya existe ODBC (Open Database Connectivity). ODBC se diseñó para ofrecer una única forma para acceder a bases de datos relacionales, y se convirtió en el estándar de acceso a datos.

Pero UDA va más allá, ahora se pretende acceder a datos que no sean únicamente relacionales, ya que cada vez se incrementa la cantidad de información que se encuentra almacenada en documentos, sistemas financieros, sistemas de correo, etc., a través de ADO 2.5 se nos permite el acceso a datos de diferentes formas.

Por lo tanto con ADO 2.5 podremos acceder a distintos tipos de datos, no sólo a datos relacionales, utilizando el mismo conjunto de objetos sencillos.

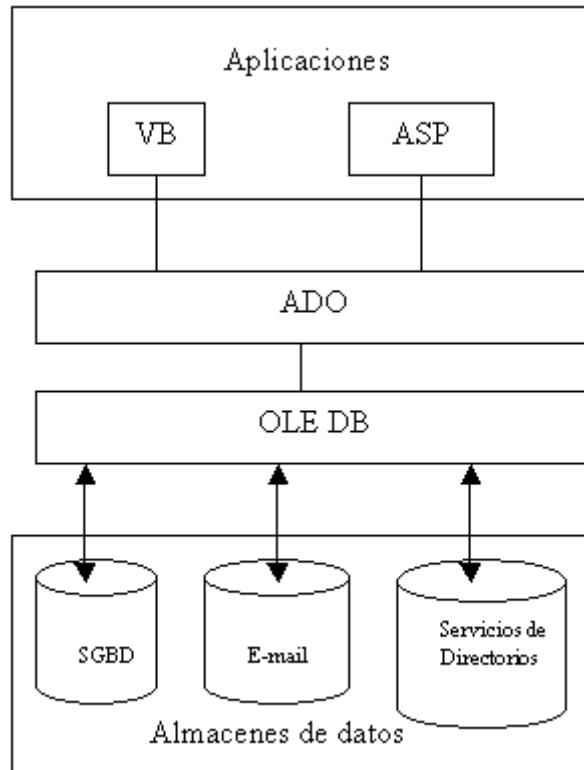


Figura 112. Acceso a distintos tipos de datos

Si vemos la figura anterior se puede comprobar que con OLE DB, que es el corazón de UDA, podemos acceder a todos los diferentes tipos de datos. La única cosa que nos limita el acceso a unos datos es la disponibilidad del proveedor OLE DB correspondiente.

## Instalación de Sql Server 7

Al igual que hicimos en los primeros capítulos, dónde se comentó la configuración básica del servidor que va a contener las aplicaciones ASP, en este capítulo se ha creído conveniente dedicar un apartado a la instalación y configuración del servidor de datos con al que se va acceder a través de ADO desde las páginas ASP.

Se podría haber elegido otro producto como sistema gestor de bases de datos (SGBD), para aplicaciones Web los más recomendables con SQL Server y Oracle. Personalmente yo he utilizado SQL Server 7 como SGBD, además el Campus Virtual Almagesto utiliza SQL Server, de todas formas todo lo referente a ADO que vamos a ver en el curso se puede aplicar a cualquier SGBD, no se debe olvidar que ADO ofrece un acceso a datos independiente del sistema gestor de bases de datos y del tipo de almacenamiento.

SQL Server 7 dispone de las denominadas *versiones* o *ediciones* de instalación, que consisten en la posibilidad de realizar una instalación adecuada a las características del sistema operativo en el que deba de ejecutarse el servidor de datos. A continuación se muestra una breve descripción de cada una de estas ediciones.

### Standard

Es la edición destinada a ejecutarse bajo Windows NT Server 4.0 o Windows 2000 Server, dispone de características de multiproceso simétrico, aprovechando la instalación en el equipo de hasta cuatro procesadores; permite la ejecución de consultas en paralelo; búsquedas avanzadas mediante Microsoft Search, etc.

Debido a que el sistema operativo utilizado en este curso ha sido Windows 2000 Server, esta edición standard del motor de datos es sobre la que se han realizado todas las pruebas.

### Enterprise

Dispone de todas las características de la anterior pero adaptada a entornos corporativos en los que se vaya a trabajar con un gran número de datos y soporten una elevada carga de transacciones. Debe instalarse en un sistema Windows NT Server Enterprise Edition 4.0 o Windows 2000 Advanced Server. Permite el uso de hasta treinta y dos procesadores en multiproceso y soporta varios servidores funcionando en clustering, técnica que permite utilizar varios servidores físicos (clusters), simulando un único servidor lógico y permitiendo el balanceo de cargas, de forma que se optimiza al máximo el rendimiento de las aplicaciones en ejecución.

### Desktop o SBS (Small Business Server)

Esta es la edición más limitada en funcionalidad, diseñada para ejecutarse bajo Windows 95-98 y para usuarios que habitualmente trabajan con aplicaciones que acceden a un servidor SQL Server Standard o Enterprise, pero que por diversos motivos deben manipular en ocasiones los datos desde el exterior, desde un portátil por ejemplo. De esta forma, pueden trabajar con sus aplicaciones habituales y realizar posteriormente la actualización de los datos en el servidor central de la empresa.

Como ya hemos dicho, en nuestro caso se ha utilizado la versión Standard en Windows 2000 Server. Una vez cumplidos los requerimientos previos a la instalación en cuanto a sistema operativo y máquina se refieren, procederemos a insertar en nuestro equipo el CD de instalación de SQL Server 7

y ejecutar el fichero AUTORUN.EXE, que nos situará en la primera pantalla del programa de instalación.



Figura 113

Antes de proceder a la instalación de los componentes de SQL Server 7 debemos asegurarnos que están instalados todos los requisitos que este gestor de datos necesita, pues de lo contrario, no podremos continuar con el programa de instalación.

Al hacer clic sobre la opción Instalar los requisitos de SQL Server 7.0 en la pantalla de instalación, podremos seleccionar el sistema operativo sobre el que vamos a instalar, del que se elegirán los requisitos correspondientes.

Respecto a Windows NT 4.0, debemos instalar en primer lugar el Service Pack 4 o superior de este sistema operativo y después Internet Explorer 4.01 Service Pack 1 o superior, en cuanto a este último podemos realizar una instalación mínima o estándar, dependiendo de los requerimientos de conectividad necesarios para SQL Server. SI tenemos Windows 2000 Server, este sistema operativo ofrece todos los requisitos, que es nuestro caso, ya que ASP 3.0 sólo funciona en servidores Web IIS 5 y sólo podemos instalar el servidor IIS 5 en un sistema operativo Windows 2000.

Para los curiosos comentaré que Almagesto se ejecuta sobre un servidor Windows 2000 Advanced Server con SQL Server 7 edición Standard. A continuación debemos elegir qué edición del gestor de datos queremos instalar, los servicios de análisis de información (OLAP) o la utilidad *English Query*, para generar consultas en lenguaje natural inglés. Seleccionaremos instalar el servidor de base de datos en su edición Standard.



Figura 114

El programa de instalación nos pregunta ahora si vamos a realizar este proceso en el equipo local o vamos a instalar en otro equipo de forma remota.



Figura 115

Después de aceptar los términos de la licencia del producto, rellenaremos una pantalla con nuestros datos e introduciremos la clave de instalación del CD.

Seguiremos con el tipo de instalación a realizar: típica, mínima o personalizada. La instalación *Típica* es la más sencilla, ya que selecciona los valores más comunes de los componentes a instalar, y será la que elegiremos en la mayoría de las ocasiones, será esta opción la que vamos a seleccionar.

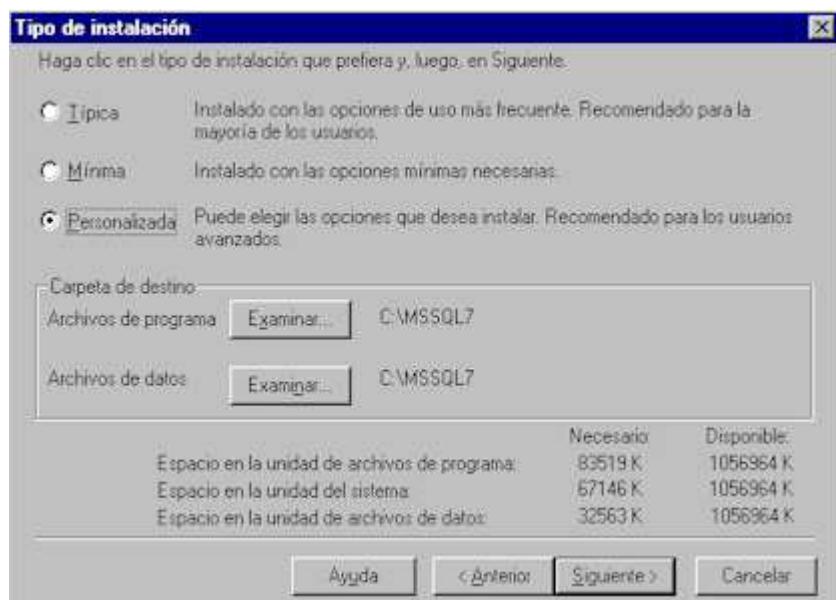


Figura 116

En este paso también podemos indicar el disco y ruta en el cual realizar la instalación, tanto para los archivos que componen el servidor como para los ficheros de bases de datos que vayamos creando. Completada esta pantalla, pulsaremos *Siguiente* para continuar con el proceso de instalación.

Seguidamente debemos especificar la cuenta de usuario definida en Windows, que servirá para iniciar los servicios de SQL Server.

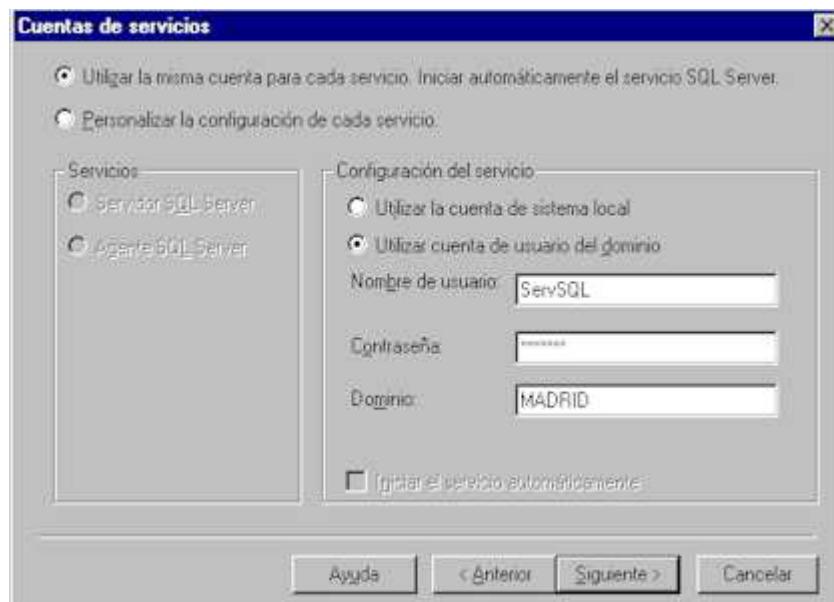


Figura 117

Como podemos ver en la Figura 117, podemos especificar cuentas separadas para los diferentes servicios, emplear una cuenta de sistema local o de usuario de dominio, que es la opción recomendada para poder efectuar operaciones a través de la red.

En el siguiente paso de la instalación debemos indicar el tipo de licencia bajo el que se ejecutará SQL Server 7.0.

- Por servidor. En este tipo de licencia, las licencias de clientes se asignan a un servidor, de forma que debemos especificar cuantas licencias de cliente se disponen para el mencionado servidor.
- Por Puesto. Esta modalidad requiere una licencia por cada equipo que vaya a realizar accesos a un servidor Windows, con la ventaja de poder acceder a cualquier servidor que haya instalado en la red.

Completado este paso, el programa nos avisa de que dispone de toda la información necesaria, tras lo que aceptamos el mensaje y comienza el proceso de instalación de SQL Server a nuestro equipo, que se realiza en los siguientes pasos:

- Copia de ficheros.
- Instalación de Microsoft Data Access Components (MDAC).
- Instalación de Microsoft Management Console (MMC).

- Instalación del coordinador de transacciones distribuidas (MDTC).
- Configuración del motor de datos.
- Comprobación de los servicios SQL Server.
- Registro de componentes ActiveX.

Los anteriores pasos son realizados automáticamente por el programa de instalación, sin que sea necesario que el usuario intervenga. Una vez terminados y si no ha sido mostrado ningún mensaje de error, la instalación de SQL Server 7.0 habrá finalizado satisfactoriamente.

# Acceso a datos con ADO: Connection

---

## Introducción

A lo largo de los siguientes capítulos vamos a ir comentando de forma detallada los principales objetos que nos ofrece ADO para el acceso a datos.

Este primer capítulo va estar dedicado al objeto Connection, luego se sucederán otros capítulos, uno dedicado al objeto Command, otro al objeto Recordset y el último dedicado a los dos nuevos objetos ofrecidos por ADO 2.5, es decir, los objetos Record y Stream.

## El objeto Connection

Un objeto Connection representa una conexión a una base de datos. Los objetos Connection se pueden crear de forma independiente sin tener en cuenta ningún objeto definido previamente. Este objeto, al igual que los objetos Command y Recordset, es creado mediante el método CreateObject del objeto Server.

Las actividades que se pueden realizar con un objeto Connection son: creación de conexiones con bases de datos, obtención de objetos Recordset sencillos y tratamiento de errores.

A continuación se van a enumerar y comentar de forma breve las diferentes propiedades, métodos y colecciones que posee este objeto. Después se verán con más detalle las propiedades, métodos y colecciones más importantes.

Lo que se ofrece a continuación no es una descripción detallada del objeto Connection, sino que es una referencia rápida del mismo.

Las propiedades del objeto Connection son:

- Attributes: indica distintas características de la conexión por ejemplo de que forma se puede iniciar una nueva transacción.
- ConnectionTimeout: tiempo de espera máximo para la realización de una conexión antes de que se produzca un error.
- ConnectionString: cadena de conexión que identifica la base de datos a la que nos queremos conectar.
- CommandTimeout: tiempo máximo de espera en la ejecución de un comando SQL que se ejecuta de forma directa sobre la conexión.
- CursorLocation: indica la localización de los cursores que se van a crear a partir de la conexión. pueden situarse en el cliente o en el servidor.
- DefaultDatabase: indica la base de datos por defecto con la que realizar la conexión.
- IsolationLevel: propiedad de sólo lectura que indica el nivel de aislamiento de la conexión.
- Mode: modo de acceso a la conexión.
- Provider: proveedor OLE DB utilizado para establecer la conexión.
- Version: propiedad de sólo lectura, indica la versión de ADO.
- State: propiedad de sólo lectura que indica el estado de la conexión, es decir, si se encuentra abierta o cerrada.

Los métodos del objeto Connection no son tan numerosos y son los siguientes:

- BeginTrans: indica el inicio de un transacción.
- CommitTrans: almacena todos los cambios y finaliza la transacción actual, representa el commit de una transacción.
- RollbackTrans: cancela todos los cambios realizados durante la transacción actual y finaliza la transacción, representa el rollback de una transacción.
- Close: cierra la conexión.
- Execute: ejecuta un comando SQL.
- Open: abre la conexión.
- Cancel: cancela la ejecución de una operación asíncrona de ejecución de una sentencia SQL o de una apertura de una conexión.
- OpenSchema: obtiene un objeto Recordset con el esquema de la base de datos.

El objeto Connection posee dos colecciones:

- Errors: colección de errores generados a través del uso de ADO.
- Properties: una colección de objetos Property referentes al objeto Connection.

La función principal de un objeto Connection es la de especificar el origen de los datos, aunque también se puede utilizar a veces para ejecutar sentencias SQL sencillas.

A lo largo de este tema se irán comentando en profundidad las distintas propiedades y métodos del objeto Connection, así como también sus propiedades.

## Realización de una conexión

Una vez creado un objeto Connection mediante la siguiente línea, podemos utilizarlo para establecer conexiones con un proveedor de datos. Decimos proveedor de datos porque la conexión no tiene que ser necesariamente a una base de datos, como vimos en el capítulo anterior ADO forma parte de la estrategia universal de acceso a datos UDA, que permite acceder a diferentes tipos de datos.

Para realizar la conexión utilizaremos el método Open del objeto Connection, al que le podemos pasar como parámetro la cadena de conexión de la base de datos (o de forma más general, del proveedor de datos u origen de los datos). También es posible especificar la información de la conexión que se va a realizar a través de la propiedad ConnectionString.

La sintaxis general del método Open es la siguiente:

```
objConexion.Open [CadenaConexion], [IDusuario], [contraseña],  
[opciones]
```

Dónde CadenaConexion es la cadena de conexión del origen de los datos, IDusuario es el nombre del usuario que va a realizar la conexión, contraseña es la contraseña de ese usuario y opciones permite indicar información adicional acerca de la conexión. Vamos comentar estos parámetros de forma detallada.

Como se puede observar en la sintaxis del método Open todos los parámetros del mismo son opcionales, ya que esta misma información puede ser facilitado por la propiedad ConnectionString. Esta situación la veremos en muchos casos al trabajar con ADO, es decir, los parámetros de los métodos se corresponden con propiedades del objeto sobre el que se lanzan.

El parámetro CadenaConexion puede contener los siguientes parámetros para la conexión:

- Provider: proveedor OLE DB que se va a utilizar.
- Data Source: el nombre del servidor dónde se encuentra la base de datos o del fichero fuente de datos.
- Initial Catalog: el nombre de la base de datos.
- User id: el nombre del usuario que va a establecer la conexión.
- Password: contraseña del usuario.
- File Name: nombre del fichero que contiene la información del proveedor de la conexión.

- URL: la URL absoluta que identifica la carpeta o fichero que se va a utilizar como origen de datos.
- DSN: nombre de la fuente de datos de ODBC que se va a utilizar.

Estos parámetros son exactos a los que puede contener la propiedad ConnectionString del objeto Connection. Además se puede observar que el usuario y contraseña pueden estar contenidos en esta cadena de conexión.

En el parámetro opciones se puede indicar una constante de ADO que aporta información adicional de la conexión, esta información indica si la conexión va a ser síncrona o asíncrona, el valor de este parámetro es adAsyncConnect, que indicará que la conexión es asíncrona, sino se indica nada la conexión será síncrona. En el entorno de ASP no tiene sentido declarar una conexión como asíncrona, ya que ASP no puede recibir los eventos que genera ADO, por lo tanto en nuestras páginas ASP el método Open nunca llevará el parámetro opciones.

Para utilizar las constantes de ADO se debe incluir el fichero de constantes correspondiente.

El fichero de constantes de ADO se llamada ADOVBS.INC y se encuentra en la ruta c:\Archivos de programa\archivos comunes\system\ado. Para incluir el fichero de constantes en nuestras páginas podemos copiarlo a nuestro sitio Web y realizar una referencia al mismo mediante la directiva INCLUDE.

```
<!--#INCLUDE FILE="ADOVBS.INC"-->
```

Código fuente 223

Esta sentencia la deberemos repetir en todas las páginas en las que queramos utilizar las constantes de ADO. Una alternativa más sencilla es hacer una referencia a la librería de tipos de ADO mediante la etiqueta METADATA dentro del fichero GLOBAL.ASA, de esta forma las constantes se encontrarán accesibles desde cualquier página ASP de la aplicación.

```
<!-- METADATA TYPE="typelib" FILE="c:\Archivos de programa\archivos comunes\system\ado\msado15.dll"-->
```

Código fuente 224

Aunque el nombre de la librería de tipos termine en 15, pertenece a la versión 2.5 de ADO.

En los distintos ejemplos del curso se supone que hemos realizado la referencia de las constantes de ADO en el fichero GLOBAL.ASA de nuestra aplicación, y por lo tanto no es necesario realizar las inclusiones del fichero de constantes en las distintas página ASP.

A continuación se va a mostrar una serie de ejemplos que realizan conexiones de distintas formas.

En este primer ejemplo se realiza una conexión con el proveedor OLE DB de SQL Server. Además de realizar la conexión se muestra el nombre del proveedor utilizado a través de la colección Properties y la versión de ADO que se ha utilizado para establecer la conexión. El código es el que muestra el Código fuente 225.

```
<%Set Conex=Server.CreateObject("ADODB.Connection")
Conex.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"%>
<b>Versión de ADO: </b> <i><%=Conex.Version%></i><br>
<b>Proveedor: </b><i><%=Conex.Properties("Provider Friendly Name")%>
(<%=Conex.Properties("Provider Name")%>)</i><br>
```

Código fuente 225

Mediante esta cadena de conexión lo que se consigue es establecer una conexión con un servidor llamado aesteban2 que tiene instalado SQL Server, el usuario conectado es sa y se conecta a la base de datos pubs.

El resultado que aparece en el navegador es:

**Versión de ADO:** 2.5  
**Proveedor:** Proveedor de Microsoft OLE DB para SQL Server  
 (sqloledb.dll)

Como se puede comprobar se muestra el nombre completo del proveedor (o nombre "amigable") y la librería en la que se encuentra. Si no indicamos ningún proveedor OLE DB a la hora de establecer la conexión por defecto se utiliza el proveedor de ODBC.

La conexión del ejemplo anterior la podríamos haber realizado de esta otra forma como indica el Código fuente 226.

```
Set Conex=Server.CreateObject("ADODB.Connection")
Conex.ConnectionString="Provider=SQLOLEDB;Data Source=aesteban2;" &_
"Initial Catalog=pubs;User id=sa"
Conex.Open
```

Código fuente 226

Y otra forma más, aparece en el Código fuente 227.

```
Conex.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs", "sa"
```

Código fuente 227

También podemos hacer uso de otras propiedades del objeto Connection, como se puede ver en el Código fuente 228, que no es nada más que otra versión distinta de nuestro ejemplo.

```
Conex.Provider="SQLOLEDB"
Conex.ConnectionString="Data Source=aesteban2;User id=sa"
Conex.Open
Conex.DefaultDatabase="pubs"
```

Código fuente 228

La propiedad DefaultDatabase, que indica la base de datos por defecto a la que nos conectamos, se puede utilizar una vez que ya está establecida la conexión. Como se puede comprobar, a la vista de las numerosas versiones de este ejemplo, el establecimiento de una conexión con ADO es muy flexible.

Ahora se va a realizar una conexión utilizando el proveedor OLE DB de ODBC, como ya hemos dicho este es el proveedor por defecto que se utiliza para el objeto Connection. Se va actuar de la misma forma que para la conexión anterior, es decir, se van a mostrar distintas formas de establecer una misma conexión.

Si suponemos que tenemos una fuente de datos ODBC (DSN de ODBC) de sistema definida en el servidor Web, podemos emplear el Código fuente 229 para establecer la conexión.

```
<%Set Conex=Server.CreateObject("ADODB.Connection")
Conex.ConnectionString="DSN=FuenteCursos;User id=cursos;Password=xxx"
Conex.Open%>
<b>Versión de ADO: </b> <i><=%=Conex.Version%></i><br>
<b>Proveedor: </b><i><=%=Conex.Properties("Provider Friendly Name")%>
(<=%=Conex.Properties("Provider Name")%>)</i><br>
```

Código fuente 229

En esta caso nos conectamos a una DSN de ODBC de sistema llamada FuenteCursos con el usuario cursos que tiene la contraseña xxx. El resultado que se obtiene ahora es el siguiente:

**Versión de ADO:** 2.5

**Proveedor:** Microsoft OLE DB Provider for ODBC Drivers (MSDASQL.DLL)

También disponemos de diversas formas de establecer una misma conexión, vamos a ver unas cuantas, en el Código fuente 230.

```
Set Conex=Server.CreateObject("ADODB.Connection")
Conex.Open "DSN=FuenteCursos;User id=cursos;Password=xxx"
Conex.Open "DSN=FuenteCursos;UID=cursos;PWD=xxx"
Conex.Open "DSN=FuenteCursos", "cursos", "xxx"
```

Código fuente 230

Como se puede observar podemos utilizar indistintamente el parámetro PWD o Password para indicar la contraseña, y el parámetro User id y UID para indicar el usuario.

Incluso se puede establecer la conexión a través del proveedor OLE DB de ODBC sin necesidad de crear la fuente de datos de ODBC en el servidor Web, es decir, se realiza la conexión con ODBC sin DSN. Veámoslo con un ejemplo (Código fuente 231).

```
Set Conex=Server.CreateObject("ADODB.Connection")
Conex.Open "driver=SQL Server;server=aesteban2;uid=cursos;pwd=xxx;database=Cursos"
```

Código fuente 231

En este caso se ha establecido una conexión a la base de datos Cursos que se encuentra en el servidor aesteban2 y se ha utilizado el driver ODBC para SQL Server, ya que nuestra base de datos se encuentra en un servidor SQL Server. Como se puede comprobar debemos utilizar un parámetro driver para indicar el driver ODBC que se va a utilizar para establecer la conexión.

En el Código fuente 232 se muestra como podemos conectarnos a una base de datos de Microsoft Access utilizando el proveedor OLE DB específico de Access.

```
Set Conex=Server.CreateObject("ADODB.Connection")
Conex.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data source=c:\tmp\musica.mdb"
```

Código fuente 232

En esta caso se debe indicar la ruta del fichero de la base de datos de Access, es decir, el fichero MDB. Y el nuevo resultado es:

Hasta ahora nos hemos conectado a distintas bases de datos y de distinta forma, pero como ya hemos dicho ADO nos permite acceder a cualquier tipo de datos para el que tengamos el proveedor OLE DB correspondiente, no tiene porque ser necesariamente bases de datos, puede ser por ejemplo una página Web que podemos obtener de una dirección URL.

Como ya se comentó en el capítulo anterior ADO 2.5 ofrece un proveedor OLE DB para acceder a servidores Web. Para establecer una conexión de esta forma deberemos escribir lo que muestra el Código fuente 233.

```
<%Set Conex=Server.CreateObject("ADODB.Connection")
Conex.Open "Provider=MSDAIPP.DSO;Data Source=http://aesteban2"%>
<b>Versión de ADO: </b> <i><=%=Conex.Version%></i><br>
<b>Proveedor: </b><i><=%=Conex.Properties("Provider Name")%>

```

Código fuente 233

En el parámetro Data Source se debe indicar la URL a la que nos queremos conectar.

En este caso se utilizan otros valores de la colección Properties, ya que este proveedor OLE DB presenta unas propiedades distintas a los anteriores, y el resultado es:

Versión de ADO: 2.5

Proveedor: Microsoft OLE DB Provider for Internet Publishing

## Configuración de la conexión

A veces, antes de abrir la conexión, podemos necesitar configurarla a través de las propiedades ConnectionString, ConnectionTimeout, CommandTimeout y Mode. La propiedad ConnectionString es otra forma de expresar la cadena de conexión para como ya hemos visto en los ejemplos anteriores.

Mediante la propiedad ConnectionTimeout indicamos en segundos el tiempo máximo de espera para la realización de una conexión, su valor por defecto es de 15 segundos. En el caso de tener una conexión a la red no muy rápida deberemos aumentar el valor de esta propiedad, de todas formas, si

nuestra aplicación ASP se va a encontrar en el entorno de Internet y no dentro de Intranet, se deberá aumentar bastante el valor de la propiedad ConnectionTimeout para no tener problemas con ningún usuario.

La propiedad CommandTimeout indica el tiempo de espera de la ejecución de un comando SQL que se ejecuta sobre la conexión mediante el método Execute del objeto Connection, el valor por defecto de esta propiedad es de 30 segundos.

Otra propiedad que podemos utilizar antes de abrir la conexión es la propiedad Mode, que indica los permisos de acceso para la conexión. Los valores que puede tener esta propiedad se identifican con una serie de constantes de ADO (ya hemos visto como incluir el fichero de constantes ADOVBS.INC), en la Tabla 19 se detallan los valores que puede tener la propiedad Mode junto con el significado de cada uno de ellos.

Constante	Valor	Descripción
AdModeUnknown	-1	Los permisos no han sido establecidos o no pueden ser determinados. Es el valor por defecto.
AdModeRead	1	Permiso sólo de lectura. Modificaciones y altas darán lugar a errores.
AdModeWrite	2	Permiso sólo de escritura.
AdModeReadWrite	3	Permisos de lectura y escritura.
adModeShareDenyRead	4	Evita que otros abran la conexión con permisos de lectura. De esta forma los cambios realizados no serán visibles por otros hasta que no se haya cerrado la conexión.
adModeShareDenyWrite	8	Evita que otros abran la conexión con permisos de escritura. Esta es la forma de asegurar que nadie está cambiando la información a la que estamos accediendo.
adModeShareExclusive	12	Evita que otros puedan abrir la conexión con permisos de lectura/escritura. Esto nos asegurará que nadie interferirá o verá nuestras lecturas o escrituras hasta que no cerremos la conexión.
adModeShareDenyNone	16	Evita que otros puedan abrir una conexión con cualquier tipo de permisos.

Tabla 19

De esta forma, si queremos establecer una conexión de sólo lectura y que tenga un tiempo de espera de 2 minutos deberemos escribir lo que muestra el Código fuente 234.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Mode=adModeRead
```

```
objConexion.ConnectionTimeout=120  
objConexion.ConnectionString="DSN=FuenteDatos;UID=pepe;PWD=xxx"  
objConexion.Open%>
```

Código fuente 234

Otra de las propiedades de Connection es State. Podremos consultar el valor de la propiedad State del objeto Connection para comprobar el estado de la misma, es decir, para comprobar si la conexión está abierta o cerrada. Los valores de State pueden ser adStateOpen (conexión abierta) o adStateClosed (conexión cerrada).

Antes de abrir la conexión podemos indicar la localización del cursor que se va a utilizar con los datos. Vamos antes a definir de forma sencilla en qué consiste un cursor: un cursor se ocupa de gestionar un conjunto de registros y la posición actual dentro de dicho conjunto, podemos decir que un Recordset y un cursor es lo mismo.

El cursor que se crea a partir de un objeto Connection puede ser un cursor localizado en el cliente o en el servidor, esto lo indicamos mediante la propiedad CursorLocation, que puede tener dos valores: adUseServer, con lo que indicamos que el cursor va a ser creado en el servidor y va ser el almacén de datos quién va a gestionar y manipular el cursor; el otro valor de CursorLocation es adUseClient, con esta constante se indica que el cursor va a ser manipulado por ADO de forma local. El valor por defecto de esta propiedad es adUseServer.

Si creamos un cursor de servidor va a ser el almacén de datos (SQL Server 7 en nuestro caso) quién va a manipular los registros, es decir, es el que se encarga de controlar el movimiento, la actualización de registros, etc.

Sin embargo, si creamos un cursor de cliente, el contenido completo del Recordset se copia al cliente y será gestionado por el servicio de cursores local. Si el conjunto de registros es muy numeroso será más lento utilizar un cursor de cliente que de servidor.

## Ejecutando comandos sobre la conexión

Una vez establecida la conexión con un origen de datos determinados, ya podemos ejecutar consultas o sentencias sobre la conexión. Aunque ya hemos comentado en el tema anterior y en este mismo, que ADO permite manipular datos de cualquier tipo, vamos a centrarnos en el acceso a datos a un sistema gestor de bases de datos relacional, en mi caso he utilizado SQL Server 7, pero el lector puede utilizar el que estime conveniente.

Esta no es la labor principal del objeto Connection, aunque se suelen ejecutar sentencias SQL sencillas directamente sobre la conexión para una mayor simplicidad del código o por comodidad del programador.

Una vez establecida la conexión, podremos ejecutar sobre ella comandos SQL (sentencias SQL, procedimientos almacenados, nombre de una tabla...), para ello utilizaremos el método Execute, que posee la siguiente sintaxis general:

```
Set resultado = conexion.Execute(CommandText [, RegistroAfectedos,  
Opciones])  
connection.Execute CommandText [, RegistroAfectedos, Opciones]
```

Como se puede observar existen dos formas distintas de utilizar el método Execute. En la primera línea se muestra de que forma se llamaría al método Execute cuando necesitamos almacenar su resultado en un objeto Recordset y en la segunda línea cuando no es necesario almacenar su resultado.

El parámetro CommandText es una cadena que representa el comando SQL que se debe ejecutar, es decir, esta cadena contendrá: la sentencia SQL o procedimiento almacenado a ejecutar o el nombre de la tabla que se quiere recuperar.

En el parámetro opcional RegistroAfectados se devolverá el número de registros a los que ha afectado la operación.

En el parámetro opcional Opciones se puede indicar mediante una constante de ADO el tipo de comando que se encuentra en el parámetro CommandText, es recomendable utilizar este parámetro para optimizar la evaluación y ejecución del comando correspondiente. Los valores que puede tomar este parámetro aparecen en la Tabla 20.

Constante	Descripción
adCmdText	Evalúa el comando como una sentencia SQL.
adCmdTable	Evalúa el comando como el nombre de una tabla.
adCmdStoredProc	Evalúa el comando como un procedimiento almacenado.
adCmdUnkown	El tipo del comando SQL es desconocido.
adCmdTableDirect	Se evalúa como el nombre de una tabla
adCmdFile	Se trata como un fichero que contiene un objeto Recordset que se había almacenado.
adExecuteNoRecords	Se utiliza cuando los comandos no devuelven registros

Tabla 20

Por lo tanto, a través del método Execute podremos ejecutar diferentes tipo de comandos SQL, a continuación se comentan mediante ejemplos algunos de los usos del método Execute.

En el Código fuente 235 se muestra como se almacenaría el contenido de una tabla en un objeto Recordset, indicando de forma explícita que el comando SQL es el nombre de una tabla.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
Set objRecordset=objConexion.Execute("provincias",,adCmdTable)%>
```

Código fuente 235

Un resultado equivalente lo obtendríamos mediante la ejecución de una sentencia SQL, que muestra el Código fuente 236.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
Set objRecordset=objConexion.Execute("Select * from provincias",,adCmdText)%>
```

Código fuente 236

Para ejecutar un procedimiento almacenado llamado borraDatos.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
objConexion.Execute "borraDatos",,adCmdStoredProc%>
```

Código fuente 237

En el Código fuente 238 se borran todas las provincias cuyo código empiece por '8', el número de provincias borradas se recoge en el parámetro registrosBorrados:

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
objConexion.Execute "DELETE FROM Provincias WHERE IDProvin LIKE '8%"_>
,,registrosBorrados,adCmdText
Response.write("Provincias borradas:"&registrosBorrados) %>
```

Código fuente 238

En este apartado también se va a comentar otro método más del objeto Connection, se trata del método OpenSchema. Este método no tiene mucho que ver con este apartado. El método OpenSchema nos ofrece la posibilidad de obtener información del esquema de nuestro proveedor de datos.

OpenSchema recibe un argumento, en forma de constante de ADO, que indica el tipo de información que queremos obtener del origen de los datos, por ejemplo podemos obtener el nombre de las tablas de la base de datos a la que nos hemos conectado, para lo que utilizaríamos la constante adSchemaTables, este ejemplo se muestra en el Código fuente 239.

```
<%Option Explicit%>
<html>
<body>
<%Dim Conex,rsTablas
Set Conex=Server.CreateObject("ADODB.Connection")
Conex.ConnectionString="Provider=SQLOLEDB;Data Source=aesteban2;" &_
"Initial Catalog=pubs;User id=sa"
Conex.Open
Set rsTablas=Conex.OpenSchema(adSchemaTables)
While Not rsTablas.EOF
    Response.Write rsTablas("TABLE_NAME") &"<br>"
    rsTablas.MoveNext
Wend
rsTablas.Close
Set rsTablas=Nothing
Conex.Close
```

```
Set Conex=Nothing%>
</body>
</html>
```

Código fuente 239

Como se puede comprobar a la vista del ejemplo anterior, el método OpenSchema devuelve la información relativa al origen de los datos en un objeto Recordset, este objeto de ADO lo veremos detalladamente en el capítulo correspondiente.

## Transacciones

En este apartado se trata otra de las funciones que puede desempeñar el objeto Connection, sin ser ésta tampoco su función principal, que como ya comentamos es la de establecer la conexión con el origen o fuente de los datos.

El objeto Connection ofrece tres métodos para manejar transacciones. Una transacción es un grupo de sentencias SQL que siempre se deben ejecutar como un conjunto, si falla alguna se cancelarán los cambios u operaciones que hayan realizado el resto de las sentencias que formen parte de la transacción.

Por ejemplo, para transferir una cantidad de dinero entre dos cuentas, se resta esa cantidad de la cuenta de origen y se le suma a la cuenta de destino. Si falla alguna de estas modificaciones, el balance de cuentas será erróneo. Pero si se tratan como una transacción se nos podrá asegurar que se producirán ambas actualizaciones o ninguna.

Las transacciones, además de para ejecutar varias sentencias como un todo, también se utilizan cuando es necesario que las sentencias se procesen con una mayor rapidez. Las transacciones son más rápidas debido a que utilizan memoria RAM para ejecutarse en lugar de acceder al disco duro para almacenar los cambios. Sin embargo, las transacciones también implican un riesgo, ya que un error en una sentencia de la transacción implicará que todas las operaciones realizadas se han perdido.

Los métodos BeginTrans, CommitTrans y RollbackTrans se usan con un objeto Connection cuando se quieren almacenar o cancelar una serie de cambios realizados sobre la fuente de datos como un todo. La sintaxis general para realizar una transacción es la siguiente:

```
[nivel de anidamiento]=conexion.BeginTrans
sentencias que forman parte de la transacción
.....
.....
conexion.{CommitTrans | RollbackTrans}
```

Al lanzar el método BeginTrans, el proveedor de datos no almacenará instantáneamente ningún cambio realizado durante la transacción, es decir, indicará el inicio de la transacción. En el caso de que el proveedor soporte transacciones anidadas, al llamar al método BeginTrans dentro de una transacción existente, creará una nueva transacción anidada. El valor de retorno del método BeginTrans indica el nivel de anidamiento de la transacción.

Una llamada al método CommitTrans almacena todos los cambios realizados dentro de la transacción y finaliza la transacción. Al lanzar el método RollbackTrans se cancelan todos los cambios realizados dentro de la transacción y finaliza la transacción. Si se llama a cualquiera de estos dos métodos y no hay una transacción abierta, se producirá un error, además, se debe indicar, que la llamada a

CommitTrans o RollbackTrans afectará a la transacción abierta más reciente, esto se debe tener en cuenta a la hora de trabajar con transacciones anidadas, se deberán ir cerrando de dentro hacia fuera.

Aunque ya advertimos desde aquí que este proceso no es demasiado usual, es decir, las transacciones se suelen tratar a otros niveles, como puede ser el propio gestor de la base de datos, estos métodos se han incluido junto con la explicación del objeto Connection para tener una visión más extensa.

Dependiendo de los valores de la propiedad Attributes del objeto Connection, la llamada al método CommitTrans o RollbackTrans podrán iniciar o no una nueva transacción de forma automática. La propiedad Attributes es una suma de dos valores: adXactCommitRetaining y adXactAbortRetaining. Si Attributes posee el primero de ellos, al lanzar el método CommitTrans se iniciará automáticamente una nueva transacción, y si además, Attributes tiene el segundo valor también, al lanzar el método RollbackTrans se dará comienzo a una nueva transacción de manera automática y sin necesidad de realizar una llamada al método BeginTrans.

En el Código fuente 240 se muestra como se realizaría una transacción compuesta por dos sentencias.

```

1 <!--#INCLUDE VIRTUAL=/ADO/ADOVBS. INC-->
2 <%
3 Set objConexion=Server.CreateObject("ADODB.Connection")
4 objConexion.Mode=adModeReadWrite
5 objConexion.ConnectionTimeout=120
6 objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
7 objConexion.Errors.Clear
8 objConexion.BeginTrans
9 On Error Resume Next
10 objConexion.Execute "INSERT INTO Provincias VALUES('10','Barcelona')"
11 objConexion.Execute "INSERT INTO Provincias VALUES('20','Córdoba')"
12 On Error Goto 0
13 if objConexion.Errors.Count>0 Then
14     objConexion.RollbackTrans
15 Else
16     objConexion.CommitTrans
17 End If
18 %>
```

Código fuente 240

Como se puede observar en el código anterior, para comprobar si alguna de las sentencias que se ejecutaron dieron algún error se consulta la propiedad Count de la colección Errors, que indica el número de errores que se han producido al manipular el objeto Connection. Antes de ejecutar las sentencias, en la línea 7 se borran todos los errores, para tener en cuenta solamente los errores producidos por la ejecución de las sentencias que forman la transacción. En la línea 9 se utiliza la sentencia On Error Resume Next para que los errores que se den al ejecutar las sentencias no terminen con la ejecución del programa, en la línea 12 se vuelve al tratamiento de errores estándar.

En el caso de que no se den errores se lanzará el método CommitTrans para almacenar los cambios realizados en la transacción y si la propiedad Count de la colección Errors es mayor que cero se lanzará el método RollbackTrans para cancelar todos los cambios realizados en la transacción.

En este punto ya hemos introducido el tema sobre el que trata el siguiente apartado, las colecciones del objeto Connection.

El objeto Connection ofrece una propiedad mediante la que podemos indicar el nivel de aislamiento de la conexión, esta propiedad es IsolationLevel. La propiedad IsolationLevel es de lectura y escritura,

los cambios realizados en el valor de esta propiedad tendrán efecto después de llamar al método BeginTrans. Por lo tanto el valor de esta propiedad afectará a las transacciones, por defecto las transacciones podrán ver los cambios realizados por otras transacciones una vez que se hayan almacenado, es decir, se ha llamado al método CommitTrans.

Si el nivel de aislamiento solicitado no está disponible, el proveedor podrá ofrecer el siguiente nivel de aislamiento. Los niveles de aislamiento se indican a través de constantes de ADO, y son los que muestra la , ordenados de un menor aislamiento a un mayor aislamiento:

Constante	Descripción
adXactUnspecified	El nivel de aislamiento no se ha podido determinar.
adXactChaos	Se pueden ver los cambios realizados por otras transacciones antes de que sean almacenados, pero no se puede modificar nada que pertenezca a transacciones de un nivel mayor de aislamiento.
adXactBrowse, adXactReadUncommitted	Desde una transacción se pueden ver los cambios que todavía no han almacenado otras transacciones.
adXactCursorStability, adXactReadCommitted	Desde una transacción se pueden ver cambios realizados por otra transacción, si estos cambios han sido almacenados (committed), es el nivel de aislamiento predeterminado.
adXactIsolated, adXactSerializable	Las transacciones se encuentran completamente aisladas entre sí.

Tabla 21

De esta forma, si tenemos en cuenta nuestro ejemplo anterior y queremos que nuestra transacción tenga el mayor nivel de aislamiento, antes de iniciar la transacción deberemos escribir lo que muestra el Código fuente 241.

```
objConexion.IsolationLevel=adXactIsolated
```

Código fuente 241

## Colecciones del objeto Connection

Como ya comentábamos al comienzo de este capítulo, el objeto Connection posee dos colecciones: Errors y Properties. Vamos a abordar ahora la primera de ellas.

La colección Errors contiene los errores que ha facilitado el origen de datos a través del proveedor correspondiente.

Si queremos tratar los errores producidos y mostrarlos al usuario deberemos recorrer la colección Errors. Cada vez que se da un error o más relacionado con una operación realizada con un objeto

Connection se crean nuevos objetos Error dentro de la colección Errors, estos errores serán generados por el proveedor de datos, es decir, no son errores de ADO sino los errores generados por la fuente de datos.

Cuando otra operación sobre el objeto Connection genere un error, la colección Errors se vaciará y se añadirá el nuevo error o conjunto de errores, creándose para ello objetos Error. Para acceder a un objeto Error dentro de la colección Errors podemos hacerlo de dos maneras diferentes.

```
conexion.Errors.Item(indice)
conexion.Errors(indice)
```

Código fuente 242

El objeto Error posee una serie de propiedades que permiten obtener información acerca del error que se ha producido, esta información consta de: descripción del error (Description), número del error (Number), fuente que ha producido el error (Source), ayuda asociada (HelpFile y HelpContext) e información sobre el origen de los datos (SQL State y NativeError). En el Código fuente 243 se muestra al usuario los errores que se han dado al realizar la transacción.

```
1 <!--#INCLUDE VIRTUAL=/ADO/ADOVBS. INC-->
2 <%
3 Set objConexion=Server.CreateObject("ADODB.Connection")
4 objConexion.Mode=adModeReadWrite
5 objConexion.ConnectionTimeout=120
6 objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
7 objConexion.Errors.Clear
8 objConexion.BeginTrans
9 On Error Resume Next
10 objConexion.Execute "INSERT INTO Provincias VALUES('10','Barcelona')"
11 objConexion.Execute "INSERT INTO Provincias VALUES('20','Córdoba')"
12 On Error Goto 0
13 if objConexion.Errors.Count>0 Then
14     Response.write("Se han producido los siguientes errores:<br>")
15     For i=0 to objConexion.Errors.Count-1
16         Response.write("Descripción:
"&objConexion.Errors(i).Description"<br>")
17         Response.write("Número de Error:
"&objConexion.Errors(i).Number"<br>")
18         Response.write("Fuente del Error:
"&objConexion.Errors(i).Source"<br>")
19         Response.write("Estado SQL:
"&objConexion.Errors(i).SQLState"<br>")
20         Response.write("Error Nativo:
"&objConexion.Errors(i).NativeError"<br>)
21         Response.write("-----<br>")
22     Next
23     objConexion.RollbackTrans
24 Else
25     objConexion.CommitTrans
26 End If
27 %>
```

Código fuente 243

La segunda colección del objeto Connection es la colección Properties, aunque esta colección no aparece únicamente en este objeto, sino que es la poseen los siguientes objetos del modelo de objetos de ADO: Command, Recordset, Field y Record.

Esta colección existe debido a la característica especial que posee ADO para conectarse a distintos orígenes de datos, cada uno de estos orígenes de datos tendrá una serie de características particulares y éstas se almacenan en la colección Properties. De esta forma los contenidos de la colección Properties variarán según el proveedor de datos que utilicemos. El Código fuente 244 pretende mostrar en pantalla las propiedades que posee el origen de datos al que nos hemos conectado.

```
<%Option Explicit%>
<html>
<body>
<%Dim Conex,rsTablas
Set Conex=Server.CreateObject("ADODB.Connection")
Conex.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Dim propiedad
For each propiedad in Conex.Properties%>
    <%=propiedad.name%>=<i><%=propiedad.value%></i><br>
<%Next
Conex.Close
Set Conex=Nothing%>
</body>
</html>
```

Código fuente 244

En este caso son propiedades específicas del sistema gestor de bases de datos SQL Server 7.

## Cerrando la conexión

Una vez que hayamos terminado de trabajar con la conexión la deberemos cerrar mediante el método Close. La llamada a Close liberará los recursos del sistema asociados a la conexión, pero no eliminará el objeto de la memoria, para eliminarlo de la memoria se deberá asignar al objeto el valor Nothing.

Cerrar una conexión mientras que existe una transacción en progreso producirá un error. Al lanzar el método Close sobre una conexión también se cerrarán todos los objetos Recordset asociados a la conexión, y también se vaciarán las colecciones Parameters de los objetos Command asociados a la conexión y su propiedad ActiveConnection tendrá el valor Nothing, estos dos objetos los veremos en detalle en sus temas correspondientes, precisamente el siguiente tema es el dedicado al objeto Command del modelo de objetos de ADO.

# Acceso a datos con ADO: Command

---

## El objeto Command

Este es el segundo objeto que se podía crear de forma independiente a la jerarquía de objetos ADO, y además era la segunda forma de obtener un objeto Recordset. Un objeto Command permite realizar la ejecución de un procedimiento almacenado de una base de datos, por lo tanto para poder utilizar el objeto Command de esta forma, el proveedor de datos debe soportar procedimientos almacenados. A través de este objeto podremos pasar parámetros a los procedimientos almacenados.

Pero este objeto no sólo permite realizar la ejecución de procedimientos almacenados, también se puede utilizar para ejecutar sentencias SQL optimizadas. El objeto Command se va a comentar de la misma forma que el objeto Connection, primero se enumerarán todas su propiedades, métodos y colecciones y luego detallaremos los más importantes según vayamos avanzando en la explicación de este objeto de ADO.

Las propiedades que posee este objeto son las que se enumeran a continuación.

- ActiveConnection: conexión a la que se encuentra asociado el objeto Command.
- CommandText: comando que va a contener el objeto Command, puede ser una sentencia SQL, el nombre de una tabla o un procedimiento almacenado.
- CommandTimeout: tiempo máximo de espera para la finalización de la ejecución de un objeto Command. Indica cuánto tiempo se esperará mientras se ejecuta un objeto Command antes de terminar su ejecución y generar un error. Se expresa en segundos, su valor por defecto es de 30 segundos.

- *CommandType*: indica el tipo del objeto Command.
- *Prepared*: indica si se va a crear un sentencia "preparada" (prepared statement), es decir, una sentencia precompilada, a partir del objeto Command antes de la ejecución del mismo.
- *State*: propiedad de sólo lectura que indica la situación actual del comando, si se está ejecutando, si está cerrado o abierto.
- *Name*: permite identificar un objeto Command para luego ejecutarlo directamente desde el objeto Connection asociado.

Los métodos del objeto Command son:

- *CreateParameter*: mediante este método crearemos un parámetro para el comando SQL a ejecutar.
- *Execute*: ejecuta el objeto Command.
- *Cancel*: cancela la ejecución asíncrona de un comando. No es posible utilizarlo en páginas ASP ya que no está permitida la ejecución asíncrona.

A continuación las colecciones del objeto Command:

- *Parameters*: esta colección contiene objetos Parameter que son cada uno de los parámetros que va a tener el objeto Command.
- *Properties*: colección de propiedades, objetos Property. Tiene la misma función que en el objeto Connection.

En el siguiente apartado vamos a ver como crear un objeto Command y se darán más detalles de sus propiedades y métodos.

## Creación del objeto command

Como se ha dicho anteriormente, un objeto Command se puede crear de forma independiente, pero debemos indicarle sobre que conexión se va a realizar la ejecución de ese comando. Para ello utilizaremos la propiedad ActiveConnection, el valor asignado a esta propiedad puede ser un objeto Connection ya creado o una cadena de conexión, en el Código fuente 245 vemos varios ejemplos con las dos posibilidades.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection=objConexion%
<%Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection="DSN=FuenteBD;UID=pepe;PWD=xxx"%>
```

Código fuente 245

Como se puede observar indicamos que el objeto Command va a utilizar una DSN de ODBC como origen de los datos, ya sea a través de un objeto Connection ya creado y abierto o bien a través de la cadena de conexión correspondiente.

En el Código fuente 246 se ofrece la creación de un objeto Command pero que se conecta a través del proveedor OLE DB de SQL Server.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection=objConexion %>
<%Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection="Provider=SQLOLEDB;Data Source=aesteban2;" &
                            "Initial Catalog=pubs;User id=sa"
```

Código fuente 246

Lo más recomendable es utilizar en la propiedad ActiveConnection un objeto Connection, de esta forma distintos objetos Command pueden compartir una misma conexión. Si utilizamos una cadena de conexión en la propiedad ActiveConnection, cada objeto Command tendrá su propia conexión (objeto Connection), lo que supondrá una mayor carga para el servidor.

Una vez que hemos indicado al objeto Command la conexión a la base de datos, le debemos indicar el comando que deseamos que contenga a través de la propiedad CommandType, además, podemos indicar el tipo de comando que se va a ejecutar a través de la propiedad CommandType, los valores de esta propiedad son las constantes que ya vimos dentro del parámetro CommandText del método Execute del objeto Connection, de todas formas se ofrecen en la Tabla 22.

Constante	Descripción
AdCmdText	Evalúa el comando como una sentencia SQL.
AdCmdTable	Evalúa el comando como el nombre de una tabla.
adCmdStoredProc	Evalúa el comando como un procedimiento almacenado.
adCmdUnkown	El tipo del comando SQL es desconocido.
adCmdTableDirect	Se evalúa como el nombre de una tabla
AdCmdFile	Se trata como un fichero que contiene un objeto Recordset que se había almacenado.
adExecuteNoRecords	Se utiliza cuando los comandos no devuelven registros

Tabla 22

De esta forma si queremos crear un comando que ejecute una sentencia SQL INSERT deberemos escribir, lo que muestra el Código fuente 247.

```
<%Set objComando=Server.CreateObject ("ADODB.Command")
objComando.ActiveConnection="DSN=FuenteBD;UID=pepe;PWD=xxxx"
objComando.CommandType=adCmdText
objComando.CommandText="INSERT INTO Provincias VALUES ('aa','Soria')"%>
```

Código fuente 247

El tiempo máximo que vamos a esperar a que un comando finalice su ejecución se indica en segundos a través de la propiedad CommandTimeout, cuando la ejecución de un comando supera este tiempo, se abortará su ejecución y se producirá un error. El valor de CommandTimeout dependerá del tráfico de la red, si es alto tendremos que aumentar el valor de esta propiedad. Esta propiedad del objeto Command permite sobreescribir el valor de la propiedad del objeto Connection para un objeto Command determinado.

Mediante la propiedad Prepared indicaremos si el objeto Command contiene una sentencia SQL precompilada. Si esta propiedad tiene el valor True, antes de ejecutar el comando se enviará a la base de datos la sentencia para que sea compilada (sentencia preparada), esto podría ralentizar la primera ejecución del objeto Command, pero una vez que el proveedor de datos a compilado el contenido del comando, el proveedor utilizará la versión compilada del comando en las sucesivas ejecuciones, por lo tanto ganaremos en eficiencia. Es recomendable asignar a Prepared el valor True cuando tengamos que ejecutar varias veces el objeto Command con una misma sentencia SQL.

La propiedad Name permite nombrar a un objeto Command para que luego se invocado desde el objeto Connection asociado como si de un método del objeto Connection se tratara. Esto se puede comprobar en el Código fuente 248.

```
<%Dim objConexion, objComando
Set objConexion=Server.CreateObject ("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objComando=Server.CreateObject ("ADODB.Command")
objComando.Name="MiComando"
objComando.ActiveConnection=objConexion
objComando.CommandText="BorraDatos"
objComando.CommandType=adCmdStoredProc
objConexion.MiComando
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 248

En este caso se ejecuta el objeto Command cuyo nombre es MiComando a través del objeto Connection al que se encuentra asociado. Se debe advertir que la propiedad Name se debe asignar antes de asignar un valor a la propiedad ActiveConnection.

Mediante la propiedad State podemos consultar en que estado se encuentra un objeto Command determinado. Los posibles valores de esta propiedad son:

- adStateClosed: el objeto Command se encuentra cerrado, es decir, desconectado de un origen de datos.
- adStateOpen: el objeto Command está abierto, es decir, conectado a un origen de datos.

- adStateConnecting: el objeto Command se está conectando al origen de datos correspondiente.

## La colección Parameters

Como se ha comentado al comienzo de este capítulo, las funciones del objeto Command eran permitir la ejecución de procedimientos almacenados y ejecución se sentencias SQL optimizadas (precompiladas o preparadas), por lo tanto debe permitir realizar el paso de parámetros a la sentencia o procedimiento SQL que se vaya a ejecutar.

Un objeto Command almacena los parámetros que va a necesitar para su ejecución en una colección llamada Parameters, esta colección está constituida por objetos Parameter. Por lo tanto antes de ejecutar un comando deberemos crear sus parámetros y añadírselos a la colección Parameters.

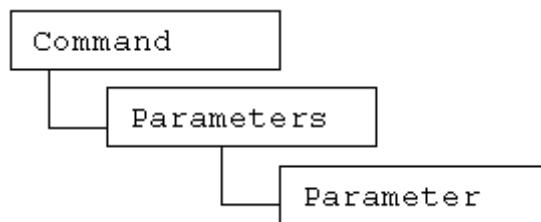


Figura 118. La colección Parameters.

Para crear un parámetro utilizaremos el método CreateParameter del objeto Command, al lanzar este método obtenemos un objeto Parameter, la sintaxis general de este método es la siguiente:

```
Set parametro=objComando.CreateParameter( [nombre] [, tipo] [, dirección] [, tamaño] [, valor] )
```

Donde parametro es un objeto Parameter en el que se va a almacenar el parámetro creado como resultado de lanzar este método, objComando es el objeto Command al que se le ha asociado ese parámetro, los siguientes argumentos son todos opcionales: nombre es una cadena que va a representar el nombre del parámetro (a través de este nombre se podrá hacer referencia al parámetro dentro de la colección Parameters), tipo indica el tipo de dato del parámetro (los tipos de datos se pueden ver en la tabla que se adjunta a continuación), dirección indica si el parámetro es de entrada o de salida (al igual que en el tipo de dato del parámetro los valores de este argumento se indicarán en una tabla) tamaño indica el tamaño máximo del parámetro y valor es el valor para ese objeto Parameter creado.

A continuación se ofrecen dos tablas, Tabla 23 y Tabla 24 con el nombre de las constantes para indicar el tipo de dato del parámetro y para indicar la dirección del parámetro.

Constante	Valor	Descripción
adBigInt	20	Entero con signo de 8 bytes.
adBinary	128	Un valor binario.
adBoolean	11	Un valor booleano.
adBSTR	8	Una cadena de caracteres terminada en nulo (Unicode).

adChar	129	Un valor de cadena.
adCurrency	6	Valor monetario (8 bytes).
adDate	7	Valor Dato.
adDBDate	133	Valor de fecha (yyyymmdd).
adDBTime	134	Valor de hora (hhmmss).
adDBTimeStamp	135	Valor time-stamp (yyyymmddhhmmss).
adDecimal	14	Un valor numérico con una precisión determinada.
adDouble	5	Un valor de doble precisión de coma flotante.
adEmpty	0	Valor no especificado.
adError	10	Código de error de 32 bits.
adGUID	72	Un identificador "globally unique" (GUID)
adIDispatch	9	Un puntero a un interfaz IDispatch en un objeto OLE.
adInteger	3	Entero con signo de 4 bytes.
adIUnknown	13	Un puntero a un interfaz IUnknown en un objeto OLE.
adLongVarBinary	205	Un valor binario largo.
adLongVarChar	201	Un valor de cadena largo.
adLongVarWChar	203	Un valor de cadena largo terminado en nulo.
adNumeric	131	Un valor numérico exacto.
adSingle	4	Un valor de coma flotante de precisión sencilla.
adSmallInt	2	Entero con signo de 2 bytes.
adTinyInt	16	Entero con signo de 1 byte.
adUnsignedBigInt	21	Entero de 8 bytes sin signo.
adUnsignedInt	19	Entero de 4 bytes sin signo.
adUnsignedSmallInt	18	Entero de 2 bytes sin signo.
adUnsignedTinyInt	17	Entero de 1 byte sin signo.
adUserDefined	132	Variable definida por el usuario.
adVarBinary	204	Un valor binario.

adVarChar	200	Un valor de cadena.
adVariant	12	Un Variant de OLE.
adVarWChar	202	Una cadena de caracteres Unicode terminada en nulo.
adWChar	130	Una cadena de caracteres Unicode terminada en nulo.

Tabla 23

Constante	Valor	Descripción
adParamInput	1	Parámetro de entrada.
adParamOutput	2	Parámetro de salida.
adParamInputOutput	3	Parámetro de entrada/salida
adParamReturnValue	4	Valor de retorno.

Tabla 24

Como se ha podido comprobar, podremos crear un objeto Parameter con todas sus propiedades especificadas en el método CreateParameter, pero también podremos lanzar el método CreateParameter sin especificar ninguna de estas propiedades (se debe recordar que todos los argumentos del método CreateParameter eran opcionales) y luego, a partir del objeto Parameter creado podremos asignar valores a sus propiedades de forma directa.

Las propiedades que posee el objeto Parameter son las siguientes:

- Name: nombre con el que se identifica al parámetro dentro de la colección Parameters. Este nombre no tiene porque coincidir con el nombre del parámetro correspondiente en el procedimiento almacenado de la base de datos.
- Type: tipo de dato del parámetro.
- Direction: indica la naturaleza (dirección) del parámetro.
- Size: es el tamaño del parámetro.
- Value: valor que se asigna al parámetro.
- NumericScale: indica el número de decimales soportado por el parámetro.
- Precision: número máximo de dígitos del parámetro.
- Attributes: es una combinación de ciertas constantes que indican otras características del parámetro. Estas constantes son adParamNullable (el parámetro acepta valores nulos), adParamSigned (el parámetro acepta valores con signo) y adParamLong (el parámetro acepta valores de datos grandes).

En el Código fuente 249 se muestran las dos posibilidades, es decir, se indican las características del parámetro a través del método CreateParameter y por otro lado se utilizan las propiedades del objeto Parameter para asignarles los valores correspondientes. En el ejemplo se pretende crear un parámetro de entrada llamado nombre de tipo char, de tamaño máximo 20 y cuyo valor es "Pepe".

```
<%Set objComando=Server.CreateObject ("ADODB.Command")
objComando.ActiveConnection="DSN=FuenteBD;UID=jose;PWD=xxx"
objComando.CommandType=adCmdText
objComando.CommandText="INSERT INTO Usuarios VALUES (?)"
Set parametro=objComando.CreateParameter("nombre",adChar,adParamInput,20,"Pepe")%>
<%Set objComando=Server.CreateObject ("ADODB.Command")
objComando.ActiveConnection="DSN=FuenteBD;UID=jose;PWD=xxx"
objComando.CommandType=adCmdText
objComando.CommandText="INSERT INTO Usuarios VALUES (?)"
Set parametro=objComando.CreateParameter
parametro.Name="nombre"
parametro.Type=adChar
parametro.Direction=adParamInput
parametro.Size=20
parametro.Value="Pepe" %>
```

Código fuente 249

Como se puede observar la primera forma de crear un parámetro y establecer sus propiedades es mucho más cómoda. También podemos apreciar que en la sentencia SQL que contiene el objeto Command, el parámetro se corresponde con el signo de interrogación (?), este es el carácter especial que indica en qué lugar se va a situar el parámetro.

Una vez creado el parámetro y establecidas sus propiedades lo deberemos añadir a la colección Parameters del objeto Command correspondiente, para ello esta colección ofrece el método Append. El orden en que se añaden los objetos Parameter a la colección Parameters es significativo, se irán sustituyendo los signos de interrogación (?) de la sentencia o los parámetros del procedimiento almacenado según se vayan añadiendo los parámetros a la colección. La sintaxis del método Append es:

```
objCommand.Parameters.Append objParametro
```

La colección Parameters, además del método Append ofrece dos métodos más: Delete y Refresh. El método Delete elimina el objeto Parameter de la colección Parameters, cuyo índice o nombre se pasa como argumento, la sintaxis de este método es:

```
objCommand.Parameters.Delete {índice|nombre}
```

El método Delete se suele utilizar para una vez ejecutado un objeto Command, eliminar sus parámetros para volver a utilizarlo con otros valores.

El método Refresh realiza una llamada al origen de los datos para obtener todos los datos correspondientes a los parámetros que necesita un objeto Command determinado. El método Refresh obtiene los nombres, tipos de datos, naturaleza y longitud de los parámetros y con esta información rellena la colección Parameters por nosotros, únicamente deberemos indicar los valores de los parámetros, no deberemos utilizar el método Append.

Sin embargo es recomendable construir la colección Parameters de forma manual, sin realizar la llamada al método Refresh, ya que esto último resulta más costoso para el origen de los datos.

La colección Parameters posee dos propiedades:

- Count: devuelve el número de objetos Parameters presentes en la colección Parameters.
- Item: es la propiedad por defecto de la colección Parameters y devuelve el objeto Parameters cuyo nombre o índice se le pasa como argumento.

En el siguiente apartado se comenta como ejecutar un objeto Command.

## Ejecución de comandos

Cuando se hayan facilitado al objeto Command todos sus parámetros ya estaremos en disposición de ejecutar el comando mediante el método Execute. Una vez ejecutado el objeto Command, si deseamos volver a utilizarlo con otros parámetros será necesario eliminar los parámetros existentes dentro de la colección Parameters, para ello se utiliza el método Delete (como ya comentamos en el apartado anterior).

Como se ha dicho con anterioridad para ejecutar un objeto Command se utiliza el método Execute, que presenta la siguiente sintaxis:

```
Set objRecordset=objComando.Execute(RegistrosAfectados,
Parámetros, Opciones)
objComando.Execute RegistrosAfectados, Parámetros, Opciones
```

En el caso de que la ejecución del objeto Command devuelva un conjunto de filas y columnas (resultado de una SELECT) lo podremos almacenar dentro de un objeto Recordset que podremos recorrer para mostrar la información. Como ya se comentó anteriormente, el objeto Command ofrecía el segundo mecanismo (el primer mecanismo lo ofrecía el objeto Connection) para obtener un objeto Recordset.

Los argumentos son todos opcionales, RegistrosAfectados es un entero que se devuelve y que indicará el número de registros que se han visto afectados por la ejecución del comando, Parámetros es un array de con los valores de los parámetros, se utiliza en el caso de que no hayamos añadido a la colección Parameters los parámetros correspondientes, Opciones indica que tipo de comando se quiere ejecutar, se utilizará si ya no lo hemos indicado en la propiedad CommandType del objeto Command, además este parámetro si se especifica sobreescribe al valor indicado en la propiedad CommandType.

Se ha comentado en el párrafo anterior otra forma de facilitar los parámetros a un objeto Command, en lugar de ir creando los parámetros con el método CreateParameter y añadirlos a la colección Parameters, podemos pasar al método Execute un array con los valores de los parámetros. Un ejemplo de esta técnica se puede comprobar en el Código fuente 250.

Esta forma de indicar los parámetros al objeto Command no es nada usual.

```
<% Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection="DSN=FuenteBD;UID=jose;PWD=xxxx"
objComando.CommandType=adCmdStoredProc
objComando.CommandText="AltaAlumno"
parametros=Array("Angel","Esteban")
objComando.Execute ,parametros,adExecuteNoRecords
```

Código fuente 250

En el Código fuente 250 se muestra como se ejecutaría una sentencia SQL que realiza una actualización sobre una tabla a partir de los parámetros que se le pasen.

```

1 <!--#INCLUDE VIRTUAL=/ADO/ADOVBS.INC-->
2 <%
3 Set objComando=Server.CreateObject("ADODB.Command")
4 objComando.ActiveConnection="DSN=FuenteBD;UID=pepe;PWD=xxx"
5 objComando.CommandType=adCmdText
6 objComando.CommandText="UPDATE Provincias SET Provincia=?&_
7                               WHERE IDProvin=?"
8 Set
parametro1=objComando.CreateParameter("nombre",adChar,adParamInput,20,"Cáceres")
9 Set parametro2=objComando.CreateParameter("clave",adChar,adParamInput,2,"10")
10 objComando.Parameters.Append parametro1
11 objComando.Parameters.Append parametro2
12 objComando.Execute
13 %>
```

Código fuente 251

En el Código fuente 252 se realiza una llamada a un procedimiento almacenado que realiza un alta sobre la tabla de provincias.

```

1 <!--#INCLUDE VIRTUAL=/ADO/ADOVBS.INC-->
2 <%
3 Set objComando=Server.CreateObject("ADODB.Command")
4 objComando.ActiveConnection="DSN=FuenteBD;UID=pepe;PWD=xxx"
5 objComando.CommandType=adCmdStoredProc
6 objComando.CommandText="altaProvincia"
7 Set
parametro1=objComando.CreateParameter("clave",adChar,adParamInput,2,"90")
8 Set
parametro2=objComando.CreateParameter("nombre",adChar,adParamInput,20,"Santander")
9 objComando.Parameters.Append parametro1
10 objComando.Parameters.Append parametro2
11 objComando.Execute
12 %>
```

Código fuente 252

Como se puede apreciar, en el caso de realizar la ejecución de un procedimiento almacenado, no se indican los parámetros con signos de interrogación, únicamente se debe proporcionar el nombre del procedimiento almacenado que queremos utilizar, y los parámetros se situarán según esté definido el procedimiento en la base de datos. Es responsabilidad del programador conocer la definición del procedimiento almacenado y por lo tanto proporcionar los parámetros de forma adecuada.

A continuación se va a ofrecer un ejemplo más completo. En esta página ASP, a partir de un par de formularios, podremos realizar operaciones de altas y bajas. Estas operaciones se van a realizar sobre una tabla llamada provincias y que posee dos campos, IDProvin y Provincia, el primero de ellos realiza la función de clave y el segundo es el nombre de la provincia.

Los dos formularios utilizados llaman a la misma página ASP en la que se encuentran, el aspecto que tiene la página ASP al acceder a ella es el que aparece en la Figura 119.

**Altas de Provincias**

ID Provincia	Provincia
<input type="text"/>	<input type="text"/>
<input type="button" value="Realizar Alta"/>	

**Bajas de Provincias**

<p style="margin: 0;">ID Provincia</p> <input type="text"/> <p style="margin: 0; text-align: center;"><input type="button" value="Realizar Baja"/></p>
--

Figura 119. Formularios para realizar las operaciones.

Al pulsar sobre el botón de la operación deseada se realizará la operación correspondiente tomando como parámetros la información que se introduce dentro de los campos de texto. En el Código fuente 253 se ofrece el código de la página ASP que construye el formulario anterior y permite realizar las operaciones a través de ADO.

```

1 <!--#INCLUDE VIRTUAL=/ADO/ADOVBS. INC-->
2 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
3 <HTML>
4 <HEAD>
5 <TITLE>Ejemplo objeto Command</TITLE>
6 </HEAD>
7 <BODY>
8 <%If Request.Form("botonAlta")="" AND Request.Form("botonBaja")="" Then%>
9   <table align="center" border="2">
10    <caption><strong>Altas de Provincias</strong></caption>
11    <tr><td>
12      <form action="comando.asp" method="POST">
13        <table border="0" align="center">
14          <tr>
15            <td>ID Provincia</td>
16            <td>Provincia</td>
17          </tr>
18          <tr>
19            <td><input type="Text" name="IDProvin" value=""></td>
20            <td><input type="Text" name="Provincia" value=""></td>
21          </tr>
22          <tr>
23            <td colspan="2" align="center"><input type="Submit" name="botonAlta"
24              value="Realizar Alta"></td>
25          </tr>
26        </table>
27      </td></tr>
28    </table>
```

```

29    <br><br><br>
30    <table align="center" border="2">
31      <caption><strong>Bajas de Provincias</strong></caption>
32      <tr><td>
33        <form action="comando.asp" method="POST">
34          <table border="0" align="center">
35            <tr>
36              <td align="center">ID Provincia</td>
37            </tr>
38            <tr>
39              <td align="center"><input type="Text" name="IDProvin" value=""></td>
40            </tr>
41            <tr>
42              <td align="center"><input type="Submit" name="botonBaja" value="Realizar
Baja"></td>
43            </tr>
44          </form>
45        </table>
46      </td></tr>
47    </table>
48  <%Else
49    Set objComando=Server.CreateObject ("ADODB.Command")
50    objComando.ActiveConnection="DSN=FuenteBD;UID=pepe;PWD=xxx"
51    objComando.CommandType=adCmdText
52    objComando.Prepared=True
53    if Request.Form("botonAlta")<>" " Then
54      idprovin=Request.Form("IDProvin")
55      provincia=Request.Form("Provincia")
56      objComando.CommandText="INSERT INTO Provincias VALUES (?,?)"
57      Set
parametro1=objComando.CreateParameter("param1",adChar,adParamInput,2,idprovin)
58      Set
parametro2=objComando.CreateParameter("param2",adChar,adParamInput,40,provincia)
59      objComando.Parameters.Append parametro1
60      objComando.Parameters.Append parametro2
61      objComando.Execute
62      Response.write("<center>Alta realizada correctamente</center>")
63    Elseif Request.Form("botonBaja")<>" " Then
64      idprovin=Request.Form("IDProvin")
65      objComando.CommandText="DELETE FROM Provincias WHERE IDProvin=?"
66      Set
parametro1=objComando.CreateParameter("param1",adChar,adParamInput,2,idprovin)
67      objComando.Parameters.Append parametro1
68      objComando.Execute numReg
69      if numReg=0 Then
70        Response.write("<center>Registro no encontrado</center>")
71      Else
72        Response.write("<center>Baja realizada correctamente</center>")
73      End If
74    End If
75  End If%>
76  </body>
77 </html>

```

Código fuente 253

En la línea 8 se comprueba si se proviene de la pulsación de uno de los botones de los formularios o por el contrario se viene de una petición de la página ASP. De la línea 9 a la 47 aparece el código HTML encargado de mostrar los formularios, es decir, la rama del If. Si la petición de la página ASP proviene de la pulsación de un botón de los formularios, se pasará a la ejecución de la rama Else.

De las líneas 49 a la 51 se crea el objeto Command encargado de realizar la operación de alta o de baja, como se puede observar el objeto Command se crea con total independencia de cualquier objeto

dentro de la jerarquía del modelo de objetos de ADO, la conexión con la fuente de datos del servidor se establece a través de la propiedad ActiveConnection del objeto Command. También indicamos el tipo de comando SQL que va a contener el objeto Command a través de su propiedad CommandType, y en la línea siguiente se indica que el contenido del objeto Command debe ser precompilado antes de su ejecución, es decir, va a ser una sentencia SQL preparada, con esto conseguimos una mayor eficiencia.

Entraremos en el If de la línea 52 si la operación a realizar es un alta. Los parámetros necesarios se recuperan a partir de la información de las cajas de texto del formulario, para ello se utiliza la colección Form del objeto Request. Una vez obtenidos los datos necesarios, se indica la sentencia SQL que se va a ejecutar, y como se indicó que era una sentencia preparada se enviará al proveedor de los datos para que realice una precompilación de la sentencia. La cadena que representa la sentencia a ejecutar es un INSERT, lo único que tiene de particular es el signo de interrogación que indica dónde irán los parámetros. Los parámetros se crean en las líneas 57 y 58, y en las líneas 59 y 60 se añaden a la colección Parameters del objeto Command.

En la línea 61 se realiza la ejecución del objeto Command, es decir, se realiza el alta con los parámetros indicados. Si hemos llegado hasta este punto podemos considerar que la operación de alta se ha realizado de forma correcta, por lo tanto, en la línea 62 le notificamos este éxito al usuario.

Si por el contrario, la operación elegida ha sido la realización de una baja entraremos en la rama ElseIf. En este caso se procede de igual manera, lo único que cambia es el contenido de la sentencia preparada, en este caso es un DELETE, y el número de parámetros a utilizar.

En la línea 68 se realiza la baja lanzando el método Execute sobre el objeto Command, pero esta vez, en la llamada a este método se utiliza un parámetro, numReg, este parámetro nos indicará el número de registros afectados por la operación realizada. En la línea 69 se comprueba el valor de esta variable, si es igual a cero, esto indicará que no se ha visto afectado ningún registro y que por lo tanto la operación de baja no se ha podido realizar porque el registro con esa clave no existía. Por el contrario, si el valor de la variable es distinto de cero, se indicará que la baja se ha realizado con éxito.

En lugar de utilizar sentencias preparadas para realizar las operaciones, podríamos haber hecho uso de procedimientos almacenados. En el Código fuente 254 se muestra en negrita lo que cambiaría al utilizar dos procedimientos almacenados llamados altaProvincia y bajaProvincia. La parte que se encarga de crear los formularios no se muestra ya que no se modifica nada.

```

48 <%Else
49     Set objComando=Server.CreateObject ("ADODB.Command")
50     objComando.ActiveConnection="DSN=FuenteBD;UID=pepe;PWD=xxx"
51     objComando.CommandType=adCmdStoredProc
52     'Se elimina, ya no es una sentencia precompilada: objComando.Prepared=True
53     if Request.Form("botonAlta")<>"" Then
54         idprovin=Request.Form("IDProvin")
55         provincia=Request.Form("Provincia")
56         objComando.CommandText="altaProvincia"
57         Set
parametro1=objComando.CreateParameter("param1",adChar,adParamInput,2,idprovin)
58         Set
parametro2=objComando.CreateParameter("param2",adChar,adParamInput,40,provincia)
59         objComando.Parameters.Append parametro1
60         objComando.Parameters.Append parametro2
61         objComando.Execute
62         Response.write("<center>Alta realizada correctamente</center>")
63     Elseif Request.Form("botonBaja")<>"" Then
64         idprovin=Request.Form("IDProvin")
65         objComando.CommandText="bajaProvincia"
66         Set

```

```

parametro1=objComando.CreateParameter("param1",adChar,adParamInput,2,idprovin)
67    objComando.Parameters.Append parametro1
68    objComando.Execute numReg
69    if numReg=0 Then
70        Response.write("<center>Registro no encontrado</center>")
71    Else
72        Response.write("<center>Baja realizada correctamente</center>")
73    End If
74 End If
75 End If%>

```

Código fuente 254

A la hora de utilizar procedimientos almacenados se deben tener en cuenta varias cosas, por un lado, que el proveedor de datos soporte la utilización de procedimientos almacenados, y por otro lado, conocer la estructura del procedimiento almacenado que se va a utilizar, es decir, su nombre, valor devuelto y el número, tipo y tamaño de los parámetros, y además si éstos son de entrada o de salida.

La forma de obtener un objeto Recordset a partir de la ejecución de un objeto Command es muy sencilla. A la hora de ejecutar el objeto Command con el método Execute podremos guardar su resultado en un objeto Recordset, de la misma forma que ocurría al ejecutar un sentencia SQL sobre un objeto Connection.

El próximo ejemplo se trata de una página ASP que realiza una SELECT sobre la tabla de provincias y guarda el resultado en un objeto Recordset para su futura inspección.

```

<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection=objConexion
objComando.CommandType=adCmdText
objComando.CommandText="SELECT * FROM Usuarios"
Set objRecordset=objComando.Execute%>

```

Código fuente 255

En el Código fuente 256 se crea un Recordset con la información devuelta por un procedimiento almacenado, en este caso, el procedimiento devuelveDomicilio devuelve una relación de los nombres de usuario junto con su domicilio.

```

<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection=objConexion
objComando.CommandType=adCmdStoredProc
objComando.CommandText="devuelveDomicilio"
Set objRecordset=objComando.Execute%>

```

Código fuente 256

A continuación se ofrecen unos cuantos ejemplos más que utilizan procedimientos almacenados para realizar las típicas operaciones de mantenimiento de una tabla, es decir, altas, bajas, consultas y modificaciones.

Al estructura de la tabla sobre la que vamos a realizar las operaciones se puede observar en el Código fuente 257 la sentencia SQL de creación de la tabla.

```
CREATE TABLE Cuentas(
    IdCuenta smallint NOT NULL ,
    Nombre varchar (25) NULL ,
    Saldo money NULL ,
    Observaciones varchar (250) NULL ,
    Usuario varchar (50) NOT NULL ,
    CONSTRAINT PK_Cuentas PRIMARY KEY (IdCuenta,Usuario)
)
```

Código fuente 257

En el Código fuente 258 se elimina un registro.

```
Set objCnx=Server.CreateObject ("ADODB.Connection")
objCnx.Open Application("CadenaConex")
Set objCmd=Server.CreateObject ("ADODB.Command")
objCmd.ActiveConnection=objCnx
objCmd.CommandText="EliminaRegistro"
Set objParam1=objCmd.CreateParameter ("param1",adSmallInt,adParamInput,2,6)
Set
objParam5=objCmd.CreateParameter ("param5",adVarChar,adParamInput,50,"anónimo")
objCmd.Parameters.Append objParam1
objCmd.Parameters.Append objParam5
objCmd.Execute numreg,,adCmdStoredProc
Response.Write("Registros eliminados: " & numreg)

objCnx.Close
Set objCnx=Nothing
```

Código fuente 258

El código del procedimiento almacenado es el Código fuente 259.

```
CREATE PROCEDURE EliminaRegistro
    @IdCuenta smallint,
    @Usuario varchar(50)
AS
    DELETE FROM Cuentas
    WHERE IdCuenta=@IdCuenta AND
        Usuario=@Usuario
```

Código fuente 259

La operación de modificación la vemos en el Código fuente 260.

```
Set objCnx=Server.CreateObject ("ADODB.Connection")
objCnx.Open Application("CadenaConex")
Set objCmd=Server.CreateObject ("ADODB.Command")
objCmd.ActiveConnection=objCnx
objCmd.CommandText="ModificaRegistro"
Set objParam1=objCmd.CreateParameter ("param1",adSmallInt,adParamInput,2,6)
```

```

Set objParam2=objCmd.CreateParameter("param2",adVarChar,adParamInput,25,"Pepa")
Set objParam3=objCmd.CreateParameter("param3",adCurrency,adParamInput,6,3000)
Set objParam4=objCmd.CreateParameter("param4",adVarChar,adParamInput,250,"nuevo
cliente")
Set objParam5=objCmd.CreateParameter("param5",adVarChar,adParamInput,50,"anónimo")
objCmd.Parameters.Append objParam1
objCmd.Parameters.Append objParam2
objCmd.Parameters.Append objParam3
objCmd.Parameters.Append objParam4
objCmd.Parameters.Append objParam5
objCmd.Execute numreg,,adCmdStoredProc
Response.Write("Registros modificados: " & numreg)

objCnx.Close
Set objCnx=Nothing

```

Código fuente 260

Procedimiento almacenado correspondiente.

```

CREATE PROCEDURE ModificaRegistro
    @IdCuenta smallint,
    @Nombre varchar(25),
    @Saldo money,
    @Observaciones varchar(250),
    @Usuario varchar(50)
AS
    UPDATE Cuentas      SET     Nombre=@Nombre,
                               Saldo=@Saldo,
                               Observaciones=@Observaciones
    WHERE IdCuenta=@IdCuenta AND
          Usuario=@Usuario

```

Código fuente 261

La operación de consulta posee el Código fuente 262.

```

Set objCnx=Server.CreateObject ("ADODB.Connection")
objCnx.Open Application("CadenaConex")

Set objCmd=Server.CreateObject ("ADODB.Command")
objCmd.ActiveConnection=objCnx
objCmd.CommandText="RecuperaRegistro"
Set objParam1=objCmd.CreateParameter("param1",adSmallInt,adParamInput,2,6)
Set
objParam5=objCmd.CreateParameter("param5",adVarChar,adParamInput,50,"anónimo")
objCmd.Parameters.Append objParam1
objCmd.Parameters.Append objParam5
Set rsRecordset=objCmd.Execute(numreg,,adCmdStoredProc)
Response.Write("Registros recuperados: " & numreg)
objCnx.Close
Set objCnx=Nothing

```

Código fuente 262

Y su procedimiento almacenado es el que vemos en el Código fuente 263.

```

CREATE PROCEDURE RecuperaRegistro
    @IdCuenta smallint,
    @Usuario varchar(50)
AS
    SELECT * FROM Cuentas
    WHERE IdCuenta=@IdCuenta AND
        Usuario=@Usuario

```

Código fuente 263

Y por último la operación de inserción de un registro.

```

Set objCnx=Server.CreateObject ("ADODB.Connection")
objCnx.Open Application("CadenaConex")

Set objCmd=Server.CreateObject ("ADODB.Command")
objCmd.ActiveConnection=objCnx

objCmd.CommandText="InsertaRegistro"
Set objParam1=objCmd.CreateParameter ("param1",adSmallInt,adParamInput,2,6)
Set objParam2=objCmd.CreateParameter ("param2",adVarChar,adParamInput,25,"Pepe")
Set objParam3=objCmd.CreateParameter ("param3",adCurrency,adParamInput,6,2000)
Set objParam4=objCmd.CreateParameter ("param4",adVarChar,adParamInput,250,"nuevo
cliente")
Set objParam5=objCmd.CreateParameter ("param5",adVarChar,adParamInput,50,"anónimo")
objCmd.Parameters.Append objParam1
objCmd.Parameters.Append objParam2
objCmd.Parameters.Append objParam3
objCmd.Parameters.Append objParam4
objCmd.Parameters.Append objParam5
objCmd.Execute numreg,,adCmdStoredProc
Response.Write("Registros insertados: " & numreg)

objCnx.Close
Set objCnx=Nothing

```

Código fuente 264

También con su procedimiento almacenado. No es necesario destruir los distintos objetos Command que hemos creado en nuestra página ASP. Al cerrar y destruir el objeto Connection asociado a los objetos Command, de forma automática se destruyen éstos también.

```

CREATE PROCEDURE InsertaRegistro
    @IdCuenta smallint,
    @Nombre varchar(25),
    @Saldo money,
    @Observaciones varchar(250),
    @Usuario varchar(50)
AS
    INSERT INTO Cuentas
    VALUES (@IdCuenta,@Nombre,@Saldo,@Observaciones,@Usuario)

```

Código fuente 265

## Tratamiento de errores en ADO

Este apartado no es específico del objeto Command, pero se ha considerado adecuado incluirlo en este mismo tema.

En el tema anterior dedicado al objeto Connection vimos que este objeto poseía una colección llamada Errors. Esta colección tiene objetos de tipo Error que contienen información sobre cada uno de los errores que se han producido en la última operación realizada sobre el objeto Connection.

Se debe señalar que el control de errores se encuentra reservado únicamente a objetos Connection, es decir, si queremos controlar los errores que se producen al manipular un objeto Recordset, lo deberemos hacer a través del tratamiento de errores estándar que ofrece VBScript mediante el objeto Err. Cuando se produce un error en tiempo de ejecución, las propiedades del objeto Err se llenan con información que identifica al error de forma única y que puede utilizarse para procesarlo.

Por lo tanto la colección Errors nos servirá para detectar errores que se han producido al realizar operaciones directamente sobre un objeto Connection, es decir, errores ofrecidos por el proveedor de datos de forma directa.

Cada objeto Error de la colección Errors posee las siguientes propiedades:

- NativeError, indica el código de error devuelto por el servidor de datos.
- Description: contiene la descripción de error (es una información valiosa).
- SQLState indica el error SQL estándar asociado.
- HelpContext: indica el contexto de ayuda del error.
- HelpFile: indica el fichero de ayuda asociado.
- Source: es una cadena que contiene el nombre del objeto o aplicación que ha originado el error.

En el Código fuente 266 se muestra una función que se encarga de la gestión de errores. La función recibe como parámetro un objeto Connection y muestra la información asociada a los errores que se hayan producido.

```
<%FUNCTION ErrorHTML(conex)
Dim hayError
hayError=false
If conex.Errors.Count<>0 Then
    hayError=true%>
    <center><h1>Se han producido errores</h1>
    <table border=1><tr><th>Código</th><th>Error</th>
    <th>Estado SQL</th><th>Fuente</th></tr>
    <%For Each objError in conex.Errors%>
        <tr><td>
            <%=objError.NativeError%>
        </td><td>
            <%=objError.Description%>
        </td><td>
            <%=objError.SQLState%>
        </td><td>
            <%=objError.Source%>
        </td></tr>
    <%Next%>
</table>
</center>
```

```

    </td></tr>
<%Next%>
</table></center>
<%End If
ErrorHTML=hayError
END FUNCTION%>

```

Código fuente 266

La función recorre la colección Errors y muestra la información del cada objeto Error en una tabla, de esta forma si se produce un error el usuario no quedará tan perplejo. La función devuelve un valor booleano para indicar si se ha producido un error o no, es decir, el número de elementos de la colección Errors es mayor que cero o no.

Este código lo podemos guardar en un fichero .ASP, para luego incluirlo en todas nuestras páginas en las que queramos controlar los errores de los objetos Connection. En el siguiente código se hace una SELECT sobre la tabla de usuarios, pero antes de mostrar la información del Recordset correspondiente, se comprueba mediante la función ErrorHTML si se han producido errores en el objeto Connection. Si no se han producido se muestran los registros de la tabla, y en caso contrario se mostrará la información asociada a cada uno de los errores.

```

1 <!--#INCLUDE FILE="ErroresADO.asp"-->
2 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
3 <HTML>
4 <HEAD>
5 <TITLE>Ejemplo objeto Recordset</TITLE>
6 </HEAD>
7 <BODY>
8 <%On Error Resume Next
9 Set objConexion=Server.CreateObject("ADODB.Connection")
10 objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;" &
                    "Initial Catalog=pubs;User id=sa"
11 Set objRecordset=objConexion.Execute("SELECT * FROM Usuarios")
12 if Not ErrorHTML(objConexion) Then%
13     <table border="1" align="center">
14         <tr>
15             <th>DNI</th>
16             <th>Nombre</th>
17             <th>Domicilio</th>
18             <th>Código Postal</th>
19         </tr>
20         <%while not objRecordset.EOF%>
21             <tr>
22                 <td><%=objRecordset("DNI")%></td>
23                 <td><%=objRecordset("Nombre")%></td>
24                 <td><%=objRecordset("Domicilio")%></td>
25                 <td align="right"><%=objRecordset("Codigo_Postal")%></td>
26             </tr>
27             <%objRecordset.MoveNext
28 WEnd%>
29     </table>
30 <%End If
31 objRecordset.close
32 Set objRecordset=Nothing
33 objConexion.close
34 Set objConexion=Nothing%>
35 </body>
36 </html>

```

Código fuente 267

Como se puede observar, incluimos el fichero ErroresADO.asp que es dónde se encuentra la función de control de errores vista anteriormente. También se debe señalar que utilizamos la instrucción On Error Resume Next para que el error que se produzca no detenga la ejecución de la página ASP.

Si modificamos la línea 11 de la siguiente forma: Set objRecordset=objConexion.Execute("SELECT \* FROM Usrios"), se producirá un error, ya que la nueva tabla no existe, y la página ASP ofrecerá el aspecto que muestra la Figura 120.

## Se han producido errores

Código	Error	Estado SQL	Fuente
208	El nombre de objeto 'Usrios' no es válido.	42S02	Proveedor de Microsoft OLE DB para SQL Server

Figura 120. Errores sobre Connection.

Si queremos tener un control sobre todos los errores que se puedan dar sobre cualquier objeto ADO deberemos modificar la función de tratamiento de errores y utilizar el control de errores de VBScript, la función resultante sería la que se muestra en el Código fuente 268.

```

1 <SCRIPT LANGUAGE=VBScript RUNAT=Server>
2 FUNCTION ErrorHTML()
3 Dim hayError
4 hayError=false
5 If Err.Number<>0 Then
6     hayError=true
7     Response.write("<center><h1>Se han producido errores</h1>")
8     Response.write("<table border=1><tr><th>Error</th></tr>")
9     Response.write("</td><td>")
10    Response.write (Err.Description)
11    Response.write("</td></tr>")
12    Response.Write("</table></center>")
13 End If
14 ErrorHTML=hayError
15 END FUNCTION
16 </SCRIPT>
```

Código fuente 268

En este caso preguntamos por la propiedad Number del objeto Err, si es cero es que no se han producido errores y si se producen se muestra el contenido de la propiedad Description, es decir, la descripción del error. Lo único que debemos modificar del código del ejemplo que mostraba el contenido de la tabla Usuarios, es la llamada a la función, ya que la nueva versión de la función ErrorHTML() no recibe ningún parámetro.

El aspecto que ofrecería la página ASP con el mismo error que antes pero con la nueva función es el que se muestra en la Figura 121.

La única diferencia es que no podemos recuperar el código de error ni el estado SQL, ya que estamos dentro de un control de errores mucho más general.

## Se han producido errores



Figura 121. Otro control de errores.



# Acceso a datos con ADO: Recordset I

---

## El objeto Recordset

Este es el tercero de los objetos principales del modelo de objetos de ADO. Como se ha visto en los dos capítulos anteriores, un objeto de este tipo se obtiene de forma automática a partir de la ejecución de un objeto Connection o un objeto Command. Pero el objeto Recordset que se obtiene es muy sencillo y limitado, representa un cursor de sólo lectura y que únicamente permite un desplazamiento hacia delante. Un objeto Recordset se puede crear de forma independiente, es decir, sin depender de ningún objeto Connection o Command.

Un objeto Recordset es un objeto tabular que contiene datos. Los valores se encuentran en las filas y los nombres de los campos en las columnas. Cada fila es un registro completo. Es importante destacar que un objeto Recordset es una representación de los datos, pero no son los datos almacenados. En un objeto Recordset se deben determinar tres cosas: el lugar en el que se encuentra, es decir, la fuente de los registros que contiene, las capacidades de navegación y la conexión con los datos. Un cursor determina el comportamiento de un objeto Recordset.

Por lo tanto, si queremos disponer de un cursor más potente, deberemos crear un objeto Recordset. al igual que creábamos un objeto Connection o Command, y a través de sus propiedades definir exactamente las características del Recorset. A continuación se pasa a mostrar de forma resumida todas las propiedades, métodos y colecciones que posee este objeto.

El objeto Recordset posee un gran número de propiedades:

- **PageSize:** número de registros del Recordset que se van a encontrar dentro de una página lógica.

- AbsolutePage: número de página del registro actual. Para movernos a una página determinada le asignaremos a esta propiedad el número de página correspondiente.
- AbsolutePosition: especifica la posición ordinal del registro actual dentro de un Recordset.
- PageCount: indica el número de páginas lógicas que posee un objeto Recordset.
- ActiveConnection: indica la conexión a la que está asociado el objeto Recordset. Esta propiedad es sólo de lectura en el caso de que la propiedad Source tenga un valor válido.
- ActiveCommand: indica el objeto Command asociado con el objeto Recordset, si es que se ha utilizado uno para crear el Recordset. Esta propiedad es de sólo lectura.
- Source: indica la procedencia de los datos que contiene el Recordset, puede ser un objeto Command, una sentencia SQL, un nombre de una tabla , un procedimiento almacenado, una dirección de Internet, etc. Esta propiedad es de lectura/escritura si el Recordset está cerrado, y sólo de lectura si está abierto.
- DataMember: especifica el nombre de un miembro de datos del que obtener datos, este miembro de datos pertenece al origen de datos especificado en la propiedad DataSource.
- DataSource: especifica un objeto que contiene datos que pueden ser representados como un objeto Recordset.
- Index: devuelve en una cadena el nombre del índice que se está utilizando actualmente.
- CursorLocation: indica la localización del motor del cursor, puede encontrarse en el cliente (adUseClient) o en el servidor (adUseServer).
- MarshalOptions: indica los registros que deben ser enviados al servidor.
- Sort: especifica el nombre del campo o campos por el que se encuentra ordenado el objeto Recordset, así como el orden.
- State: indica el estado del Recordset, si se encuentra abierto (adStateOpen) o cerrado (adStateClosed).
- LockType: indica el tipo de bloqueo que se aplicará al objeto Recordset.
- CursorType: indica el tipo de cursor que se utilizará en el Recordset.
- Bookmark: guarda una posición determinada dentro de un Recordset para volver a ella en otro momento.
- Status: indica el estado del registro actual.
- Filter: indica que se va a realizar un filtro sobre el Recordset.
- CacheSize: indica el número de registros que se encuentran en la memoria.
- EditMode: indica el proceso de edición del registro actual. Es de sólo lectura.

- MaxRecords: indica el número máximo de registros que debe contener un Recordset como resultado de una consulta. Se utiliza esta propiedad para limitar el número de registros devueltos por el proveedor desde una fuente de datos.
- RecordCount: devuelve el número de registros de un objeto Recordset.
- BOF: indica si la posición actual se encuentra antes del primer registro de un Recordset.
- EOF: indica si la posición actual se encuentra después del último registro de un Recordset.

Los métodos de estos objetos también son numerosos, y son los que aparecen a continuación:

- Open: este método abre un cursor que va a representar los registros resultantes de la realización de un comando SQL.
- Close: cierra el cursor, perdiendo todos los datos asociados.
- CompareBookmarks: compara dos Bookmark y devuelve el resultado de la comparación.
- Move: la posición actual se desplaza a la posición indicada.
- GetString: devuelve el Recordset completo dentro de una cadena.
- MoveNext: el siguiente registro en un objeto Recordset pasa a ser el actual.
- MovePrevious: el registro anterior pasa a ser el registro actual.
- MoveFirst: el primer registro dentro de un objeto Recordset especificado, pasa a ser el registro actual.
- MoveLast: el último registro dentro de un objeto Recordset especificado, pasa a ser el registro actual.
- NextRecordset: elimina el Recordset actual y se desplaza al siguiente. Esto tiene sentido cuando el comando SQL que se ha ejecutado, y cuyo resultado contiene el objeto Recordset, está compuesto de varios resultados.
- AddNew: crea un nuevo registro en un objeto Recordset actualizable.
- Delete: borra el registro actual o grupo de registros.
- Find: busca en el Recordset un registro que coincida con el criterio especificado.
- Update: almacena todos los cambios realizados sobre el registro actual.
- CancelUpdate: cancela los cambios realizados sobre el registro actual o sobre un nuevo registro sobre el que todavía no se ha lanzado el método Update.
- UpdateBatch: almacena todos los cambios pendientes de diferentes registros.
- CancelBatch: cancela todos los cambios pendientes de diferentes registros.
- GetRows: devuelve los registros de un Recordset dentro de un array de dos dimensiones.

- Supports: indica si el objeto Recordset soporta una función determinada.
- Clone: crea una copia de un objeto Recordset existente.
- Requery: actualiza los datos de un Recordset volviendo a ejecutar el comando correspondiente que creó el objeto.
- Resync: refresca los datos en el Recordset actual.
- Save: almacena el Recordset en un fichero.
- Seek: localiza un valor dentro del Recordset.
- Supports: indica si un objeto Recordset soporta la funcionalidad específica que se le pasa como argumento a este método.

El objeto Recordset posee dos colecciones:

- Fields: esta colección está formada por objetos Field. Cada objeto Field representa una columna del Recordset, es decir, un campo.
- Properties: esta colección es como la que poseían los objetos Connection y Command.

A continuación vamos a comentar las distintas características que pueden definir un Recordset.

## Tipos de Recordset

Como ya se comentó anteriormente, si queremos tener control sobre el tipo de objeto Recordset que se va a crear, lo deberemos crear de forma explícita y darle los valores deseados a sus propiedades. Las diferentes funciones que se pueden realizar con los diferentes tipos de objetos Recordset, vendrán determinadas por el valor de sus propiedades.

Antes de almacenar dentro de un Recordset los datos que provienen de la ejecución de un comando, necesitaremos definir algunas de sus propiedades. Para indicar el tipo de bloqueo que se utilizará dentro del objeto Recordset se utilizará la propiedad LockType, el bloqueo nos asegura que al mismo tiempo sólo un usuario puede estar modificando un dato. El tipo de bloqueo vendrá dado por nuestras necesidades y el entorno en el que estemos trabajando. Existen tres tipos de bloqueo:

- Sólo lectura: todos los datos del cursor se encuentran bloqueados frente a modificaciones o altas.
- Pesimista: se denomina así por que se considera el peor de los casos, es decir, un registro va a ser modificado al mismo tiempo. Este mecanismo bloquea el registro o página actual del cursor. Otros usuarios no pueden realizar cambios en el registro actual ni en la página actual de registros hasta que el usuario actual no se haya desplazado o cerrado el objeto Recordset. Este tipo de bloqueo puede causar problemas de concurrencia, ya que, si un proceso de modificación tarda mucho tiempo en realizarse, ningún usuario podrá acceder a ese registro hasta que no haya finalizado el proceso.
- Optimista: se denomina así porque se considera el mejor de los casos, la modificación de un registro es muy difícil que se produzca al mismo tiempo. El registro se bloquea únicamente en el momento en el que se va a actualizar, no durante todo el proceso de edición como ocurría

con el bloqueo pesimista. Esto mejora con creces la concurrencia pero puede darse el caso que dos personas estén editando el mismo dato al mismo tiempo.

Los valores que puede tener la propiedad LockType se identifican con las constantes de ADO que aparecen en la Tabla 25.

Tipo de Bloqueo	Valor	Descripción
adLockReadOnly	1	Sólo de lectura. No se puede modificar los datos ni añadir nuevos registros.
adLockPessimistic	2	El registro o página de registros se bloquea cuando pasa a ser el actual. Nadie más podrá editar el registro actual.
adLockOptimistic	3	Los registros se bloquean únicamente en el momento en el que son actualizados.
adLockBatchOptimistic	4	Es el bloqueo optimista pero en este caso se almacenar las modificaciones pendientes de varios registros.

Tabla 25

Además de definir el tipo de bloqueo podemos indicar el tipo de cursor que se va a utilizar. esto se realiza a través de la propiedad CursorType, esta propiedad, junto con la anterior, va a definir el nivel de aislamiento del cursor y las capacidades de navegación. En la Tabla 26 se muestra los diferentes tipos de cursores, es decir, los valores que puede tomar la propiedad CursorType.

Cursor	Constante ADO	Descripción
Forward-Only	adOpenForwardOnly	Es el tipo de cursor por defecto. El movimiento que ofrece a través del Recordset es únicamente hacia delante.
Static	adOpenStatic	Permite cualquier tipo de desplazamiento a través del Recordset. Pero no son visibles las modificaciones realizadas por otros usuarios, así como las altas y las bajas. Soporta actualización en batch.
KeySet	adOpenKeyset	Permite ver las modificaciones y bajas realizadas por otros usuarios, pero no las altas. Soporta actualización en batch.
Dynamic	adOpenDynamic	Es el más potente de todos los cursores, aunque no permite la paginación. Es como el cursor KeySet pero permite ver también altas realizadas por otros usuarios.

Tabla 26

Elegir el cursor más adecuado es una tarea muy importante de cara a la eficiencia del programa. Se debe tener en cuenta que cuanto más potente sea el cursor más cuesta mantenerlo. Se debe elegir el cursor que cubra nuestras necesidades, pero que no las sobrepase.

Si elegimos un cursor demasiado potente del que no utilizamos todas sus características estaremos desperdiando recursos.

Si se elige el cursor adecuado la información se presentará al usuario de forma más rápida, pero si dentro de Internet o una gran intranet utilizamos un cursor inadecuado el sistema podría venirse abajo debido al gran número de usuarios concurrentes.

Hay tres operaciones diferentes que podemos necesitar realizar con nuestros datos y que determinarán el tipo de cursor a utilizar. Puede que necesitemos simplemente realizar un informe mostrando los datos, puede que tengamos que editar los datos y modificarlos o puede que necesitemos añadir y eliminar registros.

También hay casos en los que realizaremos cada una de estas operaciones, en este caso puede ser recomendable utilizar un tipo de cursor diferente en la misma aplicación para cada tarea correspondiente.

La primera y más sencilla tarea es la lectura o mostrar informes. En este caso un cursor del tipo Forward-Only funcionará de forma adecuada ya que no se van a realizar modificaciones, altas ni bajas, simplemente se van a leer los datos.

Si queremos ver los cambios realizados sobre los datos que estamos leyendo podemos pensar en utilizar un cursor más potente y con un nivel de aislamiento menor, como pueden ser los del tipo KeySet o Dynamic, pero esto sería un error ya que se necesitarían muchos recursos al utilizar estos cursos, por lo tanto es mejor utilizar un cursor de tipo Forward-Only y periódicamente lanzar el método Requery del objeto Recordset que volverá a ejecutar el comando SQL que dio lugar al Recordset y por lo tanto actualizará los datos contenidos en él.

Además de utilizar un cursor de tipo Forward-Only también podremos utilizar un bloqueo de sólo lectura, la combinación de un tipo de cursor con un tipo de bloqueo adecuado nos servirá para definir de una mejor forma el comportamiento de nuestro cursor, además, podremos evitar errores y ahorrar en recursos.

En el caso de realizar una operación de edición y modificación sobre los datos el tipo de cursor y de bloqueo dependerá no sólo de la operación a realizar sino también del entorno concreto en el que nos encontramos.

Si los datos poseen un alto grado de interactividad (por ejemplo, un sistema de pedidos), es recomendable utilizar un bloqueo fuerte (bloqueo optimista) y un cursor poco aislado (KeySet). Si los cambios realizados por otros no nos afectan podremos utilizar un cursor del tipo Static o incluso Forward-Only.

Para elegir el tipo de bloqueo dependerá del tráfico de nuestra red. En el caso de poseer un alto tráfico la posibilidad de acceso y modificación de un mismo dato crece considerablemente, es este caso se podrá utilizar un bloqueo pesimista, aunque esto reducirá el nivel de concurrencia del sistema.

En sitios Web con poco tráfico lo mejor es utilizar un bloqueo de tipo optimista, ya que la posibilidad de modificar un dato al mismo tiempo por diferentes usuarios baja de forma considerable.

En el último caso, es decir, cuando es necesario realizar bajas y altas sobre los datos, estas operaciones las podremos realizar con los métodos AddNew o Delete del objeto Recordset, pero la forma más eficiente es realizar las altas y las bajas a través de comandos del origen de los datos, es decir,

utilizando sentencias SQL o procedimientos almacenados que realicen estas tareas, u otros tipos de operaciones según la naturaleza del origen de los datos. De esta forma será el proveedor de datos el encargado de manejar el cursor y realizar de forma directa el alta o la baja correspondiente.

Una vez que tenemos el Recorset creado podremos recorrerlo y navegar a través de él. En el siguiente ejemplo se puede observar como se crearía un Recordset y como se muestra la información que contiene.

En este caso el Recordset creado posee un tipo de bloqueo de sólo lectura y el cursor es Foward-Only, para movernos a través del Recordset utilizaremos su método MoveNext que desplazará el puntero de registro actual al siguiente registro.

El recorrido del objeto Recordset se realiza dentro de un bucle de tipo mientras que se terminará cuando se haya llegado al final del Recordset, esto vendrá indicado por la propiedad EOF del objeto Recordset. Esta propiedad tendrá el valor True si hemos pasado del último registro del objeto Recordset.

En este caso el Recordset va a contener el resultado de una SELECT sobre una tabla llamada usuarios que posee los campos DNI, Nombre, Domicilio y Código\_Postal. El resultado del Código fuente 269 es el que se muestra en la Figura 122.

```
<HTML>
<HEAD>
<TITLE>Ejemplo objeto Recordset</TITLE>
</HEAD>

<BODY>
<%Set objConexion=Server.CreateObject ("ADODB.Connection")
objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
Set objRecordset=objConexion.Execute ("SELECT * FROM Usuarios")%>
<table border="1" align="center">
<tr>
    <th>DNI</th>
    <th>Nombre</th>
    <th>Domicilio</th>
    <th>Código Postal</th>
</tr>
<%while not objRecordset.EOF%>
<tr>
    <td><%=objRecordset ("DNI")%></td>
    <td><%=objRecordset ("Nombre")%></td>
    <td><%=objRecordset ("Domicilio")%></td>
    <td align="right"><%=objRecordset ("Codigo_Postal")%></td>
</tr>
<%objRecordset.MoveNext
Wend
objRecordset.Close
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
</table>
</body>
</html>
```

Código fuente 269

DNI	Nombre	Domicilio	Código Postal
1111111	Don Pepito	c\Pan y Toros 25	28041
123345L	Pepito Grillo	c\Zapatero 9	28023
1883045K	Don Pimpom	c\Sesamo 8	28020
18833451Ñ	Jaime Espina	c\Silva 12	28021
1883345L	Espinete Puas	c\Sesamo 4	28020
18889945L	John Smith	c\Genova 21	28040
333333	Jose Mari	c/Sal si puedes	28041
4883345I	Angel Esteban	c\De La Lechuga	28010
90909090	Chojin	c\Fuencarral 12	28067

Figura 122. Contenido del Recordset

Esta es la forma más sencilla y directa de crear un objeto Recordset, aunque también es la menos potente, ya que obtenemos un cursor de sólo lectura y sólo podemos avanzar en un sentido, es decir, sería un cursor Forward Only y con tipo de bloqueo ReadOnly.

Pero si queremos manipular el objeto Recordset para asignarle las propiedades que deseemos, como tipo de cursor y tipo de bloqueo, deberemos crear una instancia de un objeto Recordset a través del método CreateObject del objeto Server. En el siguiente apartado veremos las distintas formas que tenemos de crear y abrir objetos RecordSet.

Vamos a utilizar un ejemplo bastante didáctico que muestra las operaciones y funcionalidades que soporta cada tipo de cursor. Para ello se ha utilizado el método Supports, esté método devolverá verdadero o falso dependiendo de si el cursor correspondiente soporta la característica o función que le pasamos como argumento.

El parámetro del método Supports va a ser una constante de ADO que se corresponderá con una función determinada, así adAddNew se corresponde a la funcionalidad que ofrece AddNew, y podemos preguntar por lo tanto mediante el método Supports si el cursor soporta añadir nuevos registros.

Para el ejemplo se han utilizado dos objetos Dictionary (ya tratados en temas anteriores) para almacenar las constantes que definen los tipos de cursor y los tipos de bloqueos, respectivamente. El tipo de cursor y de bloqueo que se van a utilizar para averiguar las funcionalidades que soporta se van a seleccionar de dos listas. Al pulsar el botón de envío se mostrará en una tabla los resultados que ofrece el método Support para cada una de las funcionalidades.

En el Código fuente 270 se ofrece el código fuente completo de este ejemplo.

```
<%Set objCursor = Server.CreateObject("Scripting.Dictionary")
objCursor.Add "adOpenForwardOnly", "0"
objCursor.Add "adOpenStatic", "3"
objCursor.Add "adOpenKeyset", "1"
objCursor.Add "adOpenDynamic", "2"
Set objBloqueo = Server.CreateObject("Scripting.Dictionary")
objBloqueo.Add "adLockReadOnly", "1"
objBloqueo.Add "adLockOptimistic", "3"
objBloqueo.Add "adLockBatchOptimistic", "4"%>
```

```

objBloqueo.Add "adLockPessimistic", "2%">
<HTML>
<BODY>
<form action="ejemploCursos.asp" method="GET">
Tipo cursor (CursorType)
<select name="cursor">
<%'Se guardan los items y las claves del diccionario en arrays y se añade a una
lista
arrCursorClave=objCursor.Keys
Response.Write Request.QueryString("cursor")
For i=0 to objCursor.Count - 1
    if arrCursorClave(i) = Request.QueryString("cursor") then%
        <option value="<%arrCursorClave(i)%>" selected><%arrCursorClave(i)%>
    <%else%>
        <option value="<%arrCursorClave(i)%>"><%arrCursorClave(i)%>
    <%end if
Next%>
</select><br>
Tipo bloqueo (LockType)
<select name="bloqueo">
<%'se repite la operación anterior para el segundo diccionario
arrBloqueoClave=objBloqueo.Keys
For i=0 to objBloqueo.Count - 1
    if arrBloqueoClave(i) = Request.QueryString("bloqueo") then%
        <option value="<%arrBloqueoClave(i)%>" selected><%arrBloqueoClave(i)%>
    <%else%>
        <option value="<%arrBloqueoClave(i)%>"><%arrBloqueoClave(i)%>
    <%end if
Next%>
</select><br>
<input type="submit" name="enviar" value="Enviar">
</form>
<%if Request.QueryString("enviar")<>" " then
    Set oRst = Server.CreateObject ("ADODB.Recordset")
    'Hay que convertir la clave en string explícitamente
    'si no no recupera los valores del diccionario
    strCursor = CStr(Request.QueryString("cursor"))
    strBloqueo = CStr(Request.QueryString("bloqueo"))
    cursor = objCursor(strCursor)
    bloqueo = objBloqueo(strBloqueo)
    'se abre el recordset con el tipo de cursor y de bloqueo indicado
    oRst.Open
"Preguntas", "DSN=FuenteCursos;UID=cursos;PWD=" , cursor, bloqueo, adCmdTable%>
    <table border="1">
    <tr>
    <th colspan="2">Cursor/Bloqueo</th>
    </tr>
    <tr>
    <td colspan="2"><%=Request.QueryString("cursor")%>
    /<%=Request.QueryString("bloqueo")%>

```

```

<tr>
<td>¿Soporta MovePrevious?</td>
<th><%=oRst.Supports(adMovePrevious)%></th>
</tr>
<tr>
<td>¿Soporta BookMark?</td>
<th><%=oRst.Supports(adBookmark)%></th>
</tr>
<tr>
<td>¿Soporta Find?</td>
<th><%=oRst.Supports(adFind)%></th>
</tr>
<tr>
<td>¿Soporta Resync?</td>
<th><%=oRst.Supports(adResync)%></th>
</tr>
</table>
<%oRst.Close
Set oRst = Nothing
End if%>
</BODY>
</HTML>

```

Código fuente 270

Un ejemplo de ejecución del Código fuente 270 podría ser la Figura 123.

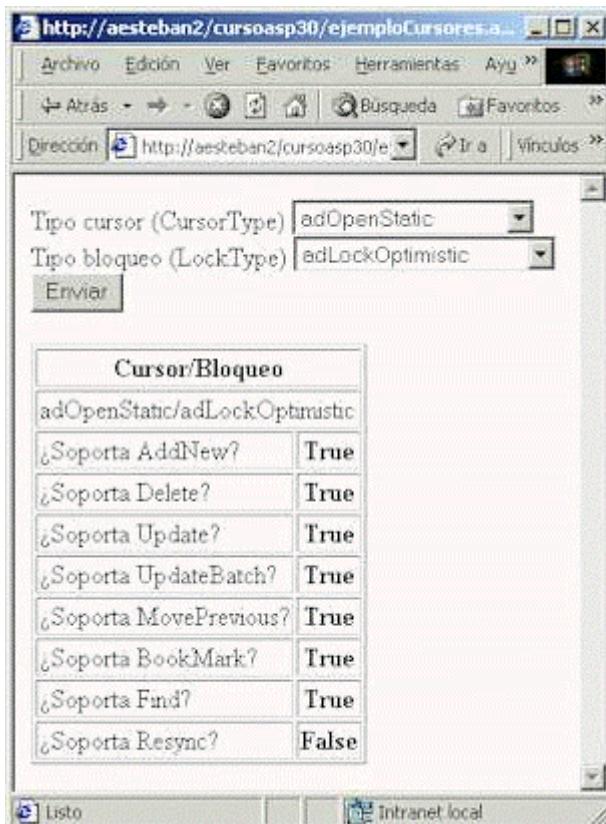


Figura 123. Características soportadas por un cursor

Como se puede observar en el ejemplo, el objeto Recordset se crea mediante el método Open, al que se pasa como parámetros el tipo de cursor y el tipo de bloqueo seleccionado en el formulario de la página.

## Creación y apertura de objetos recordset

Vamos a comenzar este apartado comentando el Código fuente 271 de ASP.

```
<HTML>
<HEAD>
<TITLE>Ejemplo objeto Recordset</TITLE>
</HEAD>
<BODY>
<%Set objRecordset=Server.CreateObject("ADODB.Recordset")>
objRecordset.Source="SELECT * FROM Usuarios"
objRecordset.CursorType=adOpenStatic
objRecordset.ActiveConnection="DSN=FuenteBD;UID=pepe;PWD=xxx"
objRecordset.Open%>
<center><strong>
Número de registros: <%=objRecordset.RecordCount%><br><br>
</strong></center>
<table border="1" align="center">
<tr>
<th>DNI</th>
<th>Nombre</th>
<th>Domicilio</th>
<th>Código Postal</th>
</tr>
<%while not objRecordset.EOF%>
<tr>
<td><%=objRecordset("DNI")%></td>
<td><%=objRecordset("Nombre")%></td>
<td><%=objRecordset("Domicilio")%></td>
<td align="right"><%=objRecordset("Codigo_Postal")%></td>
</tr>
<%objRecordset.MoveNext
Wend
objRecordset.Close
Set objRecordset=Nothing%>
</table>
</body>
</html>
```

Código fuente 271

En este ejemplo se ha creado un objeto Recordset y se han manipulado tres propiedades del mismo. A la propiedad CursorType, es decir, la propiedad que indica el tipo de cursor que va a poseer el Recordset, se le ha asignado el valor adOpenStatic. De esta forma podremos utilizar una serie de características del Recordset. En este caso se ha utilizado la propiedad RecordCount que devuelve el número de registros que existen dentro un Recordset, además, este tipo de cursor permite un desplazamiento completamente libre sobre el Recordset.

La propiedad Source indica la procedencia de los datos del Recordset, el valor de esta propiedad es un comando SQL válido, como puede ser el nombre de una tabla, una sentencia SQL o el nombre de un procedimiento almacenado, también se le puede asignar un objeto Command. En este ejemplo a esta propiedad se le ha asignado una cadena que representa un sentencia SQL, en este caso una SELECT.

En la propiedad ActiveConnection se le indica al Recordset la conexión a la que se encuentra asociado. A esta propiedad se le puede pasar un objeto Connection ya creado, o bien, una cadena de conexión, como ocurre en este ejemplo. Lo normal es asignarle un objeto Connection, ya que de esta forma se puede reutilizar el mismo objeto Connection para distintos objetos Recordset, esto mismo ocurría con la propiedad ActiveConnection del objeto Command, que tratábamos en el capítulo anterior.

Una vez definidas las propiedades del Recordset, nos disponemos a obtener los datos, para ello se utiliza el método Open. Al lanzar este método el Recordset pasará a contener los datos indicados por la propiedad Source.

En este caso el método Open se ha utilizado sin parámetros, ya que las características del objeto Recordset ya las hemos definido a través de sus propiedades.

La sintaxis general del método Open es la siguiente:

```
ObjetoRecordset.Open OrigenDatos, Conexion, TipoCursor,
TipoBloqueo, Opciones
```

Todos los parámetros son opcionales y se deberán especificar si no hemos dado algún valor a alguna de las propiedades del Recordset que representa cada uno de ellos. De esta forma, OrigenDatos es el valor que se le asignaría a la propiedad Source, Conexion representa el valor de la propiedad ActiveConnection, TipoCursor se corresponde con la propiedad CursorType, TipoBloqueo con la propiedad LockType y Opciones indica la naturaleza del origen de los datos, sus valores se pueden observar en la siguiente tabla.

A continuación se ofrecen varias formas que tenemos de obtener un objeto Recordset, en todos los casos se pretende que el contenido del objeto Recordset sea la tabla de autores (authors), y en todos los casos se va utilizar un cursor Forwar-only/Read-only.

Lo podemos obtener a partir de la ejecución directa sobre un objeto Connection.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objRecordSet=objConexion.Execute("Select * from authors",,adCmdText)%>
```

Código fuente 272

También a partir de la ejecución directa de un objeto Command.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection=objConexion
objComando.CommandText="devuelveAutores"
Set objRecordSet=objComando.Execute(,adCmdStoredProc)%>
```

Código fuente 273

De momento no hemos utilizado el método Open, pero en este nuevo ejemplo si que lo vamos a utilizar. Se trata de obtener el Recordset a partir del objeto Command, pero especificando nosotros mismos las propiedades de nuestro Recordset.

Como se puede observar en el parámetro del método Open correspondiente a la propiedad ActiveConnection del objeto Recordset no hemos indicado ninguna conexión, ya que se toma la conexión que se ha especificado para el objeto Command que utilizamos como origen de datos del objeto Recordset.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection=objConexion
objComando.CommandText="devuelveAutores"
Set objRecordSet=Server.CreateObject("ADODB.Recordset")
objRecordSet.Open objComando,,adOpenForwardOnly,adLockReadOnly,adCmdStoredProc%>
```

Código fuente 274

También podemos utilizar el método Open especificando la tabla.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.ActiveConnection=objConexion
objComando.CommandText="devuelveAutores"
Set objRecordSet=Server.CreateObject("ADODB.Recordset")
objRecordSet.Open
"authors", objConexion,adOpenForwardOnly,adLockReadOnly,adCmdTable%>
```

Código fuente 275

Y por último podemos tener un objeto Recordset completamente independiente, que posea su propia conexión y recupere la información de la tabla a través de una sentencia SQL.

```
<%strConexion="Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
strConsulta="select * from authors"
Set objRecordSet=Server.CreateObject("ADODB.Recordset")
objRecordSet.Open
strConsulta,strConexion,adOpenForwardOnly,adLockReadOnly,adCmdText%>
```

Código fuente 276

Al igual que abrimos un Recordset, debemos cerrarlo con el método Close, de esta forma liberaremos todos los recursos del sistema asociados a este objeto, pero no lo eliminaremos de la memoria, para ello deberemos asignarle al objeto Recordset el valor Nothing.

En teoría se cerramos el objeto Connection al que está asociado una serie de objetos Recordset se cerrarán todos estos objetos Recordset dependientes de forma automática, pero es mejor, para liberar memoria y recursos de forma más fiable e inmediata cerrar cada uno de los Recordset mediante estas dos sencillas líneas de código, que muestra el Código fuente 277.

```
<%objRecordset.Close
Set objRecordset=Nothing%>
```

Código fuente 277

## La colección Fields

En algunos de los ejemplos de apartados anteriores hemos ido recorriendo el objeto Recordset mediante el método MoveNext dentro de un bucle While, y en cada iteración del While hemos recuperado el valor de cada campo del registro actual, sin saberlo estábamos haciendo uso de la colección Fields del objeto Recordset. Veamos un ejemplo, en el Código fuente 278, que recupera los valores de algunos de los campos de cada registro y los muestra en una tabla de HTML.

```
<HTML>
<HEAD>
<TITLE>Ejemplo objeto Recordset</TITLE>
</HEAD>
<BODY>
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.CommandText="devuelveAutores"
objComando.ActiveConnection=objConexion
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open objComando,,adOpenForwardOnly,adLockReadOnly,adCmdStoredProc%
<table border="1" align="center">
<tr>
<th>Código autor</th>
<th>Nombre</th>
<th>Domicilio</th>
<th>Código Postal</th>
</tr>
<%while not objRecordset.EOF%>
<tr>
<td><%=objRecordset("au_id")%></td>
<td><%=objRecordset("au_lname")%><%=objRecordset("au_fname")%></td>
<td><%=objRecordset("address")%></td>
<td align="right"><%=objRecordset("zip")%></td>
</tr>
<%objRecordset.MoveNext
Wend
objRecordset.Close
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
</table>
</body>
</html>
```

Código fuente 278

En este nuevo ejemplo para recuperar el valor de cada campo del registro actual le hemos pasado directamente al objeto Recordset el nombre del campo como una cadena. En realidad estamos accediendo a su colección Fields que nos devuelve la propiedad Value de un objeto Field que se encuentra indexado a través del nombre del campo.

Por lo tanto el objeto Recordset tiene una colección por defecto denominada Fields, y cada uno de los elementos de esta colección, objetos Field, tienen una propiedad por defecto denominada Value, de todas formas más adelante en este mismo apartado veremos en detalle el objeto Field.

En la Figura 124 se muestra que lugar ocupa la colección Fields y el objeto Field junto con el objeto Recordset dentro de la jerarquía de objetos de ADO.

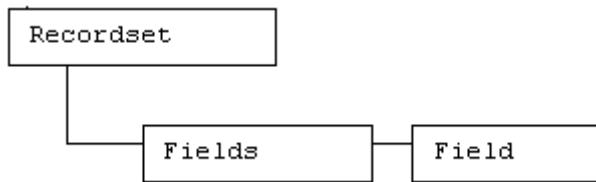


Figura 124. Jerarquía del objeto Recordset

Los objetos Field que posee la colección Fields representan cada uno de los campos del registro actual, pero no sólo permiten obtener el valor del campo correspondiente, sino que ofrece a través de sus propiedades información relativa a ese campo.

- ActualSize: tamaño real del valor del campo.
- Attributes: indica una o más características del campo, es una combinación de varias constantes de ADO, la descripción de estas constantes y sus valores se pueden observar más adelante en la tabla correspondiente.
- DataFormat: indica el formato de los datos del campo.
- DefinedSize: indica el tamaño definido del campo.
- Name: representa el nombre del campo.
- NumericScale: indica la escala de los valores numéricos.
- OriginalValue: indica el valor de un campo que se ha recuperado de un Recordset antes de realizar ningún cambio.
- Type: indica el tipo de dato del campo.
- Precision: indica el grado de precisión para los valores numéricos.
- UnderlyingValue: es el valor actual de un campo en la base de datos.
- Value: es el valor asignado a un campo. Es la propiedad por defecto del objeto Field.

El objeto Field también posee una colección Properties, al igual que ocurre con otros objetos de ADO, que recoge información específica ofrecida por el proveedor de datos.

Constante	Descripción
adFldMayDefer	Indica que el campo se aplaza, es decir, los valores del campo no se recuperan del origen de datos con todo el registro, sino solamente cuando se tiene acceso explícito a los mismos.
adFldUpdatable	Indica que se puede escribir en el campo.
adFldUnknownUpdatable	Indica que el proveedor no puede determinar si se puede escribir en el campo.
adFldFixed	Indica que el campo contiene datos de longitud fija.
adFldIsNullable	Indica que el campo acepta valores nulos.
adFldMayBeNull	Indica que se pueden leer valores nulos del campo.
adFldLong	Indica que se trata de un campo binario largo. También indica que se pueden utilizar los métodos AppendChunk y GetChunk.
adFldRowID	Indica que el campo contiene un identificador de fila persistente en el que no se puede escribir y que no tiene ningún valor significativo excepto la identificación de la fila (como por ejemplo un número de registro, un identificador único, etc.).
adFldRowVersion	Indica que el campo contiene algún tipo de marca de hora o de fecha que se utiliza para efectuar actualizaciones.
adFldCacheDeferred	Indica que el proveedor almacena los valores del campo en la memoria caché y que las lecturas siguientes se efectúan en dicha memoria .

Tabla 27

En el Código fuente 279 se puede ver como se recorre la colección Fields de un objeto Recordset, y como se recuperan los valores de distintas propiedades de cada objeto Field de la colección.

```

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open
"authors",objConexion,adOpenForwardOnly,adLockReadOnly,adCmdTable%>
<b>Información campos:</b><br>
<%For Each campo in objRecordSet.Fields%>
    Nombre: <i><%=campo.Name%></i><br>
    Valor: <i><%=campo.Value%></i><br>
    Tamaño actual: <i><%=campo.ActualSize%></i><br>
    Tamaño definido: <i><%=campo.DefinedSize%></i><br>
    Tipo: <i><%=campo.Type%></i><br>

```

```
<hr>
<%Next
objRecordset.Close
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
</BODY>
</HTML>
```

Código fuente 279

El resultado podría ser similar al siguiente:

### Información campos:

Nombre: *au\_id*  
 Valor: *172-32-1176*  
 Tamaño actual: *11*  
 Tamaño definido: *11*  
 Tipo: *200*

---

Nombre: *au\_lname*  
 Valor: *White*  
 Tamaño actual: *5*  
 Tamaño definido: *40*  
 Tipo: *200*

---

Nombre: *au\_fname*  
 Valor: *Johnson*  
 Tamaño actual: *7*  
 Tamaño definido: *20*  
 Tipo: *200*

---

Para obtener el valor de un objeto Field, es decir, el valor de un campo de un Recordset lo podemos hacer de diversas maneras, en el Código fuente 280 se ofrecen todas ellas (Item es una propiedad de la colección Fields que veremos más adelante).

```
contenido=objRst(0)
contenido=objRst(0).Value
contenido=objRst.Fields(0)
contenido=objRst.Fields(0).Value
contenido=objRst.Fields.Item(0)
contenido=objRst.Fields.Item(0).Value
contenido=objRst("nombreCampo")
contenido=objRst("nombreCampo").Value
contenido=objRst.Fields("nombreCampo")
contenido=objRst.Fields("nombreCampo").Value
contenido=objRst.Fields.Item("nombreCampo")
```

```
contenido=objRst.Fields.Item("nombreCampo").Value
```

Código fuente 280

Podemos utilizar cualquier forma, pero la más usual es la que muestra el Código fuente 281.

```
contenido=objcRst("nombreCampo")
```

Código fuente 281

Además de las propiedades vistas, el objeto Field presenta un par de métodos que podemos utilizar a la hora de manipular campos de tipo binario o de datos grandes, como puede ser por ejemplo un campo de tipo Text. Estos métodos son:

- AppendChunk: añade datos a un campo.
- GetChunk: obtiene el total o una porción de los datos de un campo.

De esta forma si queremos obtener un campo de tipo texto poco a poco, obtenemos el tamaño actual del campo y seleccionamos el tamaño de los fragmentos o porciones, una vez hecho esto se debe calcular el número de porciones y el resto de las mismas, todo esto se puede ver en el Código fuente 282, de utilización del método GetChunk:

```
<%'tamaño actual del campo a recuperar  
tam=objRecordset("nombreCampo").ActualSize  
'número de porciones  
porciones=tam/16348  
'resto del tamaño actual  
resto=tam Mod 16348  
'se recupera el resto  
texto=objRecordset("nombreCampo").getChunk(resto)  
'se van recuperando las porciones  
For cont=1 To cont<=porciones  
    texto=texto&objRecordset("nombreCampo").getChunk(16348)  
Next%>
```

Código fuente 282

Otro ejemplo de utilización de las propiedades del objeto Field es la recuperación de los nombres de los campos para mostrarlos en una tabla de HTML. Vamos a tomar Código fuente 282 y lo vamos a modificar para que el nombre de los campos se muestre de forma dinámica a partir de los valores de la propiedad Name de cada uno de los objetos Field.

```
<HTML>  
<HEAD>  
<TITLE>Ejemplo objeto Recordset</TITLE>  
</HEAD>  
<BODY>  
<%Set objConexion=Server.CreateObject("ADODB.Connection")  
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User  
id=sa"  
%>
```

```

Set objComando=Server.CreateObject("ADODB.Command")
objComando.CommandText="devuelveAutores"
objComando.ActiveConnection=objConexion
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open objComando,,adOpenForwardOnly,adLockReadOnly,adCmdStoredProc%
<table border="1" align="center">
<tr>
<%For i = 0 to objRecordset.Fields.Count - 1
    Response.Write "<TH>" & objRecordset.Fields(i).Name & "</TH>"
Next%>
</tr>
<%while not objRecordset.EOF%>
    <tr>
        <td><%=objRecordset("au_id")%></td>
        <td><%=objRecordset("au_lname")%><%=objRecordset("au_fname")%></td>
        <td><%=objRecordset("address")%></td>
        <td align="right"><%=objRecordset("zip")%></td>
    </tr>
    <%objRecordset.MoveNext
Wend
objRecordset.Close
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
</table>
</body>
</html>

```

Código fuente 283

Y si queremos hacer el proceso más dinámico todavía, podemos recuperar también los valores de los campos dentro de otro bucle For Each. El Código fuente 284 ilustra esta situación, en este caso se utiliza la propiedad Value del objeto Field.

```

<HTML>
<HEAD>
<TITLE>Ejemplo objeto Recordset</TITLE>
</HEAD>
<BODY>
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.CommandText="devuelveAutores"
objComando.ActiveConnection=objConexion
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open objComando,,adOpenForwardOnly,adLockReadOnly,adCmdStoredProc%
<table border="1" align="center">
<tr>
<%For i = 0 to objRecordset.Fields.Count - 1
    Response.Write "<TH>" & objRecordset.Fields(i).Name & "</TH>"
Next%>
</tr>
<%while not objRecordset.EOF
    Response.Write "<TR>" 
    For each campo in objRecordset.Fields
        Response.Write "<TD>" & campo.Value & "</TD>" 
    Next
    Response.Write "</TR>" 
    objRecordset.MoveNext
Wend
objRecordset.Close

```

```
Set objRecordset=Nothing  
objConexion.Close  
Set objConexion=Nothing%>  
</table>  
</body>  
</html>
```

Código fuente 284

En este nuevo ejemplo se puede ver las dos formas en las que podemos recorrer la colección Fields, con un bucle For a través de los índices de la colección, y con un bucle For Each para recuperar cada uno de los elementos de la colección.

En estos dos últimos ejemplos hemos utilizado una propiedad de la colección Fields, la propiedad Count, que devuelve el número de campos que existen en una colección Fields. Además de esta propiedad, la colección Fields ofrece la propiedad Item, que devuelve el objeto Field cuyo índice o nombre coincide con el parámetro que se le pase.

La colección Fields ofrece los siguientes métodos:

- Append: añade un campo a la colección.
- CancelUpdate: cancela cualquier cambio realizado sobre la colección.
- Delete: elimina un objeto Field de la colección.
- Refresh: actualiza los campos.
- Resync: sincroniza los datos de los campos de la colección con los de la base de datos.
- Update: guarda los cambios realizados en la colección.

Normalmente estos métodos no se utilizan, ya que se utilizan únicamente cuando se crea de forma manual un objeto Recordset, sin estar conectado a ningún origen de datos.

# Acceso a datos con ADO: Recordset II

---

## Recorriendo el Recordset

A continuación veremos como podemos movernos a través de un objeto Recordset y que métodos deberemos emplear. Se debe recordar, que para movernos en otra dirección que no sea hacia adelante, deberemos de crear el objeto Recordset utilizando Server.CreateObject, ya que, como se vio anteriormente, por defecto se crea un cursor Forward-Only.

Hasta el momento sólo nos hemos desplazado hacia adelante dentro de un Recordset utilizando el método MoveNext, es decir hemos tratado el Recordset como si fuera ForwardOnly.

Los métodos que vamos a comentar para desplazarnos a través de un objeto RecordSet son: MoveFirst, MoveLast, MoveNext, MovePrevious y Move.

Mediante los métodos MoveFirst y MoveLast nos moveremos al primer registro del RecordSet y al último registro, respectivamente. Existe una diferencia entre ambos métodos, si lanzamos el método MoveFirst sobre un objeto RecordSet el puntero de los registros se desplazará al primero de ellos, pero si lanzamos el método MoveLast el puntero no se desplazará al último de los registros, sino que lo sobrepasa, es decir, la propiedad EOF tendrá en este caso el valor True.

A través de los métodos MoveNext y MovePrevious avanzaremos un registro o retrocederemos un registro dentro del Recordset.

El método MoveNext, que nos permite ir avanzando registro a registro, es uno de los más utilizados y se suele utilizar casi siempre para recorrer un objeto RecordSet, anteriormente ya hemos visto diversos ejemplos en los que se puede apreciar del uso de este método. Se debe tener en cuenta al llamar a este

método que la propiedad EOF no tenga el valor True, es decir, que no se haya alcanzado el final del RecordSet, si EOF tiene el valor True y se lanza el método MoveNext se producirá un error.

Por otro lado tenemos el método MovePrevious para retroceder registro a registro dentro de un Recordset. En el caso de que la propiedad BOF sea igual a True, si llamamos al método MovePrevious se producirá un error.

El método Move permite movernos el número de registros que le indiquemos y también a partir de una posición determinada. Su sintaxis es la siguiente: Move numRegistros[,inicio] , con el parámetro numRegistros indicamos el número de registros que queremos desplazarnos y mediante el parámetro inicio indicamos el punto de partida, este parámetro se corresponde con un BookMark, más adelante veremos los BookMark (marcadores). En el Código fuente 285 se ofrece un ejemplo que muestra la utilización de este método.

```

1 <!--#INCLUDE VIRTUAL=/ADO/ADOVBS.INC-->
2 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
3 <HTML>
4 <HEAD>
5 <TITLE>Ejemplo método Move</TITLE>
6 </HEAD>
7 <BODY>
8 <%
9 Set objConexion=Server.CreateObject("ADODB.Connection")
10 objConexion.Open "DSN=FuenteBD;UID=pepe;PWD=xxx"
11 Set objRecordset=Server.CreateObject("ADODB.RecordSet")
12 objRecordset.Open "SELECT * FROM
Usuarios",objConexion,adOpenStatic,adLockReadOnly,
adCmdText
13 objRecordset.Move 3
14 %>
15 <table border="1" align="center">
16 <tr>
17 <th>DNI</th>
18 <th>Nombre</th>
19 <th>Domicilio</th>
20 <th>Código Postal</th>
21 </tr>
22 <%If not objRecordset.EOF Then%>
23     <tr>
24         <td><%=objRecordset("DNI")%></td>
25         <td><%=objRecordset("Nombre")%></td>
26         <td><%=objRecordset("Domicilio")%></td>
27         <td align="right"><%=objRecordset("Codigo_Postal")%></td>
28     </tr>
29 <%End If
30 objRecordset.close
31 Set objRecordset=Nothing
32 objConexion.close
33 Set objConexion=Nothing%>
34 </table>
35 </body>
36 </html>
```

Código fuente 285

La salida de esta página ASP sería la que muestra la Figura 125.

DNI	Nombre	Domicilio	Código Postal
18833451Ñ	Jaime Espina	c\Silva 12	28021

Figura 125. Nos hemos desplazado al 4º registro.

Si se observa la salida anterior con la de otros ejemplos, se puede comprobar que el registro que hemos recuperado en pantalla es el cuarto registro, esto es debido a que a el método Move le hemos pasado como parámetro el número 3, es decir, le hemos indicado que se desplazara tres registros dentro del Recordset, y como estamos situados en el primer registro al abrir el Recordset, al desplazarnos tres posiciones llegaremos hasta el cuarto registro.

Al final de este ejemplo, en las líneas 30-33, se puede apreciar que llamamos al método Close del objeto Recordset y del objeto Connection y luego asignamos Nothing. Es recomendable cerrar todos los Recordset una vez hayamos terminado de usarlos, de esta forma liberaremos recursos, y si además queremos eliminarlos de la memoria inmediatamente deberemos asignarles con Set el valor Nothing. Lo mismo ocurre con las conexiones a bases de datos, debemos cerrarlas y eliminarlas de la memoria.

Vamos a ver un nuevo ejemplo que va a aglutinar todos los métodos del objeto Recordset que se utilizan para desplazarnos dentro de él.

En ejemplo consta de un formulario con varias opciones, cada una de ellas representa el método de movimiento dentro del Recordset que se desea realizar. Al pulsar el botón de envío se realizará sobre el objeto Recordset el movimiento que se corresponde con la opción seleccionada.

El objeto Recordset lo hemos definido de tipo Static y de sólo lectura, ya que vamos a utilizar los registros para desplazarlos libremente y sólo vamos a leer los datos, no vamos a realizar ningún tipo de modificación.

Se debe señalar que por simplicidad se ha guardado el objeto Recordset en una variable se sesión, esta práctica no es muy recomendable, ya que esto supone una gran carga para el servidor, pero para este sencillo y didáctico ejemplo nos sirve.

Pasemos a ver el código completo en el Código fuente 286.

```
<HTML>
<BODY>
<%If Request.Form("Pulsado")="" Then
    'Es la primera vez que se accede a la página
    Set objConexion=Server.CreateObject("ADODB.Connection")
    objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;" & _
                    "Initial Catalog=pubs;User id=sa"
    Set objComando=Server.CreateObject("ADODB.Command")
    objComando.CommandText="devuelveAutores"
    objComando.ActiveConnection=objConexion
    Set objRecordset=Server.CreateObject("ADODB.Recordset")
    objRecordset.Open objComando,,adOpenStatic,adLockReadOnly,adCmdStoredProc
    Set Session("registros")=objRecordset
End if%>
<form method="post" action="recordset.asp">
<input type="radio" name="opcion" value="MoveNext" checked>MoveNext<br>
<input type="radio" name="opcion" value="MovePrevious">MovePrevious<br>
<input type="radio" name="opcion" value="Move">Move
<input type="text" name="posicion" size="2" value="">número de registros<br>
<input type="radio" name="opcion" value="MoveFirst">MoveFirst<br>
<input type="radio" name="opcion" value="MoveLast">MoveLast<br>
```

```

<input type="submit" name="pulsado" value="Enviar">
</form>
<%'se recupera el Recordset
Set objRecordset=Session("registros")
Select Case Request.Form("opcion")
    Case "MoveNext"
        If objRecordset.EOF Then
            Mensaje="No se puede realizar <b>MoveNext</b>, estamos después
"&_
                    "del último registro"
        Else
            objRecordset.MoveNext
            If objRecordset.EOF Then
                Mensaje="<b>MoveNext</b> realizado, estamos después "&_
                    "del último registro"
            Else
                Mensaje="<b>MoveNext</b> realizado"
            End if
        End if
    Case "MovePrevious"
        If objRecordset.BOF Then
            Mensaje="No se puede realizar <b>MovePrevious</b>, estamos "&_
                    "antes del primer registro"
        Else
            objRecordset.MovePrevious
            If objRecordset.BOF Then
                Mensaje="<b>MovePrevious</b> realizado, "&_
                    "estamos antes del primer registro"
            Else
                Mensaje="<b>MovePrevious</b> realizado"
            End if
        End if
    Case "Move"
        'On Error Resume Next
        objRecordset.Move Request.Form("posicion")
        If Err.number>0 Then
            Mensaje="No se ha realizado <b>Move</b>, se han producido "&_
                    "errores: "&Err.description
        ElseIf objRecordSet.EOF Then
            Mensaje="<b>Move</b> realizado, estamos después del último
registro"
        Elseif objRecordset.BOF Then
            Mensaje="<b>Move</b> realizado, estamos antes del primer
registro"
        Else
            Mensaje="<b>Move</b> realizado"
        End if
        On Error Goto 0
    Case "MoveLast"
        objRecordset.MoveLast
        Mensaje="<b>MoveLast</b> realizado"
    Case "MoveFirst"
        objRecordset.MoveFirst
        Mensaje="<b>MoveFirst</b> realizado"
End Select
Set Session("registros")=objRecordset%
<div align="center">
<font color="red"><%=Mensaje%></font>
</div>
<%If Not objRecordset.EOF AND Not objRecordset.BOF Then%>
    Número de registro actual: <b><%=objRecordset.AbsolutePosition%></b><br>
    Total registros: <b><%=objRecordset.RecordCount%></b>
    <table border="1" align="center">
        <tr>
            <%For i = 0 to objRecordset.Fields.Count - 1
                Response.Write "<TH>" & objRecordset.Fields(i).Name & "</TH>"
            Next%>

```

```

</tr>
<%Response.Write "<TR>" 
For each campo in objRecordset.Fields
    Response.Write "<TD>" & campo.Value & "</TD>" 
Next
Response.Write "</TR>"%>
</table>
<%End if%>
</body>
</html>

```

Código fuente 286

En la página se irá indicando si la operación se ha realizado con éxito y el registro actual en el que nos encontramos después de realizar la operación correspondiente. Además, para que quede bastante claro se indica la posición absoluta del registro actual dentro del Recordset, para ello se ha utilizado la propiedad AbsolutePosition del objeto Recordset, y para mostrar el número de registros totales se ha utilizado la propiedad RecordCount

Se ha utilizado un tipo de cursos Static, pero si hubiéramos utilizado un cursor KeySet el comportamiento dentro del ejemplo habría sido el mismo, pero si el tipo de cursor hubiera sido Dynamic, las propiedades RecordCount y AbsolutePosition habrían devuelto los valores -1, y con un cursor de tipo ForwardOnly hubiera sido imposible realizar este ejemplo, sólo se podrían utilizar los métodos MoveNext, MoveFirst y Move, pero este último siempre con valores positivos.

En la Figura 126 se puede ver un ejemplo de ejecución de la página ASP:

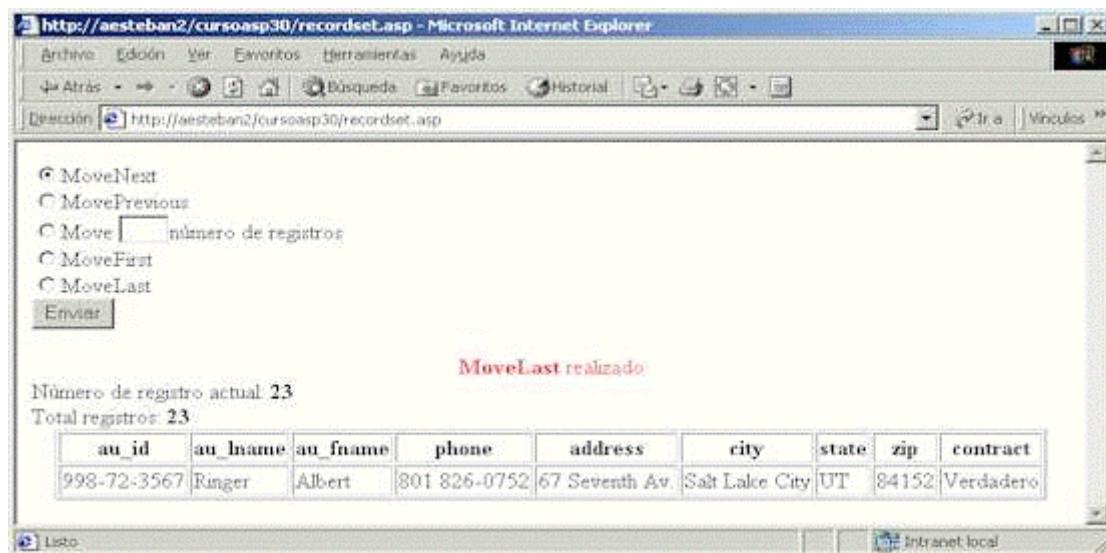


Figura 126. Desplazándonos dentro del Recordset

Como se puede comprobar en el código, antes de lanzar el método seleccionado se hace una comprobación de la situación actual para que no se produzcan errores, un caso especial es el método Move que se ha utilizado dentro de un tratamiento de errores con la sentencia On Error Resume Next.

Si ejecutamos el método Move con un número negativo el movimiento en el Recordset será hacia atrás, por lo tanto si especificamos -1 será igual a hacer un MovePrevious y se especificamos 1 será igual a ejecutar MoveNext. Si al método Move le pasamos un valor de número de registros a avanzar o retroceder (número positivo o negativo respectivamente), que sobrepasa el número de registros

actuales para realizar el desplazamiento, no se producirá un error sino que se situará en posición EOF o BOF según el tipo de movimiento que se haya realizado.

Para mostrar el nombre de los campos y sus valores se ha utilizado la forma más flexible, recorriendo la colección Fields del objeto Recordset.

Relacionado con el desplazamiento dentro de un Recordset tenemos también la propiedad BookMark. Esta propiedad permite recuperar una referencia a un registro determinado. Cada vez que se crea un nuevo objeto Recordset la propiedad BookMark se rellena de forma automática con datos de tipo Variant que permite identificar de forma única cada una de las filas del Recordset. Esto nos permite guardar una posición determinada del Recordset y luego volver a ella cuando se considere necesario.

La propiedad BookMark no se puede intercambiar entre distintos Recordset, cada objeto Recordset mantiene su propio BookMark.

En el Código fuente 287 se muestra la utilización de la propiedad BookMark. Nos desplazamos al tercer registro, obtenemos la referencia a esa fila mediante la propiedad BookMark, nos desplazamos en el Recordset, y luego asignándole a la propiedad BookMark del Recordset la referencia guardada anteriormente, volvemos al tercer registro.

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objComando=Server.CreateObject("ADODB.Command")
objComando.CommandText="devuelveAutores"
objComando.ActiveConnection=objConexion
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open objComando,,adOpenStatic,adLockReadOnly,adCmdStoredProc
objRecordset.Move 3
marca=objRecordset.Bookmark
Response.Write "1 (Move 3) - "&objRecordset("au_lname") &
"&objRecordset("au_fname") &"<br>
objRecordset.MoveLast
Response.Write "2 (MoveLast) - "&objRecordset("au_lname") &
"&objRecordset("au_fname") &"<br>
objRecordset.Bookmark=marca
Response.Write "3 (BookMark) - "&objRecordset("au_lname") &
"&objRecordset("au_fname") &"<br>
objRecordset.Close
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
</BODY>
</HTML>
```

Código fuente 287

En el ejemplo se ha utilizado un cursor de tipo Static, que los BookMark no los soportan todos los cursos, los cursos que soportan BookMark son Static y KeySet.

Después de tratar el desplazamiento dentro de un Recordset, vamos a pasar a comentar la modificación de los registros de un Recordset.

## Modificación del Recordset

Muchas veces necesitamos realizar operaciones de modificación, bajas y altas sobre una base de datos, para ello deberemos utilizar las sentencias SQL necesarias (UPDATE, DELETE e INSERT). De esta forma para insertar un nuevo registro dentro de una tabla de usuarios deberemos escribir lo siguiente:

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "DSN=FuenteBD;UID=pep;PWD=xxx"
objConexion.Execute "INSERT INTO Usuarios VALUES ('77777777','&
    ''Timoteo Smith','c/ Silvina','28032') "%>
```

Código fuente 288

En este caso la sentencia SQL se ha ejecutado directamente sobre la conexión. Pero en lugar de utilizar sentencias SQL o procedimientos almacenados de modificación, podemos utilizar también una serie de métodos que van a permitir la modificación del objeto Recordset, estos métodos son: Update, AddNew y Delete.

Aunque se recomienda utilizar sentencias SQL o procedimientos almacenados (ya que de esta forma es el origen de datos quien se encarga de los detalles de las operaciones), se va a comentar cada uno de estos métodos, más adelante veremos un ejemplo resumen de estos métodos de modificación de los datos de un objeto Recordset.

El método Update es utilizado en el proceso de modificación de un registro y en el de alta de un nuevo registro. En el primer caso lo utilizaremos con dos parámetros, el nombre del campo a modificar y su nuevo valor. Además de expresar el nombre del campo se puede expresar en un array varios campos y en el segundo parámetro un array con sus valores correspondientes. Por ejemplo, si deseamos modificar el campo Domicilio de la tabla Usuarios deberemos escribir lo que muestra el Código fuente 289.

```
<%objRecordSet.Update Domicilio, "Nuevo Domicilio" %>
```

Código fuente 289

También se puede utilizar el método Update sin parámetros, asignándole los nuevos valores directamente a los campos deseados, de esta forma, el Código fuente 289 se podría escribir como muestra el Código fuente 290.

```
<%objRecordSet("Domicilio")="Nuevo Domicilio"
objRecordSet.Update%>
```

Código fuente 290

Si queremos modificar varios campos del registro actual podemos utilizar el método Update con dos arrays, el de los campos a modificar, y el de los nuevos valores, veamos un ejemplo en el Código fuente 291.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open
"authors",objConexion,adOpenStatic,adLockOptimistic,adCmdTableDirect
objRecordset("au_lname")="au_fname","address")
valores=Array("Esteban","Angel","Silvina")
objRecordset.Update campos,valores
objRecordset.Close
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 291

Y también podemos asignar los valores a cada campo y luego lanzar el método Update.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open
"authors",objConexion,adOpenStatic,adLockOptimistic,adCmdTableDirect
objRecordset("au_lname")="Esteban"
objRecordset("au_fname")="Angel"
objRecordset("address")="Silvina"
objRecordset.Update
objRecordset.Close
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 292

Se debe tener en cuenta que para poder utilizar este método el cursor debe ser de un tipo compatible con esta operación, por lo tanto debe tener un bloqueo que no sea ReadOnly.

Si utilizamos la segunda forma que hemos visto para utilizar el método Update, es decir, modificamos los campos uno a uno y luego lanzamos el método Update, podemos cancelar estos cambios si lo deseamos llamando al método CancelUpdate en lugar de llamar al método Update, así si retomamos el Código fuente 292 tendremos el Código fuente 293.

De esta forma la modificación nunca se realizará sobre la base de datos.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User id=sa"
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open
"authors",objConexion,adOpenStatic,adLockOptimistic,adCmdTableDirect
objRecordset("au_lname")="XXXX"
objRecordset("au_fname")="XXXX"
objRecordset("address")="XXXX"
objRecordset.CancelUpdate
objRecordset.Close
```

```
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 293

Si probamos el Código fuente 294.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open
"authors",objConexion,adOpenStatic,adLockOptimistic,adCmdTableDirect
objRecordset("au_lname")="YYY"
objRecordset("au_fname")="HH"
objRecordset("address")="H"
objRecordset.MoveNext
objRecordset.Close
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 294

Veremos que la modificación se sigue realizando, aunque no hayamos lanzado el método Update, esto es debido a que al realizar un desplazamiento dentro del Recordset, si hay cambios pendientes se realizan. Si queremos modificar este comportamiento debemos definir el tipo de bloqueo del objeto Recordset como adLockBatchOptimistic.

Este tipo de bloqueo permite realizar varias actualizaciones como un proceso batch, es decir, las vamos realizando sobre el Recordset y en un momento dado podemos indicar que se realicen todas ellas sobre el origen de datos lanzando el método UpdateBatch.

Si no llamamos al método UpdateBatch, siendo el tipo de bloqueo adLockBatchOptimistic, las modificaciones sobre el origen de los datos no se realizarán.

Veamos un ejemplo de utilización del método UpdateBatch con un cursor con un tipo de bloqueo adLockBatchOptimistic, en este caso se realiza la modificación de dos registros de un Recordset.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open
"authors",objConexion,adOpenStatic,adLockBatchOptimistic,adCmdTableDirect
objRecordset("au_lname")="----"
objRecordset("au_fname")="++++"
objRecordset("address")="-----oo"
objRecordset.MoveNext
objRecordset("au_lname")="1111"
objRecordset("au_fname")="2222"
objRecordset("address")="33333"
objRecordset.UpdateBatch
objRecordset.Close
Set objRecordset=Nothing
```

```
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 295

También disponemos del método CancelBatch, con el que podremos cancelar una modificación en batch, es decir, tiene la misma misión que el método CancelUpdate. En el Código fuente 296 se puede ver un ejemplo de utilización del método CancelBatch.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
Set objRecordset=Server.CreateObject("ADODB.Recordset")
objRecordset.Open
"authors", objConexion, adOpenStatic, adLockBatchOptimistic, adCmdTableDirect
objRecordset.MoveFirst
objRecordset("au_lname") ="Esteban"
objRecordset("au_fname") ="Angel"
objRecordset("address") ="Silvina"
objRecordset.MoveNext
objRecordset("au_lname") ="Ruiz"
objRecordset("au_fname") ="Carmen"
objRecordset("address") ="Oligisto"
objRecordset.CancelBatch
objRecordset.Close
Set objRecordset=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 296

Si lo que queremos hacer es realizar una operación de alta, deberemos utilizar el método AddNew conjuntamente con el método Update, pero esta vez sin pasarle parámetros. Una vez que se ha lanzado el método AddNew sobre el objeto Recordset, podemos dar los valores al nuevo registro, asignándole a cada campo su valor correspondiente, a continuación se llama al método Update para que el nuevo registro se grabe en la base de datos. Este proceso se muestra en el Código fuente 297, dónde se abre la tabla de Usuarios y añade un usuario nuevo.

```
<HTML>

<HEAD>
<TITLE>Añade un usuario</TITLE>
</HEAD>

<BODY>
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
Set objRecordset=Server.CreateObject("ADODB.RecordSet")
objRecordSet.Open "Usuarios", objConexion, adOpenStatic, adLockOptimistic, adCmdTable
objRecordset.AddNew
objRecordset("DNI") ="223553915"
objRecordset("Nombre") ="Angel Esteban"
objRecordset("Domicilio") ="Silvina 7"
objRecordset("Codigo_Postal") ="28041"
objRecordset.Update
objRecordset.close
```

```

Set objRecordset=Nothing
objConexion.close
Set objConexion=Nothing%>
</body>
</html>

```

Código fuente 297

En este caso, al igual que con las modificaciones, el bloqueo no debe ser `ReadOnly`.

Si no lanzamos el método `Update` o `CancelUpdate`, para aceptar o cancelar el proceso de alta, al intentar cerrar el objeto `Recordset` se producirá un error. Sin embargo se realizamos un `MoveNext` el registro de dará de alta sin necesidad de llamar al método `Update`.

La forma de llamar al método `AddNew` presenta las mismas posibilidades que la llamada al método `Update` a la hora de realizar una modificación. Así por ejemplo el Código fuente 297 se podría haber escrito como nos indica el Código fuente 298.

```

<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
Set objRecordset=Server.CreateObject("ADODB.RecordSet")
objRecordSet.Open "Usuarios",objConexion,adOpenStatic,adLockOptimistic,adCmdTable
campos=Array("DNI","Nombre","Domicilio","Codigo_Postal")
valores=Array("2773893","Josep Trila","Periana 3","28045")
objRecordset.AddNew campos,valores
objRecordset.close
Set objRecordset=Nothing
objConexion.close
Set objConexion=Nothing%>

```

Código fuente 298

De esta forma el registro se añadirá de forma automática sin necesidad de llamar al método `Update` correspondiente.

Para realizar una operación de baja se utiliza el método `Delete`. Para borrar el registro deseado nos deberemos posicionar sobre él y a continuación lanzar el método `Delete` sobre el objeto `Recordset` correspondiente. Si queremos borrar los registros que satisfagan una condición, utilizaremos el método `Delete` con el parámetro `adAffectGroup`, esto borrará todos los registros que satisfagan la condición indicada en la propiedad `Filter` del `Recordset` correspondiente. La propiedad `Filter` la veremos en detalle en el próximo apartado.

Por defecto el valor que toma el método `Delete` es `adAffectCurrent`, esta constante indica que se debe eliminar el registro actual. Veamos el Código fuente 299 que utiliza este método.

```

<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
Set objRecordset=Server.CreateObject("ADODB.RecordSet")
objRecordSet.Open "Usuarios",objConexion,adOpenKeySet,adLockOptimistic,adCmdTable
objRecordset.MoveLast
objRecordset.Delete

```

```

objRecordset.close
Set objRecordset=Nothing
objConexion.close
Set objConexion=Nothing%>

```

Código fuente 299

En este caso se elimina el último registro.

Ahora vamos a tratar un ejemplo a modo de resumen que utiliza los métodos vistos para modificar objetos Recordset directamente.

En este ejemplo se permite añadir, eliminar, modificar y posicionarnos sobre los distintos registros de una tabla genérica, digo genérica porque se ha utilizado el código más general posible, sin utilizar en ningún caso un nombre o número de campos determinado.

En la parte superior de la página se muestra el contenido actual del Recordset, indicando el registro actual mediante el color amarillo. A continuación aparecen una serie de botones de navegación que nos permiten desplazarnos dentro del Recordset.

Para realizar un movimiento dentro del Recordset se utiliza siempre el método Move, al que le pasamos por parámetro la variable posición, esta variable la vamos manteniendo y actualizando a lo largo de toda la página con los valores correspondientes al botón de navegación que haya sido pulsado, así por ejemplo si pulsamos el botón de registro siguiente, se incrementará el valor de la variable posición. Más adelante, en el apartado dedicado a la paginación de un Recordset veremos otra forma de recorrer un objeto Recordset.

A continuación de los botones de navegación encontramos un formulario que permite editar (modificar) los valores del registro actual, eliminar el registro actual o dar de alta uno nuevo. Para dar de alta un nuevo registro antes debemos pulsar el botón de limpieza, para que queden los campos vacíos y poder añadir los valores del nuevo registro. Para modificar un registro simplemente nos posicionamos en él modificamos los valores que aparecen en el formulario y pulsamos modificar. Y para eliminar el registro actual pulsaremos el botón de borrado.

Para una mayor simplicidad del ejemplo se ha considerado que siempre debe existir al menos un registro en la tabla.

Veamos en el Código fuente 300, el código de este ejemplo.

```

<%'Se crea el Recordset
Set oRst = Server.CreateObject ("ADODB.Recordset")
oRst.Source = "Usuarios"
cadenaConex="Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
oRst.Open ,cadenaConex,adOpenStatic,adLockOptimistic,adCmdTable
'se recupera la posición
posición=Request.QueryString("pos")
if posición="" then posición=0
if Request.QueryString("movimiento") <> "" then
    'Nos desplazamos en el Recordset
    Select Case Request.QueryString("movimiento")
        Case " << "
            posición=0
        Case " < "
            posición=posición-1
        Case " > "

```

```

        posicion=posicion+1
    Case "    >>    "
        posicion=oRst.RecordCount-1
    End Select
elseif Request.QueryString("accion") <> "" then
    'Se modifica el Recordset
    oRst.MoveFirst
    oRst.Move posicion
    Select Case Request.QueryString("accion")
        Case "Modificar"
            For each elem in oRst.Fields
                elem.Value = Request.QueryString(elem.Name)
            Next
            oRst.Update
        Case "Añadir"
            oRst.AddNew
            For each elem in oRst.Fields
                elem.Value = Request.QueryString(elem.Name)
            Next
            oRst.Update
        Case "Eliminar"
            oRst.Delete
            If posicion<>0 Then posicion=posicion-1
    End Select
    'se vuelve a ejecutar la consulta sobre la BD
    oRst.Requery
End if%>
<HTML>
<BODY>
<TABLE border=1>
<TR>
<%oRst.MoveFirst
'se muestra el Recordset completo
For each elem in oRst.Fields
    Response.Write "<TH>" & elem.Name & "</TH>"
Next%>
</TR>
<%While not oRst.EOF
    'se marca el registro actual
    If Cint(posicion)=oRst.AbsolutePosition-1 Then
        Response.Write "<TR bgcolor='yellow'>"
    Else
        Response.Write "<TR>"
    End if
    For each elem in oRst.Fields
        Response.Write "<TD>" & elem & "</TD>"
    Next
    Response.Write "</TR>"
    oRst.MoveNext
Wend%>
</TABLE>
<%oRst.MoveFirst
oRst.Move posicion
'se crean los botones de navegación%
<FORM action="modificaciones.asp" method="GET" id=form1 name=form1>
<%if posicion>0 Then%>
    <INPUT type="submit" name="movimiento" value=" << " >
    <INPUT type="submit" name="movimiento" value=" < " >
<%End if%
If Not posicion=oRst.RecordCount-1 AND oRst.RecordCount>1 Then%>
    <INPUT type="submit" name="movimiento" value=" > " >
    <INPUT type="submit" name="movimiento" value=" >> " >
<%End if%>
<BR><BR>
<table border="0">
<%'se muestra el contenido del registro actual o campos vacíos si se
'ha pulsado limpiar

```

```

For each elem in oRst.Fields%
    <tr><td align="right">
        <%if Request.QueryString("accion")="Limpiar" then%>
            <b><%=elem.Name%></b></td><td><INPUT type="text"
name=<%=elem.Name%>" value="" maxlength=<%=elem.DefinedSize%>
size=<%=elem.DefinedSize%>">
        <%else%>
            <b><%=elem.Name%></b></td><td><INPUT type="text"
name=<%=elem.Name%>" value="<% Trim(elem.Value)%>" maxlength=<%=elem.DefinedSize%>
size=<%=elem.ActualSize%>">
        <%end if%>
    </td></tr>
<%next%>
</table>
<br>
<%'botones de modificación del Recordset%>
<INPUT type="submit" name="accion" value="Modificar">
<INPUT type="submit" name="accion" value="Añadir">
<%If oRst.RecordCount>1 Then%>
    <INPUT type="submit" name="accion" value="Eliminar">
<%End if%>
<INPUT type="submit" name="accion" value="Limpiar">
<INPUT type="hidden" name="pos" value=<%=posicion%>>
</FORM>
<%'se cierra y destruye el recordset y la conexión
oRst.Close
oRst.ActiveConnection.Close
Set oRst.ActiveConnection = Nothing
Set oRst = Nothing%>
</BODY>
</HTML>

```

Código fuente 300

The screenshot shows a Microsoft Internet Explorer window with the URL <http://aesteban2/cursoasp30/modificaciones.asp>. The page displays a table of address data with columns: DNI, Nombre, Domicilio, and Código\_Postal. A row for 'Maria Núñez' has been selected and highlighted with yellow background color.

DNI	Nombre	Domicilio	Código_Postal
111111	Angel Esteban	Silvina 3	28044
111112	Pepe Pin	Fatilo 12	28044
123445	Maria Núñez	Pan y Todos 12	28071
123445111	Carmen Ruiz	Obrigato 2	28022
4556	Ana Sánchez	Periana 11	28071

Below the table are navigation buttons: <<, <, >, >>. Underneath the table is a form with four input fields:

- DNI:
- Nombre:
- Domicilio:
- Código\_Postal:

At the bottom of the form are four buttons: Modificar, Añadir, Eliminar, and Limpiar. The status bar at the bottom of the browser window shows "Listo" and "Intranet local".

Figura 127. Modificación y movimiento en un Recordset

En este caso el Recordset tiene su propia conexión, esta conexión la cerramos y destruimos a partir de la propiedad ActiveConnection del objeto Recordset. Se ha utilizado un tipo de cursor Static con bloqueo optimista.

En el código primero se crea el Recordset, se recupera la posición y se verifica que acción quiere realizar el usuario. Esta acción puede ser de desplazamiento del Recordset o bien de modificación del mismo. Cada grupo de botones posee un nombre y un valor que especifican el tipo de operación a realizar.

Una vez realizada la operación correspondiente se muestra una tabla con los registros actuales del Recordset. El formulario que permite modificar el Recordset se construye de forma dinámica teniendo en cuenta las propiedades de los campos (ActualSize, Name, Value, etc.). En la Figura 127 se puede ver un ejemplo de ejecución del Código fuente 300.

Con este completo ejemplo se da por terminado el apartado dedicado a la modificación del objeto Recordset mediante los métodos que nos ofrece este objeto.

## Consultando el Recordset

En este apartado vamos a tratar una serie de propiedades y métodos que nos permiten manipular los datos del Recordset en lo que a consultas y recuperación de información se refiere.

Lo primero que vamos a comentar va a ser la propiedad Filter, esta propiedad la vamos a utilizar para filtrar la información que contiene el Recordset de acuerdo con el criterio especificado. Esta propiedad es interesante cuando se devuelven muchos datos en el Recordset y necesitamos obtener sólo parte de esta información, la información restante no se destruye, sigue en el Recordset pero no se encuentra visible ya que le hemos aplicado un filtro al objeto Recordset.

Una vez que hemos aplicado el filtro al Recordset, el resto de las filas es como si no existieran, sólo tendremos acceso a los registros que satisfagan el criterio especificado en la propiedad Filter. Para establecer un filtro debemos asignarle el criterio del filtro a la propiedad Filter, existen varias opciones, que pasamos a comentar a continuación:

- Asignándole una cadena que representa una sentencia SQL de tipo WHERE, también indicando un nombre campo con operadores lógicos (<, <=, >=, >>, LIKE), y valores como pueden ser fechas, cadenas, caracteres especiales, etc.
- Asignando un array de BookMarks, en este caso sólo se mostrarán los registros que haya sido marcados, es decir, los que correspondan a cada BookMark.
- Asignándole una constante de ADO para el filtrado. Las constantes se comentan en la Tabla 28.

Constante	Descripción
adFilterNone	Elimina el filtro actual y vuelve a poner todos los registros a la vista.
adFilterPending Records	Permite ver sólo los registros que han cambiado pero que no han sido enviados al servidor. Aplicable sólo para el modo de actualización por lotes.

adFilterAffected Records	Permite ver sólo los registros afectados por la última llamada a Delete, Resync, UpdateBatch o CancelBatch.
adFilterFetched Records	Permite ver los registros de la caché actual, es decir, los resultados de la última llamada para recuperar registros de la base de datos.
AdFilterConflictingRecords	Permite ver los registros que fallaron en el último intento de actualización por lotes.

Tabla 28

En el siguiente ejemplo se establece un filtro para obtener todos los nombres de autores que empiecen por la letra "A", veamos su código, en el Código fuente 301.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set oRst = Server.CreateObject("ADODB.Recordset")
cadenaConex="Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
oRst.Open "authors",cadenaConex,adOpenStatic,adLockOptimistic,adCmdTable
oRst.Filter="au_fname like 'a%'"%
<TABLE border="1">
<TR>
<%For each elem in oRst.Fields
    Response.Write "<TH>" & elem.Name & "</TH>"
Next%>
</TR>
<%While not oRst.EOF
    Response.Write "<TR>
        For each elem in oRst.Fields
            Response.Write "<TD>" & elem & "</TD>""
        Next
        Response.Write "</TR>
    oRst.MoveNext
Wend
oRst.ActiveConnection.Close
Set oRst.ActiveConnection=Nothing%>
</TABLE>
</BODY>
</HTML>
```

Código fuente 301

Un método relacionado con la consulta de datos en un Recordset es el método Find, este método se posicionará cada vez en un registro del Recordset que se corresponda con el criterio indicado. La sintaxis de este método es:

```
objRecordset.Find criterio, [saltaFilas], [direcciónBusqueda], [inicio]
```

El primer argumento es el criterio de búsqueda que se aplica para localizar el registro, se puede utilizar los mismos criterios que veíamos con la propiedad Filter, los siguientes argumentos que aparecen para Find son todos opcionales.

El segundo argumento es un valor booleano que va a indicar si deseamos o no utilizar la fila (registro) actual como parte de la búsqueda, por defecto tiene el valor false. La dirección de búsqueda permite indicar si la búsqueda en el Recordset va a ser hacia adelante (adSearchForward) o hacia detrás (adSearchBackward). Y en el último parámetro se puede especificar la posición de inicio de la búsqueda dentro del Recordset, este valor puede ser un BookMark.

Veamos un ejemplo de utilización de este método, se trata de realizar la misma operación que con la propiedad Filter del ejemplo anterior, es decir, mostrar los autores cuyo nombre comience por la letra "A". En este caso vamos a ejecutar el método Find en un bucle, ya que cada vez devolverá un registro coincidente, además le indicamos que el movimiento de la búsqueda sea hacia adelante (adSearchForward). Al no indicar en Find una posición de inicio se toma el registro actual.

De esta forma en el bucle se va lanzando el método Find que se irá situando en cada uno de los registros coincidentes con el criterio de búsqueda hasta llegar al final del objeto Recordset. En el Código fuente 302 se puede ver el código completo.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set oRst = Server.CreateObject("ADODB.Recordset")
cadenaConex="Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
oRst.Open "authors",cadenaConex,adOpenStatic,adLockOptimistic,adCmdTable%>
<TABLE border="1">
<TR>
<%For each elem in oRst.Fields
    Response.Write "<TH>" & elem.Name & "</TH>"
Next%>
</TR>
<%While not oRst.EOF
    oRst.Find "au_fname like 'a%'",,adSearchForward
    If Not oRst.EOF Then
        Response.Write "<TR>"
        For each elem in oRst.Fields
            Response.Write "<TD>" & elem & "</TD>"
        Next
        Response.Write "</TR>"
        oRst.MoveNext
    End if
Wend
oRst.ActiveConnection.Close
Set oRst.ActiveConnection=Nothing%
</TABLE>
</BODY>
</HTML>
```

Código fuente 302

Si ejecutamos este ejemplo y el anterior (el de la propiedad Filter), deberían ofrecer los mismos resultados. También podríamos haber realizado la búsqueda desde el final, como se puede ver en el Código fuente 303.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
```

```

</HEAD>
<BODY>
<%Set oRst = Server.CreateObject("ADODB.Recordset")
cadenaConex="Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
oRst.Open "authors", cadenaConex, adOpenStatic, adLockOptimistic, adCmdTable%>
<TABLE border="1">
<TR>
<%For each elem in oRst.Fields
    Response.Write "<TH>" & elem.Name & "</TH>"
Next%>
</TR>
<%oRst.MoveLast
While not oRst.BOF
    oRst.Find "au_fname like 'a%"', ,adSearchBackward
    If Not oRst.BOF Then
        Response.Write "<TR>"
        For each elem in oRst.Fields
            Response.Write "<TD>" & elem & "</TD>"
        Next
        Response.Write "</TR>"
        oRst.MovePrevious
    End if
Wend
oRst.ActiveConnection.Close
Set oRst.ActiveConnection=Nothing%>
</TABLE>
</BODY>
</HTML>

```

Código fuente 303

En este caso obtenemos la misma información pero en orden inverso.

Una propiedad relacionada con la forma en la que presenta la información un objeto Recordset es la propiedad Sort. Mediante esta propiedad podemos indicar la ordenación de los registros pertenecientes a un Recordset.

A la propiedad Sort se le asigna una cadena que se corresponde con el criterio de ordenación. Esta cadena puede estar formada por nombres de campos separados por comas, para indicar la ordenación. En esta cadena también se puede utilizar las palabras clave ASC o DESC, que indicarán que la ordenación debe ser ascendente o descendente respectivamente. Si no indicamos nada por defecto el orden es ascendente.

Mediante la propiedad Sort no cambiamos físicamente los datos, sino que simplemente permite acceder a los datos en un orden diferente. Para poder utilizar la propiedad Sort sin ningún problema se debe utilizar un Recordset de cliente, es decir, especificar en la propiedad CursorLocation del objeto Recordset la constante adUseClient.

Así por ejemplo si queremos mostrar el contenido de la tabla de autores ordenados por su nombre en orden descendente podemos utilizar el código que muestra el Código fuente 304.

```

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set oRst = Server.CreateObject("ADODB.Recordset")>

```

```

cadenaConex="Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
oRst.CursorLocation=adUseClient
oRst.Open "select * from authors", cadenaConex, adOpenStatic, adLockReadOnly, adCmdText
oRSt.Sort="au_fname DESC"
<TABLE border="1">
<TR>
<%For each elem in oRst.Fields
    Response.Write "<TH>" & elem.Name & "</TH>"
Next%>
</TR>
<%While not oRst.EOF
    Response.Write "<TR>"
    For each elem in oRst.Fields
        Response.Write "<TD>" & elem & "</TD>"
    Next
    Response.Write "</TR>"
    oRst.MoveNext
Wend
oRst.ActiveConnection.Close
Set oRst.ActiveConnection=Nothing%>
</TABLE>
</BODY>
</HTML>

```

Código fuente 304

Relacionados con la obtención de información contenida en un objeto Recordset tenemos los métodos GetRows y GetString.

El método GetRows nos permite obtener todo el contenido de un objeto Recordset dentro de un array bidimensional, siendo los campos del Recordset la primera dimensión, y el contenido de los registros la segunda dimensión. La sintaxis de este método es:

```
var=objRecorset.GetRows ( [numeroFilas] , [inicio] , [campos] )
```

El primer parámetro nos permite especificar el número de registros que deseamos obtener del Recordset, si no lo indicamos se devolverán todos los registros del Recordset, el parámetro de inicio será un BookMark que indicará a partir de que registro queremos recuperar la información, por defecto es a partir del registro actual, y el último parámetro permite indicar los campos que deseamos recuperar, puede ser una cadena con el nombre del campo a recuperar, su posición o bien un array de nombres de campos o de posiciones en el caso de que sean varios los campos a recuperar.

En el Código fuente 305 se va a utilizar el método GetRows para obtener el contenido de la tabla de autores y mostrarla por pantalla. En este caso vamos a mostrar el contenido de tres campos únicamente, por lo que al método GetRows le deberemos pasar el array con los nombre de los campos que queremos recuperar.

```

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set oRst = Server.CreateObject ("ADODB.Recordset")
cadenaConex="Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
oRst.CursorLocation=adUseClient
oRst.Open "select * from authors", cadenaConex, adOpenStatic, adLockReadOnly, adCmdText
oRSt.Sort="au_fname DESC"

```

```

campos=Array("au_fname","au_lname","au_id")
contenido=oRst.GetRows(, ,campos)
oRst.ActiveConnection.Close
Set oRst.ActiveConnection=Nothing%
<TABLE border="1">
<%For columna=0 to Ubound(contenido,2)
    Response.Write "<TR>"
    For fila=0 to UBound(contenido,1)
        Response.Write "<TD>" & contenido(fila,columna) & "</TD>"
    Next
    Response.Write "</TR>"
Next%>
</TABLE>
</BODY>
</HTML>

```

Código fuente 305

Como se puede ver no se ha indicado número de filas ni posición de inicio, de esta forma se recuperan todos los registros desde el principio. Además lo hemos utilizado junto con la propiedad Sort para establecer la ordenación de los registros.

Otro detalle de este ejemplo es que se ha cerrado el objeto Recordset (no directamente sino cerrando y destruyendo su conexión asociada), y podemos mostrar su contenido a través del array contenido, en el que tenemos todos los registros del Recordset.

El método GetString devuelve el contenido de un objeto Recordset en una cadena. La sintaxis general de este método es:

Vamos a comentar los parámetros de este método, que como se puede comprobar son todos opcionales. En el parámetro de formato de cadena sólo es posible especificar la constante adClipString, que es por lo tanto el valor por defecto. Podemos indicar en el siguiente parámetro el número de registros que se desean recuperar, por defecto son todos.

En el parámetro delimitadorColumnas se puede indicar que carácter se va a utilizar para separar los distintos campos de un registro (por defecto es el tabulador), y en delimitadorFilas se indica el carácter que delimitará cada uno de los registros (por defecto es el retorno de carro).

Y el último de los parámetros indica que expresión se va a utilizar en lugar de mostrar NULL para valores nulos de la base de datos, por defecto es una cadena vacía. El Código fuente 306 muestra la tabla de autores, pero esta vez utilizado el método GetString.

```

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set oRst = Server.CreateObject("ADODB.Recordset")
cadenaConex="Provider=SQLOLEDB;Data Source=aesteban2;Initial Catalog=pubs;User
id=sa"
oRst.Open "select * from authors",cadenaConex,adOpenStatic,adLockReadOnly,adCmdText
contenido=oRst.GetString(, , "</td><td>","</td></tr><tr><td>")
oRst.ActiveConnection.Close
Set oRst.ActiveConnection=Nothing%
<TABLE border="1">
<tr><td>
<%=contenido%>
</tr>

```

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

Código fuente 306

Como se puede ver en el código fuente del ejemplo como delimitadores de campos se ha utilizado la etiqueta HTML `<TD>` para indicar las columnas de una tabla, y como delimitador de filas se ha utilizado la etiqueta `<TR>` para indicar una nueva fila de la tabla. De esta forma simplemente tenemos que hacer un `Response.Write` de la cadena recuperada mediante `GetString` para que se muestre la información en el navegador de forma inteligible.

Este apartado que hemos dedicado a consultar el contenido del objeto Recordset termina aquí, dejando paso al siguiente apartado que muestra como realizar una navegación paginada sobre un Recordset.

## Paginación del Recordset

En algún momento podemos necesitar paginar un Recordset, es decir, ir mostrando su información poco a poco (a páginas), según el usuario lo vaya indicando. Una página en un Recordset es un conjunto lógico de registros.

Vamos a comentar un ejemplo que pagina un Recordset en páginas de 3 registros y que permite al usuario avanzar, retroceder, moverse al primer registro y al último.

En este ejemplo vamos a utilizar muchos de los conceptos vistos en lecciones anteriores, como puede ser el uso de formularios y el objeto integrado Request, utilización del objeto Session, etc, por lo tanto no profundizaremos en estos aspectos ya que se suponen conocidos por todos.

El aspecto que tendría la página ASP que queremos realizar sería el que muestra la Figura 128.

<i>Páginas: 4 Nº de Usuarios: 12 Página actual: 1</i>			
DNI	Nombre	Domicilio	Código Postal
11111111	Don Pepito	Nuevo Domicilio	28041
123345L	Pepito Grillo	c/Zapatero 9	28023
1234567	Pablo Esteban	c/Sahara 10	28041

>
>>

Figura 128. Paginación de un Recordset

Como se puede observar sólo se ofrecen los botones: desplazarnos al siguiente y al último de los registros, esto es debido a que estamos al principio del Recordset, es decir, es la página número 1.

El código que permite realizar la página ASP es el que se ofrece en el Código fuente 307.

```

1 <!--#include virtual="/ADO/ADOVBS.INC"-->
2 <html>
3 <head>
4 <title>Paginación de un Recordset</title>
5 </head>
6 <body>
7 <%
8 Dim objRst,i
9 If Request.Form("PaginaAnterior") <>"" Then
10    Set objRst = Session("objRst")
11    Session("pg") = Session("pg") -1
12    objRst.AbsolutePage = Session("pg")
13 ElseIf Request.Form("PaginaSiguiente")<>"" Then
14    Set objRst = Session("objRst")
15    Session("pg") = Session("pg") +1
16    objRst.AbsolutePage = Session("pg")
17 ElseIf Request.Form("UltimaPagina")<>"" Then
18    Set objRst = Session("objRst")
19    Session("pg") = objRst.PageCount
20    objRst.AbsolutePage = Session("pg")
21 ElseIf Request.Form("PrimeraPagina")<>"" Then
22    Set objRst = Session("objRst")
23    Session("pg") =1
24    objRst.AbsolutePage = Session("pg")
25 Else
26    Set objRst = Server.CreateObject("ADODB.Recordset")
27    objRst.Open "Select * from Usuarios", "DSN=FuenteBD;UID=pepe;PWD=xxx"_
28    ,adOpenStatic,adLockReadOnly,adCmdText
29    objRst.AbsolutePage = 1
30    Set Session("objRst") = objRst
31    Session("pg") = 1
32    Set objRst = Session("objRst")
33    objRst.PageSize = 3
34 End If
35 %>
36 <strong><em>
37 Páginas: <%=objRst.PageCount %>
38 N° de Usuarios: <%=objRst.RecordCount%>
39 Página actual: <%=Session("pg")%>
40 </em></strong>
41 <table border="1" cellpadding="5">
42 <tr>
43     <th>DNI</th>
44     <th>Nombre</th>
45     <th>Domicilio</th>
46     <th>Codigo Postal</th>
47 </tr>
48 <%For i=1 to objRst.Pagesize%>
49     <tr>
50         <td><%=objRst("DNI")%></td>
51         <td><%=objRst("Nombre")%></td>
52         <td><%=objRst("Domicilio")%></td>
53         <td><%=objRst("Codigo_Postal")%></td>
54     </tr>
55     <%objRst.MoveNext
56     if objRst.EOF Then Exit For
57 Next%>
58 </table>
59 <FORM ACTION=Paginacion.asp METHOD="POST">
60 <table border="0" cellpadding="5" cellspacing="3">
61 <tr>
62     <td>
63     <%
64     If Session("pg") >1 Then %>
65         <INPUT TYPE="Submit" Name="PrimeraPagina" Value=" << ">
66         <INPUT TYPE="Submit" Name="PaginaAnterior" Value=" < ">
```

```

67      <%
68      End If
69      If Session("pg") < objRst.PageCount Then %>
70          <INPUT TYPE="Submit" Name="PaginaSiguiente" Value=" " >
71          <INPUT TYPE="Submit" Name="UltimaPagina" Value=" " >> " "
72      <%End If%>
73      </td>
74  </tr>
75 </table>
76 </form>
77 </body>
78 </html>

```

Código fuente 307

Como se ha podido comprobar en el Código fuente 307, el objeto Recordset lo debemos crear nosotros mediante Server.CreateObject, ya que el cursor deber ser de tipo Static o Keyset para poder realizar la paginación de él sin ningún problema, si fuera un cursor Forward Only o Dynamic no sería posible paginarlo, ya que las propiedades utilizadas para la paginación no estarían disponibles en este tipo de cursosres.

Las propiedades utilizadas para poder realizar la paginación del Recordset son: PageSize, AbsolutePage, PageCount y RecordCount. Con la propiedad PageSize indicamos el tamaño de la página, en nuestro caso 3; mediante la propiedad AbsolutePage indicamos la página actual, es decir, es el valor que vamos incrementando y que guardamos dentro de la variable pg del objeto Session; la propiedad PageCount nos indica el número de páginas existentes y la propiedad RecordCount el número de registros que contiene el Recordset.

En las líneas 9-34 discernimos en que situación nos encontramos, es decir, si el usuario ha pulsado el botón de página anterior, o si ha pulsado el botón de página siguiente, o el botón de última página, o el botón de primera página o también si se ha cargado la página por primera vez. Según el caso que se de, se incrementará el contador de página actual o se decrementará, este contador se encuentra en la variable del objeto Session, a la que accedemos de la siguiente forma: Session("pg"), y si se ha pulsado para volver al principio del Recordset se asignará a este contador el valor 1 y si se ha pulsado para ir al final del Recordset se le asignará el valor de la propiedad PageCount, es decir, el número de páginas existentes.

En cualquier caso el valor de la variable Session("pg") se asignará a la propiedad AbsolutePage para hacer la página actual del Recordset la indicada por el usuario.

Una vez que nos hemos posicionado sobre la página correspondiente, deberemos mostrar los registros que contiene la página actual. Esto se puede observar en el bucle For de la línea 48, es aquí dónde interviene el valor asignado a la propiedad PageSize, en nuestro ejemplo mostraremos los registros de 3 en 3.

El objeto Recordset se guarda dentro de una variable del objeto Session, Session("objRst"), de esta forma siempre tendremos disponible el Recordset en la última posición a la que nos desplazamos. Aunque, como ya hemos comentado otras veces, está práctica no es recomendable.

En las líneas 59-76 se construye el formulario que muestra los botones que permiten al usuario navegar a través del Recordset. Según en la situación en que estemos: primera página, última página o página intermedia se crearán botones diferentes, así en la última página tendrán sentido los botones de página anterior y primera página. El aspecto de una página intermedia es el que muestra la Figura 129.

Como se ha podido observar, en ningún momento cerramos el objeto Recordset, ya que si lo cerramos en la siguiente página no podríamos acceder a sus registros. Otro detalle importante es que no se

utiliza ningún objeto Connection, a partir del objeto Recordset indicamos la cadena de conexión, por ello decimos que el Recordset es uno de los objetos de la jerarquía de ADO que es completamente independiente junto con los objetos Connection y Command, y también es uno de los más utilizados.

*Páginas: 4 N° de Usuarios: 12 Página actual: 3*

DNI	Nombre	Domicilio	Código Postal
18889945L	Jose Maria Pascual	c\Genova 21	28040
22339015	Pablo Esteban	c/Sahara 10	28041
333333	Jose Mari	c/Sal si puedes	28041



Figura 129. Página Intermedia

La paginación del Recordset la podemos realizar de una forma distinta, en la que se pueda cerrar el Recordset, esto implica no guardar el objeto Recordset en una variable de la sesión y abrirlo cada vez que se pide un movimiento a través de él. Deberemos guardar el contador de página actual y asignárselo a la propiedad AbsolutePage. Para ello deberemos modificar el Código fuente 307 que quedará como muestra el Código fuente 308.

```

1 <!--#include virtual="/ADO/ADOVBS.INC"-->
2 <html>
3 <head>
4 <title>Paginación2 de un Recordset</title>
5 </head>
6 <body>
7 <%
8 Set objRst = Server.CreateObject("ADODB.Recordset")
9 objRst.Open "Select * from Usuarios", "DSN=FuenteBD;UID=pepa;PWD=xxx"_
10 ,adOpenStatic,adLockReadOnly,adCmdText
11 objRst.PageSize = 3
12 If Request.Form("PaginaAnterior") <>"" Then
13     Session("pg") = Session("pg") -1
14     objRst.AbsolutePage = Session("pg")
15 ElseIf Request.Form("PaginaSiguiente")<>"" Then
16     Session("pg") = Session("pg") +1
17     objRst.AbsolutePage = Session("pg")
18 ElseIf Request.Form("UltimaPagina")<>"" Then
19     Session("pg") = objRst.PageCount
20     objRst.AbsolutePage = Session("pg")
21 ElseIf Request.Form("PrimeraPagina")<>"" Then
22     Session("pg") =1
23     objRst.AbsolutePage = Session("pg")
24 Else
25     objRst.AbsolutePage = 1
26     Session("pg") = 1
27 End If
28 %>
29 <strong><em>
30 Páginas: <%=objRst.PageCount %>
31 N° de Usuarios: <%=objRst.RecordCount%>
32 Página actual: <%=Session("pg")%>
```

```
33 </em></strong>
34 <table border="1" cellpadding="5">
35 <tr>
36     <th>DNI</th>
37     <th>Nombre</th>
38     <th>Domicilio</th>
39     <th>Codigo Postal</th>
40 </tr>
41 <%For i=1 to objRst.PageSize%>
42     <tr>
43         <td><%=objRst("DNI")%></td>
44         <td><%=objRst("Nombre")%></td>
45         <td><%=objRst("Domicilio")%></td>
46         <td><%=objRst("Codigo_Postal")%></td>
47     </tr>
48     <%objRst.MoveNext
49     if objRst.EOF Then Exit For
50 Next%>
51 </table>
52 <FORM ACTION=Paginacion.asp METHOD="POST">
53 <table border="0" cellpadding="5" cellspacing="3">
54 <tr>
55     <td>
56     <%
57     If Session("pg") >1 Then %
58         <INPUT TYPE="Submit" Name="PrimeraPagina" Value=" << ">
59         <INPUT TYPE="Submit" Name="PaginaAnterior" Value=" < ">
60     <%
61     End If
62     If Session("pg") < objRst.PageCount Then %
63         <INPUT TYPE="Submit" Name="PaginaSiguiente" Value=" > "gt;
64         <INPUT TYPE="Submit" Name="UltimaPagina" Value=" >> "gt;
65     <%End If%>
66     </td>
67 </tr>
68 </table>
69 <%objRst.close
70 Set objRst=Nothing%>
71 </form>
72 </body>
73 </html>
```

Código fuente 308

En esta nueva versión de la paginación, liberaremos recursos al cerrar el Recordset, pero deberemos abrir un Recordset nuevo cada vez que el usuario pulse un botón del formulario indicando el movimiento que desea realizar dentro del Recordset, siendo esta opción la más recomendable.



# ADO 2.5: Record

---

## Introducción

En este capítulo vamos a tratar el primero de los dos nuevos objetos que ofrece ADO 2.5, antes de nada se debe aclarar que desde ASP 3.0 todavía no se pueden utilizar estos dos objetos de forma satisfactoria.

Se debe señalar que para utilizar con éxito los objetos Record y Stream desde ASP 3.0 se debe incluir la cuenta del usuario anónimo de Internet (IUSR\_nombreMaquina) dentro del grupo de Administradores. Esto supone un grave problema de seguridad y puede ser válido para servidores Web de prueba o de desarrollo, pero para servidores Web en producción resulta inadmisible. Supongo que se trata de un error en la versión 3.0 de ASP o bien un error del proveedor OLEDB Internet Publishing, pero lamentablemente hasta la fecha no existe otra solución.

## ActiveX Data Objects 2.5 (ADO 2.5)

Junto con ASP 3.0 se ofrece la nueva versión de los componentes de servidor para el acceso a datos, es decir, ActiveX Data Objects 2.5 (ADO 2.5).

Lo más destacable es que ADO 2.5 amplía su modelo de objetos con dos objetos más: Record y Stream.

Hasta ahora mediante ADO accedíamos sobre todo a datos estructurados como puede ser un conjunto de registros de una tabla en una base de datos, pero también nos puede interesar acceder a datos que no

sean tan homogéneos como puede ser un sistema de ficheros o un sistema de correo, en este caso estaremos ante datos semi-estructurados.

Una característica nueva que forma parte de los datos semi-estructurados es lo que se denomina Internet Publishing. La versión 2.1 de ADO ya ofrecía el proveedor OLE DB para Internet Publishing, pero con una funcionalidad muy limitada. Ahora en su nueva versión, ADO ofrece la funcionalidad completa.

Mediante el proveedor OLE DB para Internet Publishing podemos manipular recursos Web desde ADO, es decir, podemos construir nuestras propias aplicaciones para manipular sitios Web. Todas estas innovaciones entran dentro de la estrategia de acceso a datos de Microsoft, UDA (Universal Data Access).

Veamos ahora como se modela en objetos este nuevo acceso a datos semi-estructurados por parte de ADO.

Normalmente el almacenamiento semi-estructurado sigue una organización de árbol, con nodos, subnodos y archivos. Si consideramos un sitio Web vemos que presenta carpetas, subcarpetas y archivos. Si pensamos que este sistema de almacenamiento debe ser modelado mediante ADO, podemos elegir como candidato el objeto RecordSet, ya que puede contener un conjunto de datos.

Pero si lo pensamos más detenidamente vemos que un objeto RecordSet puede contener un conjunto de carpetas, pero luego cada una de las carpetas tendrá distintos archivos de distintos tipos cada uno con unas características.

Es en este momento cuando entra en juego el objeto Record. En la situación vista anteriormente, la colección de directorios y/o archivos será representada por un objeto Recordset pero cada directorio o archivo será representado mediante un objeto Record, ya que pueden tener propiedades únicas y distintas. Como se puede ver este es un nuevo concepto que puede ser necesario madurar por parte del lector.

Los valores de las propiedades que contiene el objeto Record referentes al archivo o directorio que representa, se pueden recuperar a modo de campos del objeto Record. Además, el objeto Record ofrece una serie de métodos que permiten manipular el registro: CopyRecord, MoveRecord y DeleteRecord.

Una vez visto la necesidad del objeto Record, para describir y representar las características únicas y variadas de cada elemento, es sencillo ver la utilidad del nuevo objeto Stream.

El objeto Stream permite acceder a los contenidos de cada elemento, que puede ser un directorio, un archivo, un mensaje de correo, etc. Para ello este objeto posee una serie de métodos como pueden ser: ReadText, WriteText, LoadFromFile, etc.

La inclusión de estos dos nuevos objetos se basa en la filosofía de UDA (Universal Data Access), es decir, no nos importa el origen de los datos, lo podremos manipular de la misma manera, ya sea una URL, un sistema de ficheros, una base de datos, un sistema de correo, etc.

Ahora veremos en más detalle estos dos objetos.

## El objeto Record

Ya hemos comentado que este nuevo objeto de ADO va a representar un fichero, carpeta o mensaje de correo, es decir, un recurso determinado. Permite a los desarrolladores acceder y manejar datos organizados de forma jerárquica dentro de sistemas de ficheros o sistemas de correo.

Mediante el objeto Record podremos renombrar, eliminar, mover o copiar ficheros o carpetas. Las conexiones para utilizar el objeto Record se hacen mediante el proveedor OLE DB para la publicación en Internet (OLE DB Provider for Internet Publishing).

A continuación vamos a comentar brevemente las propiedades que ofrece este objeto:

- *ActiveConnection*: indica la conexión que se utiliza para obtener los datos, puede ser un objeto Connection o una cadena de conexión, tiene el mismo significado que la propiedad de mismo nombre del objeto Recordset.
- *Mode*: indica el modo en el que se ha abierto la conexión y que permisos se tendrán sobre el objeto, es decir, lectura, modificación, etc.
- *ParentURL*: cadena que contiene la URL padre del objeto Record.
- *RecordType*: constante que indica si el objeto Record representa un documento estructurado, un objeto simple o una colección de objetos que contiene hijos.
- *Source*: indica el origen de los datos, puede ser una URL o una referencia a un objeto Recordset abierto.
- *State*: indica el estado del objeto Record, es decir, si se encuentra abierto o cerrado.

Los métodos del objeto Record son:

- *Cancel*: cancela la ejecución de una operación de apertura o ejecución asíncronas.
- *Close*: cierra el objeto Record.
- *CopyRecord*: copia el objeto que representa el objeto Record (fichero o directorio) de una localización a otra.
- *DeleteRecord*: elimina el elemento que representa el objeto Record.
- *GetChildren*: devuelve un objeto Recordset que contiene todos los ficheros y directorios que se encuentran en el directorio representado por el objeto Record.
- *MoveRecord*: mueve el elemento representado por el objeto Record de una localización a otra.
- *Open*: crea un nuevo fichero o directorio o abre uno existente.

Además el objeto Record posee las siguientes colecciones:

- *Fields*: colección de objetos Field que contienen información descriptiva del objeto Record.
- *Properties*: colección de objetos Property presente en muchos de los objetos de ADO.

## Creación y apertura del objeto Record

Para instanciar un objeto Record utilizaremos la sentencia que muestra el Código fuente 309.

```
Set objRecord=Server.CreateObject ("ADODB.Record")
```

Código fuente 309

Y para abrir un objeto Record utilizaremos su método Open que ofrece la siguiente sintaxis general:

```
objRecord.Open Fuente, ConexiónActiva, Modo, OpcionesCreación,  
Opciones, Usuario, Contraseña
```

Muchos de los parámetros son similares a los que aparecen en el método Open del objeto Recordset, y además aquí ocurre lo mismo, son todos los parámetros opcionales, y se corresponden con propiedades del objeto. Veamos el significado de todos estos parámetros.

El parámetro fuente se corresponde con la propiedad Source e indica la fuente (origen) de los datos, puede ser el nombre de un fichero o directorio, también puede ser un objeto Recordset abierto. La conexión activa se corresponde con la propiedad ActiveConnection y puede ser una cadena de conexión o un objeto Connection que indica al objeto Record de dónde puede encontrar la fuente de los datos.

El argumento modo, que se corresponde con la propiedad Mode del objeto Record, indica de qué forma se va a abrir el objeto Record y qué operaciones se pueden realizar con él, en la Tabla 29 se muestran y describen las distintas constantes de ADO que podemos utilizar para especificar el modo. El modo puede ser una combinación de varios valores, según los permisos que deseemos aplicar al objeto Record. Como se puede comprobar son las mismas constantes que las que se utilizan para el modo del objeto Connection.

Constante	Descripción
adModeUnknown	Es el modo por defecto e indica que los permisos no se han establecido
adModeRead	Permisos de sólo lectura
adModeWrite	Permisos de sólo escritura
adModeReadWrite	Permisos de lectura y escritura
adModeShareDenyRead	Evita que otros usuarios puedan abrir una conexión con permisos de lectura
adModeShareDenyWrite	Evita que otros usuarios puedan abrir una conexión con permisos de escritura
adModeShareExclusive	Evita que otros establezcan una conexión con este objeto Record
adModeShareDenyNone	Evita que otros establezcan conexiones con cualquier permiso

adModeRecursive	Se utiliza conjuntamente con constantes ShareDeny para propagar restricciones a todos los sub-records del actual Record
-----------------	---

Tabla 29

Las opciones de creación indica si el fichero o directorio debe ser abierto o creado, al igual que para el argumento anterior se ofrece una tabla con la descripción de las constantes válidas de ADO para este parámetro tenemos la Tabla 30.

Constante	Descripción
AdCreateOverwrite	Sobreescribe el Record (fichero o carpeta) existente en la URL especificada
AdOpenIfExists	Abre un Record existente, sino existe lo crea.
AdCreateCollection	Crea un directorio en la URL especificada
adCreateNonCollection	Crea un fichero en la URL especificada
AdFailIfNotExists	Se produce un error si el recurso indicado en la URL no existe
AdCreateStructDoc	Crea un nuevo documento estructurado en la URL especificada

Tabla 30

El siguiente parámetro, opciones, indica en que modo se abre el objeto Record, si se abre de forma síncrona o asíncrona, a partir de una URL etc, a continuación se ofrece una tabla con las constantes existentes para este parámetro. En el entorno de ASP sólo va a ser posible abrir objetos Record de manera síncrona.

Constante	Descripción
adOpenAsync	Abre el objeto Record de forma asíncrona
adOpenSource	Abre el documento indicado en la URL, en lugar de los contenidos ejecutados
adOpemURLBind	Indica que la cadena de conexión contiene una URL

Tabla 31

Los parámetros usuario y contraseña indica el usuario que va a abrir el Record correspondiente, y poseerá por lo tanto unos permisos determinados.

Se debe tener en cuenta que para utilizar los nuevos objetos de ADO, el servidor Web al que nos vamos a conectar debe ser un servidor que soporte la nueva tecnología de Microsoft DAV (Distributed Authoring and Versioning). El servidor IIS 5 se puede decir que es un servidor WebDAV. DAV es un

especificación que extiende el protocolo HTTP 1.1 para incluir comandos que permiten gestionar ficheros en el servidor Web.

Sobre WebDAV se construye el proveedor OLEDB para publicación en Internet (Internet Publishing), para poder utilizar los nuevos comandos a través de los nuevos objetos de ADO 2.5, es decir, los objetos Record y Stream.

A continuación vamos a ver distintas formas de abrir un objeto Record.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "URL=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject("ADODB.Record")
objRecord.Open ,objConexion,adModeRead,adFailIfExists,adOpenSource
objRecord.Close
Set objRecord=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 310

En el Código fuente 310 se abre un objeto Record a partir de un objeto Connection existente. Como se puede apreciar la cadena de conexión no indica ningún proveedor OLEDB sino que utiliza la cadena URL. Al utilizar URL en la cadena de conexión se toma de forma automática el proveedor OLEDB para publicación en Internet. También podemos especificar de forma explícita como se ve en el Código fuente 311.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "Provider=MSDAIPP.DSO;Data Source=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject("ADODB.Record")
objRecord.Open ,objConexion,adModeRead,adFailIfExists,adOpenSource
objRecord.Close
Set objRecord=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 311

O también es válido el Código fuente 312.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Provider="MSDAIPP.DSO"
objConexion.Open "Data Source=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject("ADODB.Record")
objRecord.Open ,objConexion,adModeRead,adFailIfExists,adOpenSource
objRecord.Close
Set objRecord=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 312

Para abrir directorios puede ser necesario configurar el sitio Web desde las herramientas de administración de IIS 5 para que permita la inspección de directorios.

En los casos anteriores se ha abierto un directorio, ya que ni en el método Open ni en la cadena de conexión especificábamos un fichero, sino un directorio, en este caso el directorio es cursoASP30. Si hubiéramos querido abrir un fichero podríamos utilizar el Código fuente 313.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "URL=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject("ADODB.Record")
objRecord.Open "L1.jpg",objConexion,adModeRead,adFailIfNotExists,adOpenSource
objRecord.Close
Set objRecord=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 313

En este caso si el fichero no existe fallará la ejecución del método Open, más adelante veremos como crear ficheros y directorios mediante el método Open. Un código equivalente al anterior, pero que utiliza las propiedades del objeto Record es el que muestra Código fuente 314.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Open "URL=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject("ADODB.Record")
objRecord.ActiveConnection=objConexion
objRecord.Mode=adModeRead
objRecord.Source="L1.jpg"
objRecord.Open ,,,adFailIfNotExists,adOpenSource
objRecord.Close
Set objRecord=Nothing
objConexion.Close
Set objConexion=Nothing%>
```

Código fuente 314

Hasta ahora hemos abierto un objeto Record a partir de un objeto Connection existente, pero también podemos crear un objeto Record completamente independiente, estableciendo la cadena de conexión a su propiedad ActiveConnection o utilizándola en el parámetro correspondiente del método Open. En el Código fuente 315 se puede ver esta última alternativa.

```
<%Set objRecord=Server.CreateObject("ADODB.Record")
strConex="URL=http://aesteban2/cursoasp30"
objRecord.Open "L1.jpg",strConex,adModeRead,adFailIfNotExists,adOpenSource
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 315

El fichero que utilizamos para la propiedad source (o parámetro correspondiente del método Open) en el ejemplo anterior, podemos incluirlo también dentro de la cadena de conexión, como se puede comprobar en el Código fuente 316.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
strConex="URL=http://aesteban2/cursoasp30/L1.jpg"
objRecord.Open ,strConex,adModeRead,adFailIfExists,adOpenSource
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 316

Siempre que se utiliza la propiedad Source toma como URL padre la especificada dentro de la cadena de conexión.

Mediante el método Open también podemos crear nuevos elementos (ficheros o directorios) utilizando los parámetros adecuados. También se debe tener en cuenta que debemos tener permisos de lectura/escritura en la zona del servidor Web en la que vamos a crear los ficheros o directorios, para realizar esta comprobación podemos consultar las propiedades del Web o directorio correspondiente.

En el Código fuente 317 se muestra como crear un fichero en una URL determinada.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
strConex="URL=http://aesteban2/cursoasp30/L4.jpg"
objRecord.Open ,strConex,adModeReadWrite,adCreateNonCollection
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 317

Como se puede comprobar el fichero que se desea crear se llama L1.jpg, y se ha incluido dentro de la cadena de conexión. El modo del objeto Record se ha establecido de lectura/escritura (adModeReadWrite) para poder crear el nuevo fichero y además se ha indicado el tipo de Record para indicar que no es una colección (adCreateNonCollection), es un único fichero.

Para crear un directorio el proceso es similar pero el tipo de objeto Record que se va a crear en este caso es una colección (adCreateCollection), ya que podrá contener otros directorios y/o ficheros. En este caso se especifica el directorio a crear en la propiedad Source del objeto Record, y se creará por lo tanto a partir de la URL de conexión.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
strConex="URL=http://aesteban2/cursoasp30"
objRecord.Source="NuevoDir"
objRecord.Open ,strConex,adModeReadWrite,adCreateCollection
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 318

Para terminar este apartado vamos a ver una forma más para abrir un objeto Record, esta última posibilidad que consiste en abrir un objeto Record a partir de un objeto Recordset.

En este ejemplo se utiliza un objeto Connection para obtener un objeto Recordset que va a contener múltiples filas, una por cada fichero o directorio en la URL utilizada para establecer la conexión. Después este objeto Recordset, una vez abierto, se establece como origen de datos del objeto Record. Todo esto se puede observar en el Código fuente 319.

```
<%Set objConex=Server.CreateObject ("ADODB.Connection")
Set objRecordset=Server.CreateObject ("ADODB.Recordset")
Set objRecord=Server.CreateObject ("ADODB.Record")
strConex="URL=http://aesteban2/cursoasp30"
objConex.Open strConex
objRecordset.Open "/",objConex
Set objRecord.Source=objRecordset
objRecord.Open
objRecord.Close
Set objRecord=Nothing
objRecordset.Close
Set objRecordset=Nothing
objConex.Close
Set objConex=Nothing%>
```

Código fuente 319

## La colección Fields

La colección Fields del objeto Record contiene objetos Field indicando cada uno de ellos información descriptiva del objeto Record al que pertenecen, no se debe olvidar que cada objeto Record puede presentar unas propiedades distintas, ya

Se va a ofrecer un código ASP que muestra los campos que posee un objeto Record que representa a un directorio de un sitio Web y otro que representa a un fichero, junto con sus valores correspondientes. Mediante el objeto Record vamos a abrir en primer lugar una URL y mostrar los campos que posee con sus valores y a continuación se hace lo mismo con un fichero.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
objRecord.Open " ", "URL=http://aesteban/cursoASP30"%>
<html>
<body>
<div align="center"><b>Campos de un directorio</b></div>
<table align="center" border="2" cellspacing="2" cellpadding="5">
<tr>
<th>Campo</th>
<th>Valor</th>
<%For Each campo in objRecord.Fields%>
<tr>
<td><%=campo.Name%></td>
<td><%=campo.Value%></td>
</tr>
<%Next%>
</table>
<%objRecord.Close
objRecord.Open "L1.jpg", "URL=http://aesteban2/cursoASP30", adModeRead,
adFailIfNotExists, adOpenURLBind%>
<div align="center"><b>Campos de un fichero</b></div>
```

```


| Campo           | Valor            |
|-----------------|------------------|
| <%=campo.Name%> | <%=campo.Value%> |


<%objRecord.Close
Set objRecord=Nothing%>
</body>
</html>

```

Código fuente 320

El resultado de la ejecución del código anterior se puede ver en la Tabla 32 y en la Tabla 33 .

Campo	Valor
RESOURCE_PARSENAMESPACE	cursoASP30
RESOURCE_PARENTNAME	http://aesteban2
RESOURCE_ABSOLUTEPARSENAME	http://aesteban2/cursoASP30
RESOURCE_ISHIDDEN	False
RESOURCE_ISREADONLY	
RESOURCE_CONTENTTYPE	
RESOURCE_CONTENTCLASS	application/octet-stream
RESOURCE_CONTENTLANGUAGE	
RESOURCE_CREATIONTIME	20/07/2000 11:37:38
RESOURCE_LASTACCESSTIME	
RESOURCE_LASTWRITETIME	08/08/2000 10:15:33
RESOURCE_STREAMSIZE	0
RESOURCE_ISCOLLECTION	True
RESOURCE_ISSTRUCTUREDDOCUMENT	
DEFAULT_DOCUMENT	
RESOURCE_DISPLAYNAME	cursoASP30

RESOURCE_ISROOT	
RESOURCE_ISMARKEDFOROFFLINE	False
DAV:getcontentlength	0
DAV:creationdate	20/07/2000 11:37:38
DAV:displayname	cursoASP30
DAV:getetag	"309b6e96211c01:26d8"
DAV:getlastmodified	08/08/2000 10:15:33
DAV:ishidden	False
DAV:iscollection	True
DAV:getcontenttype	application/octet-stream

Tabla 32. Campos de un directorio

Campo	Valor
RESOURCE_PARSENAMESPACE	L1.jpg
RESOURCE_PARENTNAME	http://aesteban2/cursoASP30
RESOURCE_ABSOLUTEPARSENAME	http://aesteban2/cursoASP30/L1.jpg
RESOURCE_ISHIDDEN	False
RESOURCE_ISREADONLY	
RESOURCE_CONTENTTYPE	
RESOURCE_CONTENTCLASS	image/jpeg
RESOURCE_CONTENTLANGUAGE	
RESOURCE_CREATONTIME	08/08/2000 8:37:39
RESOURCE_LASTACCESSTIME	
RESOURCE_LASTWITETIME	08/08/2000 8:37:39
RESOURCE_STREAMSIZE	154692
RESOURCE_ISCOLLECTION	False
RESOURCE_ISSTRUCTUREDDOCUMENT	

DEFAULT_DOCUMENT	
RESOURCE_DISPLAYNAME	L1.jpg
RESOURCE_ISROOT	
RESOURCE_ISMARKEDFOROFFLINE	False
DAV:getcontentlength	154692
DAV:creationdate	08/08/2000 8:37:39
DAV:displayname	L1.jpg
DAV:getetag	"506a43e9131c01:26d8"
DAV:getlastmodified	08/08/2000 8:37:39
DAV:ishidden	False
DAV:iscollection	False
DAV:getcontenttype	image/jpeg

Tabla 33. Campos de un fichero

Como se puede comprobar hay dos tipos de campos, los que comienzan por RESOURCE y los que comienzan por DAV, hay algunos campos que son comunes para ambos tipos. Los campos RESOURCE son ofrecidos por el proveedor OLEDB para publicación en Internet, y los campos DAV son ofrecidos por el servidor IIS 5, que como ya hemos dicho es un servidor WebDAV. Vamos a comentar brevemente el significado de los campos RESOURCE.

- RESOURCE\_PARSENAMESPACE: la URL del recurso.
- RESOURCE\_PARENTNAME: la URL absoluta del padre del recurso.
- RESOURCE\_ABSOLUTEPARSENAME: la URL absoluta del recurso, es la suma de los dos campos anteriores.
- RESOURCE\_ISHIDDEN: indica si el recurso es un recurso oculto o no, por ejemplo se puede tratar de un fichero oculto.
- RESOURCE\_ISREADONLY: indica si el recurso es de sólo lectura o modificable.
- RESOURCE\_CONTENTTYPE: contiene el tipo MIME del recurso, como puede ser text/html o text/plain.
- RESOURCE\_CONTENTCLASS: indica la plantilla a partir de la que se ha creado el recurso.
- RESOURCE\_CONTENTLANGUAGE: el lenguaje en el que se encuentra almacenado el contenido del recurso.
- RESOURCE\_CREATONTIME: la fecha y hora de creación del recurso.

- RESOURCE\_LASTACCESSTIME: la fecha y hora de la última vez que se accedió al recurso.
- RESOURCE\_LASTWRITETIME: la fecha y hora de la última vez que se escribió en el recurso.
- RESOURCE\_STREAMSIZE: tamaño en bytes del recurso.
- RESOURCE\_ISCOLLECTION: contendrá el valor verdadero si el recurso es una colección (por ejemplo un directorio) y no un documento simple o estructurado.
- RESOURCE\_ISSTRUCTUREDDOCUMENT: tendrá el valor verdadero si el recurso es un documento estructurado y no una colección o un documento simple.
- DEFAULT\_DOCUMENT: URL del documento por defecto de un directorio.
- RESOURCE\_DISPLAYNAME: nombre que se muestra del recurso.
- RESOURCE\_ISROOT: verdadero si el recurso es la raíz de una colección o documento estructurado.

Especial interés tiene el campo RESOURCE\_ISCOLLECTION, consultando este campo sabremos si el recurso es un directorio que contiene hijos, estos hijos puede ser ficheros u otros directorios. Este campo será consultado cuando necesitemos recorrer la estructura completa de un objeto Record, y se utilizará en colaboración con el método GetChildren del objeto Record. En un próximo apartado veremos la utilización del método GetChildren para obtener los hijos que contiene un objeto Record.

También podemos consultar la propiedad RecordType para averiguar si un objeto Record es una colección que contiene elementos hijo, el caso más común de colección es un directorio con ficheros y subdirectorios. Existe un campo especial al que se puede acceder mediante la constante adRecordURL y que devuelve la URL a la que nos encontramos conectados, así el Código fuente 321.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
strConex="URL=http://aesteban2/cursoasp30"
objRecord.Open "L1.jpg", strConex, adModeRead, adFailIfNotExist, adOpenSource
Response.Write objRecord.Fields(adRecordURL).Value
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 321

Devolverá esta salida: <http://aesteban2/cursoasp30/L1.jpg>. También es posible añadir nuestros propios campos a un objeto Record, para ello utilizaremos el método Append de la colección Fields.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
strConex="URL=http://aesteban2/cursoasp30"
objRecord.Open "L1.jpg", strConex, adModeRead, adFailIfNotExist, adOpenSource
objRecord.Fields.Append "NuevoCampo", adVarChar, 20
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 322

## Manipulación de objetos Record

El objeto Record nos ofrece una serie de métodos que nos permiten copiar, mover, renombrar y eliminar los recursos representados, ya sean ficheros o directorios. Vamos a comentar cada uno de estos métodos mostrando ejemplos de uso de los mismos.

Para copiar un recurso válido (fichero o directorio) a un destino que se encuentre accesible utilizaremos el método CopyRecord. Este método tiene la siguiente sintaxis general:

```
valorRetorno=objRecord.CopyRecord( [Origen], Destino,
[NombreUsuario], [Contraseña], [Opciones], [Síncrono] )
```

El valor de retorno de este método es una cadena que contiene el nombre completo (URL absoluta) del nuevo fichero resultante de la operación de copia. En origen se indica el fichero que se desea copiar y en destino la localización y nombre que va a tener el nuevo fichero copiado. El destino de la copia podrá ser relativo a la URL en la que se encuentra abierto el objeto Record correspondiente, ya que antes de realizar la operación de copia se debe abrir el objeto Record de alguna de las formas ya vistas en apartados anteriores. Tanto el origen y el destino, si se tratan de dos servidores distintos, deben soportar ambos DAV, es decir, deben ser servidores WebDAV.

Si no especificamos el parámetro origen, se copiará el fichero o directorio que se encuentre representado por el objeto Record.

En nombre usuario y contraseña se puede especificar un usuario con permisos, en le parámetro opciones podemos encontrar una serie de constantes relativas a la operación de copia que se pasan a comentar en la Tabla 34.

Constante	Descripción
adCopyUnspecified	Es la opción por defecto e indica que se realizará una copia recursiva de los directorios que contienen subdirectorios y el directorio o fichero de destino no se sobreescibirá si ya existe
adCopyOverwrite	Si el directorio o fichero de destino existen se sobreescibirán
adCopyNonRecursive	Se copiará el directorio correspondiente pero sin incluir sus subdirectorios.
adCopyAllowEmulation	Si el método de copia falla se simulará el proceso de copia utilizando un mecanismo de subida y bajada de ficheros. Esto puede ser posible cuando el destino no soporta el mecanismo DAV.

Tabla 34

Y el parámetro síncrono es para indicar si la operación de copia se va a llevar a cabo de forma síncrona o asíncrona, en diversos puntos hemos comentado que el entorno de ASP no permite la realización de operaciones asíncronas, así que este último parámetro no se utilizará.

Veamos este método en acción con un ejemplo que copia el fichero L1.jpg a un directorio dentro de la URL actual llamado NuevoDir, el nombre que se le da al nuevo fichero copiado es copia.jpg. Veamos el Código fuente 323.

```
<%Set objConexion=Server.CreateObject("ADODB.Connection")
objConexion.Provider="MSDAIPP.DSO"
strConex="URL=http://aesteban2/cursoasp30"
objCOnexion.Open strConex
Set objRecord=Server.CreateObject("ADODB.Record")
Set objRecord.ActiveConnection=objConexion
origen="L1.jpg"
destino="nuevadir/copia.jpg"
objRecord.Open
nombreFichero=objRecord.CopyRecord( origen,destino,,,adCopyOverWrite)
objRecord.Close
Set objRecord=Nothing
objConexion.Close
Set objConexion=Nothing%>
El nuevo fichero es <%=nombreFichero%>
```

Código fuente 323

En este caso se ha abierto el objeto Record a partir de un objeto Connection existente. Podemos realizar la misma operación que el código anterior pero utilizando un objeto Record de forma independiente de un objeto Connection.

```
<%strConex="URL=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject("ADODB.Record")
origen="L1.jpg"
destino="nuevadir/copia.jpg"
objRecord.Open ,strConex
nombreFichero=objRecord.CopyRecord(origen,destino,,,adCopyOverWrite)
objRecord.Close
Set objRecord=Nothing%>
El nuevo fichero es <%=nombreFichero%>
```

Código fuente 324

Si deseamos copiar el directorio completo a otra localización también podemos utilizar el método CopyRecord, en este caso el origen es el directorio images y la copia se encontrará en NuevoDir/imagescopia. En este caso el nombre de fichero devuelvo será la URL absoluta del nuevo directorio copiado.

```
<%strConex="URL=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject("ADODB.Record")
origen="images"
destino="nuevadir/imagescopia"
objRecord.Open ,strConex
nombreFichero=objRecord.CopyRecord(origen,destino,,,adCopyOverWrite)
objRecord.Close
Set objRecord=Nothing%>
El nuevo fichero es <%=nombreFichero%>
```

Código fuente 325

Al indicar la constante adCopyOverWrite parámetro de opciones se copiarán todos los subdirectorios del directorio de origen y se sobreescribirán en el destino.

El siguiente método que vamos a comentar en este apartado de manipulación de objetos Record, es el método MoveRecord. Este método nos va a permitir mover un origen válido a cualquier destino que se encuentre accesible. La sintaxis de este método es muy similar a la de CopyRecord teniendo todos sus parámetros el mismo significado que en el caso anterior pero para la operación de mover o renombrar un fichero o directorio.

```
valorRetorno=objRecord.MoveRecord( [Origen], Destino,
[NombreUsuario], [Contraseña], [Opciones], [Síncrono] )
```

El parámetro que tiene algún cambio es el de opciones que posee una serie de constantes que definen la operación de movimiento, estas constantes aparecen comentadas en la Tabla 35.

Constante	Descripción
adMoveUnspecified	Es la opción por defecto e indica que el directorio o fichero de destino no se debería sobreescibir si existe y los enlaces (hipervínculos) se deberían actualizar automáticamente para tener en cuenta la nueva localización
adMoveOverwrite	Los directorios y ficheros de destino se deben sobreescibir si ya existen
adMoveDontUpdateLinks	Se actualizarán los enlaces en el nuevo destino
adMoveAllowEmulation	Si el método falla a la hora de realizar la operación de movimiento, el proveedor del origen intentará simular el proceso utilizando los mecanismos de subida y bajada de ficheros

Tabla 35

Así si deseamos mover el fichero L1.jpg al directorio NuevoDir podemos utilizar el Código fuente 326.

```
<%Set objConexion=Server.CreateObject ("ADODB.Connection")
objConexion.Provider="MSDAIPP.DSO"
strConex="URL=http://aesteban2/cursoasp30"
objCOnexion.Open strConex
Set objRecord=Server.CreateObject ("ADODB.Record")
Set objRecord.ActiveConnection=objConexion
origen="L1.jpg"
destino="nuevadir/movidoL1.jpg"
objRecord.Open
nombreFichero=objRecord.MoveRecord(origen,destino,,,adMoveUnspecified)
objRecord.Close
Set objRecord=Nothing
objConexion.Close
Set objConexion=Nothing%>
El fichero se ha movido a <%=nombreFichero%>
```

Código fuente 326

Como se puede comprobar además de mover el fichero lo hemos renombrado a movidoL1.jpg. Esta misma operación la podemos llevar a cabo con un objeto Record independiente como indica el Código fuente 327.

```
<%strConex="URL=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject ("ADODB.Record")
origen="L1.jpg"
destino="nuevadir/movidoL1.jpg"
objRecord.Open ,strConex
nombreFichero=objRecord.MoveRecord(origen,destino,,,adMoveUnspecified)
objRecord.Close
Set objRecord=Nothing%>
El nuevo fichero es <%=nombreFichero%>
```

Código fuente 327

También podemos mover el contenido completo de un directorio a otro. En este caso se ha movido el directorio images, con todos sus subdirectorios, al directorio NuevoDir. Además se renombra el directorio images a imagesMovido.

```
<%strConex="URL=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject ("ADODB.Record")
origen="images"
destino="nuevadir/imagesMovido"
objRecord.Open ,strConex
nombreFichero=objRecord.MoveRecord(origen,destino,,,adMoveUnspecified)
objRecord.Close
Set objRecord=Nothing%>
El nuevo fichero es <%=nombreFichero%>
```

Código fuente 328

Como se puede comprobar ofrece la misma funcionalidad que CopyRecord pero para operaciones de movimiento de ficheros y directorios. Hemos visto los métodos de copia y movimiento de objetos Record, nos queda el método que permite eliminar objetos Record.

El objeto Record ofrece el método DeleteRecord que nos permitirá borrar un fichero o directorio a partir de una localización de terminada. Este método es más sencillo que los anteriores y presenta la siguiente sintaxis:

```
objRecord.DeleteRecord [origen] , [síncrono]
```

En segundo parámetro, como ya hemos comentado, no se va a utilizar, y si no se especifica el primero de ellos se eliminará el recurso representado por el objeto Record actual. Veamos un sencillo ejemplo, en el Código fuente 329, que elimina el fichero L1.jpg

```
<%Set objConexion=Server.CreateObject ("ADODB.Connection")
objConexion.Provider="MSDAIPP.DSO"
strConex="URL=http://aesteban2/cursoasp30"
objCOnexion.Open strConex
Set objRecord=Server.CreateObject ("ADODB.Record")
Set objRecord.ActiveConnection=objConexion
origen="L1.jpg"
```

```
objRecord.Open  
objRecord.DeleteRecord origen  
objRecord.Close  
Set objRecord=Nothing  
objConexion.Close  
Set objConexion=Nothing%>
```

Código fuente 329

También podemos realizar la misma operación pero utilizando un objeto Record independiente y sin especificar el parámetro origen, por lo que se borrará el recurso al que se encuentra conectado el objeto Record.

```
<%strConex="URL=http://aesteban2/cursoasp30"  
Set objRecord=Server.CreateObject("ADODB.Record")  
objRecord.Open "L1.jpg",strConex  
objRecord.DeleteRecord  
objRecord.Close  
Set objRecord=Nothing%>
```

Código fuente 330

Así mismo también podemos eliminar directorios con todo su contenido, en este caso se elimina el directorio images.

```
<%strConex="URL=http://aesteban2/cursoasp30"  
Set objRecord=Server.CreateObject("ADODB.Record")  
origen="images"  
objRecord.Open ,strConex  
objRecord.DeleteRecord origen  
objRecord.Close  
Set objRecord=Nothing%>
```

Código fuente 331

En este punto se da por finalizado el apartado dedicado a la manipulación de objetos Record, en el siguiente apartado veremos el interesante método GetChildren para tener acceso a los distintos ficheros y directorios que contiene un objeto Record que es un directorio.

## El método Getchildren

Hemos dedicado un apartado al método GetChildren del objeto Record debido a la importancia de este método. El método GetChildren devuelve un objeto Recordset con los ficheros y directorios que se encuentran dentro del objeto Record correspondiente, que será lógicamente un directorio. Además veremos un ejemplo que muestra la estructura completa de una URL determinada.

Para aplicar con éxito el método GetChildren sobre un objeto Record debemos asegurarnos que el objeto Record es un directorio, para ello podemos consultar el valor del campo RESOURCE\_ISCOLLECTION o bien el valor de la propiedad RecordType.

Veamos un sencillo ejemplo que comprueba si el objeto Record es un directorio, y en caso afirmativo muestra los ficheros y directorios que contiene.

En el ejemplo se utiliza las dos posibilidades para averiguar si un objeto Record es un directorio o no.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%strConex="URL=http://aesteban2/cursoasp30"
Set objRecord=Server.CreateObject("ADODB.Record")
objRecord.Open ,strConex
'nos aseguramos que el Record es un directorio
If objRecord.RecordType=adCollectionRecord Then
    Set objRSHijos=objRecord.GetChildren
    Response.Write "Contenido de
"&objRecord("RESOURCE_ABSOLUTEPARSENAME") &":<br>
    While not objRSHijos.EOF
        Response.Write objRSHijos.Fields("RESOURCE_DISPLAYNAME")
        If objRSHijos.Fields("RESOURCE_ISCOLLECTION") Then
            Response.Write " (directorio)<br>"
        Else
            Response.Write " (fichero)<br>"

        End if
        objRSHijos.MoveNext
    Wend
    objRSHijos.Close
    Set objRSHijos=Nothing
End if
objRecord.Close
Set objRecord=Nothing%>
</BODY>
</HTML>
```

Código fuente 332

El método GetChildren devuelve un objeto Recordset formado por objetos Record, cada uno de estos objetos representará un fichero o directorio dentro del objeto Record actual, sobre el que se ha lanzado el método GetChildren.

Si se aplica el método GetChildren sobre un objeto Record que no representa un directorio se producirá un error.

En este ejemplo se muestra al lado del nombre de cada recurso si se trata de un fichero o de un directorio, para realizar un recorrido completo de la estructura de un directorio determinado debemos hacer uso de la recursividad, para ir mostrando los distintos ficheros y subdirectorios contenidos en otros directorios.

El siguiente ejemplo muestra a partir de la URL especificada su estructura (jerarquía) completa. Para permitir acceder a los contenidos de todos los directorios se debe hacer uso de la recursividad.

El procedimiento recursivo que se utiliza se llama MuestrarJerarquia, que recibe como parámetro un objeto Record que representa a un directorio.

La llamada recursiva a este método se producirá cuando un de los hijos del Record sea a su vez un directorio, y por lo tanto debemos obtener también sus hijos. Este procedimiento recursivo tiene el Código fuente 333.

```
Sub MuestraJerarquia(objPadre)
    Set objHijo=Server.CreateObject("ADODB.Record")
    'se muestra el nombre del objeto Record padre
    Response.Write Espacios(objPadre("RESOURCE_PARSENAME")) & " (directorio)<br>"
    'se obtienen los hijos del Record padre
    Set objRSHijos=objPadre.GetChildren
    'incrementamos el nivel de anidación
    nivel=nivel+1
    'recorremos el Recordset de objetos Record hijo
    While not objRSHijos.EOF
        'abrimos el objeto Record hijo a partir del Recordset
        objHijo.Open objRSHijos
        If objHijo("RESOURCE_ISCOLLECTION") Then
            'es un directorio y hacemos la llamada recursiva
            MuestraJerarquia objHijo
        Else
            'es un fichero, se muestra su nombre
            Response.Write Espacios(objHijo("RESOURCE_PARSENAME")) &"<br>"
        End if
        objHijo.Close
        objRSHijos.MoveNext
    Wend
    'decrementamos el nivel de anidación, para la vuelta de la recursividad
    nivel=nivel-1
    Set objHijo=Nothing
    objRSHijos.Close
    Set objRSHijos=Nothing
End Sub
```

Código fuente 333

Vamos a tener un formulario en el que vamos a indicar la URL de la que queremos obtener toda su estructura, al pulsar el botón correspondiente se mostrará toda la jerarquía completa de la URL.

Para mostrar claramente la jerarquía de la URL se ha utilizado una función que añade una serie de espacios (en HTML &nbsp;) delante de cada elemento, para situarlo a un nivel adecuado y que se pueda apreciar de forma visual.

La función es la que muestra el Código fuente 334.

```
Function Espacios(elemento)
    cadenaEspacios=Space(nivel*2)
    cadenaEspacios=Replace(cadenaEspacios, " ", "&nbsp;&nbsp;&nbsp;&nbsp; ")
    Espacios=cadenaEspacios&elemento
End Function
```

Código fuente 334

El código completo de la página es el que se muestra en el Código fuente 335. En la Figura 130 se puede ver un ejemplo de ejecución:

```

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<form method="post" action="jerarquia.asp">
<input type="text" name="URL" size="30">
<input type="submit" name="enviar" value="Muestra jerarquía">
</form>
<%If Request.Form("URL")<>"" Then
    nivel=0
    'creamos el objeto Record raíz
    Set objRaiz=Server.CreateObject("ADODB.Record")
    objRaiz.Open , "URL=&Request.Form("URL")
    'mostramos la jerarquía
    MuestraJerarquia objRaiz
    objRaiz.Close
    Set objRaiz=Nothing
End if%>
<%Sub MuestraJerarquia(objPadre)
    Set objHijo=Server.CreateObject("ADODB.Record")
    'se muestra el nombre del objeto Record padre
    Response.Write Espacios(objPadre("RESOURCE_PARSENAME")) &" (directorio)<br>"
    'se obtienen los hijos del Record padre
    Set objRSHijos=objPadre.GetChildren
    'incrementamos el nivel de anidación
    nivel=nivel+1
    'recorremos el Recordset de objetos Record hijo
    While not objRSHijos.EOF
        'abrimos el objeto Record hijo a partir del Recordset
        objHijo.Open objRSHijos
        If objHijo("RESOURCE_ISCOLLECTION") Then
            'es un directorio y hacemos la llamada recursiva
            MuestraJerarquia objHijo
        Else
            'es un fichero, se muestra su nombre
            Response.Write Espacios(objHijo("RESOURCE_PARSENAME")) &"<br>"
        End if
        objHijo.Close
        objRSHijos.MoveNext
    Wend
    'decrementamos el nivel de anidación, para la vuelta de la recursividad
    nivel=nivel-1
    Set objHijo=Nothing
    objRSHijos.Close
    Set objRSHijos=Nothing
End Sub
Function Espacios(elemento)
    cadenaEspacios=Space(nivel*2)
    cadenaEspacios=Replace(cadenaEspacios, " ", " &nbsp;&nbsp;&nbsp;")
    Espacios=cadenaEspacios&elemento
End Function%>
</BODY>
</HTML>

```

Código fuente 335

Este ha sido el último apartado de este capítulo dedicado al objeto Record de ADO, en el siguiente veremos el otro nuevo objeto de ADO 2.5, el objeto Stream, que como ya hemos comentado lo vamos a utilizar para acceder a los contenidos del objeto Record.

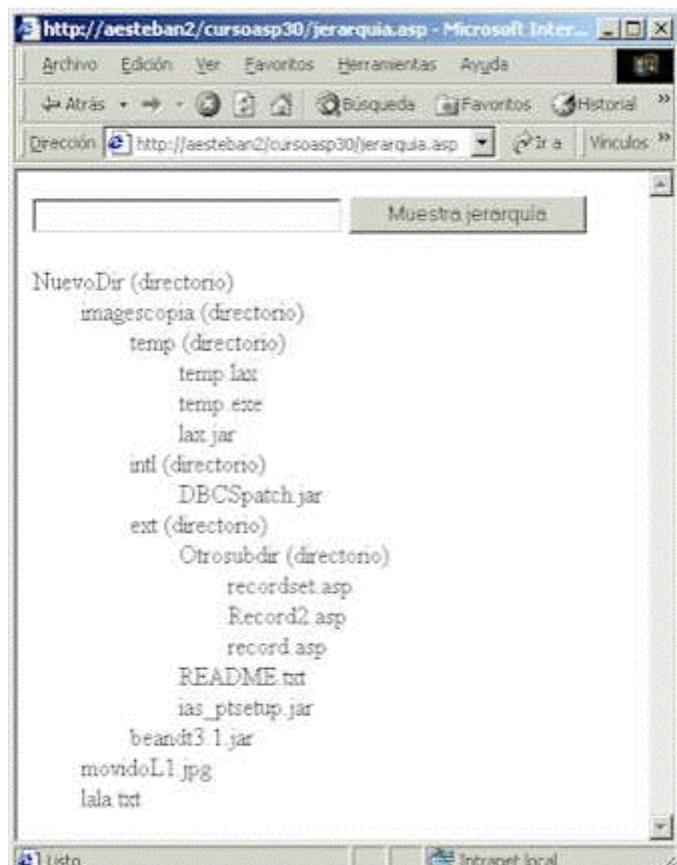


Figura 130. Jerarquía de un objeto Record

# ADO 2.5: Stream

---

## Introducción

En este capítulo vamos a tratar el segundo nuevo objeto ofrecido por ADO 2.5, se trata del objeto Stream, que como veremos se encuentra muy relacionado con el objeto visto en el capítulo anterior, es decir, con el objeto Record, ya que su labor principal va a ser la de permitir acceder a los contenidos de un objeto Record determinado.

Este es el último tema dedicado al acceso a datos con ADO desde ASP.

## El objeto Stream

En el capítulo anterior vimos que con el objeto Record podemos acceder a ficheros y directorios, para realizar un gran número de operaciones con ellos, pero no existe la posibilidad con el objeto Record de leer o escribir en los ficheros que representan, es en este momento cuando el objeto Stream hace su aparición, un objeto Stream nos va a permitir acceder al contenido de los objetos Record.

Un objeto Stream es realmente un contenedor para un bloque de memoria temporal, ofrece la posibilidad de escribir, leer y manipular flujos binarios de bytes o datos en formato texto que forman parte de un fichero.

A continuación vamos a exponer de forma breve las propiedades y métodos que ofrece el objeto Stream.

Las propiedades que presenta el objeto Stream son:

- *Charset*: identifica el conjunto de caracteres utilizado y el formato de los datos, por defecto utiliza el formato UNICODE.
- *EOS*: esta propiedad tiene el valor true si nos encontramos al final del Stream.
- *LineSeparator*: indica el carácter que se utiliza para separar líneas en un Stream de texto, por defecto es el salto de línea y retorno de carro (vbCrLf), otras constantes que se pueden asignar son adCr (retorno de carro) y adLf (salto de línea).
- *Mode*: indica los permisos disponibles para acceder al fichero, por defecto tiene permisos de sólo lectura (adModeRead). Los valores que puede tener esta propiedad se corresponden con los de la propiedad Mode del objeto Connection.
- *Position*: especifica la posición actual en el Stream.
- *Size*: indica el tamaño en bytes del Stream.
- *State*: indica el estado en el que se encuentra el objeto Stream, abierto o cerrado.
- *Type*: indica si contiene datos binarios o texto.

Y sus métodos aparecen a continuación:

- *Cancel*: cancela cualquier operación asíncrona pendiente.
- *Close*: cierra un objeto Stream abierto.
- *CopyTo*: copia caracteres o bytes de un objeto Stream a otro.
- *Flush*: vacía el contenido del búfer del Stream y lo envía al fichero correspondiente.
- *LoadFromFile*: carga un objeto Stream desde un fichero.
- *Open*: abre un objeto Stream a partir de una URL o de un objeto Record existente, o también crea un objeto Stream en blanco.
- *Read*: lee un número determinado de bytes del Stream.
- *ReadText*: lee un número determinado de caracteres de un Stream de texto.
- *SaveToFile*: guarda en un fichero un objeto Stream abierto.
- *SetEOS*: establece la posición actual como la posición de finalización del Stream.
- *SkipLine*: salta una línea cuando se está leyendo de un Stream de tipo texto.
- *Write*: escribe datos binarios en un Stream.
- *WriteText*: escribe texto en un Stream.

## Creación y apertura del objeto Stream

Un objeto Stream se crea mediante la sentencia que muestra el Código fuente 336. Un objeto Stream lo podemos obtener (abrir) desde diversas fuentes. Veamos brevemente las distintas opciones de las que disponemos:

```
Set objStream=Server.CreateObject ("ADODB.Stream")
```

Código fuente 336

- A partir de una URL que apunta a un recurso.
- De forma independiente, para almacenar datos que no dependen de ningún origen de datos.
- A partir de un objeto Record, obteniendo su Stream por defecto.
- A partir de un objeto Record abierto.

El método Open permite abrir un objeto Stream, su sintaxis general es la siguiente:

```
objStream.Open [Fuente] , [Modo] , [Opciones] , [Usuario] , [Contraseña]
```

El parámetro fuente indica el origen de los datos del Stream, puede ser una cadena con una URL válida de un fichero o bien un objeto Record abierto. Si no indicamos este parámetro opcional, el objeto Stream no se encontrará asociado a un origen de datos. El parámetro modo se corresponde con la propiedad Mode y especifica el modo en el que se va a abrir el objeto Stream, en lo que a permisos de acceso se refiere. Por defecto se abre sólo para lectura, y puede presentar las siguientes constantes que se muestran en la Tabla 36.

Constante	Descripción
adModeUnknown	Es el modo por defecto e indica que los permisos no se han establecido
adModeRead	Permisos de sólo lectura
adModeWrite	Permisos de sólo escritura
adModeReadWrite	Permisos de lectura y escritura
adModeShareDenyRead	Evita que otros usuarios puedan abrir una conexión con permisos de lectura
adModeShareDenyWrite	Evita que otros usuarios puedan abrir una conexión con permisos de escritura
adModeShareExclusive	Evita que otros establezcan una conexión con este objeto Stream
adModeShareNone	Evita que otros establezcan conexiones con cualquier permiso

Tabla 36

El parámetro opciones indica cómo y desde dónde se va a abrir el objeto Stream, su valor por defecto es la constante adOpenStreamUnspecified, los valores que puede tomar este parámetro se describen en la siguiente tabla de constantes de ADO.

Constante	Descripción
adOpenStreamAsync	Abre un objeto Stream de forma asíncrona, en el entorno de ASP no es posible utilizar tareas asíncronas
adOpenStreamFromRecord	Abre un Stream utilizando un objeto Record existente como fuente
adOpenStreamFromURL	Abre un Stream utilizando como origen una URL válida
adOpenStreamUnspecified	No es específica el origen del objeto Stream

Tabla 37

Los dos últimos parámetros, usuario y contraseña, indican el usuario que va a acceder al recurso.

Veamos diversos ejemplos que abren un objeto Stream.

```
<%Set objStream=Server.CreateObject ("ADODB.Stream")
objStream.Open
"URL=http://aesteban2/cursoasp30/alta.asp", adModeReadWrite, adOpenStreamFromURL
objStream.Close
Set objStream=Nothing%>
```

Código fuente 337

En este primer ejemplo se ha abierto un objeto Stream a partir de una URL que indica la localización de un fichero, como se puede comprobar se ha abierto sólo para lectura y se ha utilizado la constante adOpenStreamFromURL, para indicar que la fuente del objeto Stream es una URL. La URL debe hacer referencia a un fichero existente, si hace referencia a un directorio o a un fichero no existente se producirá un error.

Al utilizar la cadena URL en el parámetro fuente del método Open estamos indicando que se va a utilizar el proveedor OLEDB para publicación en Internet (OLEDB provider for Internet publishing).

Otra forma distinta de abrir un objeto Stream es a partir de un objeto Record abierto existente. Y dentro de esta posibilidad hay dos variantes: utilizando el objeto Record como origen de datos del objeto Stream u obteniendo el objeto Stream por defecto que ofrece un objeto Record abierto.

En el Código fuente 338 se muestra un ejemplo del primer caso.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
strConex="URL=http://aesteban2/cursoasp30"
objRecord.Open "alta.asp", strConex, adModeRead, adFailIfNotExists
Set objStream=Server.CreateObject ("ADODB.Stream")
```

```

objStream.Open objRecord,adModeReadWrite,adOpenStreamFromRecord
objStream.Close
Set objStream=Nothing
objRecord.Close
Set objRecord=Nothing%>

```

Código fuente 338

En este caso el objeto Record que vamos a utilizar debe representar un fichero válido, si representa un directorio a la hora de abrir el objeto Stream a partir de este objeto Record se producirá un error. Como se puede ver se ha utilizado la constante adOpenStreamFromRecord dentro del parámetro opciones del método Open del objeto Stream.

También podemos obtener un objeto Stream a partir de lo que se denomina el Stream por defecto de un objeto Record. El objeto Record posee un campo especial que se puede acceder a él a través de la constante adDefaultStream, este campo contiene un objeto Stream que se corresponde con el fichero que representa el objeto Record. En el Código fuente 339 se ofrece un sencillo ejemplo.

```

<%Set objRecord=Server.CreateObject ("ADODB.Record")
strConex="URL=http://aesteban2/cursoasp30"
objRecord.Open "alta.asp",strConex,adModeRead,adFailIfExists
Set objStream=Server.CreateObject ("ADODB.Stream")
objStream=objRecord.Fields(adDefaultStream)
objStream.Close
Set objStream=Nothing
objRecord.Close
Set objRecord=Nothing%>

```

Código fuente 339

En este caso no se utiliza el método Open para abrir el objeto Stream, sino que se obtiene directamente del correspondiente objeto Record abierto.

Otra forma de abrir un objeto Stream es abrir un objeto Stream completamente independiente, sin que esté relacionado con ningún origen de datos. En este caso se abre un objeto Stream vacío que se utilizará para escribir datos en él, ya sean de tipo texto o binarios. En el Código fuente 340 se ofrece un ejemplo.

```

<%Set objStream=Server.CreateObject ("ADODB.Stream")
objStream.Open
objStream.Close
Set objStream=Nothing%>

```

Código fuente 340

En el siguiente apartado en el que tratamos la manipulación del objeto Stream, veremos, entre otras muchas más cosas, los métodos que ofrece para escribir en un Stream. En los ejemplos de este apartado únicamente hemos abierto y cerrado un Stream sin hacer nada con él, en el siguiente apartado veremos como manipular el objeto Stream.

## Manipulación del objeto Stream

En este apartado vamos a comentar los distintos métodos que nos ofrece el objeto Stream para escribir, leer y grabar en disco, es decir, todos los métodos necesarios para manipular un objeto Stream.

Una vez que hemos abierto un objeto Stream, utilizando cualquiera de las opciones vistas en el apartado anterior, podemos examinar los contenidos del fichero que representa este objeto.

Para obtener los contenidos de un objeto Stream podemos utilizar los métodos Read y ReadText, el primero de ellos lo se debe utilizar cuando el objeto Stream contiene datos binarios, y el segundo cuando contiene datos de tipo texto.

La propiedad Type del objeto Stream nos indica el formato de los datos, si queremos leer los datos de un Stream con contenido binario la propiedad Type debe tener el valor de la constante adTypeBinary, y si contiene datos de tipo texto deberá tener el valor adTypeText. Si conocemos a priori el formato que va tener el fichero representado por el objeto Stream es recomendable asignarle a la propiedad Type el valor correspondiente.

Los métodos Read y ReadText reciben un parámetro que indica el número de bytes/caracteres que se deben leer, si no indicamos nada como parámetro se leerá el contenido del objeto Stream completo. También podemos indicar que se lea todo el contenido de un objeto Stream utilizando como parámetro la constante adReadAll.

El Código fuente 341 consiste en mostrar el contenido de un un objeto fichero en un área de texto. En este caso el tipo de datos del objeto Stream es texto.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<%Set objStream=Server.CreateObject ("ADODB.Stream")>
objStream.Open "URL=http://aesteban2/cursoASP30/alta.asp", adModeRead, adOpenFromURL
IF objStream.Type=adTypeText Then
    objStream.Charset="ascii"%
    <textarea cols="100" rows="20">
    <%=objStream.ReadText(adReadAll)%>
    </textarea>
<%Else
    Response.Write "Formato binario de datos"
End if
objStream.Close
Set objStream=Nothing%>
</BODY>
</HTML>
```

Código fuente 341

Como se puede comprobar en este caso se ha abierto un objeto Stream a partir de una URL que hace referencia a un fichero. Se ha utilizado la propiedad Charset para indicar el juego de caracteres que corresponde al formato de datos del fichero, en este caso se ha utilizado caracteres ASCII. En este ejemplo se comprueba que los datos del Stream sean de tipo texto para poder leerlos con el método ReadText y no se produzcan resultados no deseados.

En el Código fuente 342 se lee el contenido de un objeto Stream que representa a un fichero de imagen, por lo tanto la lectura se realiza sobre datos binarios.

```
<%Set objStream=Server.CreateObject ("ADODB.Stream")
objStream.Open "URL=http://aesteban2/cursoASP30/L1.jpg", adModeRead, adOpenFromURL
objStream.Type=adTypeBinary
contenido=objStream.Read
Response.BinaryWrite contenido
objStream.Close
Set objStream=Nothing%>
```

Código fuente 342

En este caso se lee todo el contenido y se envía al navegador, por lo tanto al cargar esta página el navegador Web mostrará la imagen correspondiente al fichero representado por el objeto Stream. En este caso hemos especificado el tipo de objeto Stream mediante la propiedad Type.

La propiedad Position del objeto Stream indica la posición actual dentro de un objeto Stream abierto. Esta posición se calcula teniendo como referencia el número de bytes completos de un objeto Stream, así por ejemplo si utilizamos el conjunto de caracteres ASCII (1 byte para cada carácter), el Código fuente 343 nos devolvería 10, cada vez que se va leyendo un número determinado de caracteres (o bytes) se va actualizando el valor de Position.

```
<%Set objStream=Server.CreateObject ("ADODB.Stream")
objStream.Open "URL=http://aesteban2/cursoASP30/alta.asp", adModeRead, adOpenFromURL
objStream.Charset="ascii"
objStream.ReadText(10)%>
POSICIÓN ACTUAL:<=%objStream.Position%>
<%objStream.Close
Set objStream=Nothing%>
```

Código fuente 343

La propiedad Position es de lectura y escritura, y podemos modificarla cuando lo consideremos necesario, en el Código fuente 344 se lee a partir de una posición determinada.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<form method="post" action="stream.asp">
<input type="text" name="posicion" size="5">
<input type="submit" name="enviar" value="MuestraTexto">
<%If Request.Form("posicion")<>"" Then
    Set objStream=Server.CreateObject ("ADODB.Stream")
    objStream.Open
"URL=http://aesteban2/cursoASP30/alta.asp", adModeRead, adOpenFromURL
    objStream.Charset="ascii"
    objStream.Position=Request.Form("posicion")
    If Not objStream.EOS Then%
        <textarea cols="80" rows="20">
        <=%objStream.ReadText(objStream.Size-objStream.Position)%>
        </textarea>
```

```

<br>POSICIÓN ACTUAL:<%=objStream.Position%>
<%End if
objStream.Close
Set objStream=Nothing
End if%>
</form>
</BODY>
</HTML>

```

Código fuente 344

Como se puede ver la posición determinada se obtiene a partir de un formulario, y se realiza un control para verificar que con la posición indicada no se ha sobrepasado el final del Stream, para ello hacemos uso de la propiedad EOS, que será verdadera cuando hayamos llegado al final del Stream, así si Position es mayor que el tamaño del Stream no se mostrará el texto. Para leer el resto de caracteres hasta el final desde la posición indicada, hacemos uso de la propiedad Size, y que al estar los datos del Stream es formato ASCII, el número de bytes coincide con el número de caracteres.

Otro método relacionado con la lectura de objetos Stream es el método SkipLine, este método salta una línea de un Stream con datos de tipo texto, y se sitúa en la siguiente línea.

Para escribir en un objeto Stream disponemos de los métodos Write y WriteText, que permiten escribir datos binarios y texto respectivamente. El método Write recibe como argumento un array de bytes, y el método WriteText una cadena de texto. El método WriteText posee un segundo parámetro que se corresponde con una constante que indica el tipo de escritura que se va a realizar, si utilizamos la constante adWriteLine, se escribe el texto y se salta de línea, y si utilizamos la constante adWriteChar simplemente se escribe el texto.

En ambos casos los datos se empiezan a escribir en el objeto Stream a partir de la posición actual, definida como ya hemos visto por la propiedad Position. Si queremos forzar que se comience a escribir en el principio del Stream estableceremos la propiedad Position a cero, pero si lo que queremos es vaciar por completo el Stream antes de escribir algo en él, además de establecer Position a cero, deberemos lanzar el método SetEOS, este método establecerá la posición actual como la de finalización del Stream. Veamos un sencillo ejemplo, en el Código fuente 345 que vacía un Stream antes de escribir en él.

```

<%Set objRecord=Server.CreateObject ("ADODB.Record")
objRecord.Open
"alta.asp", "URL=http://aesteban2/cursoASP30", adModeWrite, adFailIfExists
Set objStream=Server.CreateObject ("ADODB.Stream")
objStream.LineSeparator=adCRLF
objStream.Open objRecord, adModeWrite, adOpenStreamFromRecord
objStream.Position=0
objStream.SetEOS
objStream.WriteText "Esta es la primera1 linea",adWriteLine
objStream.WriteText "Esta es la segunda linea",adWriteLine
objStream.WriteText "Esta es la tercera linea",adWriteLine
objStream.WriteText "FIN.",adWriteChar
objStream.Close
Set objStream=Nothing
objRecord.Close
Set objRecord=Nothing%>

```

Código fuente 345

En este caso hemos abierto un objeto Stream para escritura a partir de un objeto Record existente, que se ha tenido que abrir también en modo de escritura. El separador de línea se ha establecido como salto de línea y retorno de carro. En el Código fuente 346 copiamos el contenido de un Stream a otro.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
objRecord.Open
"alta.asp","URL=http://aesteban2/cursoASP30",adModeRead,adFailIfExists
Set objStreamLectura=Server.CreateObject ("ADODB.Stream")
objStreamLectura.Open objRecord,adModeRead,adOpenStreamFromRecord
objRecord.Close
objRecord.Open
"prue.asp","URL=http://aesteban2/cursoASP30",adModeWrite,adFailIfExists
Set objStreamEscritura=Server.CreateObject ("ADODB.Stream")
objStreamEscritura=objRecord(adDefaultStream)
objStreamEscritura.Position=0
objStreamEscritura.SetEOS
While Not objStreamLectura.EOS
    objStreamEscritura.WriteText objStreamLectura.ReadText(10)
Wend
objStreamLectura.Close
Set objStreamLectura=Nothing
objStreamEscritura.Close
Set objStreamEscritura=Nothing
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 346

En este caso el objeto Stream para lectura se ha obtenido de un objeto Record abierto, y el objeto Stream de escritura a partir del Stream por defecto de un objeto Record abierto. El Stream de escritura se ha vaciado de contenido antes de empezar a escribir en él, y se han ido escribiendo de 10 en 10 caracteres. Para ir leyendo del Stream de origen y escribiendo en el de destino hemos utilizado un bucle While que se ejecutará hasta que la propiedad EOS tenga un valor verdadero. Este mismo ejemplo lo podríamos realizar de una forma más sencilla, leyendo todo el contenido y escribiéndolo una única vez, como se puede comprobar en el Código fuente 347.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
objRecord.Open
"alta.asp","URL=http://aesteban2/cursoASP30",adModeRead,adFailIfExists
Set objStreamLectura=Server.CreateObject ("ADODB.Stream")
objStreamLectura.Open objRecord,adModeRead,adOpenStreamFromRecord
objRecord.Close
objRecord.Open
"prue.asp","URL=http://aesteban2/cursoASP30",adModeWrite,adFailIfExists
Set objStreamEscritura=Server.CreateObject ("ADODB.Stream")
objStreamEscritura=objRecord(adDefaultStream)
objStreamEscritura.Position=0
objStreamEscritura.SetEOS
objStreamEscritura.WriteText objStreamLectura.ReadText
objStreamLectura.Close
Set objStreamLectura=Nothing
objStreamEscritura.Close
Set objStreamEscritura=Nothing
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 347

También podemos hacer uso del método CopyTo para copiar contenidos de un objeto Stream a otro objeto Stream, ambos objetos deben encontrarse abiertos. El método CopyTo se lanzará sobre el Stream de origen y se le pasa por parámetro el Stream de destino y el número de caracteres/bytes a copiar, si no se indica este segundo parámetro se copiará completo es Stream de origen. Veamos un sencillo ejemplo, en el Código fuente 348, que obtenemos a partir de la modificación del ejemplo anterior.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
objRecord.Open
"alta.asp", "URL=http://aesteban2/cursoASP30", adModeRead, adFailIfExists
Set objStreamLectura=Server.CreateObject ("ADODB.Stream")
objStreamLectura.Open objRecord,adModeRead,adOpenStreamFromRecord
objRecord.Close
objRecord.Open
"prue.asp", "URL=http://aesteban2/cursoASP30", adModeWrite,adFailIfExists
Set objStreamEscritura=Server.CreateObject ("ADODB.Stream")
objStreamEscritura=objRecord(adDefaultStream)
objStreamEscritura.Position=0
objStreamEscritura.SetEOS
objStreamLectura.CopyTo objStreamEscritura
objStreamLectura.Close
Set objStreamLectura=Nothing
objStreamEscritura.Close
Set objStreamEscritura=Nothing
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 348

Podemos cargar el contenido de un fichero determinado en un objeto Stream abierto haciendo uso del método LoadFromFile. A este método se le debe pasar como parámetro la ruta física completa del fichero que se desea cargar, este fichero debe encontrarse lógicamente en el servidor Web.

```
<%Set objRecord=Server.CreateObject ("ADODB.Record")
objRecord.Open
"alta.asp", "URL=http://aesteban2/cursoASP30", adModeWrite,adFailIfExists
Set objStream=Server.CreateObject ("ADODB.Stream")
objStream.Open objRecord,adModeWrite,adOpenStreamFromRecord
objStream.Position=0
objStream.SetEOS
objStream.LoadFromFile "c:\Inetpub\wwwroot\cursoASP30\prue.asp"
objStream.Close
Set objStream=Nothing
objRecord.Close
Set objRecord=Nothing%>
```

Código fuente 349

Mediante el método LoadFromFile se sobrescribe el contenido existente en el objeto Stream correspondiente, pero no se cambia el enlace que existe entre el objeto Stream y el fichero que representa, es decir, se modifica el contenido del fichero pero no el nombre del fichero. Como se puede observar se debe indicar que el objeto Stream se ha abierto para escritura.

Relacionado también con la manipulación de objetos Stream tenemos el método SaveToFile. Este método permite almacenar el contenido de un objeto Stream en un fichero. El método SaveToFile recibe dos parámetros, el primero de ellos es la ruta completa del fichero en el que queremos almacenar de forma permanente el contenido del objeto Stream correspondiente, lógicamente el fichero se almacenará en el servidor Web. Y el segundo parámetro son una serie de constantes que indican la forma en la que se va grabar el fichero, estas constantes son: adSaveCreateNotExist, crea un nuevo fichero si no existe ya, adSaveCreateOverWrite, sobrescribe el fichero si ya existe.

En el Código fuente 350 se guarda en un fichero el contenido de un objeto Stream que hemos abierto a partir de una URL.

```
<%Set objStream=Server.CreateObject("ADODB.Stream")
objStream.Open
"URL=http://aesteban2/cursoASP30/L1.jpg", adModeRead, adOpenStreamFromURL
objStream.Type=adTypeBinary
objStream.SaveToFile "c:\tmp\stream.jpg", adSaveCreateNotExist
objStream.Close
Set objStream=Nothing%>
```

Código fuente 350

Como se puede observar antes de guardar el contenido del objeto Stream debemos especificar en su propiedad Type, el tipo de los datos que contiene, en este caso son binarios, ya que se trata de una imagen.

En este otro ejemplo se almacena en un fichero el contenido de un objeto Stream que se ha creado de forma independiente sin estar asociado a ningún fichero.

En este caso el tipo de dato que contiene el Stream es texto, y en el método SaveToFile indicamos que si el fichero no existe se cree y si además existe se sobrescribe.

```
<%Set objStream=Server.CreateObject("ADODB.Stream")
objStream.Open
objStream.Type=adTypeText
objStream.WriteLine "Este es un Stream independiente,", adWriteLine
objStream.WriteLine "y le estamos añadiendo contenido", adWriteLine
objStream.WriteLine "para almacenarlo en un fichero, ", adWriteLine
objStream.WriteLine "con el método SaveToFile."
objStream.SaveToFile "c:\tmp\stream.txt", adSaveCreateOverWrite
objStream.Close
Set objStream=Nothing%>
```

Código fuente 351

Con este ejemplo terminamos no sólo este apartado ni este tema, sino todo el ciclo de temas dedicados al acceso a datos con ADO desde ASP. En el próximo capítulo trataremos la herramienta que nos propone Microsoft para la programación de páginas ASP, Visual InterDev 6. En los primeros capítulos la presentamos de forma breve, pero el siguiente capítulo va a consistir en un monográfico de esta herramienta de desarrollo.



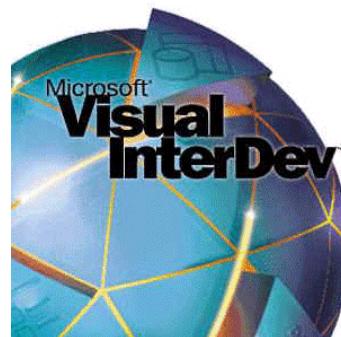
# ASP y Visual InterDev 6.0

---

## Introducción

Visual Interdev 6.0 forma parte del paquete Visual Studio 6.0 de Microsoft. Este paquete incluye una serie de herramientas de desarrollo visual especialmente diseñadas para ayudar al programador a crear aplicaciones dentro de un entorno integrado. Las herramientas que proporciona este paquete son las siguientes:

- Visual Basic
- Visual C++
- Visual FoxPro
- Visual J++
- Visual InterDev
- Visual SourceSafe
- Biblioteca MSDN



Visual InterDev nos permite desarrollar páginas Web basadas en HTML y aplicaciones basadas en ASP. Ofreciendo, además, un entorno de desarrollo integrado donde podremos realizar nuestra creación de páginas ASP a partir de un interfaz de usuario muy sencillo. Permite el desarrollo RAD

(Rapid Application Development). Ofrece herramientas que hacen mucho más agradable el desarrollo de aplicaciones orientadas a datos. Permite el desarrollo de soluciones integradas que pueden incluir componentes COM creados con Visual Basic, Visual C++, Visual J++ o Visual FoxPro.

Las herramientas para el manejo de bases de datos que nos ofrece Visual InterDev facilitan la utilización de los objetos ActiveX para bases de datos, es decir, los objetos ADO (ActiveX Data Objects) que veíamos en los temas anteriores correspondiente, además permite la conexión con el gestor de bases de datos SQL Server 7. Desde Visual InterDev podemos acceder a fuentes de datos definidas con ODBC y podemos utilizarlo como un administrador de bases de datos de SQL Server.

Otra característica de Visual InterDev es que permite el diseño en tiempo real de controles ActiveX.

Se facilita el desarrollo de páginas de manera independiente por parte de los equipos Web, manteniendo accesible una versión principal de las páginas. Se puede llevar a cabo, además, una programación de manera independiente en modo local.

Se facilita enormemente la programación gracias a una serie controles enlazados a datos y a un Modelo de objetos de secuencias de comandos.

## Proyectos y soluciones

El aspecto que ofrece Visual InterDev es común al de otras herramientas de desarrollo como Visual J++ y Visual C++. Dentro de Visual InterDev existen un par de conceptos que es necesario aclarar antes de empezar a desarrollar una aplicación con Visual InterDev. Si se ha utilizado alguna herramienta de la familia Visual Studio estos conceptos ya estarán claros, de todas formas los vamos a comentar a continuación.

El primero de estos conceptos se denomina proyecto. Todo lo que realizamos en Visual InterDev forma parte de un proyecto, no se puede crear un fichero sin que sea parte de un proyecto. Un proyecto es una colección de todo nuestro trabajo (páginas HTML, páginas ASP, ficheros gráficos, sonido, etc.) que se encuentra almacenado físicamente en un servidor Web en una estructura denominada sitio Web. Una Web o sitio Web es simplemente un directorio virtual que se encuentra localizado en un servidor. El proyecto es realmente un único fichero (\*.vip) en la máquina cliente que hace referencia al servidor y al sitio Web.

Los proyectos administran dos copias de la aplicación Web: local y maestra. Todos los ficheros de la aplicación Web maestra se almacenan en el servidor Web.

Cuando se edita un fichero, Visual InterDev descarga una copia de trabajo en la máquina cliente desde el servidor Web vía protocolo HTTP. Cuando se graban los cambios el fichero es actualizado automáticamente vía HTTP.

Debido a que los ficheros se encuentran almacenados en el servidor Web, varias personas pueden trabajar en el mismo proyecto al mismo tiempo, es decir, pueden hacer referencia a la misma aplicación Web maestra. De esta forma un miembro de un equipo de desarrollo puede estar editando una página ASP para introducir contenidos HTML desde FrontPage, mientras que otro miembro está realizando la programación de esa misma página. Si se está trabajando en un entorno de equipo, será necesario refrescar a menudo el panel donde se muestra el proyecto.

El segundo concepto que es necesario conocer es el de solución. Una solución es un conjunto de uno o más proyectos y sus elementos asociados, dentro de la solución se suelen integrar todos los proyectos que pertenezcan a una misma aplicación. Cuando se crea un nuevo proyecto, este proyecto debe ir

asociado a una solución, se puede crear una nueva solución o añadir proyectos a una solución existente. Los proyectos que se encuentren dentro de una misma solución pueden compartir información entre ellos.

Una solución pueden albergar proyectos heterogéneos, es decir, puede contener un proyecto de una base de datos, otro proyecto de Visual Basic y varios proyectos Web.

La solución actual está representada por el nodo de nivel superior de la ventana "Explorador de proyectos". Toda la información relativa a los proyectos y elementos que componen una solución está incluida en un archivo de solución .SLN.

La estructura formada por una solución y los proyectos que contiene se puede observar dentro de la ventana del Explorador de proyectos de Visual InterDev (Figura 131).

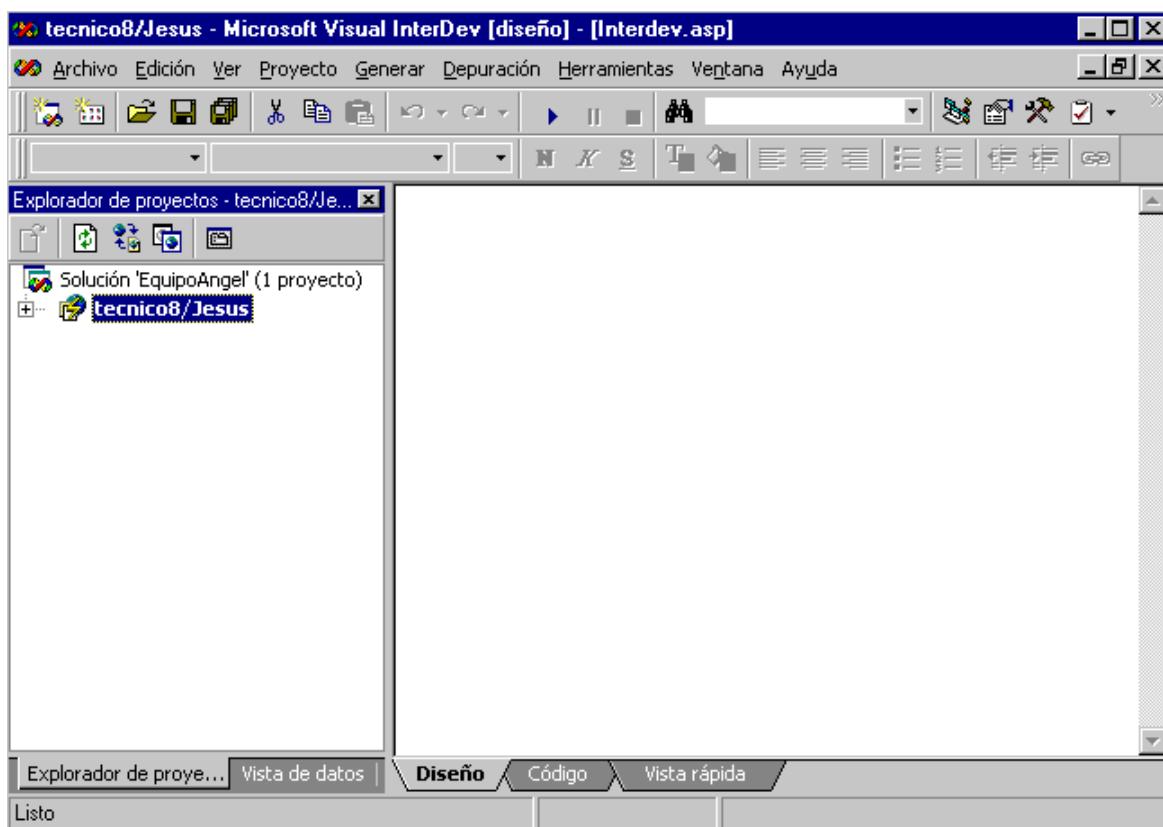


Figura 131

Desde el Explorador de proyectos podremos observar los diferentes ficheros y conexiones a bases de datos que posee un proyecto. Si se hace un doble clic sobre uno de estos ficheros se iniciará el editor por defecto. Si se pulsa con el botón derecho del ratón aparecerá un menú que permitirá editar, realizar una vista previa del fichero o ver sus propiedades entre otras operaciones sobre el fichero seleccionado.

Los proyectos de bases de datos y las conexiones a bases de datos aparecen en la vista denominada Vista de datos, que permite observar los diferentes objetos dentro de los objetos de datos, estos objetos pueden ser tablas, procedimientos almacenados, vistas o diagramas de bases de datos, como se observa en la Figura 132.

La vista de datos muestra todos los objetos de la base de datos a la que estamos conectados. Permite modificar el diseño de la base de datos (según permisos), crear nuevas tablas, borrar tablas, crear relaciones, procedimientos almacenados, etc., todo ello es posible gracias a la conexión en tiempo real que se establece con la base de datos. Además se ofrece un listado de todos los objetos de la base de datos de una manera jerarquizada.

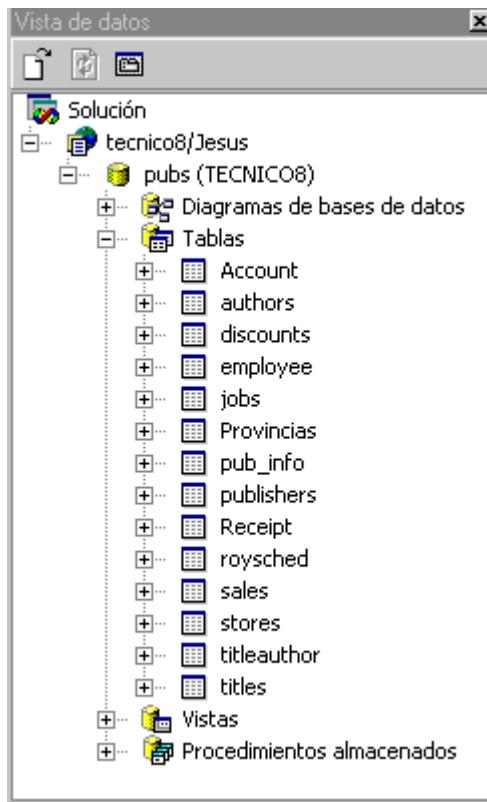


Figura 132

Se puede considerar que uno de los puntos fuertes de Visual InterDev es la integración con un sistema gestor de bases de datos. La integración entre las bases de datos y las páginas ASP es un tema bastante importante a la hora de desarrollar aplicaciones ASP, y Visual InterDev nos facilita bastante el trabajo en este ámbito. En los apartados siguientes se explicará el uso de las herramientas que Visual InterDev ofrece para el acceso y gestión de bases de datos.

El aspecto que ofrece la vista de datos es muy similar al que ofrecen Microsoft Access y SQL Server en su opción Enterprise Manager. Muchas de las tareas que podemos realizar con éste último las podremos llevar a cabo también desde la vista de datos, de esta forma, pulsando dos veces sobre una tabla podremos editar su contenido y si pulsamos con el botón derecho del ratón podremos entrar en el diseño de la misma a través de la opción Diseño.

## Crear un proyecto Web

- Desde el menú Archivo de Visual Interdev se debe seleccionar la opción Nuevo proyecto...
- En el cuadro de diálogo Nuevo proyecto, seleccione Proyecto de Visual Interdev en el panel izquierdo y Nuevo proyecto Web en el panel derecho.

- En el cuadro de texto Nombre, escriba un nombre para el proyecto y en el cuadro de texto Ubicación, una ubicación local para el mismo.
- Al pulsar en Abrir, se iniciará el Asistente para aplicaciones Web.

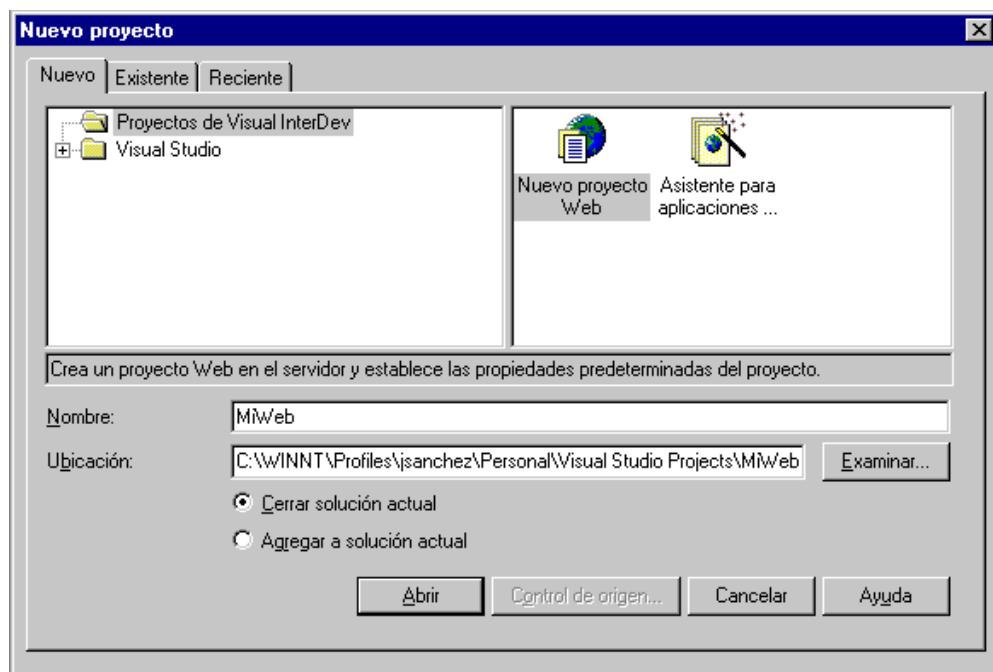


Figura 133

Paso 1 de 4: En este primer paso se debe especificar el nombre del servidor en el que se va a publicar el nuevo Web y el modo de trabajo a utilizar: Maestro o Local.

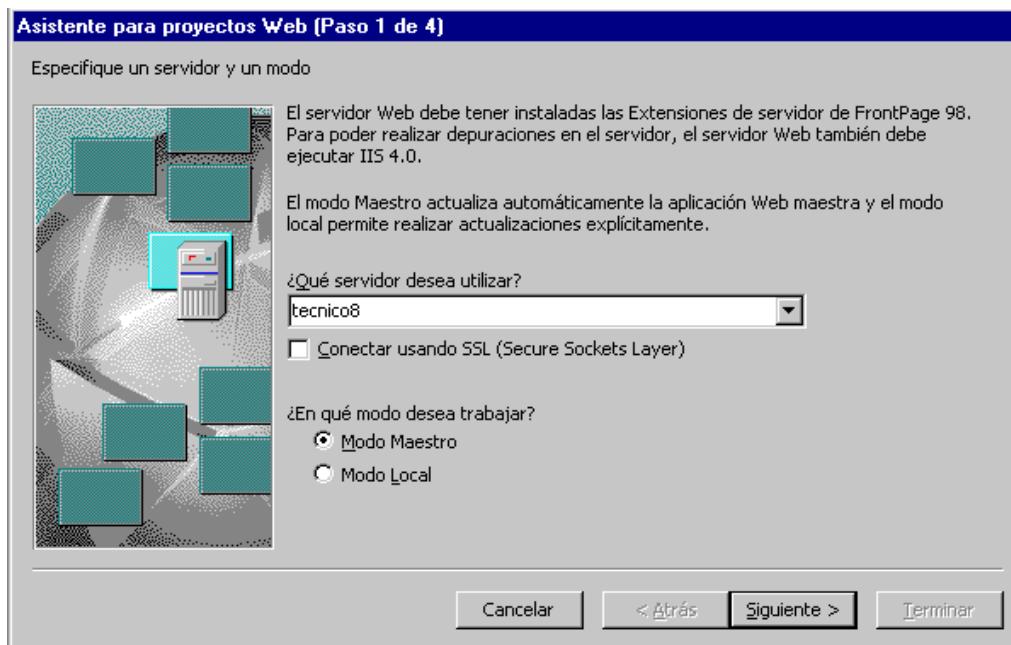


Figura 134

Paso 2 de 4: En este segundo paso se debe especificar el nombre de la aplicación Web, en el caso de querer crear una nueva, o bien se puede establecer una conexión con una aplicación Web ya existente en el servidor seleccionado.

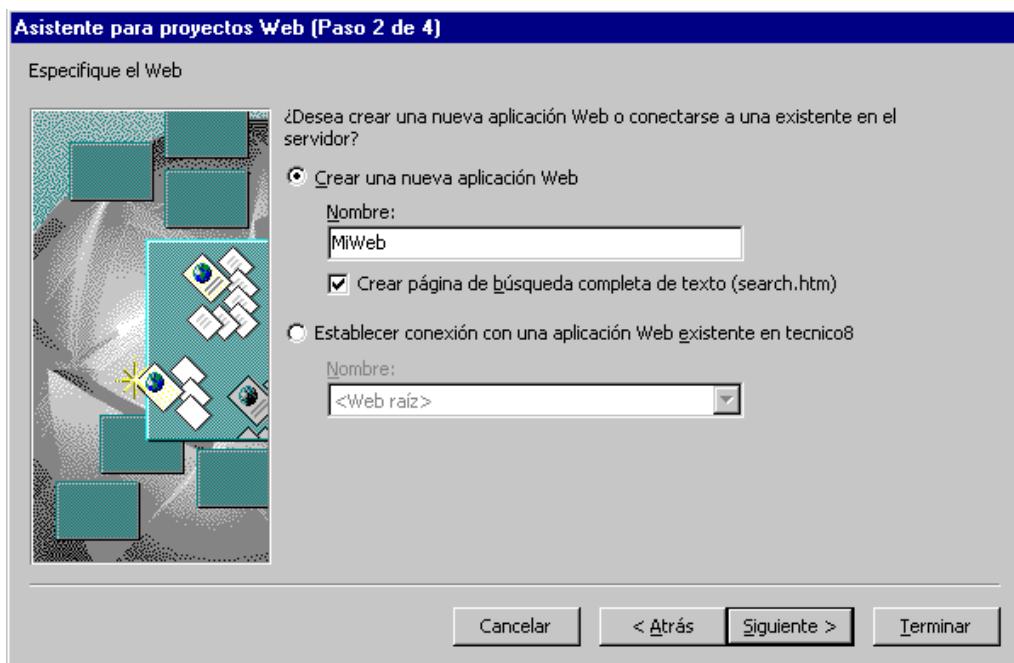


Figura 135

Paso 3 de 4: En este tercer paso se puede seleccionar un diseño predefinido para todas las páginas del Web. Puede seleccionar "Ninguno" en este momento y aplicar un diseño más adelante.

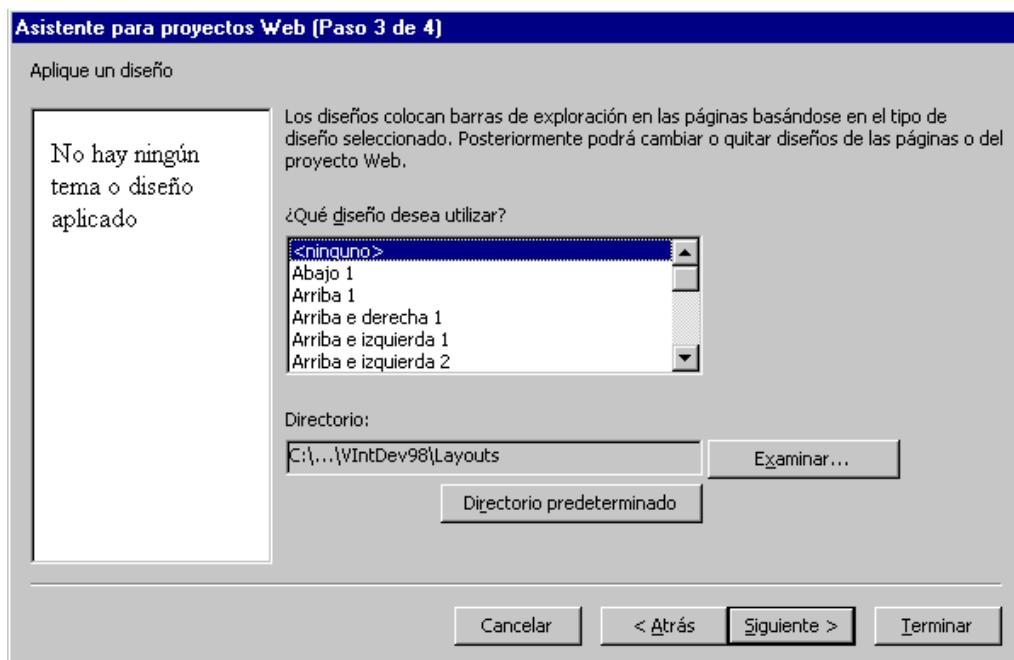


Figura 136

Paso 4 de 4: En este cuarto paso podrá aplicar un tema. Igualmente podrá seleccionar "Ninguno" y aplicar un tema concreto más adelante.

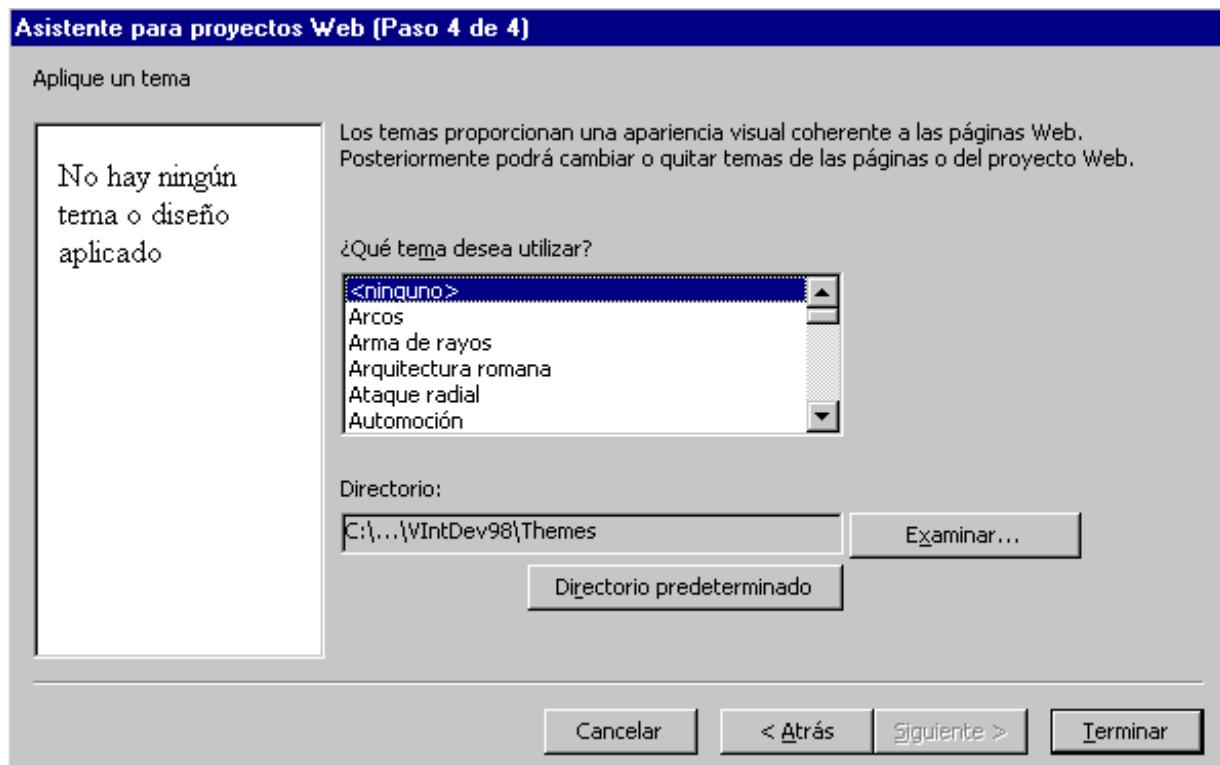


Figura 137

Como resultado, tendremos una solución nueva en el Explorador de proyectos, como vemos en la Figura 138.

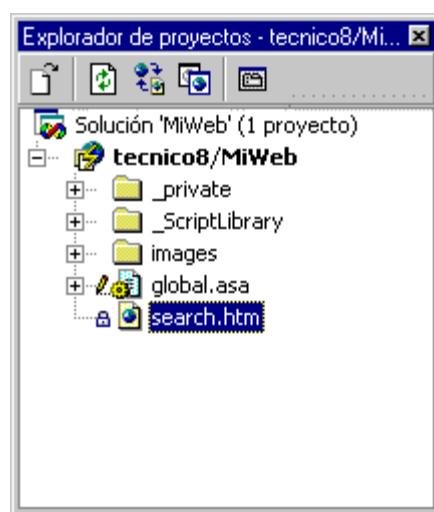


Figura 138

El asistente ha creado por defecto una serie de carpetas y ficheros en el nuevo sitio Web:

- \_private: Esta carpeta contendrá ficheros internos de trabajo del Web de Interdev.
- ScriptLibrary: Contiene una serie ficheros fuente para los objetos de script usados por los controles de Visual Interdev.
- images: Carpeta vacía que debería usarse para almacenar las imágenes del Web.
- global.asa: Fichero que representa a la aplicación ASP. A través de este fichero podremos agregar el script que queramos que se ejecute al producirse eventos de aplicación o de sesión en aplicaciones ASP.
- search.htm: Página de búsqueda de texto en el Web.

Una vez creado nuestro proyecto Web podremos agregar archivos de la siguiente forma:

- Dentro de la ventana del explorador de proyectos, debemos seleccionar el proyecto o la subcarpeta dentro del proyecto en la que vamos a colocar el archivo.
- Desplegamos el menú Proyecto, y seleccionamos la opción Agregar elemento (Figura 139)
- Dentro de la ficha Nuevo, en su panel derecho, debemos seleccionar el tipo de archivo a crear y el cuadro de texto Nombre: indicar un nombre para el archivo respetando la extensión por defecto en función del tipo de archivo seleccionado.
- Finalmente pulsaremos en el botón Abrir para agregar el archivo al proyecto. A partir este momento podremos, a través de la vista código, editar el código html y el script de la página .ASP que acabamos de crear.

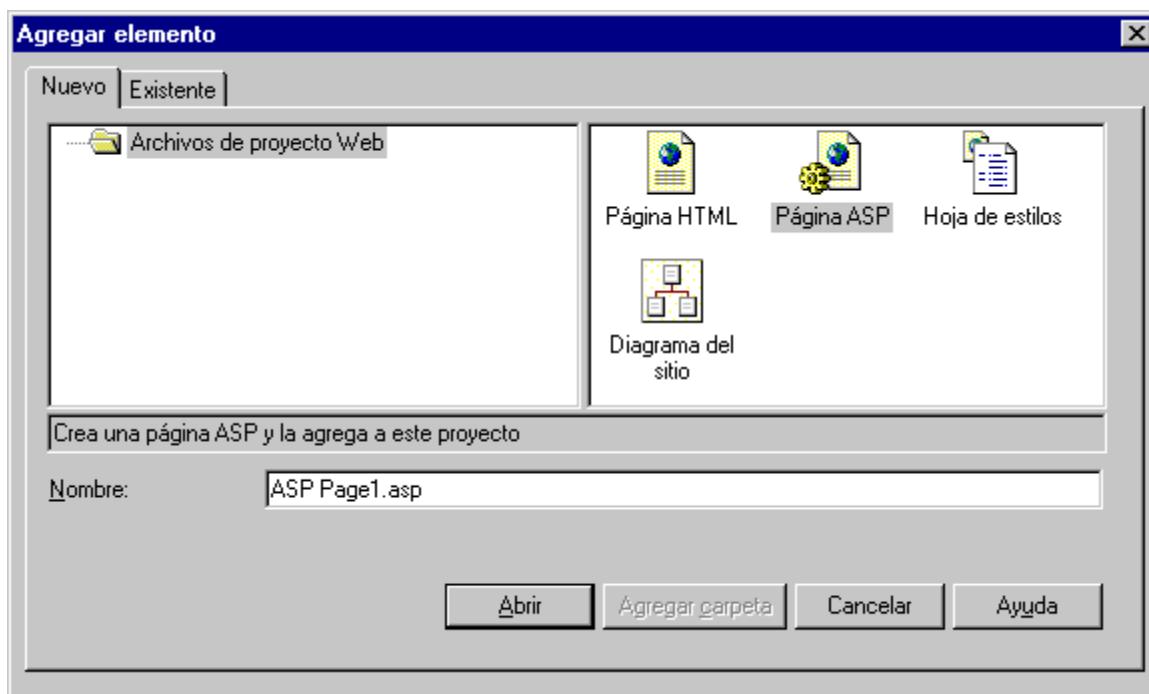


Figura 139

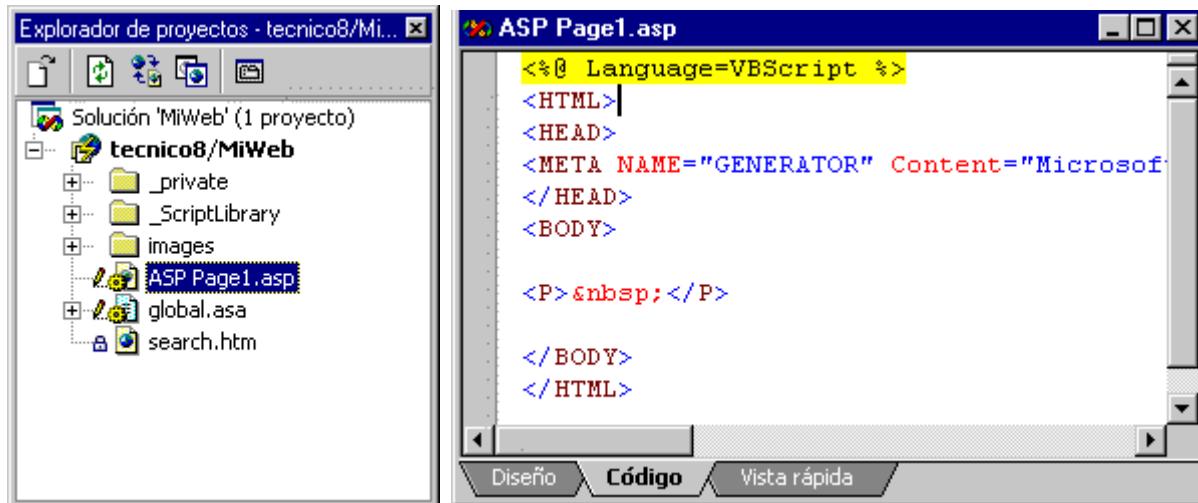


Figura 140

## El acceso a datos a través de Visual InterDev

A través de Visual Interdev podremos acceder y manipular los datos contenidos en la mayoría de las bases de datos: Access, SQL Server, Oracle .... A través del diseñador de consultas, podremos diseñar, ejecutar y almacenar consultas utilizando un formato gráfico muy similar al que nos podemos encontrar en Microsoft Access. Gracias a que la conexión con la base de datos es activa, podremos editar, insertar y borrar información en la base de datos. Además, Visual Interdev no permitirá crear, editar y modificar directamente objetos de la base de datos tales como tablas, vistas y procedimientos almacenados en tiempo de diseño.

Además de todo esto, Visual Interdev pone a disposición del programador una serie de controles en tiempo de diseño enlazados a datos que nos permitirán seleccionar un conjunto de registros de la base de datos y mostrarlos en una página .html o .asp, con tan sólo arrastrar el control a la página correspondiente e indicarle al control el enlace a datos correspondiente.

### Agregar una conexión a datos

Para poder acceder y manipular la base de datos, lo primero será establecer una conexión con la misma. Para ello se debe crear una DSN (Data Source Name) de sistema en el cliente y en el servidor. En el servidor es necesaria para poder acceder a ella a través de conexiones en nuestra página ASP, pero la de cliente únicamente es necesaria para Visual Interdev. Esta DSN de cliente es necesaria para que desde Visual Interdev podamos tener acceso a la base de datos en tiempo de diseño.

Partiendo del supuesto de que ya disponemos de una DSN de sistema en el servidor llamada "FuenteBD", que nos va a permitir acceder a la base de datos "Pubs" de SQL Server. Los pasos para crear una DSN de cliente y establecer la conexión con la base de datos antes comentada son los siguientes:

- En el menú Proyecto, seleccionamos Agregar conexión de datos. Aparecerá el cuadro de diálogo Seleccionar origen de datos (Figura 141).

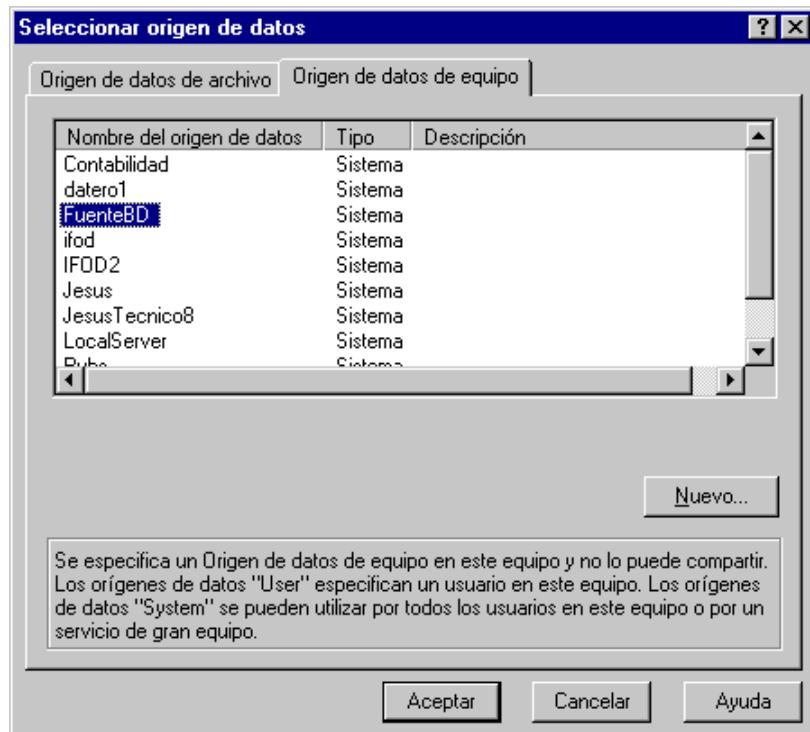


Figura 141

- Dentro de la ficha Origen de datos de equipo, pinchamos en el botón Nuevo.
- En el primer cuadro de diálogo que nos aparecerá seleccionaremos: Origen de datos de sistema, con lo cual crearemos un origen de datos específico del equipo y utilizable por cualquier usuario.



Figura 142

- Seleccionaremos un controlador para el origen de datos. En nuestro caso: SQL Server:

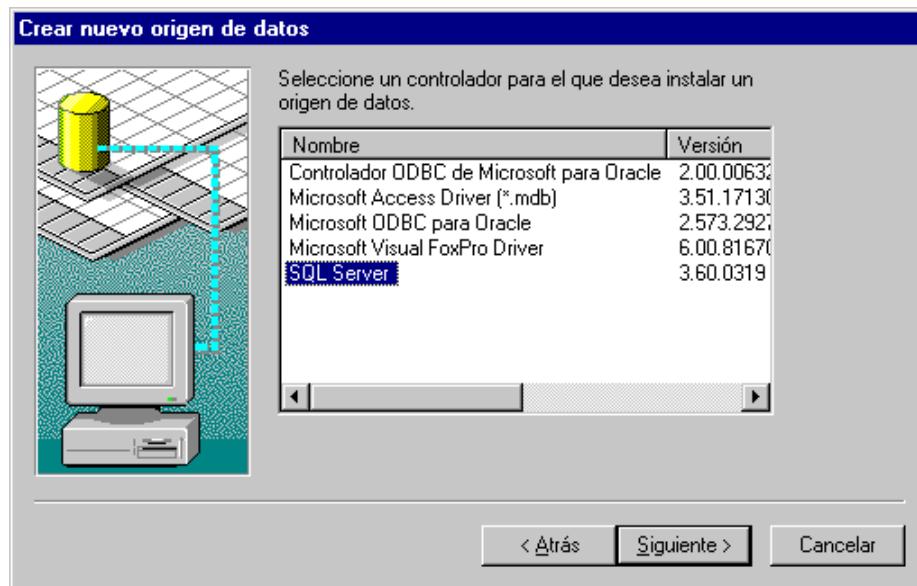


Figura 143

- En el resto de cuadros de diálogo tendremos que suministrar, entre otros datos: el nombre del origen de datos ("FuenteBD"), el servidor SQL Server en el que residen los datos a los que vamos a acceder, la forma en que SQL Server ha de comprobar la autenticidad del Id. de inicio de sesión y finalmente el nombre de la base de datos a la que vamos a acceder ("Pubs"). En el resto de cuadros de diálogo podríamos aceptar la opciones por defecto.
- Al finalizar todo el proceso de creación de la DSN de sistema en el cliente, regresaremos al cuadro de diálogo Seleccionar origen de datos, en el cual ya aparecerá el nombre del origen de datos que acabamos de crear. Selecciona dicho nombre y pincha en el botón Aceptar. Aparecerá un cuadro de diálogo con los parámetros de la conexión de datos, denominado Inicio de sesión para SQL Server:

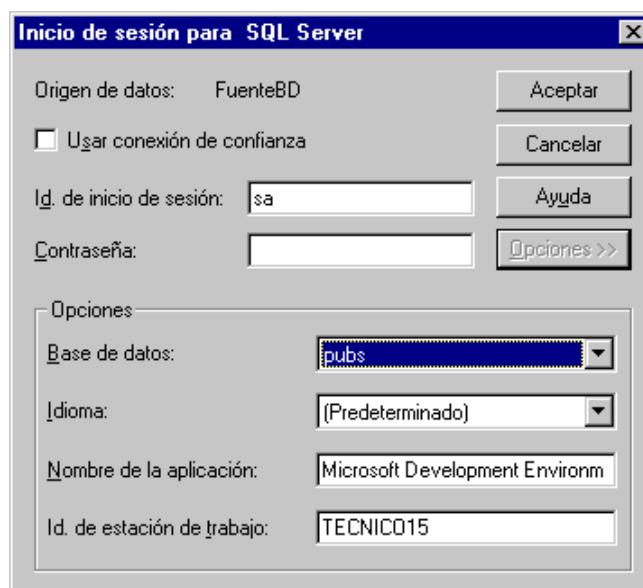


Figura 144

- Al aceptar el cuadro de diálogo anterior, obtendremos otro cuadro de diálogo correspondiente a la conexión que se va a crear y a las propiedades de la misma. Dale un nombre a la conexión y pincha en Aceptar.

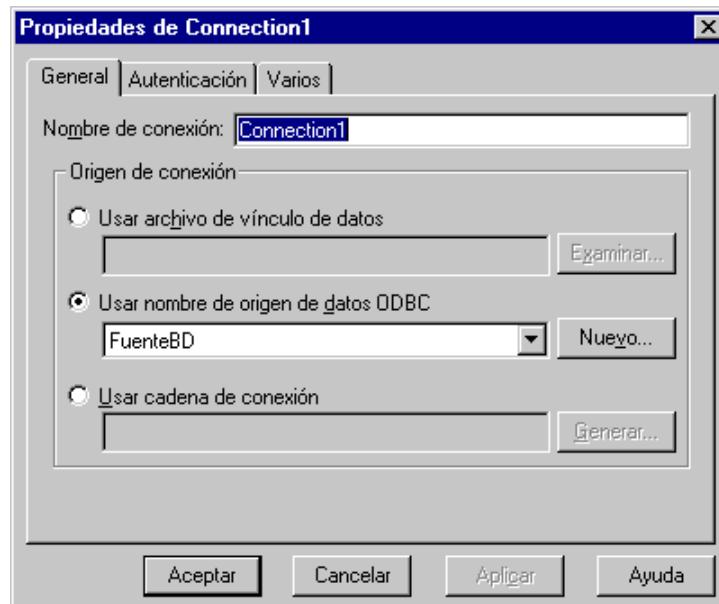


Figura 145

Después de todos los pasos anteriores, ya disponemos de una conexión a la base de datos desde nuestro proyecto. La conexión de datos aparecerá bajo el ícono DataEnvironment del proyecto, debajo del archivo GLOBAL.ASA.

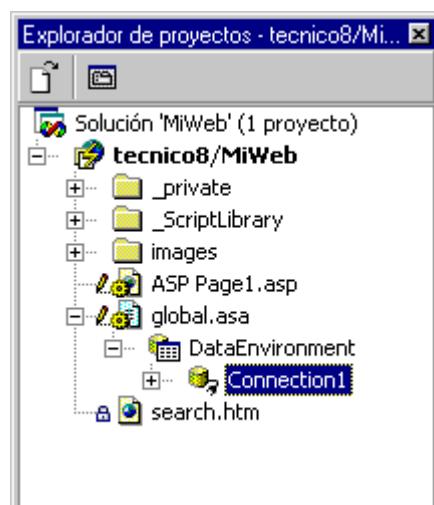


Figura 146

La conexión está asociada al fichero GLOBAL.ASA, ya que es allí donde se encuentran definidos los parámetros de la conexión. Si hacemos doble clic sobre el fichero GLOBAL.ASA podremos visualizar su contenido.

```

<SCRIPT LANGUAGE='VBScript' RUNAT='Server'>
'Puede agregar controladores de evento especiales a este archivo que se ejecuten
automáticamente al
'producirse eventos de páginas Active Server. Para ello, cree una subrutina y déle
un nombre de la
'lista siguiente que se corresponda con el evento que desea utilizar. Por ejemplo,
para crear un
'controlador de evento para Session_OnStart, debe insertar el siguiente código en
'este archivo (sin comentarios) :

'Sub Session_OnStart
'***Inserte el código aquí ***
'End Sub

'EventName           Descripción
'Session_OnStart     Se ejecuta la primera vez que un usuario abre cualquier
página de la aplicación
'Session_OnEnd       Se ejecuta cuando finaliza el tiempo de espera de la sesión
de un usuario o éste
'sale de la aplicación
'Application_OnStart Se ejecuta una vez cuando un usuario abre por primera vez la
primera página de
la aplicación
'Application_OnEnd   Se ejecuta una vez cuando se apaga el servidor Web
</SCRIPT>

<SCRIPT LANGUAGE=VBScript RUNAT=Server>
Sub Application_OnStart
    '==Visual InterDev Generated - startspan==
    '--Project Data Connection

    Application("Connection1_ConnectionString") =
    "DSN=FuenteBD;UserId=sa;PASSWORD=;SERVER=TECNICO8;UID=sa;APP=Microsoft Development
    Environment;WSID=TECNICO15;DATABASE=pubs"
    Application("Connection1_ConnectionTimeout") = 15
    Application("Connection1_CommandTimeout") = 30
    Application("Connection1_CursorLocation") = 3
    Application("Connection1_RuntimeUserName") = "sa"
    Application("Connection1_RuntimePassword") = ""

    '-- Project Data Environment
    'Set DE = Server.CreateObject("DERuntime.DERuntime")
    'Application("DE") = DE.Load(Server.MapPath("Global.ASA"),
    "_private/DataEnvironment/DataEnvironment.asa")
    '==Visual InterDev Generated - endspan==
End Sub
</SCRIPT>

```

Código fuente 352

Como podemos ver, toda la información necesaria para establecer la conexión se guarda en variables del objeto Application.

## La vista de datos

Si pinchamos dos veces sobre nuestra conexión "Connection1" accederemos a la Vista de datos, a través de la cual podremos explorar, editar y modificar directamente los distintos objetos de la base de datos asociada a la conexión.

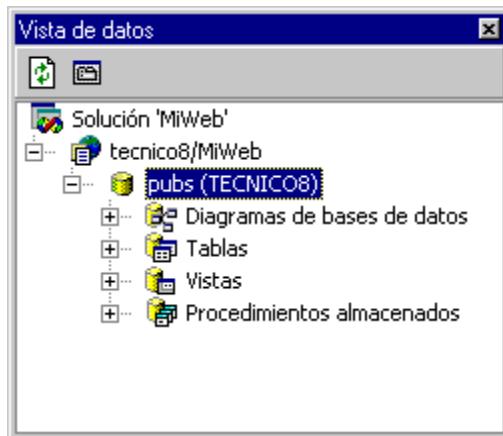


Figura 147

En cualquier momento podemos expandir el árbol de objetos asociados a la base de datos, seleccionar algún objeto y con el botón derecho acceder a un menú contextual a través del cual realizar distintas operaciones sobre el elemento seleccionado.

Siguiendo este procedimiento, podríamos abrir cualquier tabla de la base de datos y dar de alta, baja o modificar registros, o bien acceder al diseño de la tabla y modificar las características de alguno de sus campos, añadir campos nuevos, definir campos clave etc:

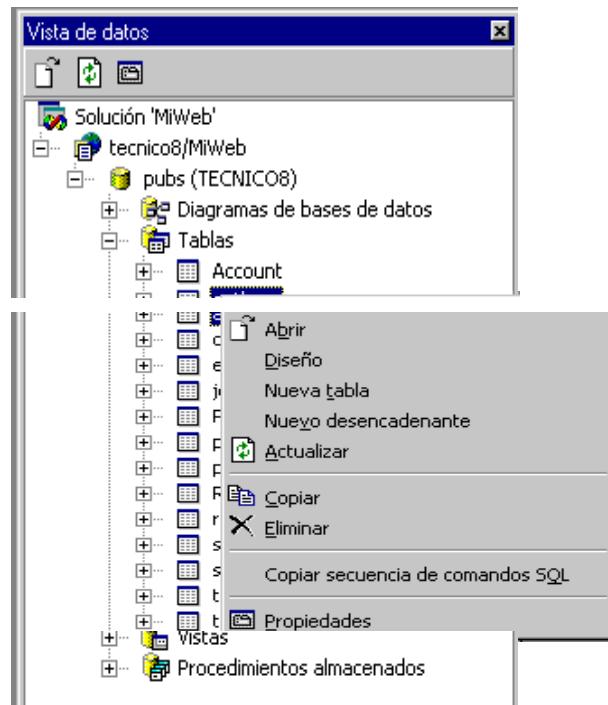


Figura 148

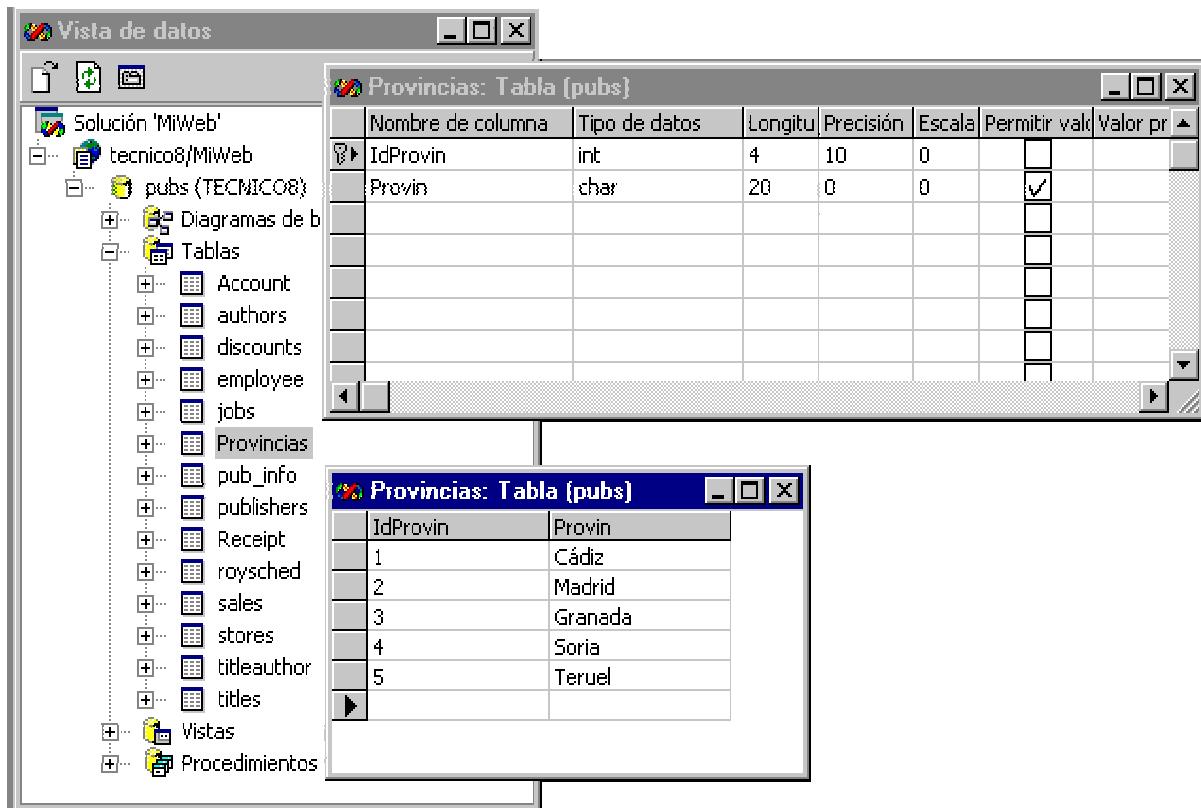


Figura 149

## El Diseñador de Consultas

A través del diseñador de consultas podremos diseñar, ejecutar y almacenar consultas, todo ello en un formato gráfico similar al que nos podemos encontrar en Microsoft Access.

Para acceder a este diseñador active la opción Consulta del menú Ver/Barras de herramientas, una vez activa la barra de herramientas de consulta abra la primera tabla con la vaya a trabajar desde la vista de datos.

El diseñador de consultas consta de cuatro paneles: el panel de diagrama, el panel de cuadrícula, el panel SQL y el panel de resultados, todos ellos activables desde la barra de herramientas de consulta.

El panel de diagrama es utilizado para mostrar las tablas de la base de datos. Podríamos manipular tablas de la base de datos arrastrando la tabla deseada desde la vista de datos hasta el panel de diagrama. Una vez tengamos las tablas en este panel podríamos establecer relaciones entre las distintas tablas.

El panel de cuadrícula es usado para mostrar los campos seleccionados en forma de columnas. El panel SQL muestra la consulta SQL en su formato correspondiente. Una vez ejecutada la consulta el resultado de la misma se muestra en el panel de resultados.

Estos cuatro paneles se encuentran completamente sincronizados, es decir, se actualizan todos ellos automáticamente cuando se produce alguna modificación en cualquiera de ellos.

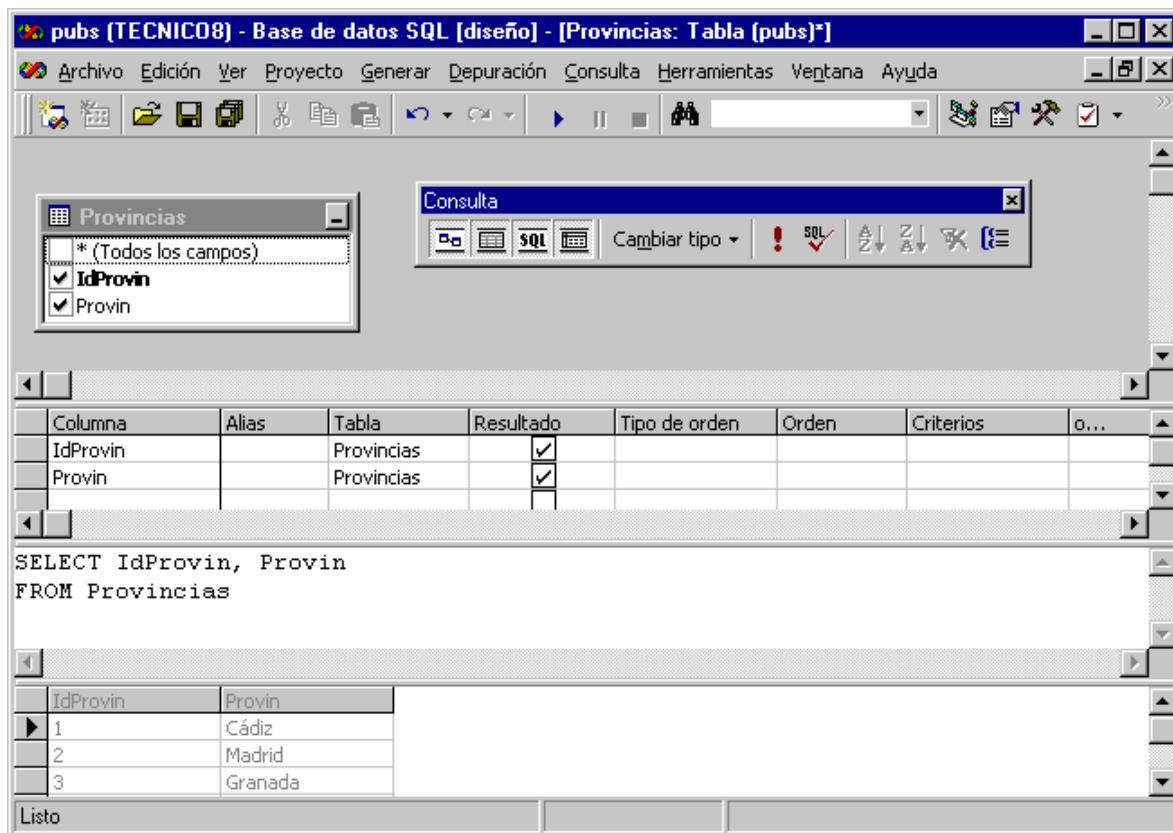


Figura 150

## El Diseñador de bases de datos

A través de esta herramienta podremos, de una manera directa, crear, editar y borrar objetos de bases de datos, utilizando para ello diagramas de bases de datos. Estos diagramas ofrecen un método visual para representar tablas, columnas y relaciones de la base de datos.

Para crear un diagrama nuevo seleccionaremos la rama del árbol de la vista de datos correspondiente a "Diagramas de bases de datos" y con el botón derecho accederemos a la opción de menú "Nuevo Diagrama". Después de esto tendremos una ventana de Diagrama de bases datos vacía, a la cual podemos arrastrar, desde la vista de datos, las tablas que conformarán el diagrama.

En el diagrama de la base de datos podremos modificar las propiedades de las tablas; crear, modificar o borrar relaciones entre tablas; crear tablas nuevas; eliminar tablas de la base de datos; crear índices y claves de tablas.

Según se va realizando modificaciones en el diagrama de la base de datos, Visual Interdev va generando un script SQL que recoge todos los cambios realizados. Si cerramos el diagrama y elegimos guardar cambios Visual Interdev ejecutará el script y los cambios se realizarán sobre la base de datos. Si se quiere grabar o visualizar el script generado deberemos pulsar el botón Guardar archivo de cambios dentro de la barra de herramientas Diagrama de base de datos.

Además de los elementos antes comentados, Visual Interdev pone a nuestra disposición un Diseñador de Vistas el cual nos permitirá crear visualmente la instrucción SQL que define una vista, y un Editor de procedimientos almacenados. A ambos elementos podremos acceder desde el árbol de la vista de datos.

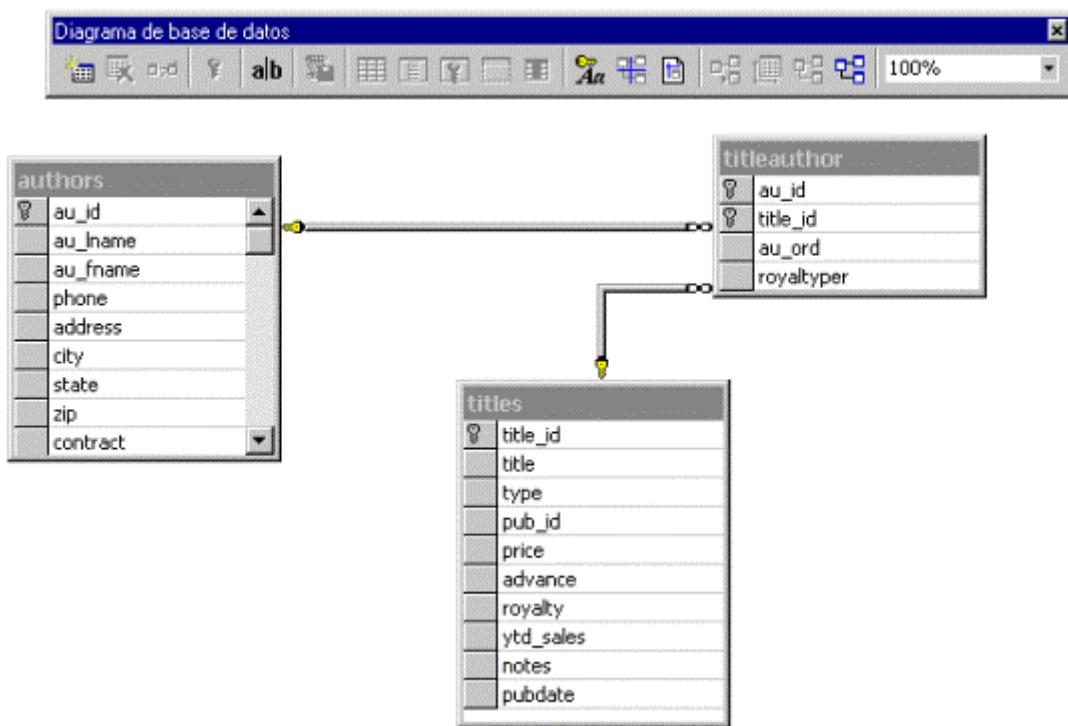


Figura 151

## Controles en tiempo de diseño

Visual Interdev 6.0 incluye una serie de controles en tiempo de diseño enlazados a datos. Una vez establecida en el proyecto Web una conexión con la base de datos, a través de los controles en tiempo de diseño podremos fácilmente presentar y editar los datos, simplificando en gran medida la programación de la aplicación.

Para poder trabajar con los controles en tiempo de diseño se ha de habilitar el modelo de objetos de secuencias de comandos de Visual Interdev. A través de este modelo de objetos se proporciona una interfaz para la automatización del acceso a bases de datos, creando un modelo de alto nivel para el acceso y manipulación de datos.

Entre los controles en tiempo de diseño podemos destacar los siguientes:

- **Recordset:** A través de este control podremos hacer referencia a un objeto de la base de datos y extraer un conjunto de registros. Funciona como el control principal a través del cual se enlazarán a los datos el resto de controles.
- **Controles individuales:** A través de estos controles podremos mostrar el contenido de un campo de la base de datos. Son lo cuadros de texto, cuadros de lista, botones de radio y casillas de verificación.
- **RecordsetNavBar:** A través de este control podremos fabricarnos los típicos botones de navegación a través de registros: desplazamiento al registro anterior, siguiente, primero y último.
- **Grid:** Nos permitirá mostrar un conjunto de registros a través de una tabla.

Veamos algún ejemplo ilustrativo del manejo de estos controles: Con nuestra conexión ya establecida con la base de datos pubs, creamos una página nueva Datos.asp y activamos la vista Diseño en la ventana de la página. Además activamos el cuadro de herramientas seleccionando la opción Cuadro de herramientas del menú Ver y dentro del cuadro de herramientas la pestaña Controles en tiempo de diseño. Finalmente organizaremos la ventanas como muestra la Figura 152.

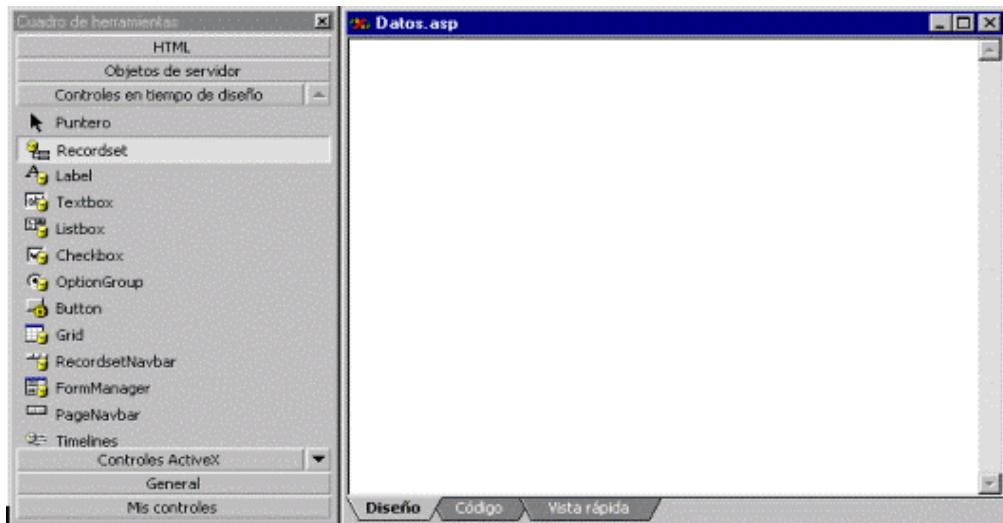


Figura 152

Vamos ahora a agregar un control Recordset a la página. A través de este control estableceremos el enlace con los datos, sirviendo de puente para el acceso a datos del resto de controles. En nuestro ejemplo vamos a establecer un enlace con los datos de la tabla Provincias de la base de datos pubs. Para ello simplemente arrastramos el control Recordset desde el cuadro de herramientas a la vista diseño de la página Datos.asp y respondemos que sí a la pregunta de si habilitamos el modelo de objetos de secuencias de comandos Visual Interdev.

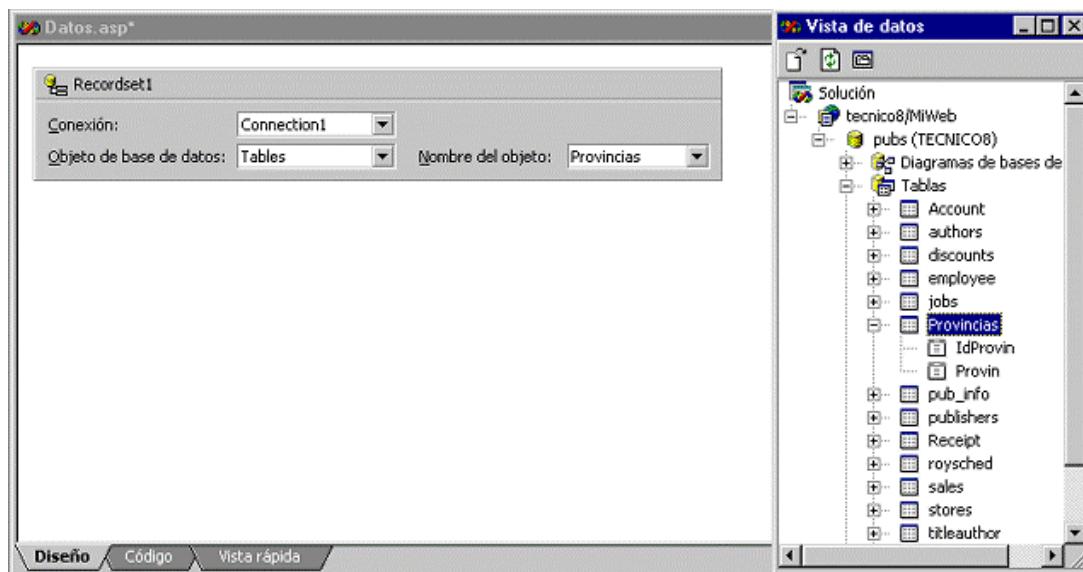


Figura 153

Una vez insertado el control Recordset basta con indicarle la conexión a la que están asociados los datos (Connection1), el tipo de objeto del que vamos a recuperar los datos (Tables) y el nombre del objeto (Tabla Provincias). Vamos ahora a incluir en la página dos controles Textbox a través de los cuales vamos a visualizar el contenido de los dos campos de la tabla: IdProvin y Provin. Para ello simplemente arrastramos los controles Textbox desde el cuadro de herramientas a sus posiciones correspondientes en la página DATOS.ASP.



Figura 154

Una vez insertado los controles Textbox, debemos seleccionarlos y con el botón derecho del ratón acceder a sus propiedades.

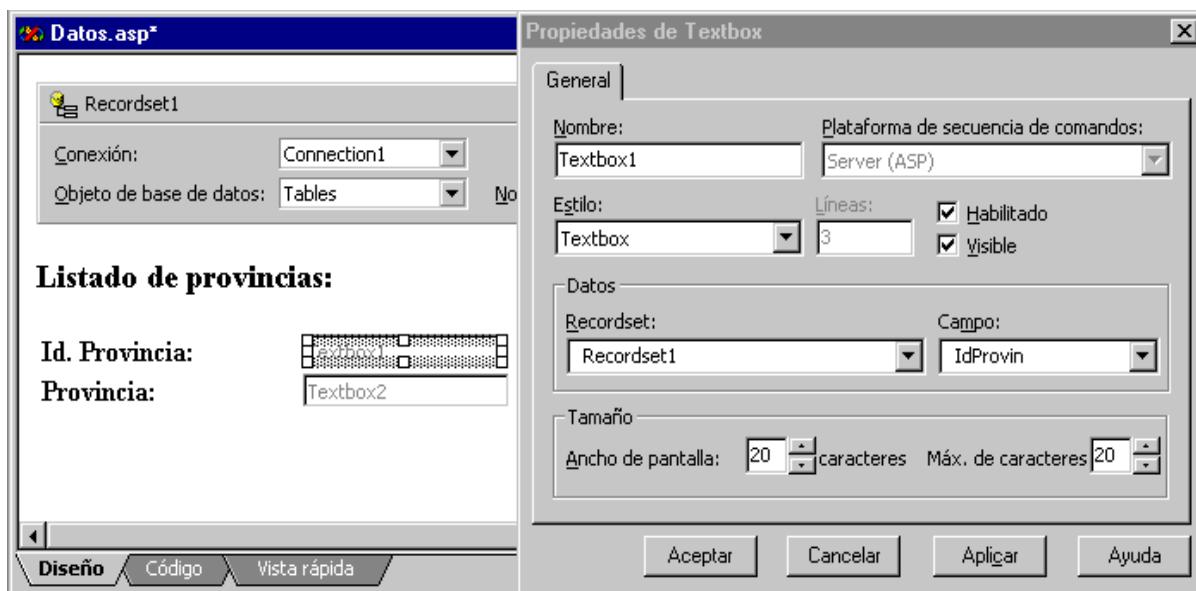


Figura 155

En propiedades de cada textbox debemos indicar el recordset al que está asociado (Recordset1) y el campo con el que está enlazado (IdProvin o Provin).

Después de esto, ya podría cargar la página en el explorador y ver el resultado:

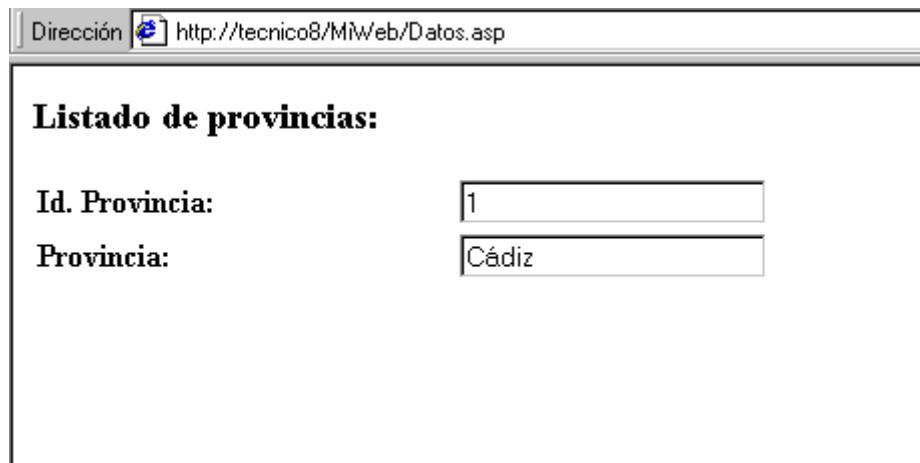


Figura 156

Vamos a completar el ejemplo incluyendo unos botones de navegación que nos permitan desplazarnos por los registros de la tabla provincias, para ello simplemente tenemos que arrastrar un control RecordsetNavBar desde el cuadro de herramientas a la página DATOS.ASP y configurar sus propiedades:

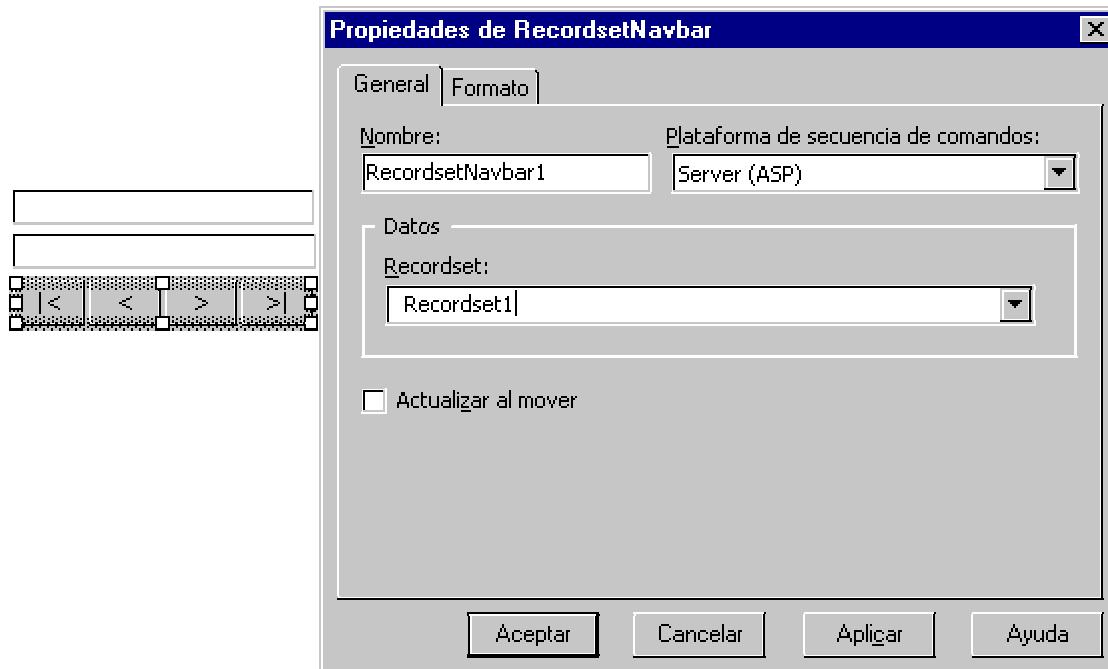


Figura 157

El resultado sería el siguiente el que muestra la Figura 158.

Como ejemplo del uso del control Grid vamos a crear una nueva página llamada Grid.asp y le vamos a incluir un control de este tipo, enlazándolo con los datos de la tabla Authors de la base de datos Pubs.

Dirección <http://tecnico8/MiWeb/Datos.asp>

### Listado de provincias:

<b>Id. Provincia:</b>	<input type="text" value="5"/>
<b>Provincia:</b>	<input type="text" value="Teruel"/>
<input type="button" value=" &lt;"/> <input type="button" value="&lt;"/> <input type="button" value="&gt;"/> <input type="button" value=" &gt;"/>	

Figura 158

La secuencia de trabajo es análoga a la seguida en el ejemplo anterior:

- Primero arrastramos a la vista diseño de la página Grid.asp un control Recordset y lo enlazamos con la tabla Authors asociada a la conexión Connection1.
- Seguidamente arrastramos sobre la página un control Grid y accedemos a sus propiedades.
- En el cuadro de diálogo de propiedades del control Grid activamos la ficha Datos, y en ella indicamos el recordset al que estará enlazado el control (Recordset1) y los campos a visualizar (au\_id, au\_lname, au\_fname, phone). El resto de opciones del cuadro de diálogo las podríamos aceptar por defecto.

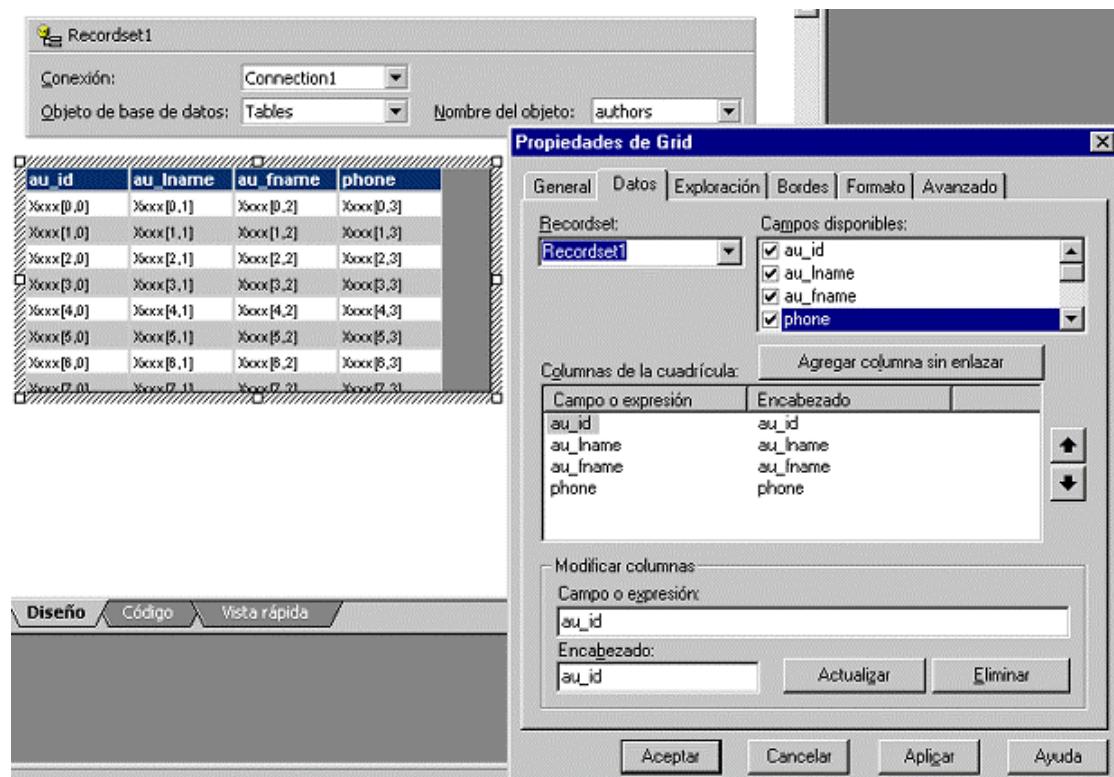
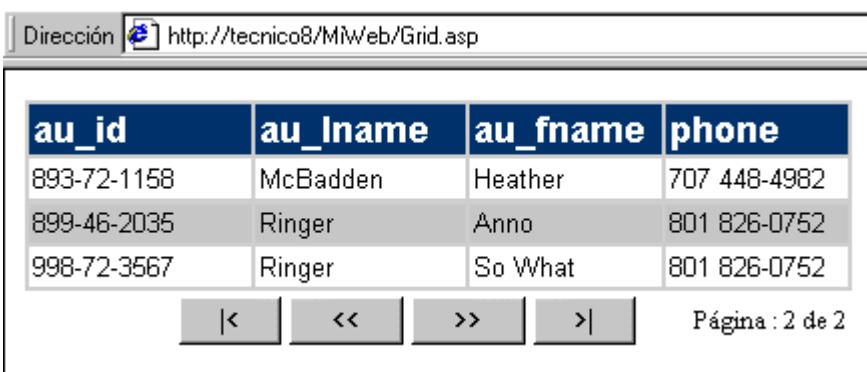


Figura 159

Si cargamos la página GRID.ASP en el navegador, el resultado sería el que muestra la Figura 160.



The screenshot shows a web browser window with the URL <http://tecnico8/MiWeb/Grid.asp> in the address bar. The page displays a data grid with four columns: **au\_id**, **au\_lname**, **au\_fname**, and **phone**. The data is as follows:

<b>au_id</b>	<b>au_lname</b>	<b>au_fname</b>	<b>phone</b>
893-72-1158	McBadden	Heather	707 448-4982
899-46-2035	Ringer	Anno	801 826-0752
998-72-3567	Ringer	So What	801 826-0752

Below the grid are navigation buttons: <|, <<, >>, and >|. To the right of the buttons, it says "Página : 2 de 2".

Figura 160

## Proyecto de base de datos

Si queremos manipular y tener una conexión a una base de datos, no es indispensable crear un proyecto Web previamente y después añadir una conexión de datos al mismo. Tenemos la posibilidad de crear un proyecto de base de datos.

Mediante este tipo de proyecto se ofrece un enlace vivo, es decir, una conexión en tiempo real con la base de datos indicada en la fuente ODBC elegida.

Para crear un proyecto de este tipo debemos elegir la opción Nuevo Proyecto... del menú Archivo de Visual Interdev. En el cuadro de diálogo Nuevo proyecto, dentro de la ficha Nuevo, desplegar la opción Proyectos de base de datos. Seleccionar en el panel derecho Nuevo proyecto de base de datos y finalmente especificar un nombre y ubicación para el mismo:

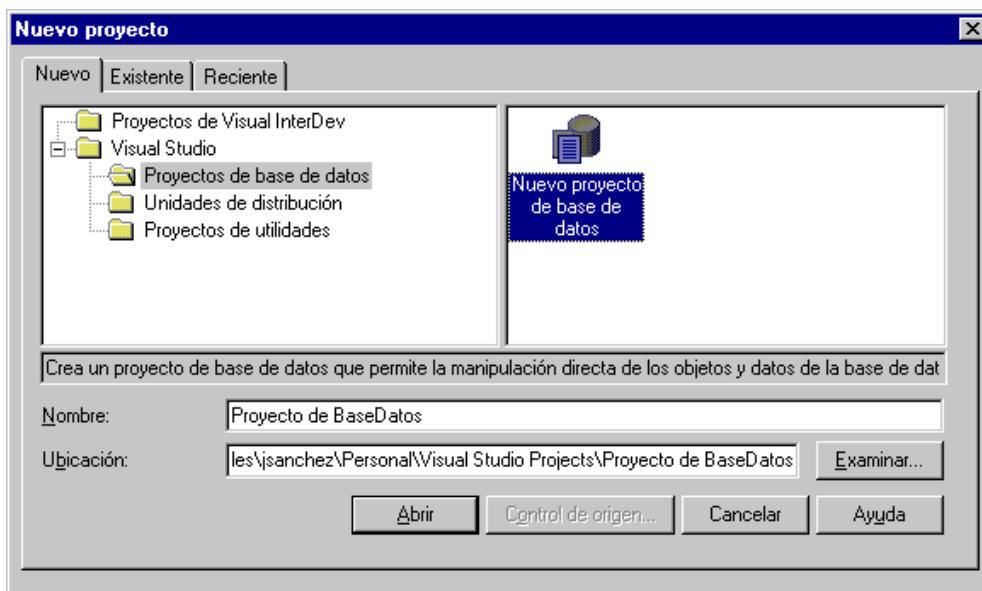


Figura 161

A continuación se nos dará la opción de crear una fuente ODBC nueva o bien de seleccionar una existente. Nosotros vamos a usar la fuente ODBC "FuenteBD" que ya creamos en apartados anteriores.

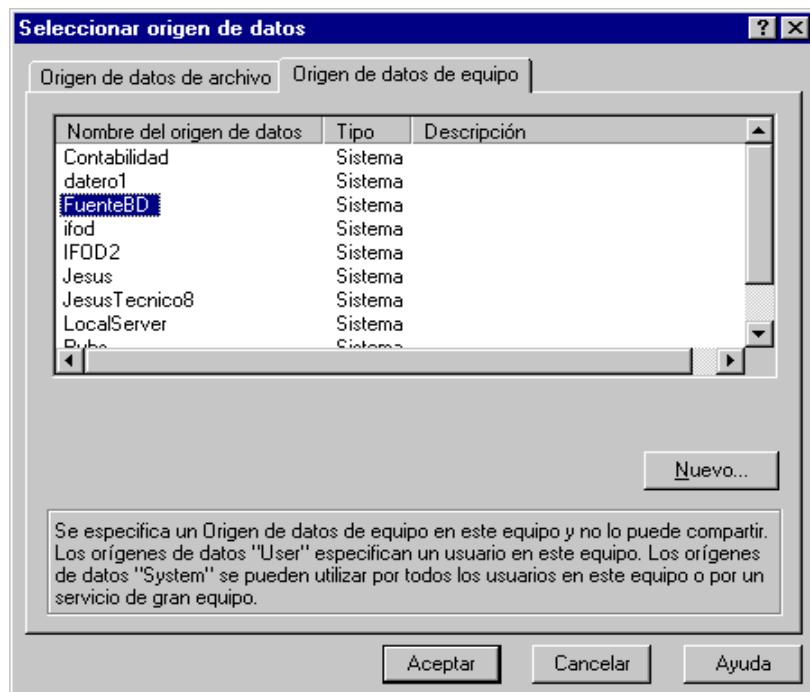


Figura 162

Después de esto, disponemos de un proyecto de base de datos desde el cual podemos manipular la base de datos asociada a la DSN "FuenteBD" y realizar sobre ella todas las operaciones que tengamos permitidas. Podremos utilizar el diseñador de consultas, el diagrama de bases de datos, entrar directamente en el diseño de las tablas, etc.

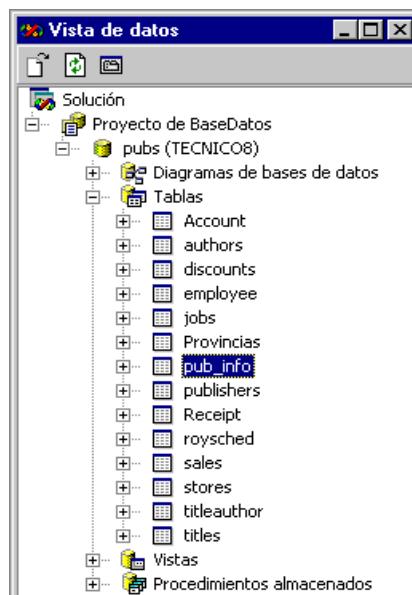


Figura 163

## Desarrollo de aplicaciones web en equipo

Ya sabemos que en Visual Interdev los proyectos administran dos copias de la aplicación Web: maestra y local, dando lugar a los modos de trabajo Maestro y Local.

En el modo de trabajo Maestro todos los ficheros de la aplicación se almacenan en el servidor Web. En el modo Local se realizan los cambios en una copia con permiso de escritura de la aplicación, sin afectar a la aplicación maestra.

Debido a que en el modo de trabajo Maestro, los ficheros se encuentran almacenados en el servidor Web, varias personas pueden trabajar en el mismo proyecto al mismo tiempo, es decir, pueden hacer referencia a la misma aplicación Web maestra. De esta forma un miembro de un equipo de desarrollo puede estar editando una página ASP para introducir contenidos HTML, mientras que otro miembro está realizando la programación de esa misma página.

En el modo de trabajo Maestro se pueden dar problemas de concurrencia, es decir, que un miembro del equipo de desarrollo interfiera con el trabajo de otro. Para solventar esto, se podría trabajar en modo Local para crear nuevas versiones de la aplicación Web sin cambiar los archivos Web maestros y, de esta forma, no interferir con el trabajo realizado por otros programadores.

Para trabajar en modo Local, se deben seguir los siguientes pasos:

- Seleccionar el proyecto Web en el explorador de proyectos y pulse el botón derecho del ratón.
- En el menú contextual seleccione: Modo de trabajo / Local.

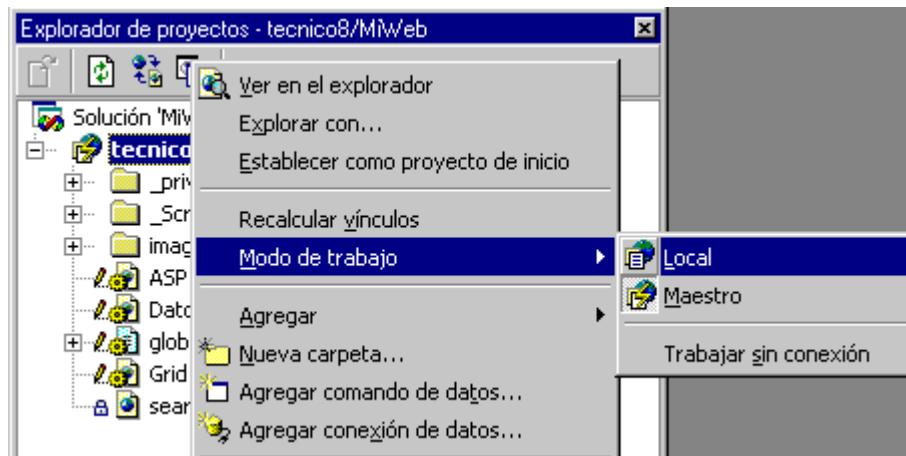


Figura 164

Después de esto, los cambios realizados en cualquier página del proyecto Web no afectarán a la copia maestra que reside en el servidor. Si queremos obtener una vista previa en el programa navegador de alguna página del proyecto, y no existe un servidor Web en la estación de trabajo, la página se cargará con una dirección URL de fichero (file:/// ) y el script se servidor no funcionará.

Trabajando en el modo Local, en cualquier momento podremos chequear las diferencias entre las versiones local y maestra de una página. Para ello simplemente seleccionamos la página a chequear en el explorador de proyectos y en el menú contextual que obtenemos al pulsar el botón derecho

seleccionamos la opción: Comparar con el Web maestro. Obtendremos una caja de diálogo informándonos de las diferencias entre ambas versiones de la página.

```

Diferencias entre ispA.tmp y ASP Page1.asp.

C:\...\Visual Studio Projects\MiWeb\ispA.tmp          C:\...\jsanchez\Personal\Visual Studio Projects\MiWeb\MiWeb_Local\ASP Page1.asp
1  <%@ Language=VBScript %>                         1  <%@ Language=VBScript %>
2  <HTML>                                           2  <HTML>
3  <HEAD>                                         3  <HEAD>
4  <META NAME="GENERATOR" Content="Microsoft Visual St 4  <META NAME="GENERATOR" Content="Microsoft Visual St
5  </HEAD>                                         5  </HEAD>
6  <BODY>                                         6  <BODY>
7  <P>&nbsp;</P>                                     7
8  <P>&nbsp;</P>                                     8  <h1>Página modificada en el modo de trabajo local.
9  <P>&nbsp;</P>                                     9
10 </BODY>                                         10 </BODY>
11 </HTML>                                         11 </HTML>
12

```

Líneas eliminadas | Líneas modificadas | Líneas insertadas | Lín 9, Col 1

Figura 165

Cuando se considere que se ha completado la versión trabajo de la aplicación Web, podríamos actualizar la versión maestra. Para ello:

- Seleccione el proyecto Web en el explorador de proyectos y pulse el botón derecho del ratón.
- En el menú contextual seleccione: Modo de trabajo / Maestro.

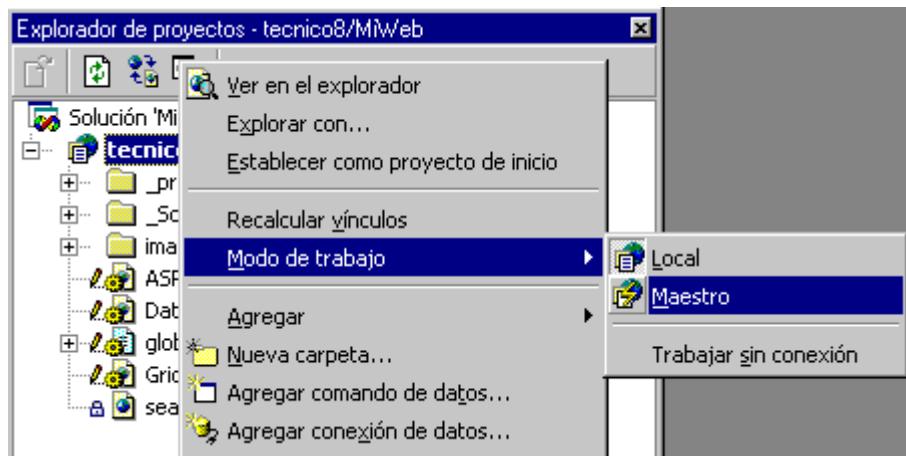


Figura 166

Después de esto, la copia maestra del proyecto Web se actualizará con la versión modificada de todas las páginas de la copia local y se volverá al modo de trabajo Maestro.

## Distribuir la aplicación

Una vez se ha completado la aplicación Web, podemos hacerla disponible para los usuarios finales en un servidor de producción. Al distribuir una aplicación Web lo que estamos haciendo es un duplicado de la aplicación en otra ubicación. Para llevar a cabo este proceso de distribución el servidor de producción debe tener instaladas las Extensiones de servidor de FrontPage.

Antes de distribuir la aplicación, debemos tener en cuenta una serie de puntos:

- Comprobación del correcto funcionamiento de los hiperenlaces entre las páginas de la aplicación. Para ello lo recomendable es el uso de rutas de acceso relativas.
- Asegurar la transportabilidad de los datos y la conexión de los datos a la base de datos de producción.
- Asegurarnos de incluir en el servidor de producción todos los archivos de los que hace uso la aplicación.
- Registrar en el servidor de producción todos los componentes necesarios.
- Comprobar que el servidor de producción dispone de todo el software necesario para el funcionamiento de la aplicación a distribuir, como por ejemplo la extensiones de servidor de FrontPage, los controladores ODBC necesarios, etc.
- Configurar el sistema de permisos del sistema operativo y del servidor Web de cara a la correcta publicación de la aplicación.

Una vez nos hemos asegurado que la aplicación está preparada para su distribución, seguiremos los siguientes pasos:

- Seleccionamos la aplicación a distribuir en el Explorador de proyectos.

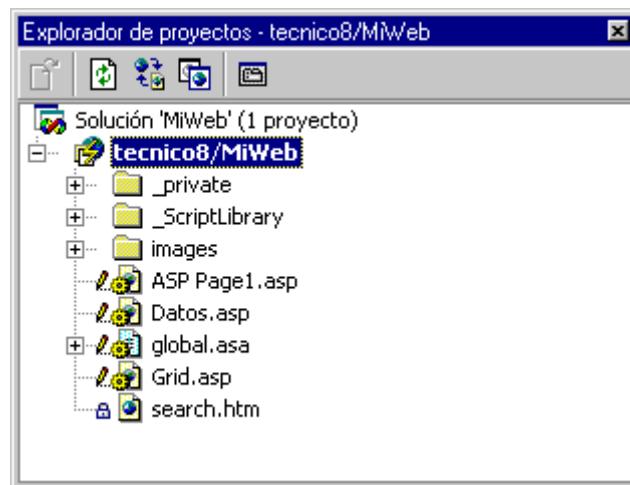


Figura 167

- Seleccionamos la opción Proyecto / Proyecto Web / Copiar Aplicación Web... Lo cual nos llevará al cuadro de diálogo Copiar Proyecto.

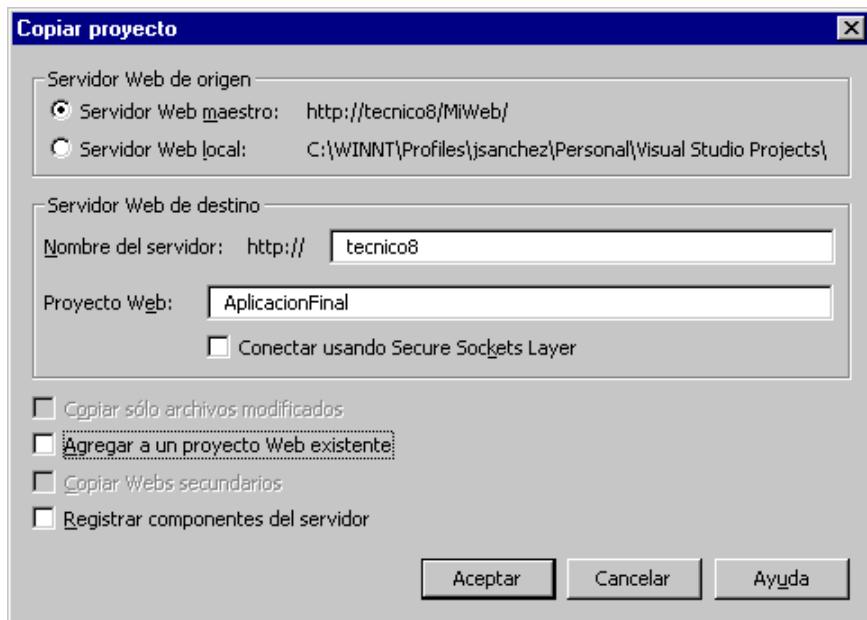


Figura 168

En este cuadro de diálogo habría que especificar el nombre del servidor de producción en el que se publicará nuestra aplicación de cara al usuario final (supongamos que nuestro servidor de producción es <http://tecnico8>) y el nombre de la aplicación en el servidor (AplicacionFinal).

Después de todo este proceso nuestra aplicación ya está disponible para los usuarios en el servidor de producción. Así por ejemplo, cualquier usuario podría cargar la página DATOS.ASP desde su programa navegador en la nueva ubicación:

<http://tecnico8/AplicacionFinal/Datos.asp>

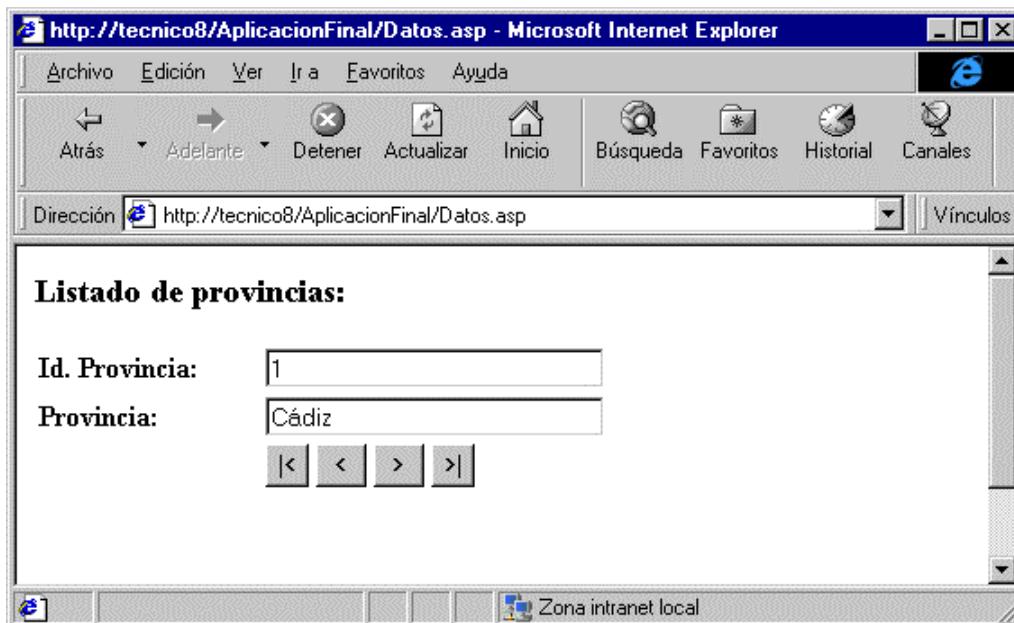


Figura 169



# ASP y servicios de componentes

---

## Introducción

En este último tema vamos a mostrar como crear nuestros propios componentes de servidor con Visual Basic 6, para luego utilizarlos desde nuestras aplicaciones ASP. Comentaremos también los distintos modelos de aplicaciones.

También veremos como utilizar estos componentes dentro de páginas ASP transaccionales y cómo registrarlos y configurarlos en Servicios de componentes de Windows 2000, que se podría decir que es la nueva versión del servidor de transacciones Microsoft Transaction Server 2.0, que se ofrece como parte de Windows 2000.

## Creando componentes ActiveX Server

Además de poder contar con los distintos tipos de componentes ActiveX de servidor y con los componentes de terceras partes, como hemos visto en capítulos anteriores, también podemos crear nuestros propios componentes ActiveX Server o componentes de servidor.

Cualquier lenguaje con el que se puedan crear controles ActiveX como puede ver Visual C++, Visual Basic, Visual J++ y Delphi, nos servirá para construir controles ActiveX Server. A continuación se va a mostrar paso a paso el proceso de creación de un control ActiveX con Visual Basic 6.0. Se ha elegido esta herramienta de desarrollo por su similitud del lenguaje utilizado con VBScript.

El primer paso para crear nuestro componente es iniciar VB 6.0 y en la ventana de Nuevo Proyecto elegir la opción ActiveX DLL y pulsar el botón etiquetado como Abrir. Por defecto VB crea un

proyecto llamado Project1, cambiaremos el nombre del proyecto por otro más descriptivo, como puede ser SimpleCompo. Acto seguido se debe modificar el nombre de la clase del proyecto (Class1) por el nombre Multiplica.

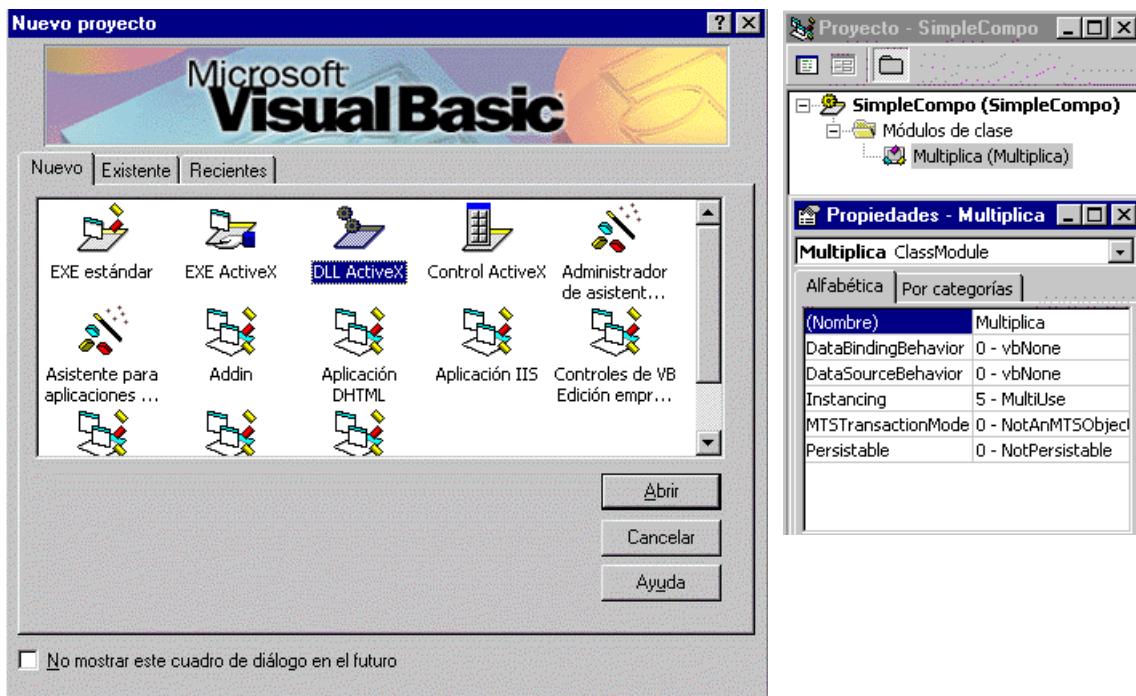


Figura 170. Creación de un ActiveX en VB 6.0

Una vez realizadas estas tareas iniciales ya podremos empezar a escribir código para nuestro componente. En este primer ejemplo vamos ha realizar un componente bastante simple que únicamente realiza la multiplicación de dos números que se pasan como parámetro, lo importante no es la complejidad del componente sino tener claros los diferentes pasos que se debe seguir. El código que debemos escribir dentro de la clase Multi es el Código fuente 353.

```
Function Multi(X As Variant, Y As Variant) As Variant
    Multi = X * Y
End Function
```

Código fuente 353

Una vez escrito este código debemos elegir la opción Generar SimpleCompo.dll dentro de la opción de menú Archivo. Esto creará la librería, fichero .DLL, que contendrá a nuestro componente.

Para registrar el componente en el servidor Web, se debe mover el fichero SimpleCompo.dll al subdirectorio: winnt\system32 dentro del servidor Web, y ejecutar la utilidad regsvr32. En este caso desde la línea de comandos debemos ejecutar el siguiente comando: regsvr32 SimpleCompo.dll, acto seguido aparecerá una ventana indicándonos si el componente se ha registrado con éxito.

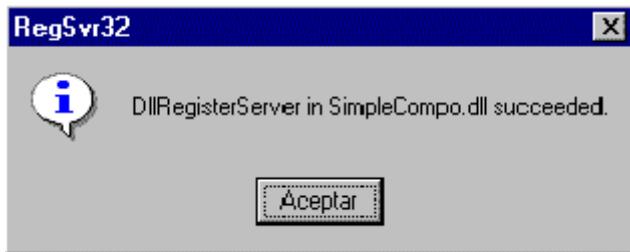


Figura 171. El componente se ha registrado.

Una vez que se ha registrado el componente ActiveX, en el servidor, ya pasa a ser un ActiveX Server que podemos utilizar desde una página ASP utilizando para ello la sentencia `Server.CreateObject`. En el Código fuente 354 se ofrece un código de ejemplo que utiliza nuestro componente ActiveX de servidor.

```
<HTML>
<HEAD>
    <TITLE>Instanciación de un componente ActiveX Server</TITLE>
</HEAD>

<BODY>
<%X=5
Y=50%>
El valor de X:<%=X%><br>
El valor de Y:<%=Y%>
<p>&nbsp;</p>
<%Set objeto=Server.CreateObject("SimpleCompo.Multiplica")%>
El Resultado de X*Y es: <%=objeto.Multi(X,Y)%>
</BODY>
</HTML>
```

Código fuente 354

En el siguiente ejemplo se construye un componente ActiveX de servidor con varias funciones en las que permite dar un formato a la entrada que se le pasa. Las funciones que ofrece nuestro componente son: a partir de una hora en formato AM/PM devuelve la hora completa; a partir de una fecha en formato día/mes/año, devuelve la fecha completa y a partir de un texto lo convierte todo a mayúsculas. Para realizar estas funciones se utiliza la función `Format` de Visual Basic, es decir, estamos encapsulando funciones propias de un lenguaje determinado dentro de un control ActiveX de servidor que vamos a poder utilizar desde nuestra aplicación ASP.

En este caso se han seguido los mismos pasos que en el ejemplo anterior, el proyecto se ha nombrado como Componente2 y la clase como Formatos. En el Código fuente 355, se muestra el código incluido en esta clase.

```
Public Function HoraCompleta(hora As Variant) As Variant
    HoraCompleta = Format(hora, "hh:mm:ss")
End Function

Public Function FechaCompleta(fecha As Variant) As Variant
    FechaCompleta = Format(fecha, "dddd, mmmm d yyyy")
End Function

Public Function Mayusculas(texto As Variant) As Variant
```

```

    Mayusculas = Format(texto, ">")
End Function

```

Código fuente 355

Un ejemplo de utilización de este componente, una vez registrado en el servidor, puede ser la página de ASP que muestra el Código fuente 356.

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
2
3 <HTML>
4 <HEAD>
5 <TITLE>Otro Componente ActiveX Server</TITLE>
6 </HEAD>
7
8 <BODY>
9 <%Set componente=Server.CreateObject("Componente2.Formatos")%>
10 <table border="1" align="center">
11 <tr>
12     <th>Función Utilizada</th>
13     <th>Entrada</th>
14     <th>Salida</th>
15 </tr>
16 <tr>
17     <td>Función HoraCompleta</td>
18     <td>8:30 PM</td>
19     <td><%=componente.HoraCompleta("8:30 PM")%></td>
20 </tr>
21 <tr>
22     <td>Función FechaCompleta</td>
23     <td>2/9/98</td>
24     <td><%=componente.FechaCompleta("2/9/98")%></td>
25 </tr>
26 <tr>
27     <td>Función Mayúsculas</td>
28     <td>Cambia las letras a mayúscula</td>
29     <td><%=componente.Mayusculas("Cambia las letras a mayúscula")%></td>
30 </tr>
31 </table>
32 </BODY>
33 </HTML>

```

Código fuente 356

Y su salida se muestra en la Figura 172.

Función Utilizada	Entrada	Salida
Función HoraCompleta	8:30 PM	20.30.00
Función FechaCompleta	2/9/98	miércoles, sep 2 1998
Función Mayúsculas	Cambia las letras a mayúscula	CAMBIA LAS LETRAS A MAYÚSCULA

Figura 172. Utilización del componente Componente2.

Más adelante veremos un ejemplo más complejo que accede a bases de datos y además utilizaremos transacciones.

## Introducción a COM

Desde hace años viene imponiéndose un nuevo modelo de organización de las aplicaciones. Es lo que habitualmente llamamos Programación Orientada al Objeto. Dentro de una manera relativamente estándar de abordar las cosas, cada fabricante de productos ha implementado sus modelos de objetos de modos distintos.

Pues bien, para los productos de Microsoft, los objetos se organizan a través de lo que se denomina COM (*Component Object Model*). Frente a COM se encuentran otras mecánicas de fabricantes o grupos de fabricantes diferentes, como es el caso de CORBA, otro modelo de organización de objetos donde se encuentran empresas como Sun, IBM, etc. Por tanto, dejemos claro el primer concepto, COM es el modelo común de objetos (componentes) del que participan los distintos productos de Microsoft.

Pero COM, en sus primeras implementaciones no soportaba distribución de objetos, es decir que algo pueda ejecutarse de manera independiente a la máquina donde esté instalado. Así, un objeto COM, por ejemplo cualquier control para Visual Basic a los que los programadores estamos tan acostumbrados, tiene que registrarse y cargarse en la máquina local donde se va a ejecutar. A partir de la aparición de Windows NT 4.0 comienza a disponerse de lo que se denomina tecnología DCOM (que también está disponible para Windows 95). DCOM es el modelo COM distribuido (Distributed Common Object Model) que nos abre la posibilidad a que los objetos puedan residir en distintos servidores y ejecutarse en los mismos sin necesidad de que la máquina cliente que va a usar el componente tenga que cargarlo físicamente en su memoria.

Con la aparición de DCOM comienza a ser posible lo que los teóricos del cliente/servidor vienen denominando desde hace años cliente/servidor en tres capas. Cuando hasta ahora hablamos de cliente/servidor solemos hacerlo del modelo convencional en dos capas, o lo que es lo mismo, la lógica de la aplicación y la lógica de los datos. Nuestros desarrollos se caracterizan porque almacenamos los datos en servidores de datos tipo Oracle o SQL Server; en cuanto a las reglas de negocio de la aplicación (validaciones, tratamientos con los datos, etc.) se incorporan bien al servidor de datos en forma de triggers o procedimientos almacenados, bien a la propia aplicación donde se mezclan con la lógica de la presentación.

Al hablar ahora de cliente/servidor en tres capas nos referimos a un sistema que permite un alto nivel de distribución donde, igualmente, tenemos servidores de datos, pero donde la lógica de la aplicación se encuentra ya dividida en funciones u objetos diferenciados, unos pertenecen claramente a la lógica de la presentación y se ejecutan sobre las máquinas clientes y otros pertenecen a la lógica del negocio y se ejecutan sobre otros servidores intermedios cuya finalidad es simplemente la de albergar y ejecutar estos componentes.

La aplicación así se modulariza y distribuye al máximo, de forma que muchas máquinas cooperan para lograr la finalidad del correcto despliegue de la misma. Al hallarse distribuido el proceso, ganamos en rapidez ya que hay más procesadores cooperando en la resolución de las tareas, así como en modularidad, ya que la aplicación se componentiza más aún del nivel al que hasta ahora estamos acostumbrados, separando y dividiendo tareas que hasta ahora teníamos que tratar como una: presentación y reglas de negocio. La Figura 173 trata de mostrarnos un resumen de todo esto.

El siguiente paso de COM es COM+. COM+ es un conjunto de servicios que combina COM con Microsoft Transaction Server (MTS) en sistemas Windows 2000. De esta forma el servidor transaccional MTS ha dejado de existir como una entidad separada en Windows 2000, toda su funcionalidad es ahora una parte básica del sistema operativo.

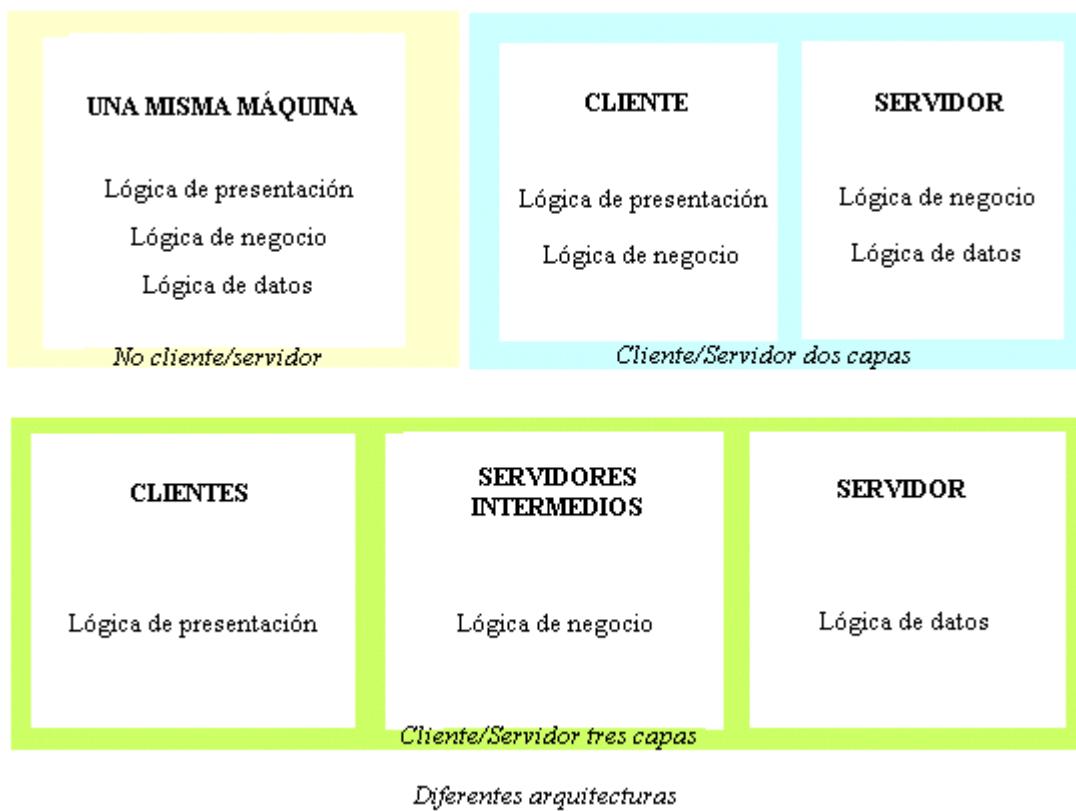


Figura 173

MTS aparece ahora en Windows 2000 bajo el monitor denominado Servicios de componentes.

Para quién no conozca el servidor MTS, en el siguiente apartado se ofrece una introducción al predecesor de Servicios de componentes.

## Antecedentes: Microsoft Transaction Server

Microsoft Transaction Server (MTS) es un servidor para aplicaciones distribuidas que gestiona de modo automático toda la interacción que se produce entre esas tres capas del cliente/servidor antes reseñado, es decir, entre los clientes que simplemente contienen lógica de presentación, los componentes que contienen reglas de negocio y que se encuentran en servidores intermedios y los datos que se encuentran en bases de datos instaladas en servidores de datos como SQL Server.

El producto se concibe como un servidor que contiene los componentes COM que se van a encargar de gestionar todas las reglas de negocio de la empresa. Dichos componentes pueden ubicarse en cualquier servidor NT de la red y MTS se encarga de controlar su ejecución cuando ésta es requerida desde cualquier aplicación. Con esto se pretende, además de dar soporte a este entorno distribuido reseñado, que se gestionen de forma automática cuestiones como el control de errores, la complejidad multiusuario de las aplicaciones, la seguridad de las mismas, etc.

Una vez escrito un componente y ubicado en MTS, dicho componente es compartido, además, por todas las aplicaciones que necesitan su uso. De este modo se descarga considerablemente la parte cliente de las aplicaciones para engrosar la parte servidora de las mismas. Se pretende con ello potenciar el rol de servidor de aplicaciones de NT haciendo clientes mucho menos pesados en cuanto al trabajo que tienen que resolver.

La elección del nombre para esta herramienta quizá sea algo confusa en tanto que podemos confundirla con un monitor transaccional convencional y, sin duda, es algo más. MTS es una combinación de dos tecnologías: monitor transaccional y Object Request Brokers (ORB), y ofrece un componente o servidor intermedio (middleware) para desarrollar aplicaciones distribuidas.

Un monitor transaccional se utiliza para gestionar el acceso a bases de datos y controlar la consistencia de los datos a través de transacciones. Un ORB gestiona servicios que se encuentran distribuidos en una o más máquinas, ofreciendo a las aplicaciones un único servicio al que conectarse de forma transparente. Esto libera a la aplicación de conocer en qué lugar de la red se encuentran cada uno de los servicios utilizados, por lo tanto no debe localizarlos.

Cuando se utiliza MTS, las aplicaciones se dividen en interfaz de usuario y en reglas de negocio, que se comunican entre sí. Las reglas de negocio se implementan como componentes COM que MTS hospeda, componentes ActiveX de servidor que se ejecutan en servidores de aplicaciones MTS. El interfaz de usuario es una aplicación que invoca a componentes que residen en MTS. De esta forma, una aplicación bancaria podría invocar a un componente hospedado por MTS que carga o abona en una cuenta.

MTS ofrece al programador la capacidad de agrupar un conjunto de acciones en una única operación denominada transacción. O bien todos los pasos asociados a la transacción se completan satisfactoriamente o los cambios realizados en la transacción se deshacen.

## Características de servicios de componentes

A continuación se comentan las características principales de los Servicios de componentes de Windows 2000:

- Simplicidad en el ciclo de desarrollo: cualquier regla de negocio construida como un componente COM y gestionada por los Servicios de componentes puede escribirse de un modo muy sencillo. El programador puede dedicarse sólo a pensar los aspectos funcionales que dicho programa debe cubrir, olvidándose de temas como el control de usuarios, la gestión de la concurrencia, etc. De todos estos aspectos se encargan los Servicios de componentes. Digamos que sólo tenemos que escribir sencillos componentes monousuario que pueden convertirse en complicados procesos multiusuario sin necesidad de que el programador toque una sola línea del código de los mismos.
- Soporte completo de ActiveX: Servicios de componentes soporta cualquier DLL ActiveX desarrollada con cualquier herramienta que sirva para generar dichos componentes, como por ejemplo Visual C++, Visual J++ o Visual Basic.
- Dos sencillas API para el desarrollo de aplicaciones: el desarrollador no necesita conocer en profundidad la complejidad de sistemas como COM+ o Win32. Se pueden construir potentes aplicaciones transaccionales simplemente usando dos sencillas API (CoCreateInstance para Visual C++ y CreateObject para Visual Basic).
- Gestión automática de los recursos del servidor: el manejo de los recursos necesarios para los objetos gestionados por Servicios de componentes es automático. Por ejemplo, sin Servicios de componentes una aplicación Visual Basic que obtiene un objeto conexión para acceder por ADO a una base de datos SQL Server está consumiendo recursos del servidor. Cada cliente que se conecta vuelve a obtener un objeto conexión y los recursos se siguen consumiendo. En cambio, con Servicios de componentes tenemos un sólo componente ActiveX especializado en obtener la conexión a la base de datos. Dicho componente es compartido por todos los clientes

y los recursos asignados a la conexión son administrados directamente por Servicios de componentes. La ganancia que se produce en los procesos del servidor es espectacular.

- Escalabilidad: Como consecuencia de lo anterior, una aplicación cliente/servidor de tres o más capas con ADO consumirá recursos del servidor. Con Servicios de componentes y la gestión automática de los recursos el problema se acaba y la escalabilidad de las aplicaciones se convierte en un hecho. Al administrar de forma automática los recursos que cada objeto necesita, Servicios de componentes asigna y descarta según lo juzga conveniente. De este modo podemos olvidarnos de los problemas inherentes a la escalabilidad de las aplicaciones. Nosotros desarrollamos para 1 usuario y Servicios de componentes se encarga del resto. Servicios de componentes proporciona a los componentes servicios de gestión de threads (hilo de ejecución o hebra) y de control de concurrencia.
- Organización en aplicaciones: si Servicios de componentes se encarga de gestionar ActiveX que son llamados por las aplicaciones desde los clientes, uno de los principales problemas con que nos encontraremos es que tendremos multitud de componentes dispersos difícilmente catalogables. La solución a esto son las aplicaciones COM+ (en MTS 2.0 se denominaban paquetes). Una aplicación COM+ es un conjunto de componentes que se agrupan para dar servicio a una misma aplicación bajo una misma política de seguridad. Servicios de componentes permite la instalación de paquetes ya pre-construidos por otras herramientas o, asimismo, permite crearlos seleccionando componentes entre aquellos registrados en el servidor donde se pretende instalar la aplicación COM+.
- Uso de DTC para transacciones distribuidas: Servicios de componentes soporta transacciones distribuidas a través del MS DTC (*Microsoft Distributed Transaction Coordinator*) de Windows 2000. La gestión de las transacciones se realiza de forma absolutamente sencilla para el programador que solo necesita indicar en sus componentes los puntos de éxito (SetComplete) o fracaso (SetAbort) de las mismas. Si se usa una herramienta de base de datos como SQL Server podemos eludir el uso de los BEGIN TRANSACTION ... END TRANSACTION o cualquier otro elemento de complejidad transaccional.
- Aplicaciones en Internet: para Servicios de componentes es transparente el hecho de que estemos llamando a un componente ActiveX desde una aplicación en nuestra red corporativa o que lo hagamos desde Internet. En el primer caso se usa DCOM como soporte para el proceso distribuido y en el segundo se usa HTTP.
- Descargando al cliente: una de las más desagradables consecuencias que ha tenido el desarrollo cliente/servidor en conjunción con los entornos visuales ha sido el de cargar enormemente a los clientes con demasiada parte de la lógica de las aplicaciones. Esto ha hecho que estos equipos se tornen pesados por la cantidad enorme de piezas software que deben contener para resolver todo su trabajo. En cambio, la filosofía de los navegadores de Internet es opuesta a esto en su totalidad. Los navegadores son herramientas poco pesadas que descargan todo lo que necesitan desde los servidores Web. La filosofía de Servicios de componentes va exactamente en esa misma dirección. Al estar el grueso de las reglas de negocio de cada aplicación instaladas en servidores intermedios gestionados por Servicios de componentes, el cliente queda altamente descargado de muchas tareas, quedando en sus manos sólo lo que hace referencia a la lógica de la presentación de las aplicaciones. Las aplicaciones cliente que trabajan con Servicios de componentes no necesitan ninguna librería adicional para acceder al uso de la gestión transaccional que dicho producto proporciona.
- Configuración dinámica: otra característica muy importante de Servicios de componentes es la posibilidad de configuración dinámica o lo que es lo mismo la inserción de clientes y servidores en el entorno de una aplicación distribuida sin necesidad de volver a crear la aplicación o ni siquiera detener la ejecución de la misma.

- Compatibilidad con otros entornos: Servicios de componentes trabaja perfectamente en conjunción con SQL Server y es compatible con el protocolo XA para transacciones, lo que permite que las aplicaciones de Servicios de componentes funcionen con IBM DB2, Informix y otras bases de datos compatibles con este protocolo que se ejecutan tanto en Windows 2000 como en sistemas operativos que no son de Microsoft a través de ODBC.
- Plena integración con Internet Information Server 5.0. Lo cual redunda en una serie de características relacionadas con la creación de aplicaciones ASP, como son:
  - Ejecución de secuencias de comandos de páginas asp dentro de una transacción gestionada por Servicios de componentes.
  - Protección contra errores de las aplicaciones Web de IIS. Al poder ejecutarse cada aplicación dentro de su propia aplicación COM+ de Servicios de componentes, se permite un aislamiento de los procesos implicados.
  - Inclusión de eventos transaccionales en las páginas ASP.
- Características relacionadas con la administración:
  - La utilidad explorador de Servicios de componentes, que permitirá administrar las aplicaciones COM+.
  - Cierre de aplicaciones COM+ de manera individual, sin necesidad de cerrar el proceso del servidor.
  - Configuración de la activación de aplicaciones COM+.
  - Desplazamiento de componentes entre aplicaciones COM+.
- Características relacionadas con la programación:
  - Se puede llevar a cabo, de manera automática, procedimientos administrativos como la instalación de aplicaciones COM+, mediante objetos de administración programables a través de secuencias de comandos.

## El explorador de servicios de componentes

En la Figura 174 se muestra una vista de la consola que nos permite administrar los Servicios de componentes. En dicha figura se puede apreciar como asociado a Servicios de componentes existe una carpeta con el conjunto de ordenadores definidos en la red, que tiene instalado a su vez Servicios de componentes. En cada uno de estos ordenadores podremos colocar componentes que conformen la aplicación distribuida.

Una vez seleccionada la máquina sobre la que vamos a actuar podemos instalar en ella aplicaciones COM+ o crear aplicaciones COM+ vacías a las cuales iremos añadiendo componentes. Esto es tan sencillo como pulsar con el botón derecho del ratón sobre la carpeta aplicaciones COM+ y seleccionar la opción Nuevo|Aplicación del menú contextual.

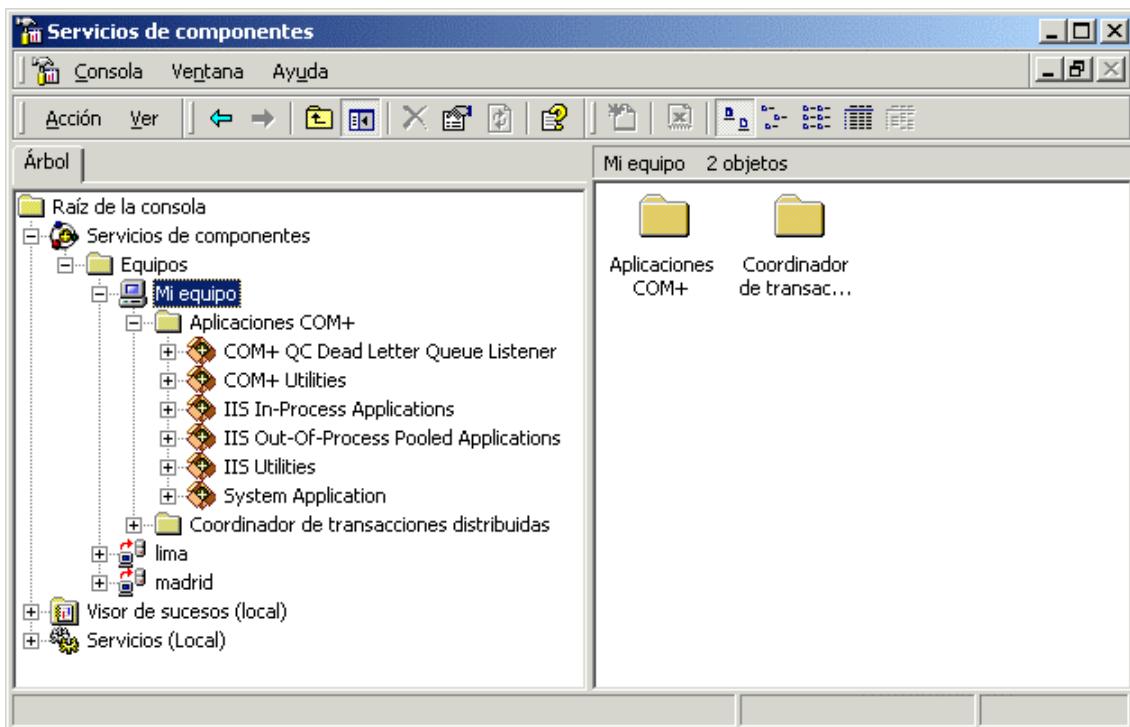


Figura 174. Servicios de componentes



Figura 175. Inicio del asistente de creación de una nueva aplicación COM+

En este momento aparecerá un asistente (Figura 175) para incorporar aplicaciones COM+ pregeneradas o para crear una aplicación vacía que luego podemos llenar añadiendo componentes. Una aplicación COM+ vacía debe poblararse siempre con componentes previamente instalados en la máquina donde se encuentra Servicios de componentes.



Figura 176. Creando una aplicación COM+

Las aplicaciones COM+ existentes en el sistema podemos verlas navegando por la ventana del explorador de Servicios de componentes. Si hacemos doble clic sobre una de las aplicaciones y seleccionamos la carpeta componentes, nos aparecen los componentes que componen dicha aplicación. La Figura 177 nos muestra los componentes de la aplicación COM+ llamada IIS Utilities que se corresponde con una aplicación de utilidades para IIS que nos proporciona Servicios de componentes.

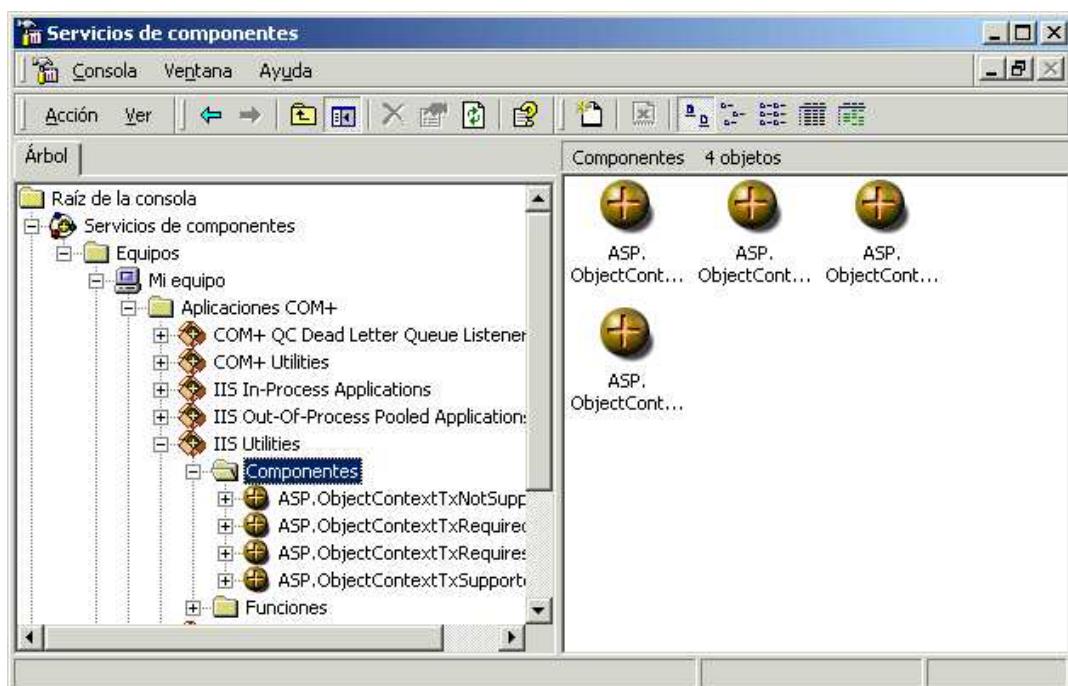


Figura 177. Componentes de una aplicación COM+

El comportamiento de cada componente es configurable en su conjunto. Pulsando el botón derecho del ratón sobre cualquiera de estos componentes podremos acceder a sus propiedades, y determinar aspectos relativos a la seguridad de los mismos, al entorno distribuido donde se ejecutarán, así como su carácter transaccional. En la Figura 178 podemos las hojas propiedades de un componente determinado.

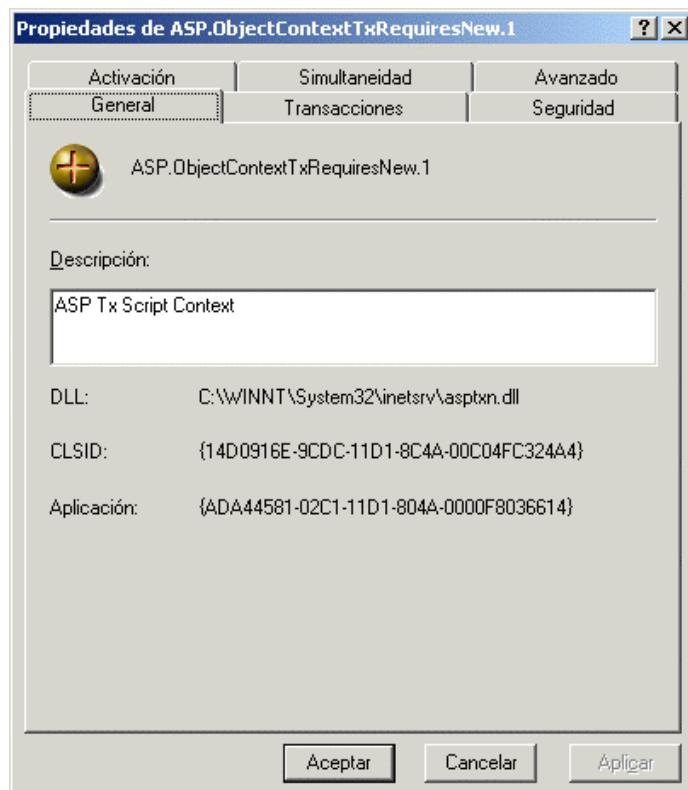


Figura 178. Propiedades de un componente

En la imagen siguiente podemos ver una muestra de un resumen de propiedades de todos los componentes de la aplicación seleccionada.

Servicios de componentes					
Árbol	Componentes				
	Id. de prog.	CLSID	Transacción	S...	Sincronizar
Raíz de la consola					
Servicios de componentes					
Equipos					
Mi equipo					
Aplicaciones COM+					
COM+ Dead Letter Queue Listener					
COM+ Utilities					
IIS In-Process Applications					
IIS Out-Of-Process Pooled Application					
IIS Utilities					
Componentes					
ASP.ObjectCont...	{14D09170-9CD...	No se admite	Sí	Necesario	
ASP.ObjectCont...	{14D0916D-9CD...	Necesario	Sí	Necesario	
ASP.ObjectCont...	{14D0916E-9CD...	Necesita nuevo	Sí	Necesario	
ASP.ObjectCont...	{14D0916F-9CD...	Compatible	Sí	Necesario	
Prácticas					

Figura 179. Resumen de propiedades

Cada componente posee una característica transaccional (Figura 180), a la cual accederemos pulsando con el botón derecho del ratón sobre el componente y seleccionando la opción Propiedades del menú desplegable.

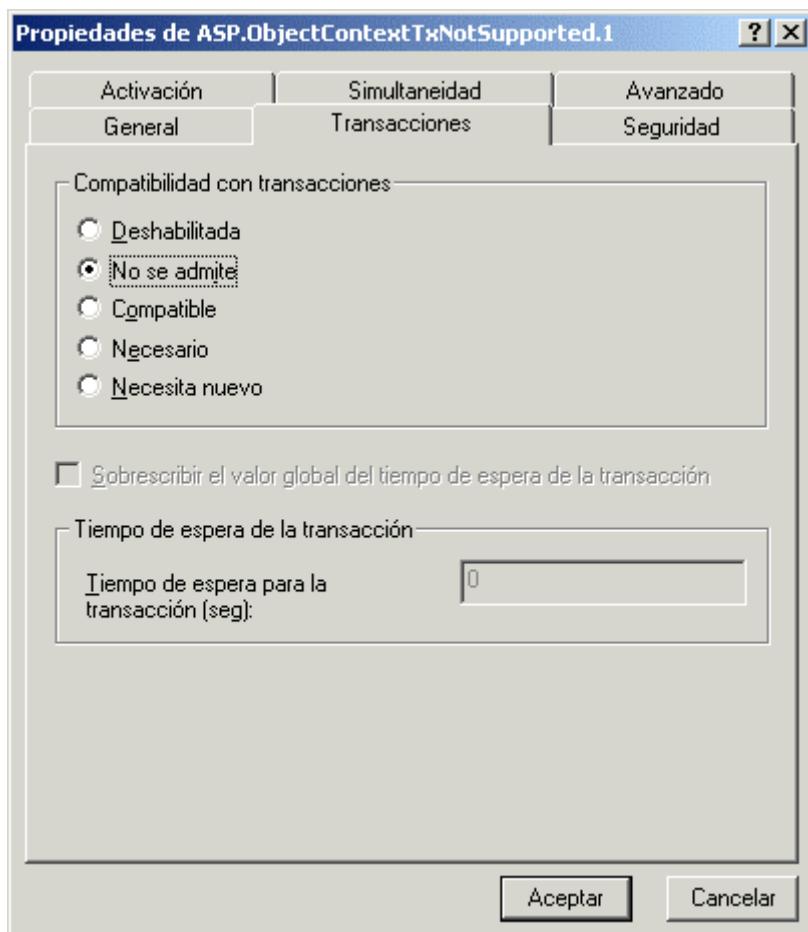


Figura 180. Transacciones en un componente

Las aplicaciones se pueden configurar para determinar dónde se ejecutarán sus componentes, y la propia aplicación. En Servicios de componentes existen dos posibles opciones, según la clase de aplicación:

- Aplicación de Biblioteca: permite ejecutar las instancias de sus componentes, en el propio espacio de proceso que el cliente que las crea. Esta opción sólo está disponible, en aquella máquina en la que se encuentra instalada la aplicación. Esta opción suele ser recomendable en tiempo de desarrollo de los componentes.
- Aplicación de Servidor: las instancias de los componentes pertenecientes a la aplicación, se ejecutan en el espacio de trabajo correspondiente al Server Process que ejecuta la aplicación, el cual no tiene nada que ver con los clientes que los utilizan. Esta es la opción por excelencia en un entorno distribuido, ya que por lo general los clientes serán remotos.

En la Figura 181 se muestran las opciones comentadas.

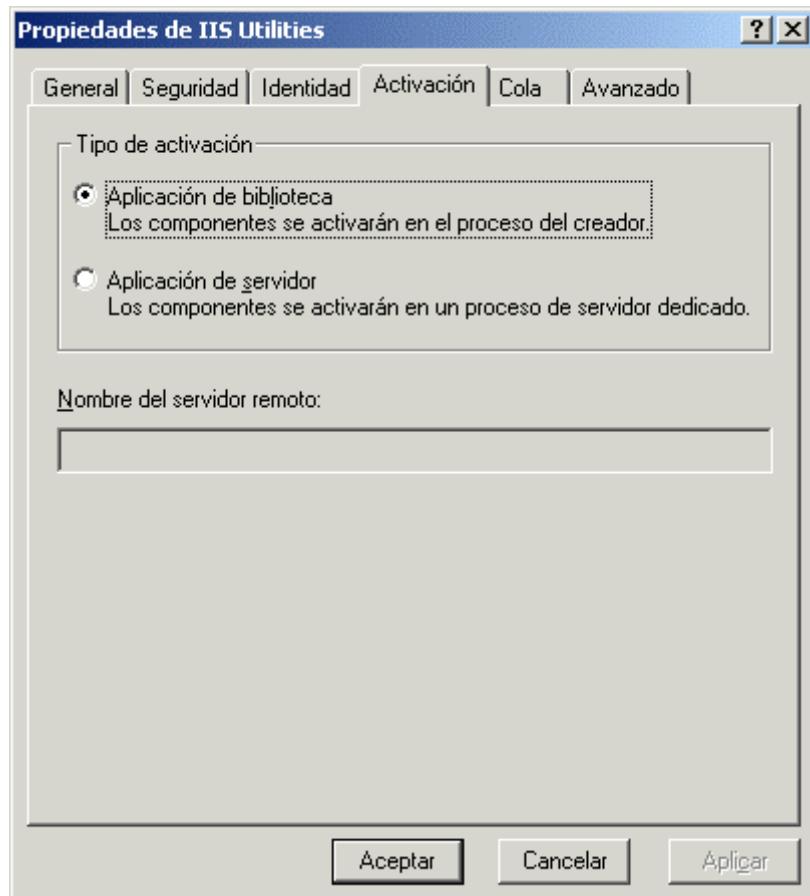


Figura 181. Activación de los componentes de una aplicación

## Gestión de transacciones

La gestión automática de transacciones realizada por Servicios de componentes facilita en gran medida la tarea de programación, ya que es el propio Servicios de componentes quien se encarga de arrancar y finalizar las transacciones, sin necesidad de intervenir el programador en dicha gestión. Además, la gestión de transacciones es más segura si es Servicios de componentes quien la gestiona.

La especificación del carácter transaccional de los objetos controlados por Servicios de componentes no se realiza mediante programación, sino desde el explorador del propio Servicios de componentes, en la opción de propiedades de un componente, como puede verse en la Figura 182.

Para los componentes sujetos a un tratamiento transaccional (por defecto ninguno lo está) Servicios de componentes arranca una transacción, en caso de ser un objeto perteneciente a un contexto aislado, o bien aprovecha la existente en un contexto desde el que ha sido instanciado. Son cuatro las opciones disponibles referentes al tratamiento transaccional automático, a saber:

- *Deshabilitada*: se ignorarán los requerimientos transaccionales del componente, y nunca va a participar en un contexto transaccional.
- *No se admite*: no se ejecutarán sus operaciones bajo una transacción, en ningún momento. La diferencia con el caso anterior es que en este caso siempre se crea un nuevo contexto para estos componentes.

- Compatible: cuando un objeto de este tipo es instanciado, en caso de pertenecer a un contexto transaccional, se hará uso de dicha transacción en sus operaciones, en caso contrario, no será arrancada ninguna transacción.
- Necesario: cuando se utiliza un objeto definido de esta manera, Servicios de componentes arranca una transacción para las operaciones que realice. Si hay una transacción existente en su contexto se aprovecha esta transacción, en caso contrario se crea una nueva transacción.
- Necesita nuevo: la utilización de un objeto de este tipo tiene como consecuencia el arranque de una nueva transacción con independencia del origen de su instancia. Esto permite independizar transacciones dentro de un contexto transaccional existente.

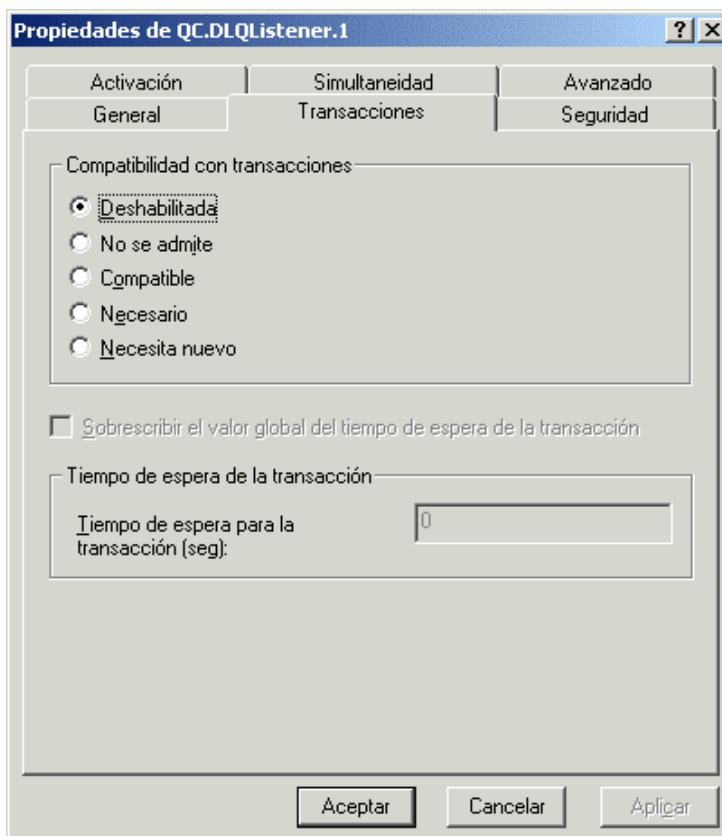


Figura 182. Carácter transaccional de un componente

La especificación de estas opciones tiene una incidencia importante en la programación del componente, en tanto en cuanto, cuando un componente se ha declarado transaccional y sin embargo, su código no contempla dicha capacidad, sus transacciones se intentarán comprometer (realizar, commit) por parte de Servicios de componentes, al descargar el objeto, pero no se tendrá conocimiento de cual fue el resultado de la transacción.

Lo más idóneo es incluir las funciones pertinentes en el código de los métodos del componente, para asegurarnos que las transacciones se realizan (SetComplete) o deshacen (SetAbort). Las mismas funciones que realizan dicha tarea, tienen incidencia en la optimización de los recursos, pues son éstas las que permiten una activación en el momento (Just In-time Activation).

En caso de tener ya escrito un componente que no fue diseñado para trabajar bajo Servicios de componentes, pero que se desea que sí lo haga actualmente, lo más conveniente es colocarlo como no

transaccional, dado que en cualquier caso sería él mismo, el que originalmente se encargase de sus propias transacciones. Lo que no se conseguiría, en cualquier caso, es una activación en el momento, al ser un objeto ActiveX normal y corriente.

La interfaz IObjectContext permite capturar una referencia al objeto de contexto asociado al objeto instanciado, cuyos métodos nos permitirán hacer uso de las características de Servicios de componenetes. Para hacer uso de esta interfaz desde VB debe incluirse en el proyecto la referencia COM+ Services Type Library.

Para obtener una referencia al objeto de contexto, se declara una variable de tipo ObjectContext y después se llama a la función GetObjectContext, como indica el Código fuente 357.

```
Dim ctxObj As ObjectContext  
Set ctxObj = GetObjectContext()
```

Código fuente 357

Los métodos del objeto de contexto más utilizados son los siguientes:

- CreateInstance: crea una instancia de la clase que se envía como parámetro, con un contexto de idénticas características. Las transacciones abiertas bajo este contexto engloban a las del nuevo contexto, en caso de que el nuevo componente sea transaccional y no requiera una transacción independiente.
- SetComplete: este método obliga a realizar las transacciones, una vez se abandone el módulo en que es ejecutado. De la misma forma al salir de este módulo, el objeto será desactivado liberando así sus recursos y permitiendo posteriormente su reactivación, en caso necesario. Aún en el caso de tratarse de un componente no transaccional, este método es de uso aconsejable..
- SetAbort: su funcionamiento es similar al de SetComplete desde el punto de vista de gestión de recursos. A diferencia de SetComplete, este método obliga a deshacer las transacciones realizadas en el contexto en el que es ejecutado.
- EnabledCommit: Indica que las modificaciones son susceptibles de ser realizadas aunque debe mantenerse el estado interno del objeto, lo que significa que no se desactivará éste, hasta su descarga o desactivación explícita.
- DisabledCommit: indica que el trabajo realizado por el objeto, no debe ser llevado a cabo y guardado en la base de datos, hasta que se ejecute un EnabledCommit. Tampoco tendrá lugar su desactivación, al igual que con el método anterior.

## Páginas ASP transaccionales. Ejemplo de desarrollo

Como ya sabemos, a partir de la versión 2.0 de ASP se incluye un nuevo objeto integrado, el objeto ObjectContext. A través de este objeto integrado de ASP podremos tratar y gestionar las transacciones que se realicen en las páginas ASP, pudiendo construir, por tanto, páginas ASP transaccionales. A través de este objeto podremos deshacer o llevar a cabo las transacciones gestionadas por Servicios de componentes.

Para indicar la naturaleza transaccional de una página ASP debemos incluir, como primera línea de la página, el Código fuente 358.

```
<% @ TRANSACTION = valor %>
```

Código fuente 358

Donde el argumento valor, puede tomar los valores:

- Requires: Se requiere una transacción.
- Requires\_New: Se requiere una transacción nueva.
- Supported: Soporta transacciones.
- Not\_Supported: No soporta transacciones.
- *Disabled*: Transacciones deshabilitadas, no se participará en ninguna transacción.

Para más información sobre el objeto ObjectContext, se puede consultar el tema dedicado a este objeto integrado

## Ejemplo de desarrollo transaccional con ASP y Servicios de componentes

Vamos a programar una página ASP transaccional (Transferencias.asp) que, a partir de una información bancaria suministrada por un usuario mediante formulario (Formulario.asp), nos permita llevar a cabo depósitos o retiradas de fondos en una serie de cuentas bancarias perteneciente al usuario en cuestión.

La información referente a las cuentas bancarias del usuario y al estado de las mismas se registrará en una tabla "Cuentas" de una base de datos de SQL Server, que residirá en un servidor de datos. Siendo la estructura de dicha tabla la mostrada en la Figura 183.

Para llevar a cabo la gestión comentada, debemos suministrar dos operaciones. Una operación que permita depositar fondos en la cuenta y otra que permita retirarlos, asegurando además que la cuenta tenga fondos suficientes para cubrir la transferencia. En caso contrario se debería informar al usuario de que la operación no se puede llevar a cabo.

Estas dos operaciones las vamos a implementar por medio de un componente en Visual Basic 6.0. Este componente será el encargado de interactuar, a través de ADO, con la tabla de "Cuentas" en SQL Server, incrementando o decrementando el Saldo de la cuenta en función de la operación efectuada. Es decir, será el encargado de implementar la "Lógica de negocio" de la aplicación. Todo ello se llevará a cabo dentro de un contexto transaccional por lo que el componente será registrado en Servicios de componentes, donde instalaremos la aplicación correspondiente.

Finalmente la "Lógica de presentación", la llevaremos a cabos a través de las páginas ASP. Tendremos una página Formulario.asp que se encargará de recoger del usuario la información financiera a tratar y otra página Transferencias.asp que se encargará de efectuar las llamadas correspondientes a las

funciones encapsuladas en el componente, efectuando todas estas llamadas, a su vez, dentro de un contexto transaccional.

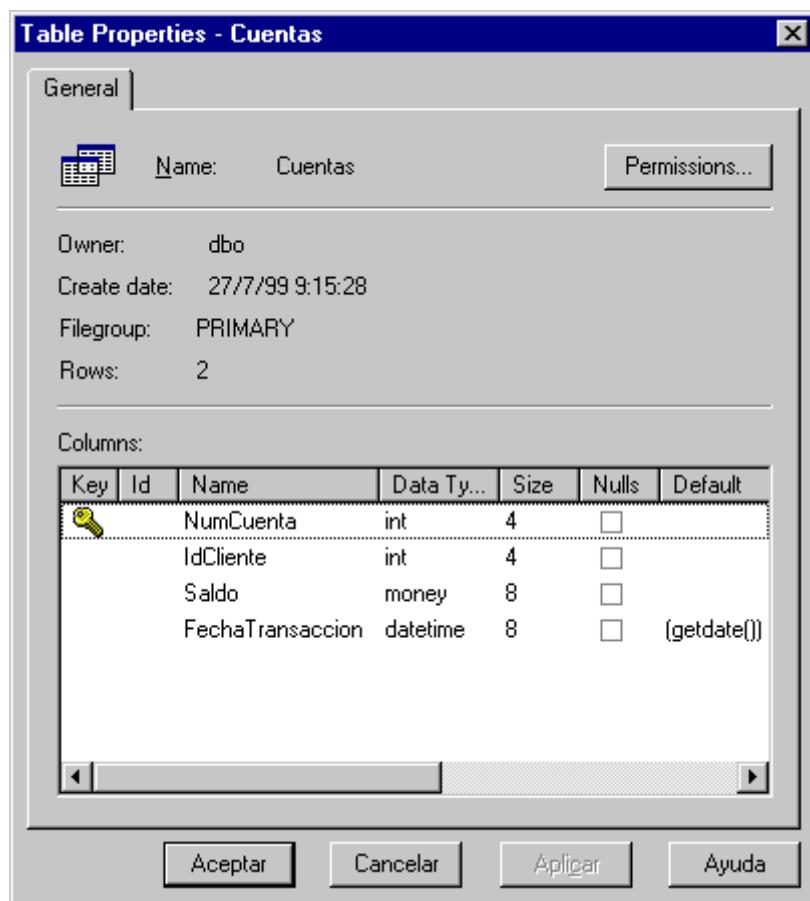


Figura 183

Vamos a empezar creando el componente en Visual Basic 6.0. Para ello crearemos un nuevo proyecto "ActiveX DLL". Al proyecto lo llamaremos "Cuentas", y al módulo de clase "Cuenta":

Debemos asegurarnos de poner la propiedad Instancing del módulo de clase Cuenta al valor MultiUse. Y, en las propiedades del proyecto, seleccionar "Subprocesos independientes" de la opción Modelo de subprocesos.

Además, deberán ser incluidas las siguientes referencias en el proyecto:

- Microsoft ActiveX Data Objects 2.5 Library
- Microsoft ActiveX Data Objects Recordset 2.5 Library
- COM+ Services Type Library

Vamos ahora a implementar, a través del módulo de clase Cuenta, las funciones RetirarFondos y DepositarFondos, que se encargarán de decrementar o incrementar, respectivamente, el Saldo de la cuenta correspondiente. El código del módulo es el Código fuente 359.

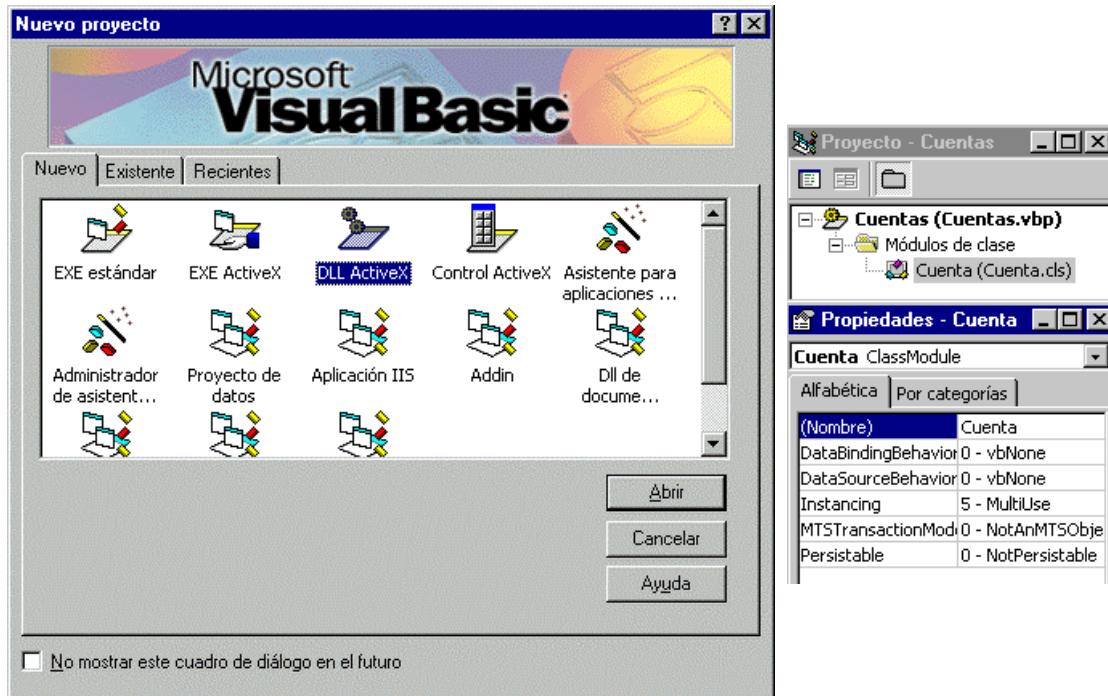


Figura 184

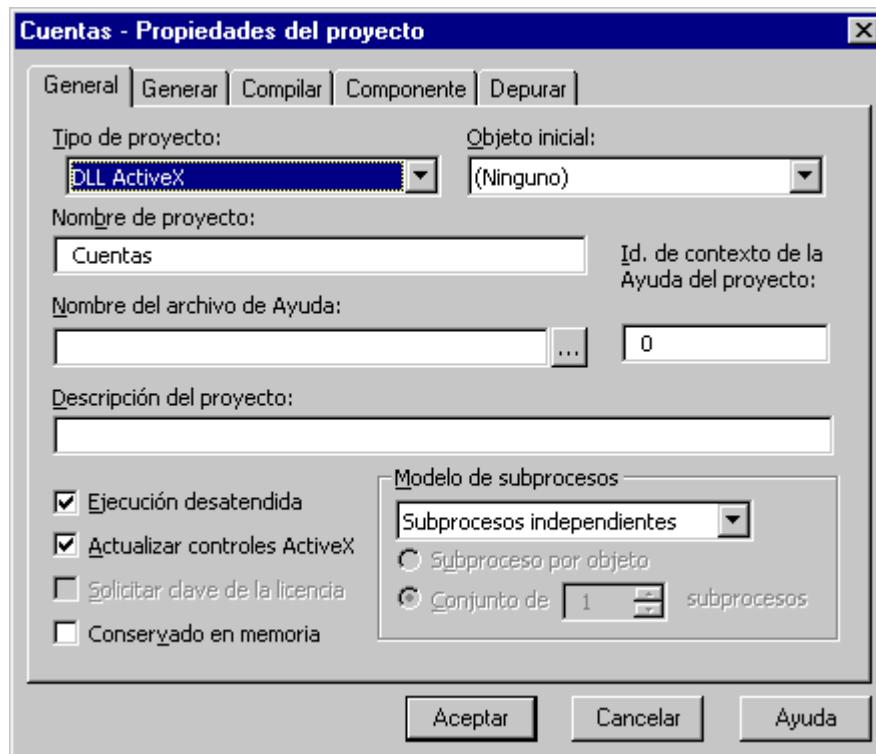


Figura 185

```
Option Explicit
Private oObjectContext As ObjectContext
Private oConex As ADODB.Connection
```

```

Private oRs As ADODB.Recordset
Private vSql As Variant
Const vCadConex As Variant = "DSN=Cuentas;UID=sa;PWD=;
Public Function RetirarFondos(ByVal vNumCta As Variant, ByVal vCantidad As Variant)
As Variant

    On Error GoTo Errores

    Set oObjectContext = GetObjectContext()
    Set oConex = oObjectContext.CreateInstance("ADODB.Connection")
    oConex.Open vCadConex

    vSql = "select * from Cuentas where NumCuenta = " & vNumCta

    Set oRs = oObjectContext.CreateInstance("ADODB.Recordset")
    oRs.Open vSql, oConex, adOpenDynamic, adLockOptimistic

    If oRs.EOF Then
        Err.Raise vbObjectError + 500, "RetirarFondos", "Número de cuenta no válido"
    Else
        If CCur(vCantidad) > CCur(oRs("Saldo")) Then
            Err.Raise vbObjectError + 510, "RetirarFondos", "Fondo insuficiente"
        Else
            Dim vNuevoSaldo As Variant
            vNuevoSaldo = CCur(oRs("Saldo")) - vCantidad
            oRs("Saldo") = vNuevoSaldo
            oRs("FechaTransaccion") = Now
            oRs.Update
        End If
    End If

    RetirarFondos = vCantidad
    oObjectContext.SetComplete

    Exit Function
Errores:
    oObjectContext.SetAbort
    RetirarFondos = "ERROR: " & Err.Description
End Function

Public Function DepositarFondos(ByVal vNumCta As Variant, ByVal vCantidad As Variant)
As Variant

    On Error GoTo Errores
    Set oObjectContext = GetObjectContext()
    Set oConex = oObjectContext.CreateInstance("ADODB.Connection")
    oConex.Open vCadConex

    vSql = "select * from Cuentas where NumCuenta = " & vNumCta

    Set oRs = oObjectContext.CreateInstance("ADODB.Recordset")
    oRs.Open vSql, oConex, adOpenDynamic, adLockOptimistic

    If oRs.EOF Then
        Err.Raise vbObjectError + 600, "DepositarFondos", "Número de cuenta no
válido"
    Else
        Dim vNuevoSaldo As Variant
        vNuevoSaldo = CCur(oRs("Saldo")) + vCantidad
        oRs("Saldo") = vNuevoSaldo
        oRs("FechaTransaccion") = Now
        oRs.Update
    End If
    DepositarFondos = vCantidad
    oObjectContext.SetComplete

    Exit Function

```

```

    Errores:
        oObjectContext.SetAbort
        DepositarFondos = "ERROR: " & Err.Description
End Function

```

Código fuente 359

El código completo de la clase se puede obtener [aquí](#).

Una vez confeccionada la clase, vamos a generar el archivo .dll correspondiente a la misma. Para ello simplemente seleccionamos la opción: Archivo / Generar Cuentas.dll...

Procedamos ahora al siguiente paso, que será añadir el componente a Servicios de componentes. Para ello debemos proceder a la creación de la aplicación COM+ correspondiente seleccionando la carpeta "aplicaciones COM+" y accediendo a la opción "nuevo / aplicación" del menú contextual que obtendremos pulsando con el botón derecho del ratón sobre la carpeta.

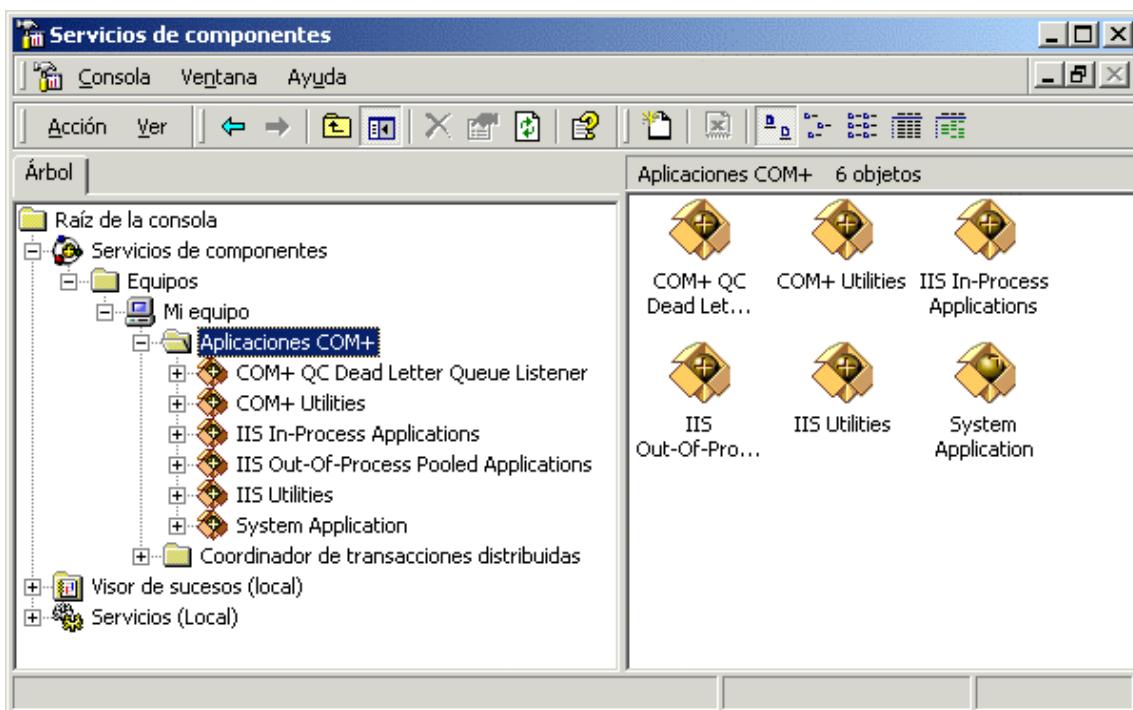


Figura 186

Esto nos llevará a un asistente para la creación de aplicaciones COM+. Seleccionaremos la opción "Crear una aplicación vacía" y llamaremos a la nueva aplicación: GestiónCuentas.

En las opciones correspondientes a la definición de la identidad del paquete seleccionaremos la cuenta de "Usuario interactivo: usuario conectado actualmente".

Después de esto, ya tendremos la aplicación COM+ vacía GestiónCuentas.

Una vez creada la aplicación COM+, le añadiremos el componente seleccionando la opción "nuevo / componente" del menú contextual correspondiente en la carpeta "componentes". Esto nos llevará a un asistente para la instalación de componentes.

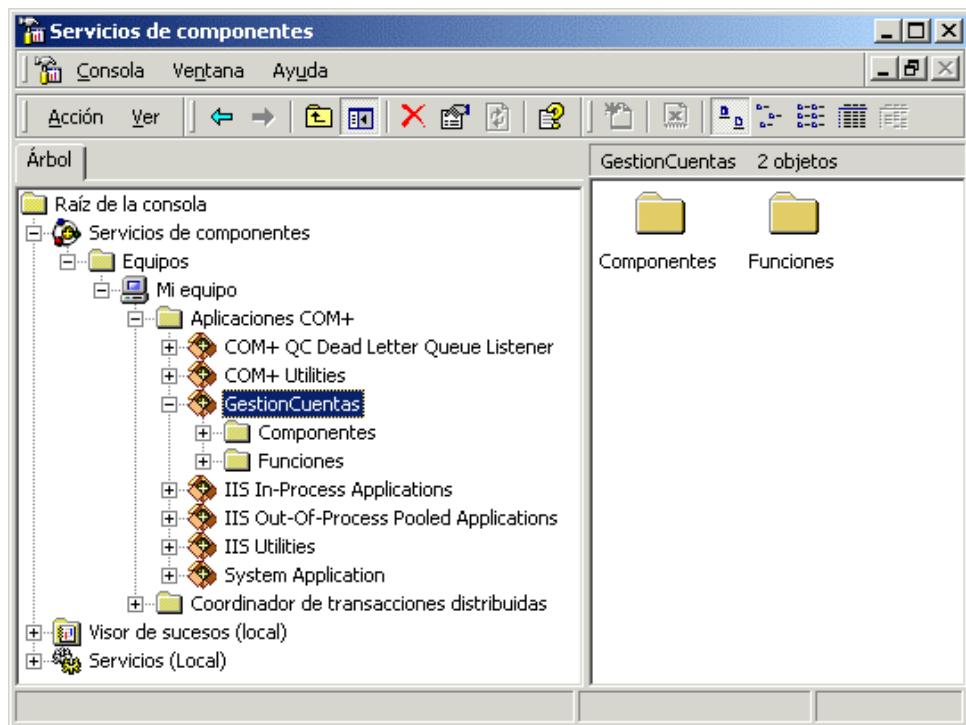


Figura 187

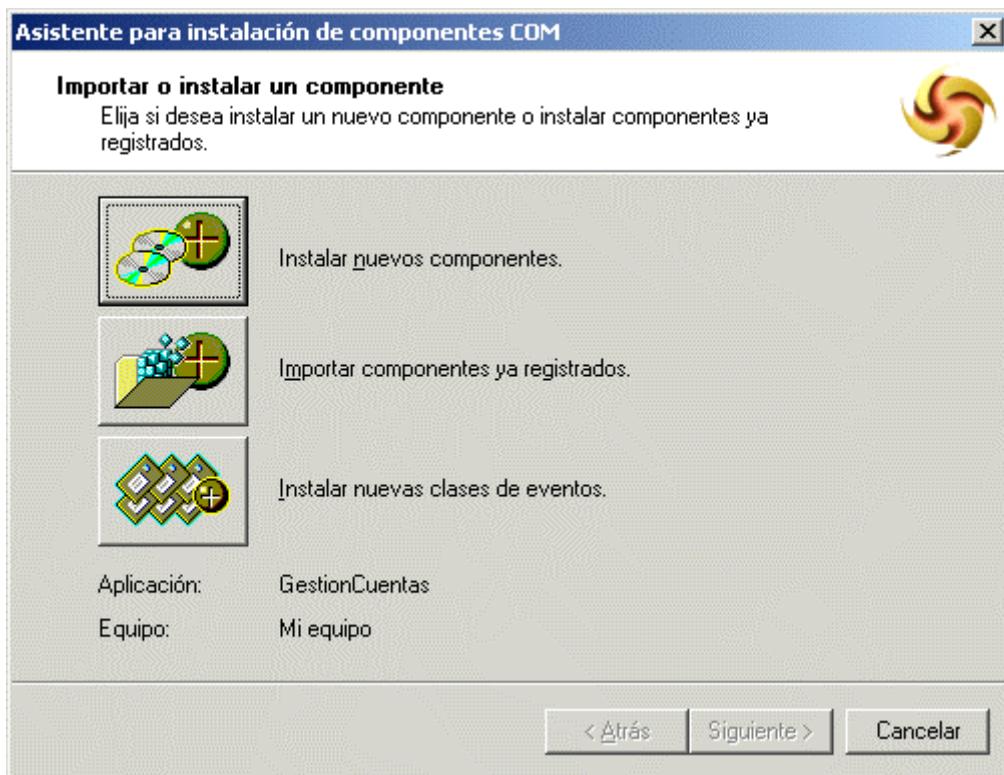


Figura 188

Si seleccionamos la opción Instalar nuevos componentes, pasaremos a un cuadro de diálogo desde el cual podremos agregar el fichero .dll correspondiente a nuestro componente con "Agregar archivos...":

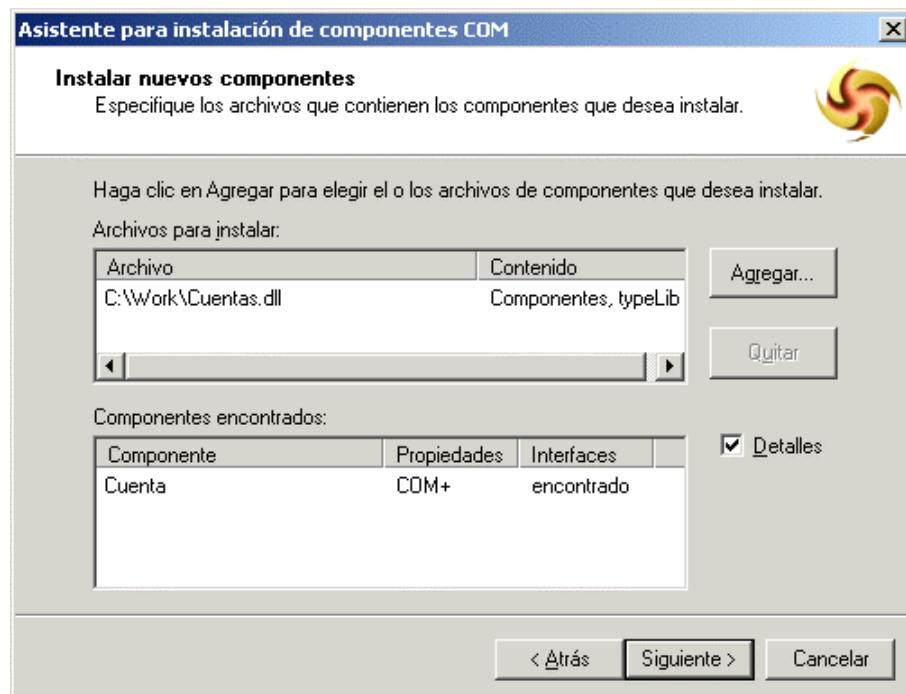


Figura 189

Después de todo este proceso, ya tendremos instalado nuestro componente en Servicios de componentes.

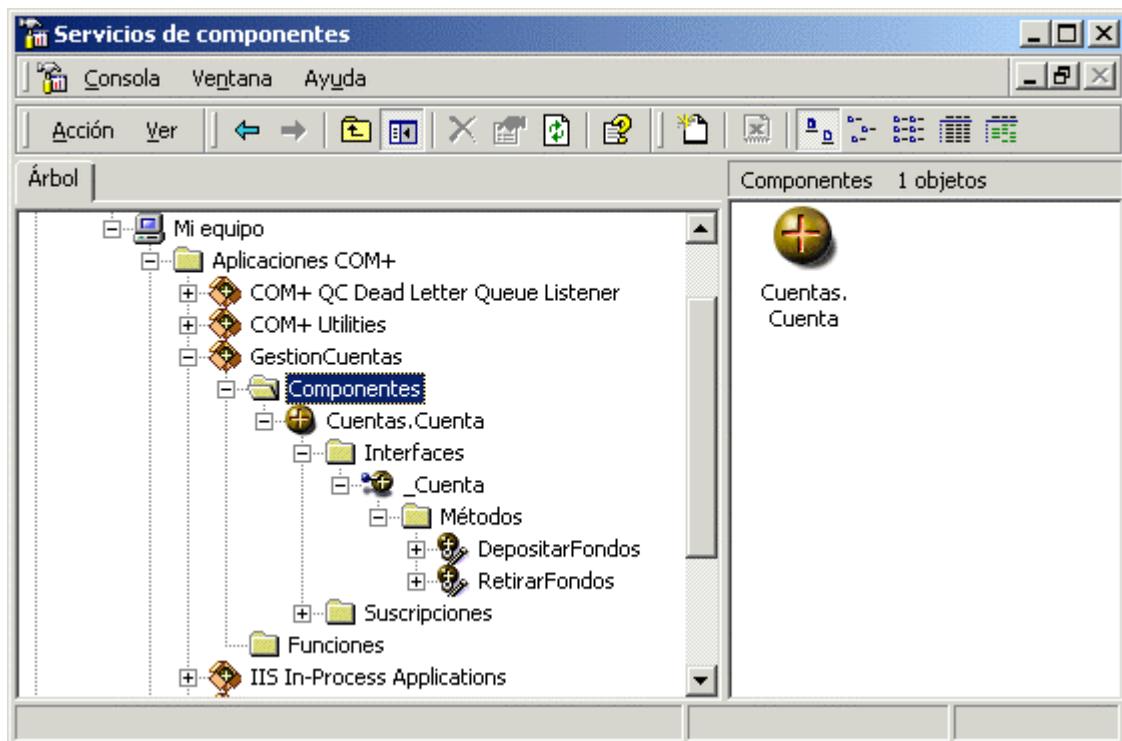


Figura 190

El último paso será acceder a las propiedades del componente y configurar su compatibilidad con transacciones, sin olvidarnos, antes de hacer referencia a nuestro componente desde las páginas ASP, de iniciar el DTC (*Distributed Transaction Coordinator*), en el caso de que no lo estuviera.

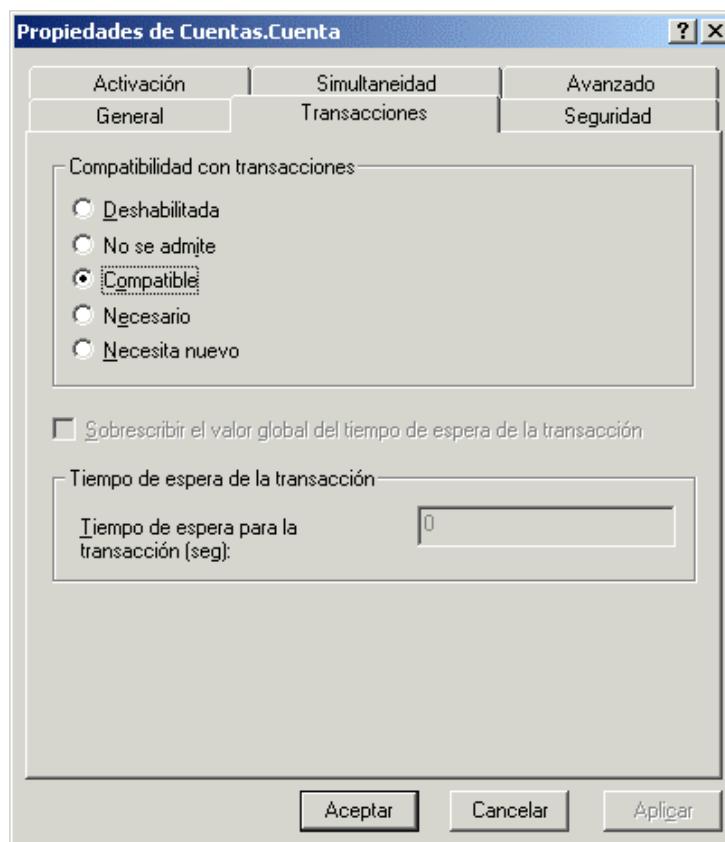


Figura 191

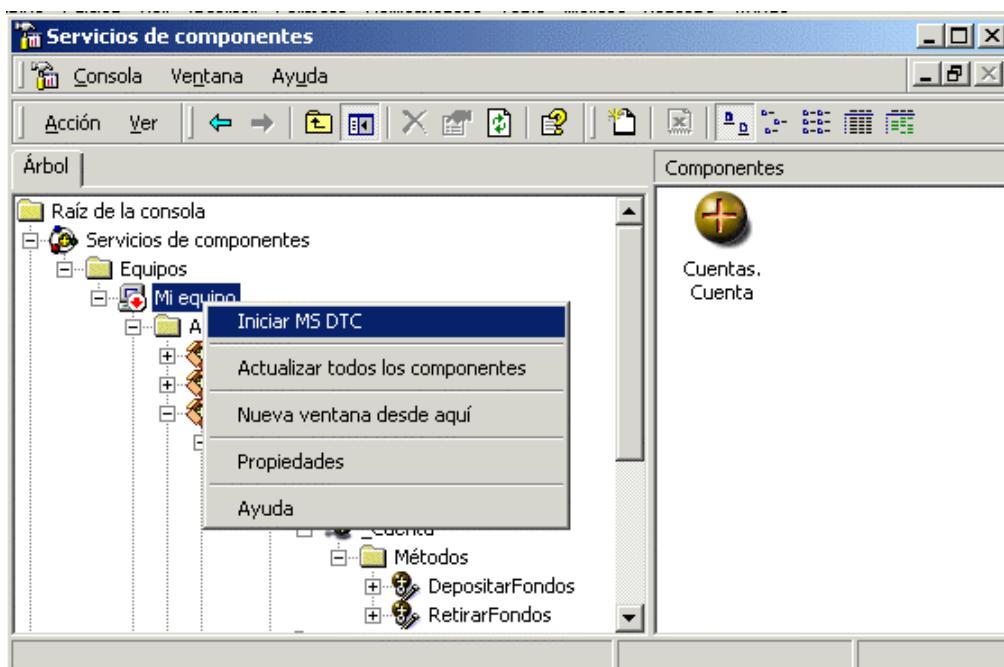


Figura 192

Como parte final del desarrollo, nos quedan por construir las páginas ASP que constituirán la "Lógica de presentación". Como ya comentamos tendremos una página Formulario.asp que se encargará de recoger del usuario la información financiera a tratar y otra página Transferencias.asp que se encargará de efectuar las llamadas correspondientes a las funciones RetirarFondos y DepositarFondos encapsuladas en el componente MTS, efectuando todas estas llamadas, a su vez, dentro de un contexto transaccional. Realmente esta página (Transferencias.asp) podríamos decir que está a medio camino entre la "Lógica de presentación" y la "Lógica de negocio". El formulario a través del cual solicitaremos la información bancaria a tratar (Formulario.asp) lo podemos ver en la Figura 193.



Figura 193

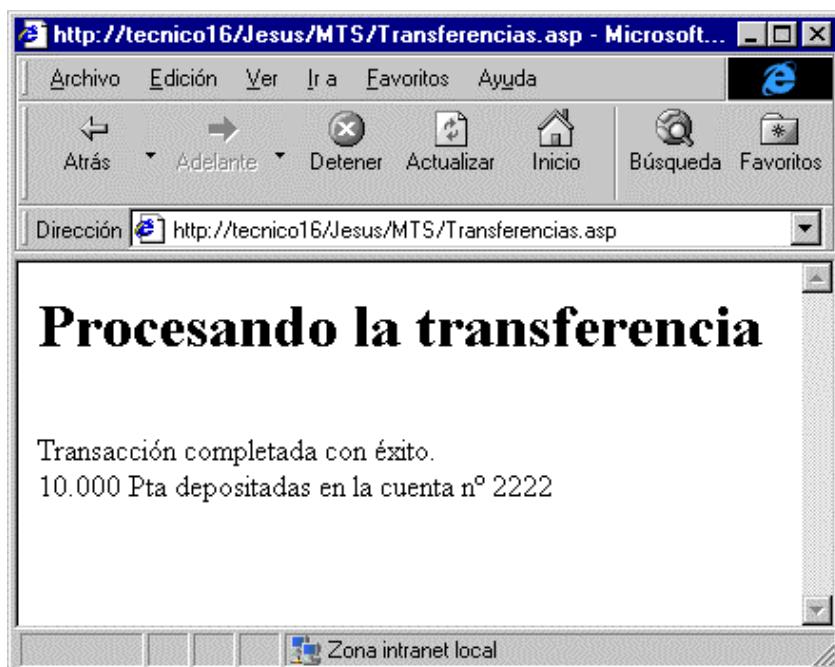


Figura 194

Cuando el usuario pulse en el botón "Ejecutar transferencia", los datos del formulario serán enviados a la página "TRANSFERENCIAS.ASP".

Esta página ASP se encargará de recoger dichos datos, instanciar al componente "Cuentas" y llamar a la función correspondiente del componente: DepositarFondos o RetirarFondos. Además, informará al usuario del resultado de la transferencia en función de si ésta se ha completado de manera satisfactoria o no, a través de los eventos del objeto ObjectContext: OnTransactionCommit y OnTransactionAbort.

La salida correspondiente a la entrada de formulario anterior la podemos ver en la Figura 194.

El código ASP de la página FORMULARIO.ASP es el que muestra el Código fuente 360.

```
<html>
<head>
<title>Formulario</title>
</head>
<body>
<form action="Transferencias.asp" method="post">
<b><p>Nº de Cuenta:</p></b> <input type="text" name="NumCuenta" ><br>
<b>Cantidad a transferir:</b> <input type="text" name="Cantidad" ><br>
<br>
<b><b>Selecciona el tipo de transferencia:</b></b>
<input type="radio" name="TipoTransferencia" value="meter" checked> Depósito de fondos <br>
<input type="radio" name="TipoTransferencia" value="sacar"> Retirada de fondos<br>
<br>
<input type="submit" name="BotonEnvio" value="Ejecutar transferencia">
</b>
</form>
</body>
</html>
```

Código fuente 360

Y el código ASP de la página TRANSFERENCIAS.ASP lo vemos en el Código fuente 361.

```
<%@ TRANSACTION = Required %>
<%Option Explicit%>
<html>
<head>
<title>Transferencias</title>
</head>
<body>
<h1>Procesando la transferencia</h1>
<%Dim Fondos, ObjetoCuenta, CadenaError, NumCuenta, Cantidad, TipoTransferencia _ , CadenaTransferencia
Set ObjetoCuenta = Server.CreateObject("Cuentas.Cuenta")
NumCuenta = Request.Form("NumCuenta")
Cantidad = Request.Form("Cantidad")
TipoTransferencia = Request.Form("TipoTransferencia")
If TipoTransferencia = "meter" Then
    Fondos = ObjetoCuenta.DepositarFondos(NumCuenta, Cantidad)
    CadenaTransferencia = " depositadas en "
Else
    Fondos = ObjetoCuenta.RetirarFondos(NumCuenta, Cantidad)
    CadenaTransferencia = " retiradas de "
```

```
End If
If Left(Cstr(Fondos),5) = "ERROR" Then
    CadenaError = Fondos
   ObjectContext.SetAbort
End If%>
<%sub OnTransactionCommit()
    Response.Write ("<br>Transacción completada con éxito.<br>")
    Response.Write (FormatCurrency(Fondos) & CadenaTransferencia & "la cuenta nº " &
NumCuenta)
end sub
sub OntransactionAbort()
    Response.Write ("Transacción no completada <br>")
    Response.write (CadenaError)
end sub%>
</body>
</html>
```

Código fuente 361

El ejemplo completo se encuentra en el siguiente [fichero](#), que contiene las páginas ASP y el proyecto de Visual Basic con la clase correspondiente.





Si quiere ver más textos en este formato, visítenos en: <http://www.lalibreriadigital.com>. Este libro tiene soporte de formación virtual a través de Internet, con un profesor a su disposición, tutorías, exámenes y un completo plan formativo con otros textos. Si desea inscribirse en alguno de nuestros cursos o más información visite nuestro campus virtual en: <http://www.almagesto.com>.

Si quiere información más precisa de las nuevas técnicas de programación puede suscribirse gratuitamente a nuestra revista **Algoritmo** en: <http://www.algoritmodigital.com>. No deje de visitar nuestra revista **Alquimia** en <http://www.eidos.es/alquimia> donde podrá encontrar artículos sobre tecnologías de la sociedad del conocimiento.

Si quiere hacer algún comentario, sugerencia, o tiene cualquier tipo de problema, envíelo a la dirección de correo electrónico [lalibreriadigital@eidos.es](mailto:lalibreriadigital@eidos.es).