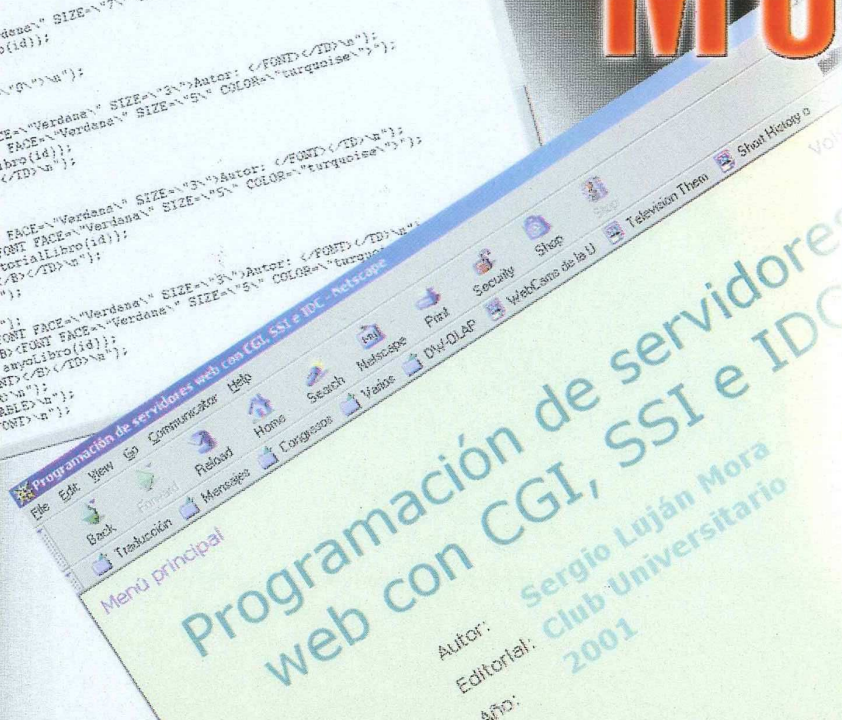
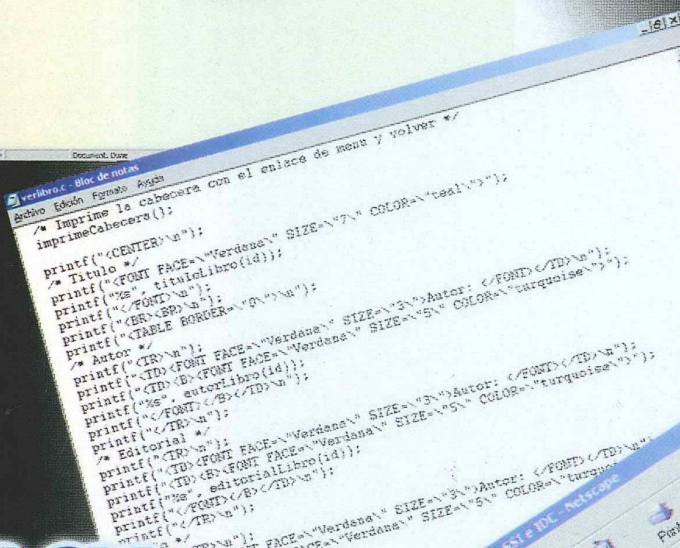
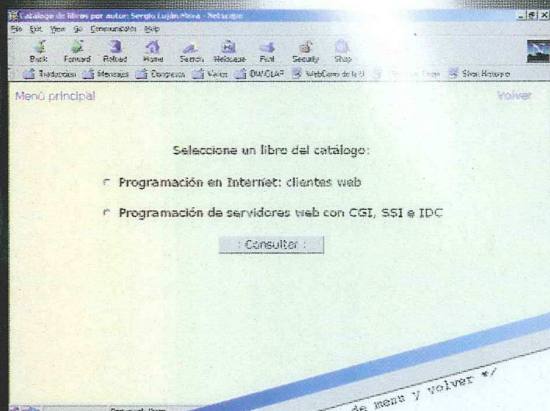


PROGRAMACIÓN DE SERVIDORES WEB

con CGI, SSI e IDC

Sergio Luján Mora



ECU
EDITORIAL
CLUB
UNIVERSITARIO

Programación de servidores
web con CGI, SSI e IDC
Autor: Sergio Luján Mora
Editorial: Club Universitario
Año: 2001

PROGRAMACIÓN DE SERVIDORES WEB

La creación de páginas web se ha modificado en pocos años: de páginas estáticas, que siempre mostraban el mismo contenido, se ha pasado a páginas dinámicas que permiten la creación de verdaderas aplicaciones que se ejecutan en la web. Las aplicaciones de este tipo se conocen como "aplicaciones web".

Las aplicaciones web permiten la generación automática de contenido, la creación de páginas personalizadas según el perfil del usuario o el desarrollo del comercio electrónico. Además, una aplicación web permite interactuar con los sistemas de una empresa, como puede ser gestión de clientes, contabilidad o inventario, a través de una página web.

La programación de las aplicaciones web se encuadra dentro de las arquitecturas cliente/servidor. Por una parte, tenemos el cliente web (el navegador) que solicita servicios. Por otro lado, el servidor web que ofrece servicios y responde a las peticiones de los clientes.

Este libro contempla la programación de la parte servidor de las aplicaciones web. De las diversas tecnologías que existen, se centra en las primeras que se emplearon en la web: CGI, SSI e IDC. El libro contiene gran cantidad de ejemplos que ayudan a comprender el empleo de estas tecnologías y posee varios índices que permiten su empleo como obra de referencia.

El contenido de este libro complementa a "Programación en Internet: Clientes Web", publicado por el autor en esta misma editorial.



ECU®
EDITORIAL CLUB UNIVERSITARIO

www.ecu.fm

NOTA DEL AUTOR

Este libro fue publicado originalmente con copyright (todos los derechos reservados) por el autor y el editor.

La publicación actual de este libro se realiza bajo la licencia Creative Commons Reconocimiento-NoComercial-SinObrasDerivadas 3.0 España que se resume en la siguiente página. La versión completa se encuentra en la siguiente dirección:

<http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.es>



[Creative Commons](#)

Creative Commons License Deed

Reconocimiento-NoComercial-SinObraDerivada 3.0 España (CC BY-NC-ND 3.0)

Usted es libre de:



copiar, distribuir y comunicar públicamente la obra

Bajo las condiciones siguientes:



Reconocimiento — Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciadore (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).



No comercial — No puede utilizar esta obra para fines comerciales.



Sin obras derivadas — No se puede alterar, transformar o generar una obra derivada a partir de esta obra.

Entendiendo que:

Renuncia — Alguna de estas condiciones puede [no aplicarse](#) si se obtiene el permiso del titular de los derechos de autor

Dominio Público — Cuando la obra o alguno de sus elementos se halle en el [dominio público](#) según la ley vigente aplicable, esta situación no quedará afectada por la licencia.

Otros derechos — Los derechos siguientes no quedan afectados por la licencia de ninguna manera:

- Los derechos derivados de [usos legítimos](#) u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.
 - Los derechos [morales](#) del autor;
 - Derechos que pueden ostentar otras personas sobre la propia obra o su uso, como por ejemplo [derechos de imagen](#) o de privacidad.
- **Aviso** — Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.

Esto es un resumen legible por humanos del texto legal (la licencia completa) disponible en los idiomas siguientes:

[Asturian](#) [Castellano](#) [Catalán](#) [Euskera](#) [Gallego](#)

Título: Programación de servidores web con CGI, SSI e IDC

Autor: © Sergio Luján Mora

I.S.B.N.: 84-8454-136-3

Depósito Legal: A-1496-2001

Edita: Editorial Club Universitario

www.ecu.fm

Printed in Spain

Imprime: Imprenta Gamma - Telf.: 965 67 19 87

C/. Cottolengo, 25 – San Vicente (Alicante)

e. mail: gamma@gamma.fm

www.gamma.fm

Reservados todos los derechos. Ni la totalidad ni parte de este libro puede reproducirse o transmitirse por ningún procedimiento electrónico o mecánico, incluyendo fotocopia, grabación magnética o cualquier almacenamiento de información y sistema de reproducción, sin permiso previo y por escrito de los titulares del Copyright.

Programación de servidores web con CGI, SSI e IDC

Sergio Luján Mora

Prefacio

Las aplicaciones web (*web-based application*) se clasifican dentro de las aplicaciones cliente/servidor. Por un lado, se tiene el navegador (*browser*) que hace el papel de cliente; por otro lado, se tiene el servidor web que representa la parte servidor. Para crear cada una de las partes, cliente y servidor, se emplean distintas tecnologías. Así, por ejemplo, para programar un cliente web se suele utilizar HTML, JavaScript o *applets* en Java, mientras que para programar un servidor web se emplea CGI, SSI, ASP o JSP.

En este libro se repasan las tecnologías que fueron esenciales en la programación de los servidores web durante los primeros años de la web. Las tres tecnologías que se presentan en este libro, CGI, SSI e IDC, permiten crear páginas web dinámicas.

Mientras que CGI y SSI aún se emplean muy a menudo, IDC ha sido superado por tecnologías que han aparecido posteriormente. Sin embargo, debido a la sencillez de IDC, he considerado que es un punto de inicio muy adecuado para afrontar el estudio de tecnologías más avanzadas pero a su vez más complicadas.

Para afrontar correctamente el estudio de los temas tratados en este libro, hace falta poseer unos mínimos conocimientos sobre HTML. Existen multitud de libros sobre HTML, pero recomiendo la consulta del libro *Programación en Internet: Clientes Web* que he publicado en Editorial Club Universitario. En él, se trata la programación de la parte cliente de las aplicaciones web y en particular se estudian HTML y JavaScript.

El contenido de este libro se ha dividido en tres capítulos y un apéndice. Los tres capítulos son independientes, por lo que se pueden leer en cualquier orden. El libro además posee una serie de índices que permiten su empleo como obra de referencia.

El capítulo primero trata sobre CGI: presenta el estándar CGI, describe

las distintas formas que existen de enviar información a un programa CGI, explica cómo emplear las variables de entorno y comenta algunos consejos que pueden ayudar a lograr programas CGI más seguros. El lenguaje empleado para programar los CGI de ejemplo que contiene este capítulo es *C*, por lo que es necesario poseer unos conocimientos mínimos de *C* o *C++* para comprenderlos.

El segundo capítulo está dedicado a SSI. Se explica su uso, los comandos más comunes (no todos los servidores web aceptan los mismos comandos) y se incluyen varios ejemplos.

El tercer capítulo explica la tecnología IDC de Microsoft y cómo generar páginas web dinámicas a partir de la información almacenada en una base de datos.

Por último, el único apéndice del libro complementa el capítulo tres, ya que explica como crear un DSN para acceder a una base de datos mediante ODBC.

Para finalizar, quisiera mandar un abrazo a mi familia y a Marisa, la gente que quiero; un saludo a los amigos y compañeros del Laboratorio Multimedia (mmlab), con los que trabajé y disfruté de buenos momentos, y otro saludo a los amigos y compañeros del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Alicante, con los que trabajo (y espero seguir trabajando).

Alicante, 11 de noviembre de 2001

Sergio Luján Mora

Índice general

Prefacio	III
Índice general	V
Índice de cuadros	IX
Índice de figuras	XI
Índice de acrónimos	XIII
1. CGI	1
1.1. Introducción	2
1.2. Un ejemplo	5
1.3. Aplicaciones	5
1.4. Qué necesito para programar un CGI	6
1.5. Lenguaje de programación	7
1.5.1. Independencia de plataforma	8
1.5.2. Independencia de servidor	8
1.6. Razones para emplear CGI	9
1.7. Razones para no emplear CGI	10
1.8. El primer CGI	11
1.9. Cómo comunicarse directamente con el cliente	17
1.10. Cómo envía el servidor información a un CGI	17
1.10.1. A través de la línea de comandos	18
1.10.2. Cómo tratar los formularios	22
1.10.3. A través de la URL	24
1.10.4. A través de la entrada estándar	25

1.10.5. A través de información de ruta	26
1.11. Variables de entorno CGI	26
1.11.1. Específicas del servidor	26
1.11.2. Específicas del cliente	27
1.11.3. Específicas de la petición	28
1.11.4. Cómo acceder a las variables desde C	29
1.12. Un ejemplo más complejo	31
1.13. Seguridad	36
1.13.1. Permisos de ejecución	36
1.13.2. Examina el código	39
1.13.3. Versiones estables	39
1.13.4. Las presunciones son peligrosas	39
1.13.5. Programa defensivamente	40
1.13.6. Limpia los datos antes de usarlos	40
1.13.7. Limpia los datos antes de pasarlos a otro programa	42
1.13.8. Cuidado con HTML	42
1.13.9. Nivel de privilegio	42
1.13.10. Nivel de prioridad	43
1.13.11. Usa un ordenador para los CGIs	43
1.13.12. Consulta listas de correo y grupos de noticias	43
1.13.13. Nunca olvides el código fuente	43
1.14. WinCGI	44
2. SSI	47
2.1. Introducción	48
2.2. Qué necesito para programar mediante SSI	48
2.3. Procesamiento de los archivos	49
2.4. Comentarios HTML y comandos SSI	50
2.5. Comandos SSI más comunes	51
2.5.1. config	51
2.5.2. echo	54
2.5.3. exec	58
2.5.4. flastmod	59
2.5.5. fsize	61
2.5.6. include	61
2.6. Ejemplo de programa SSI	62

3. IDC	67
3.1. Introducción	68
3.2. Cómo funciona	68
3.3. Qué necesito para programar mediante IDC	69
3.4. Un IDC sencillo	72
3.5. El archivo .idc	74
3.5.1. Campos obligatorios	74
3.5.2. Campos opcionales	75
3.5.3. Campos opcionales avanzados de ODBC	77
3.6. El archivo .htx	77
3.6.1. Valor de un campo en un formulario	78
3.6.2. Variables integradas	79
3.7. Cómo procesar los campos de un formulario	79
3.8. Un IDC más complejo	80
3.8.1. Ejemplo 1	80
3.8.2. Ejemplo 2	81
3.8.3. Ejemplo 3	84
A. Cómo crear un DSN	87
A.1. ODBC	87
A.2. Creación de un DSN	91
Bibliografía	99
Índice alfabético	101

Índice de cuadros

1.1. Diferencias entre una página HTML normal y una página generada a partir de un CGI	4
1.2. Lenguajes de programación más comunes	8
1.3. Tipos MIME más comunes	12
1.4. Códigos de estado HTTP más usuales	16
1.5. Caracteres especiales en la codificación URL	24
2.1. Modificadores de timefmt	55
2.2. Ejemplos de distinto formato fecha	56
2.3. Modificadores de sizefmt	56
2.4. Parámetros del comando flastmod, fsize e include	60
3.1. Operadores de las expresiones lógicas	78

Índice de figuras

1.1. Esquema básico de una aplicación web basada en CGI	4
1.2. Mensaje de error porque el encabezado no es correcto	15
1.3. Ejecución desde una ventana de MS-DOS	19
1.4. Página con cuadro de texto ISINDEX para realizar una búsqueda	21
1.5. Página de respuesta a una búsqueda ISINDEX	22
1.6. Ejemplo de variables de entorno	30
1.7. cgi-select: página 1	37
1.8. cgi-select: página 2	38
1.9. Permisos de ejecución en Microsoft Personal Web Server	38
2.1. Permisos de ejecución en Microsoft Personal Web Server	50
2.2. Mensaje de error por defecto	53
2.3. Mensaje de error personalizado	53
2.4. Ejemplo de comando echo	57
2.5. Ejemplo de comando exec	60
2.6. Ejemplo de programa ejecutado mediante exec	65
3.1. Esquema básico de una aplicación web basada en IDC	70
3.2. Mensaje de error porque no hay permisos de ejecución	71
3.3. Permisos de ejecución en Microsoft Personal Web Server	72
3.4. Ejemplo de un IDC sencillo	73
3.5. Mensaje de error porque no existe DNS	74
3.6. Formulario de toma de datos para inserción	83
3.7. Formulario de acceso a la parte privada	85
A.1. Mecanismos de acceso a bases de datos	89
A.2. Arquitectura de ODBC	90
A.3. Fuentes de datos ODBC	92

A.4. Pantalla principal de Fuentes de datos ODBC	93
A.5. Selección del controlador	94
A.6. Creación de un DSN para Microsoft Access	95
A.7. Seleccionar una base de datos	96
A.8. Crear una base de datos	97

Índice de acrónimos

API *Application Program Interface*

Interfaz de programación de aplicaciones. Conjunto de constantes, funciones y protocolos que permiten programar aplicaciones. Una buena API facilita la tarea de desarrollar aplicaciones, ya que facilita todas las piezas y el programador sólo tiene que unir las para lograr el fin que desea.

ASP *Active Server Pages*

Páginas activas de servidor. Tecnología de MICROSOFT que permite crear páginas web dinámicas en el servidor. Se puede decir que las páginas **ASP** son similares a los programas **CGI**. Las páginas **ASP** suelen estar programadas en *VBScript*, aunque también se pueden programar en otros lenguajes.

ASCII *American Standard Code for Information Interchange*

Código binario utilizado para representar letras, números, símbolos, etc. A cada carácter se le asigna un número del 0 al 127 (7 bits). Por ejemplo, el código **ASCII** para la A mayúscula es 65. Existen códigos **ASCII** extendidos de 256 caracteres (8 bits), que permiten representar caracteres no ingleses como las vocales acentuadas o la ñe. Los caracteres de la parte superior (128 a 255) de estos códigos **ASCII** extendidos varían de uno a otro. Por ejemplo, uno de los más extendidos es **ISO** Latin-1 (oficialmente ISO-8859-1).

CGI *Common Gateway Interface*

Interfaz de pasarela común. Estándar que permite el intercambio de información entre un servidor y un programa externo al servidor. Un programa **CGI** es un programa preparado para recibir y enviar datos desde y hacia un servidor web según este estándar. Normalmente se programan

en *C* o en *Perl*, aunque se puede usar cualquier lenguaje de propósito general.

DLL *Dynamic Link Library*

Librería de enlace dinámico. Fichero que almacena funciones ejecutables o datos que pueden ser usados por una aplicación en **Microsoft Windows**. Una **DLL** puede ser usada por varios programas a la vez y se carga en tiempo de ejecución (no en tiempo de compilación).

DNS *Domain Name System*

Sistema de nombres de dominio. Servicio de Internet que traduce los nombres de dominio en direcciones **IP**. Cada vez que se emplea un nombre de dominio, un servidor de **DNS** tiene que traducir el nombre de dominio en su correspondiente dirección **IP**. Por ejemplo, el nombre de dominio **www.ua.es** se corresponde con la dirección **IP 193.145.233.99**.

DSN *Data Source Name*

Nombre de origen de datos. Un **DSN** representa toda la información necesaria para conectar una aplicación con una base de datos mediante **ODBC**.

HTML *HyperText Markup Language*

Lenguaje de etiquetado de hipertexto. Lenguaje compuesto de una serie de etiquetas o marcas que permiten definir el contenido y la apariencia de las páginas web. Aunque se basa en **SGML**, no se puede considerar que sea un subconjunto. Existen cientos de etiquetas con diferentes atributos. **W3C** se encarga de su estandarización. El futuro sustituto de **HTML** es **XHTML**.

HTTP *HyperText Transfer Protocol*

Protocolo de transferencia de hipertexto. Es el protocolo que se emplea en **WWW**. Define como se tienen que crear y enviar los mensajes y que acciones debe tomar el servidor y el navegador en respuesta a un comando. Es un protocolo *stateless* (sin estado), porque cada comando se ejecuta independientemente de los anteriores o de los posteriores. Actualmente, la mayoría de los servidores soportan **HTTP 1.1**. Una de las principales ventajas de esta versión es que soporta conexiones persistentes: una vez que el navegador se conecta al servidor, puede recibir múltiples ficheros a través de la misma conexión, lo que aumenta el rendimiento de

la transmisión hasta en un 20 %. Se puede consultar el estándar en **RFC 2616** (junio 1999).

IDC *Internet Database Connector*

Conector de bases de datos de Internet. Tecnología propietaria de MICROSOFT que permite generar páginas web dinámicas a partir de la información almacenada en una base de datos. Es el precursor de **ASP**.

IP *Internet Protocol*

Protocolo de Internet. Protocolo básico de Internet perteneciente a la familia **TCP/IP**. Especifica el formato de los paquetes (datagramas) y el esquema de direccionamiento.

ISAPI *Internet Server Application Program Interface*

Un API para el servidor Microsoft Internet Information Server. Permite programar aplicaciones web.

ISO *International Organization for Standards*

Organización fundada en 1946, cuyos miembros son las organizaciones nacionales de normalización (estandarización) correspondientes a los países miembros. Entre sus miembros se incluyen la ANSI (Estados Unidos), BSI (Gran Bretaña), AFNOR (Francia), DIN (Alemania) y UNE (España).

JSP *Java Server Pages*

Tecnología de SUN MICROSYSTEMS que permite crear páginas web dinámicas en el servidor. Equivale a la tecnología **ASP** de MICROSOFT. Se programan en *Java*.

MIME *Multipurpose Internet Mail Extensions*

Se usa en el correo electrónico desde 1992 para enviar y recibir ficheros de distinto tipo. Se puede consultar el estándar en **RFC 1341**, **RFC 1521** y **RFC 1522**.

ODBC *Open Database Connectivity*

Conectividad abierta de bases de datos. **ODBC** es un estándar *de facto* para el acceso a base de datos en entornos cliente/servidor. Mediante **ODBC**, se puede cambiar la parte servidor (la base de datos) sin tener que cambiar el cliente.

RFC *Request for Comments*

Medio de publicar propuestas sobre Internet. Cada **RFC** recibe un número. Algunos se convierten en un estándar de Internet.

SGBD *Sistema Gestor de Bases de Datos*

Programa (o programas) que permite almacenar, modificar y extraer información contenida en una base de datos. Los **SGBD** se pueden clasificar según la forma que tienen de almacenar internamente los datos: modelo relacional, en red, jerárquico, etc.

SGML *Standard Generalized Markup Language*

Lenguaje que permite organizar y etiquetar los distintos elementos que componen un documento. Se emplea para manejar grandes documentos que sufren constantes revisiones y se imprimen en distintos formatos. Desarrollado y estandarizado por **ISO** en 1986 (ISO 8879:1986).

SQL *Structured Query Language*

Lenguaje de consulta estructurado. Lenguaje estandarizado de acceso a bases de datos. Basado en SEQUEL (*Structured English Query Language*), diseñado por IBM en 1974. Existen distintas versiones, siendo la más conocida SQL-92 y la última publicada y estandarizada SQL-1999.

SSI *Server Side Include*

Directivas de inclusión del servidor. Comandos que se incluyen en una página **HTML** y que son ejecutados por el servidor web antes de transmitir la página al cliente. Permite generar páginas web dinámicas.

TCP/IP *Transmission Control Protocol/Internet Protocol*

Familia de protocolos que se emplean en las comunicaciones de Internet.

URL *Universal Resource Locator*

También conocido como *Uniform Resource Locator*. Sistema de direccionamiento de máquinas y recursos en Internet. Es decir, se trata de una dirección que permite localizar cualquier máquina o documento que se encuentre accesible a través de Internet.

W3C *World Wide Web Consortium*

Consortio internacional de compañías involucradas en el desarrollo de Internet y en especial de la **WWW**. Su propósito es desarrollar estándares y “poner orden” en Internet.

WWW *World Wide Web*

Sistema de servidores web conectados a Internet (no todos los ordenadores conectados a Internet forman parte de la **WWW**). Su protocolo de comunicación es **HTTP**, su lenguaje de creación de documentos **HTML** y su sistema de direccionamiento de los recursos **URL**. Los navegadores web (*browsers*) permiten navegar por la web.

XHTML *Extensible HyperText Markup Language*

HTML escrito según las normas que marca **XML**. Por tanto, se trata de una aplicación concreta de **XML** y no tienen que confundirse entre sí.

XML *Extensible Markup Language*

Metalinguaje de etiquetado basado en **SGML**. Diseñado específicamente para la **WWW** por **W3C**. Permite que un usuario diseñe sus propias etiquetas, con sus atributos y las reglas de construcción de documentos (sintaxis).

Capítulo 1

CGI

El interfaz CGI permite que un cliente web (un navegador) ejecute un programa en el servidor web. Por medio de CGI se pueden crear páginas web dinámicas. El programa CGI y el servidor web se comunican a través de la salida y entrada estándar. Los programas CGI pueden ser escritos mediante diferentes lenguajes de programación.

Índice General

1.1. Introducción	2
1.2. Un ejemplo	5
1.3. Aplicaciones	5
1.4. Qué necesito para programar un CGI	6
1.5. Lenguaje de programación	7
1.5.1. Independencia de plataforma	8
1.5.2. Independencia de servidor	8
1.6. Razones para emplear CGI	9
1.7. Razones para no emplear CGI	10
1.8. El primer CGI	11
1.9. Cómo comunicarse directamente con el cliente . . .	17
1.10. Cómo envía el servidor información a un CGI . . .	17
1.10.1. A través de la línea de comandos	18
1.10.2. Cómo tratar los formularios	22

1.10.3. A través de la URL	24
1.10.4. A través de la entrada estándar	25
1.10.5. A través de información de ruta	26
1.11. Variables de entorno CGI	26
1.11.1. Específicas del servidor	26
1.11.2. Específicas del cliente	27
1.11.3. Específicas de la petición	28
1.11.4. Cómo acceder a las variables desde C	29
1.12. Un ejemplo más complejo	31
1.13. Seguridad	36
1.13.1. Permisos de ejecución	36
1.13.2. Examina el código	39
1.13.3. Versiones estables	39
1.13.4. Las presunciones son peligrosas	39
1.13.5. Programa defensivamente	40
1.13.6. Limpia los datos antes de usarlos	40
1.13.7. Limpia los datos antes de pasarlos a otro programa	42
1.13.8. Cuidado con HTML	42
1.13.9. Nivel de privilegio	42
1.13.10. Nivel de prioridad	43
1.13.11. Usa un ordenador para los CGIs	43
1.13.12. Consulta listas de correo y grupos de noticias	43
1.13.13. Nunca olvides el código fuente	43
1.14. WinCGI	44

1.1. Introducción

Common Gateway Interface (**CGI**) es un interfaz que permite transferir información entre un servidor web y un programa externo al servidor. ¿Por qué es necesario el estándar **CGI**? Si queremos acceder desde un servidor web a una aplicación externa, una primera solución puede ser incluir en el servidor web un interfaz para cada una de las aplicaciones externas que se quiera ejecutar. Pero esta solución es claramente inviable: es difícil y laborioso programar un

servidor web para que pueda acceder a todas las posibles aplicaciones existentes y, además, mantenerlo “al día” según surjan nuevas aplicaciones. En vez de ello, mediante **CGI** se establece un conjunto de normas (protocolo) que deben de seguir los servidores web y las aplicaciones para poder interactuar entre sí.

En la Figura 1.1 está representado el funcionamiento básico de una aplicación web basada en **CGI**:

1. El cliente web (el navegador) lanza una petición nueva mediante *HyperText Transfer Protocol* (**HTTP**). Esta petición puede ir acompañada de datos codificados por el navegador (por ejemplo, información introducida por el usuario en un formulario).
2. El servidor web recibe la petición, analiza la *Universal Resource Locator* (**URL**) y detecta que se trata de un programa **CGI**. Ejecuta el **CGI** y le pasa los datos codificados.
3. El **CGI** recibe los datos codificados, los descodifica y realiza su función (en algunos casos, es posible que un programa **CGI** no necesite recibir datos para cumplir su misión). La función que realiza el programa **CGI** se puede clasificar en procesamiento directo (el programa **CGI** realiza por sí mismo todo el procesamiento de los datos recibidos) y procesamiento indirecto (el programa **CGI** interactúa con otras aplicaciones que son las verdaderas destinatarias de los datos recibidos, como por ejemplo, un *Sistema Gestor de Bases de Datos* (**SGBD**)).
4. El programa **CGI** genera su resultado: una página *HyperText Markup Language* (**HTML**), una imagen, un archivo de sonido, etc. y lo envía al servidor web.
5. El servidor web procesa la información recibida del programa **CGI**: le añade el código necesario para formar un encabezado **HTTP** correcto¹.
6. El servidor web reenvía el resultado del programa **CGI** al cliente web.
7. El cliente web muestra la salida del programa **CGI**.

¹Más adelante veremos que se puede evitar este procesamiento y “hablar” directamente al cliente.

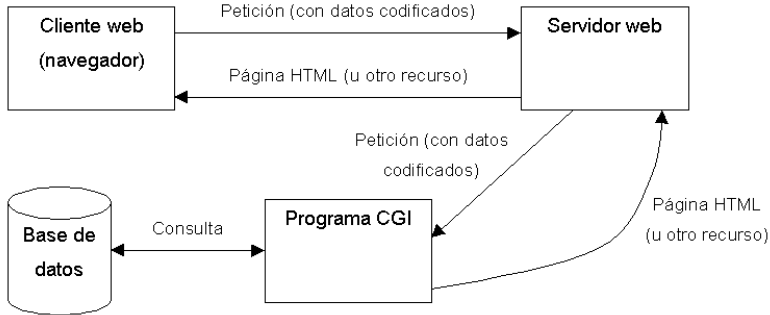


Figura 1.1: Esquema básico de una aplicación web basada en CGI

El uso de **CGI** supone un aumento en la complejidad de los sitios web, ya que se requieren conocimientos de programación y de administración de permisos de los sistemas operativos empleados en los servidores web.

La versión actual de este estándar es **CGI/1.1**. Las principales diferencias entre una página **HTML** normal y una página generada a partir de un programa **CGI** aparecen resumidas en el Cuadro 1.1.

Página HTML	CGI
El servidor web recupera la página	El servidor web ejecuta el programa CGI
El contenido es estático	El contenido puede ser dinámico

Cuadro 1.1: Diferencias entre una página HTML normal y una página generada a partir de un CGI

¿Qué se puede hacer con un **CGI**? En principio, no hay limitaciones. Pero siempre hay que tener en cuenta la siguiente recomendación: *cualquier cosa que haga un CGI, lo tiene que hacer rápidamente y empleando la menor cantidad posible de recursos*. Si no, el usuario se desesperará, se conectará a otra página y se prometerá a sí mismo no volver a visitar esa web donde las páginas tardaban una eternidad.

1.2. Un ejemplo

Cuando se introduce en el formulario de un buscador (por ejemplo, GOOGLE, ALTAVISTA o YAHOO!) un término a buscar, el navegador (Microsoft Internet Explorer o Netscape Communicator) envía una petición al servidor web (Apache o Microsoft Internet Information Server²) en la que se solicita una página nueva y que se acompaña del término a buscar.

El servidor web recibe la solicitud, comprueba que la página que se solicita es un programa **CGI** y lo ejecuta pasándole el término a buscar. Además del término a buscar, le pasa información auxiliar en forma de variables de entorno, como por ejemplo la dirección *Internet Protocol* (**IP**) del cliente, método que ha empleado para enviar el término a buscar, etc.

El programa **CGI** realiza una búsqueda en una base de datos (o en un fichero plano) y localiza la información solicitada. El programa **CGI** genera de forma dinámica y en tiempo real una página **HTML** nueva a partir de la información encontrada y envía el resultado al servidor web.

El servidor web reenvía la página generada por el programa **CGI** al navegador.

1.3. Aplicaciones

El uso de programas **CGI** permite incorporar interacción en un sitio web: en vez de un sitio web estático se puede tener un sitio interactivo que se adapte a las necesidades de los distintos usuarios (páginas web dinámicas según el perfil de cada usuario). Las aplicaciones de los programas **CGI** son múltiples:

- Gestión de un libro de visitas o firmas (*guestbook*). Mediante un **CGI** se puede recuperar la información introducida en un formulario de un libro de visitas, almacenarla en un fichero y mostrar en una página web todas las visitas recibidas.
- Gestión de anuncios (*banners*). Mediante un **CGI** se pueden mostrar de forma aleatoria o de forma prefijada (por ejemplo, según la hora del día o según la dirección **IP** del cliente) distintos anuncios con distintas direcciones de enlace. Además, se puede controlar el número de pulsaciones (*clicks*) que recibe cada anuncio.

²Normalmente se conoce por sus siglas: IIS.

- Gestión de contadores (*hit counters*). Ya sea contadores en modo texto o modo gráfico (el programa **CGI** devuelve una imagen que contiene el valor del contador).
- Imágenes sensibles procesadas en el servidor web³. Las imágenes sensibles o mapas de imágenes son imágenes que contienen zonas activas que actúan como enlaces: en función de la zona de la imagen en la que pulse el usuario, se activa un enlace hacia un documento u otro. También se puede hacer zoom en una imagen mediante esta técnica.
- Acceso a bases de datos. Se puede emplear un programa **CGI** como pasarela (de ahí el nombre de *gateway*) para acceder a una base de datos. De este modo, se pueden crear aplicaciones como buscadores, comercio electrónico, etc.

1.4. Qué necesito para programar un CGI

Para poder programar un **CGI** y probarlo hacen falta los siguientes programas:

- Un editor de textos como Bloc de notas de Microsoft Windows o joe de Linux para crear las páginas **HTML** que conectan con el programa **CGI** y para crear el propio código del programa **CGI**.
- Si se va a programar el **CGI** mediante un lenguaje compilado (*C*, *C++*, *Pascal*, etc.), hace falta el correspondiente compilador. Si se va a programar mediante un lenguaje interpretado (*Perl*, *shell* de Unix, etc.), hace falta el correspondiente intérprete.
- Un servidor web (ya sea local o remoto) en el que se puedan ejecutar programas **CGI**. Por ejemplo, Microsoft Personal Web Server, Microsoft Internet Information Server o Apache.
- Por último, un navegador como Netscape Communicator o Microsoft Internet Explorer para poder comprobar las páginas **HTML** y los programas **CGI**.

³También existen las imágenes sensibles procesadas en el cliente.

No es necesario disponer de una conexión a Internet, ya que se puede comprobar localmente el código creado.

Lo que sí que es recomendable es utilizar un buen editor de textos, que sea cómodo, configurable, soporte macros, etc. y que sea *syntax highlight*. Esta última característica significa que el editor es capaz de comprender el lenguaje en el que se programa, y colorea las palabras diferenciándolas según sean variables, palabras reservadas, comentarios, etc.

1.5. Lenguaje de programación

Como un **CGI** es un programa que se ejecuta en el servidor, se puede programar en cualquier lenguaje que permita crear ejecutables para el sistema operativo del servidor. Lo único que se le exige al lenguaje de programación es que sea capaz de:

- Leer datos de la entrada estándar.
- Acceder a las variables de entorno.
- Escribir en la salida estándar.

Por tanto, la elección de un lenguaje se basa principalmente en qué lenguajes se conocen y qué lenguajes están disponibles en el sistema. Probablemente, *C* y *Perl* son los lenguajes más empleados a la hora de programar **CGI**.

Por razones históricas, a los programas **CGI** se les suele llamar también *scripts*⁴, porque al principio se programaban con lenguajes de *script*. Mucha gente prefiere escribir los programas **CGI** con lenguajes de *script* en vez de lenguajes compilados, porque son más fáciles de depurar, modificar y mantener que un programa compilado. Sin embargo, los programas compilados son más rápidos a la hora de ejecutarse, ya que los *scripts* son interpretados.

Por tanto, la lista de lenguajes de programación que se pueden emplear no tiene límite; en el Cuadro 1.2 se muestran los más empleados en la programación de **CGI** (la lista no es excluyente: nada nos impide programar un **CGI** en *Python*, *Fortran*, *Pascal*, *TCL* o en nuestro lenguaje favorito).

Como la tecnología **CGI** se encuentra muy extendida en el mundo Internet, existen multitud de librerías en los distintos lenguajes de programación que

⁴Normalmente, se emplea la palabra programa para denotar aplicaciones y código “largo y compilado” mientras que *script* hace referencia a código “corto y no compilado”.

Lenguaje	Sistema	Tipo
Cualquier shell de Unix	Unix	Interpretado
Perl	Unix, Windows, MacOS	Interpretado
C, C++	Unix, Windows, MacOS	Compilado
Visual Basic	Windows	Compilado
AppleScript	MacOS	Interpretado
REXX	OS2	Interpretado

Cuadro 1.2: Lenguajes de programación más comunes

facilitan la creación de programas **CGI**: *cgi-lib (Perl)*, *CGI-HTML (C)*, *AHTML (C++)*, etc.

1.5.1. Independencia de plataforma

La independencia de plataforma implica la capacidad de ejecutar el código de un **CGI** en distinto *hardware* o *software* (sistema operativo) sin tener que modificarlo. La mejor forma de lograrlo es por medio de un “lenguaje universal” y no empleando código específico del sistema (llamadas al sistema operativo, por ejemplo).

Esto se traduce en el uso de lenguajes como *C* y *Perl*, que están disponibles prácticamente en cualquier plataforma. Si se tienen que emplear llamadas al sistema operativo, es conveniente aislar el código que las realiza en módulos independientes, de forma que al trasladar el código de una plataforma a otra se minimizan y facilitan los cambios necesarios.

1.5.2. Independencia de servidor

La independencia de servidor significa que el código se puede ejecutar en distintos servidores web sobre el mismo sistema operativo sin tener que modificarlo. Esta independencia es más sencilla de conseguir que la anterior, pero hay que observar una serie de recomendaciones:

- No asumir que el programa se ejecutará en un directorio concreto.
- No asumir que algunos directorios se hallan siempre en la misma ruta. Por ejemplo, suponer que el directorio temporal se encuentra siempre en

C:\TEMP o que el directorio principal del servidor web es C:\INETPUB\WWWROOT es muy peligroso.

- No asumir que el programa se va a ejecutar con unos permisos (privilegios) concretos.
- No asumir la existencia de configuraciones de red concretas: direcciones **IP**, dominios, etc.
- No asumir la presencia de programas externos, como por ejemplo, suponer que está disponible en cualquier instalación el programa `sendmail` de Unix.

Si deseamos distribuir un programa **CGI** que hemos desarrollado, para evitar todos estos problemas, la mejor solución es proporcionar al usuario la posibilidad de configurar los valores dependientes del servidor mediante un fichero de configuración.

1.6. Razones para emplear CGI

En los primeros años de la era web (1992-1997), **CGI** era la única posibilidad que se tenía de añadir interactividad y dinamismo a los sitios web. Pero desde entonces han surgido distintas soluciones que sustituyen completamente este estándar. Entonces, ¿por qué seguir usando **CGI**? Existen diversas razones:

1. **CGI** es el método más rápido cuando se ejecuta mucho código. Sin embargo, cuando el código que se tiene que ejecutar es pequeño y poco complejo, las páginas activas como *Active Server Pages (ASP)*, *Java Server Pages (JSP)* o PHP son la mejor solución, debido a la sobrecarga que supone ejecutar una aplicación externa al servidor web.
2. **CGI** es un estándar, compatible con la mayoría (por no decir la totalidad) de los servidores web. Podemos crear un programa **CGI** que se ejecute en distintos servidores web en distintas plataformas.
3. **CGI** es un estándar compatible con todos los clientes web.

4. Un programa **CGI** se puede escribir prácticamente en cualquier lenguaje. Por tanto, si se conoce un lenguaje de programación, se puede escribir un **CGI** desde el primer día.
5. Cómo es una tecnología establecida y probada (es decir, “antigua”), existen multitud de recursos, tales como tutoriales, programas **CGI** gratuitos, librerías, etc. La mayoría de los problemas que nos pueden surgir ya han sido resuelto y sólo hay que buscar qué soluciones se han planteado y cuál es la mejor.

1.7. Razones para no emplear CGI

Como se ha comentado en el apartado anterior, existen una serie de ventajas a la hora de emplear **CGI**. Sin embargo, el estándar **CGI** también tiene sus inconvenientes:

1. **CGI** es una tecnología obsoleta. Desde su nacimiento, han surgido otras posibilidades: *applets*, *servlets*, **ASP**, ColdFusion, **JSP**, **PHP**, etc.
2. **CGI** no mantiene el estado automáticamente⁵. Otras tecnologías (**ASP**, por ejemplo) mantienen el estado, lo que facilita la programación de aplicaciones web como “carritos de la compra” (*market cart*) o lectores de correo a través de la web (*webmail*). Para resolver esta carencia, se suelen emplear los campos ocultos de los formularios⁶: en ellos se almacenan las selecciones del usuario o un identificador único (*id*) que permite seguir su actividad de una página a otra.
3. La integración entre un programa **CGI** y el servidor web es muy débil. La única comunicación que se establece entre ambos es para transmitir los datos de entrada y la salida producida por el programa⁷.

⁵Está es una limitación que se debe realmente al protocolo **HTTP** y no a **CGI**. El protocolo **HTTP** es un protocolo “sin estado” (*stateless*): cada vez que un cliente solicita un recurso (una página **HTML**, por ejemplo) al servidor web, es como si fuera la primera vez que lo hace. Entre las distintas peticiones no se almacena ningún tipo de información sobre el cliente en el servidor.

⁶`<INPUT TYPE="HIDDEN">`.

⁷Esto es una desventaja y a la vez una ventaja: gracias a que la integración es tan débil, un programa **CGI** bien hecho es independiente de la plataforma y se puede usar sin problemas en distintos servidores web.

4. Cada vez que se tiene que ejecutar un programa **CGI**, se crea una instancia nueva del programa en memoria.

1.8. El primer CGI

La salida o resultado que produce un programa **CGI** se tiene que dirigir a la salida estándar (**stdout**). Un programa **CGI** puede devolver cualquier tipo de documento. Cada documento que un **CGI** envía a un servidor web debe contener una cabecera (también llamado encabezado **HTTP**) al principio del mismo que indica el tipo de documento que es y así tanto el servidor como el cliente web⁸ lo pueden procesar adecuadamente. El tipo del documento se expresa mediante los tipo **MIME**. Los tipos **MIME** básicos (**text**, **multipart**, **message**, **application**, **image**, **audio**, **video**) se dividen en subtipos. En el Cuadro 1.3 se muestran los tipos **MIME** más comunes y las extensiones asociadas a esos tipos.

La cabecera de la respuesta se compone de una serie de líneas con texto *American Standard Code for Information Interchange* (**ASCII**) separadas entre sí por saltos de línea. Muy importante: al final de la cabecera se tiene que dejar una línea en blanco⁹, que indica donde termina la cabecera y empieza el cuerpo del mensaje de respuesta. A continuación viene el cuerpo de la respuesta, que puede estar en cualquier formato (texto **ASCII**, formato binario para una imagen, archivo de sonido, etc.).

Por ejemplo, la salida que tiene que generar un programa **CGI** para enviar una página **HTML** sencilla con la frase ¡Hola mundo! es:

Ejemplo 1.1

```
1 Content-type: text/html
2
3 <HTML>
4 <BODY>
5 ¡Hola mundo!
```

⁸Los navegadores web usan los tipos *Multipurpose Internet Mail Extensions* (**MIME**) para saber con que programa tienen que mostrar un documento que no pueden tratar directamente (por ejemplo, un documento de Microsoft Word). Estos programas pueden ser externos al navegador o estar incluidos en él en forma de *plug-ins*.

⁹La línea en blanco se puede indicar con un salto de línea (**LF**) o con un retorno de carro y un salto de línea (**CR + LF**).

Tipo	Extensión
application/msword	doc
application/octet-stream	bin exe
application/pdf	pdf
application/x-shockwave-flash	swf
audio/midi audio/x-midi	midi mid
image/gif	gif
image/jpeg	jpeg jpe jpg
text/html	html htm
text/plain	txt
text/richtext	rtx
text/vnd.wap.wml	wml
text/xml	xml xsl
video/mpeg	mpeg mpg mpe
video/quicktime	qt mov
video/msvideo video/x-msvideo	avi

Cuadro 1.3: Tipos MIME más comunes

```
6 </BODY>
7 </HTML>
```

En la primera línea se indica el tipo **MIME** del contenido de la respuesta. El formato que se emplea es **Content-type: tipo MIME**. En este caso, como se trata de una página **HTML** empleamos `text/html`. A continuación, como la cabecera de la respuesta ha terminado, se tiene que dejar una línea en blanco. Por último, se incluye el contenido de la respuesta. El siguiente **CGI** programado en *C* genera como salida la página **HTML** anterior¹⁰. La línea en blanco que separa la cabecera del cuerpo de la respuesta se crea mediante la instrucción `printf("\n");` de la línea 6; esta instrucción se ha dejado en una línea sola a propósito.

Ejemplo 1.2

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     printf("Content-type: text/html\n");
6     printf("\n");
7     printf("<HTML>\n<BODY>\n");
8     printf(";Hola mundo!\n");
9     printf("</BODY>\n</HTML>\n");
10    return 0;
11 }
```

Por otro lado, no confundir el salto de línea `\n` con la instrucción salto de línea `
` del código **HTML**. Las tres instrucciones que generan el código de la respuesta se pueden resumir en una sola sin ningún salto de línea:

Ejemplo 1.3

```
1 printf("<HTML><BODY>;Hola mundo!</BODY></HTML>");
```

¹⁰Para generar la salida, se puede emplear la instrucción `printf(...)` o `fprintf(stdout, ...)`.

Los saltos de línea los incluimos para facilitar la lectura del código **HTML** si lo visualizamos directamente desde el navegador¹¹.

Otra posibilidad que se ofrece es redirigir (*redirect*) la respuesta a otra página¹². En vez de generar el documento de salida, se puede simplemente indicar al cliente web donde puede encontrarlo. Para ello se emplea el formato `Location: URL`, donde `URL` puede ser una dirección de cualquier tipo (absoluta, relativa, a otro servidor, etc.). Por ejemplo, el siguiente programa **CGI** en *C* redirige la respuesta a la dirección `http://www.ua.es`.

Ejemplo 1.4

```

1 #include <stdio.h>
2
3 int main(int argc, char *argv[])
4 {
5     printf("Content-type: text/html\n");
6     printf("Location: http://www.ua.es\n");
7     printf("\n");
8     printf("<HTML>\n<BODY>\n");
9     printf("Nueva dirección: ");
10    printf("<A HREF=\"http://www.ua.es\">http://www.ua.es</A>\n");
11    printf("</BODY>\n</HTML>\n");
12    return 0;
13 }
```

Algunos navegadores antiguos no aceptan la redirección (no reconocen la instrucción `Location`). Por ello, es conveniente, tal como se ha hecho en el código anterior, incluir la posibilidad de que los navegadores antiguos también puedan acceder a la información (¡aunque de forma manual a través de un enlace!). Si no se quiere “dar soporte” a los navegadores antiguos, el código anterior se puede reducir al siguiente:

Ejemplo 1.5

```

1 #include <stdio.h>
2
```

¹¹Netscape Communicator: botón derecho del ratón y elegir **View Source**; Microsoft Internet Explorer: botón derecho del ratón y seleccionar **Ver código fuente**. También se puede acceder a través de los menús.

¹²Aunque parezca una posibilidad poco útil, se puede aprovechar esta posibilidad para mantener un registro de los enlaces que selecciona un usuario. También se puede emplear para redirigir de forma aleatoria.

```
3 int main(int argc, char *argv[])
4 {
5     printf("Location: http://www.ua.es\n");
6     printf("\n");
7     return 0;
8 }
```

Aunque parezca repetitivo, hay que recordar siempre dejar una línea en blanco al final de la cabecera, incluso aunque no haya cuerpo. En la Figura 1.2 se muestra el mensaje de error que muestra el navegador Microsoft Internet Explorer 5.5 cuando en el programa **CGI** anterior se elimina la línea `printf("\n");` de la línea 6.

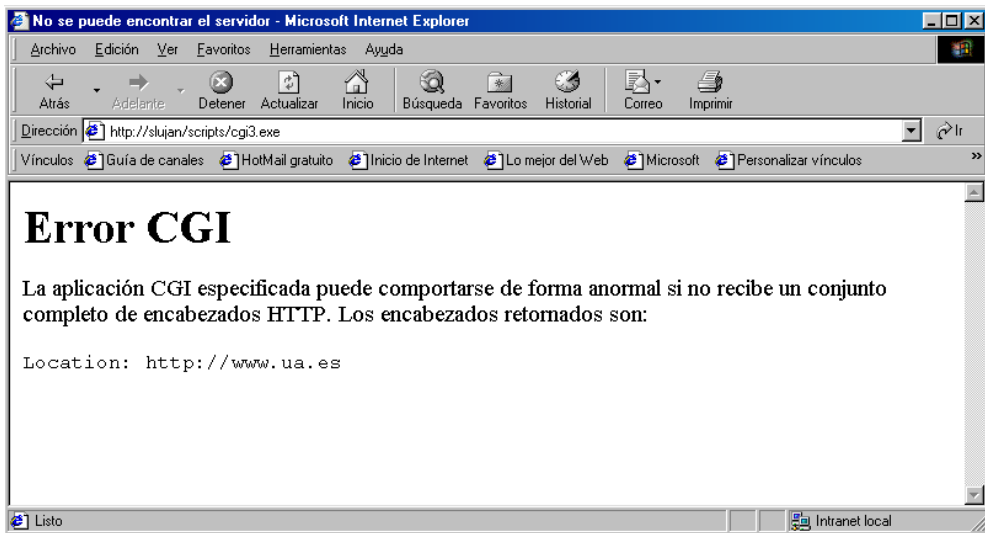


Figura 1.2: Mensaje de error porque el encabezado no es correcto

Existe una última directiva que permite a un programa **CGI** comunicar un código y mensaje de error. Para ello, se emplea la instrucción `Status: nnn xxxxx`, donde `nnn` es un código de estado de tres dígitos y `xxxxx` es un mensaje de error. En el Cuadro 1.4 mostramos algunos de los códigos más usuales.

Código	Resultado	Descripción
200	OK	Ningún problema
202	Accepted	La petición se está procesando, pero ha sido aceptada
204	No Response	El servidor no desea enviar ninguna respuesta
301	Moved	El documento se ha trasladado a un nuevo sitio
302	Found	El documento no está donde se esperaba, pero se ha encontrado en algún otro sitio en el servidor
400	Bad Request	La sintaxis de la petición HTTP no es correcta
401	Unauthorized	El documento requiere unos permisos que no posee el usuario
403	Forbidden	El servidor deniega el acceso al documento
404	Not Found	El servidor no puede encontrar el documento
500	Server Error	El servidor ha generado un error
502	Service Overloaded	El servidor está muy ocupado y no puede servir la petición

Cuadro 1.4: Códigos de estado HTTP más usuales

1.9. Cómo comunicarse directamente con el cliente

Cuando el programa **CGI** envía su salida al servidor web, éste le añade las instrucciones necesarias para formar un mensaje **HTTP** correcto. En algunos casos, se puede querer evitar esta sobrecarga y “hablar” directamente con el cliente web. En este caso, el programa **CGI** es el responsable de crear un mensaje **HTTP** correcto.

Para que el servidor web sepa distinguir unos programas **CGI** de otros, cuando se desee hablar directamente con el cliente, el nombre del programa **CGI** debe comenzar por `nph`¹³. Por ejemplo, las siguientes instrucciones representan un mensaje **HTTP** correcto:

Ejemplo 1.6

```
1 HTTP/1.0 200 OK
2 Server: IIS/4.0
3 Content-type: text/html
4
5 <HTML><BODY>
6 Esto es un mensaje HTTP correcto
7 </BODY></HTML>
```

1.10. Cómo envía el servidor información a un CGI

Un programa **CGI** puede recibir información desde un servidor web de cuatro formas distintas:

- A través de la línea de comandos (*command line*).
- A través de la **URL** (`QUERY_STRING`).
- A través de la entrada estándar (`stdin`).
- A través de información de ruta (`PATH_INFO`).

Un programa **CGI** tiene que saber como va a recibir la información, ya que en cada caso tiene que actuar de distinta forma. Los dos métodos más populares son a través de la **URL** (también llamado método `GET`) y a través de la entrada estándar (método `POST`).

¹³ *No Parse Header*: no se debe analizar la cabecera.

1.10.1. A través de la línea de comandos

La línea de comandos se emplea únicamente en el caso de una búsqueda **ISINDEX**. En estas consultas, el programa **CGI** recibe una lista de términos separados por espacios en blanco. Esta lista se recibe de dos formas:

- Por la línea de comandos: cada término es un argumento de la línea de comandos. Además, los términos se encuentran descodificados.
- Por la **QUERY_STRING**: el servidor crea una variable de entorno¹⁴ llamada **QUERY_STRING** y le asigna una cadena que contiene los términos de la búsqueda. La cadena no se encuentra descodificada.

Desde el cliente web, se puede enviar una petición de consulta **ISINDEX** al servidor web de dos formas: mediante el uso de la etiqueta **ISINDEX** o directamente en la **URL**.

La sintaxis de la etiqueta **ISINDEX** es:

Ejemplo 1.7

```
1 <ISINDEX PROMPT="texto">
```

donde **texto** es el texto que acompaña al cuadro de texto que el usuario puede emplear para introducir términos de búsqueda. Esta etiqueta tiene que emplearse en la cabecera del documento **HTML** (**<HEAD> ... </HEAD>**). En la Figura 1.4 podemos ver como el siguiente código **HTML** que contiene esta etiqueta se muestra en un navegador:

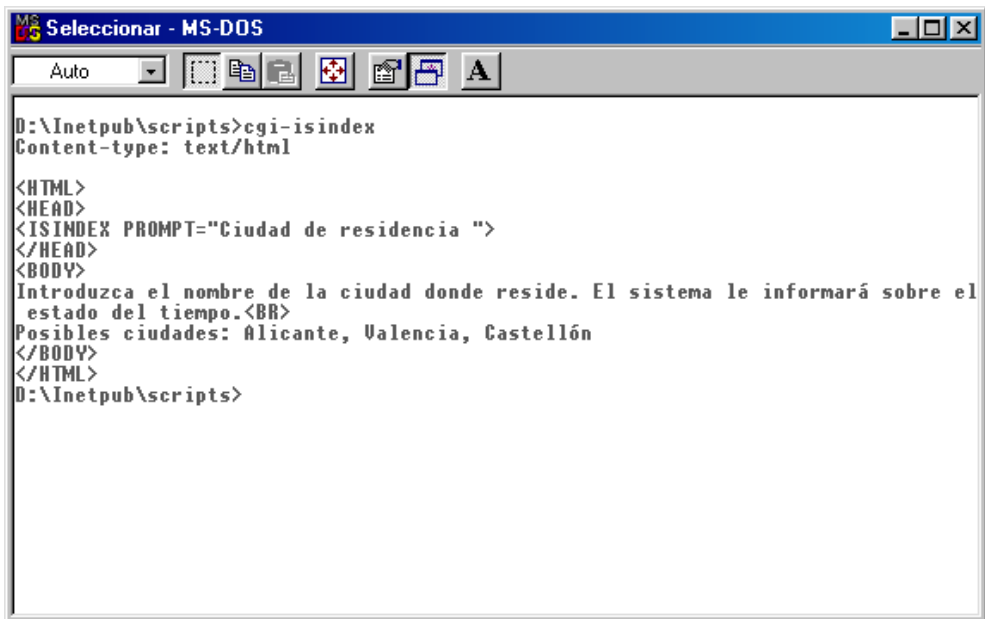
Ejemplo 1.8

```
1 <HTML>
2 <HEAD>
3 <ISINDEX PROMPT="Ciudad de residencia ">
4 </HEAD>
5 <BODY>
6 Introduzca el nombre de la ciudad donde reside. El sistema
7 le informará sobre el estado del tiempo.<BR>
8 Posibles ciudades:
9 Alicante, Valencia, Castellón
10 </BODY>
11 </HTML>
```

¹⁴Más adelante se explican las variables de entorno CGI.

Como se puede observar, en ninguna parte se indica el programa **CGI** que se tiene que ejecutar cuando el cliente realice una consulta (que se realizará cuando el usuario pulse la tecla **Enter** (↵) y el foco esté situado en el cuadro de texto que representa la etiqueta **ISINDEX**). La página **HTML** se va a llamar así misma, así que para que haya procesamiento de algún modo, el código anterior lo tiene que haber generado previamente un programa **CGI**.

El siguiente código en *C* es un programa **CGI** que muestra la primera vez que se ejecuta el documento **HTML** de la Figura 1.4. En la Figura 1.3 podemos ver la salida que produce este programa cuando se ejecuta directamente desde una ventana de MS-DOS.



```
MS-DOS Seleccionar - MS-DOS
Auto
D:\inetpub\scripts>cgi-isindex
Content-type: text/html

<HTML>
<HEAD>
<ISINDEX PROMPT="Ciudad de residencia ">
</HEAD>
<BODY>
Introduzca el nombre de la ciudad donde reside. El sistema le informará sobre el
estado del tiempo.<BR>
Posibles ciudades: Alicante, Valencia, Castellón
</BODY>
</HTML>
D:\inetpub\scripts>
```

Figura 1.3: Ejecución desde una ventana de MS-DOS

Cuando se realiza una consulta, se vuelve a ejecutar el programa **CGI** y éste detecta que se le pasa alguna información a través de la línea de comandos: en el código del programa se puede observar como se consulta la variable **argc** en la línea 9 para saber si se han recibido parámetros a través de la línea de comandos. Por ejemplo, en la Figura 1.5 podemos ver la página que se genera cuando se introduce en el campo de entrada la cadena **Alicante**.

Ejemplo 1.9

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(int argc, char *argv[])
6 {
7     int i;
8
9     if(argc == 1)
10    {
11        printf("Content-type: text/html\n\n");
12        printf("<HTML>\n");
13        printf("<HEAD>\n");
14        printf("<ISINDEX PROMPT=\"Ciudad de residencia \">\n");
15        printf("</HEAD>\n");
16        printf("<BODY>\n");
17        printf("Introduzca el nombre de la ciudad donde reside. ");
18        printf("El sistema le informará sobre el estado del tiempo.");
19        printf("<BR>\n");
20        printf("Posibles ciudades: Alicante, Valencia, Castellón\n");
21        printf("</BODY>\n</HTML>");
22    }
23    else
24    {
25        printf("Content-type: text/html\n\n");
26        printf("<HTML>\n<BODY>\n");
27        if(!strcmp(argv[1], "Alicante"))
28        {
29            printf("<CENTER><IMG SRC=\"nubes.gif\"></CENTER>\n");
30            printf("Cielo nublado. ");
31            printf("Posibilidad de precipitación al anochecer.");
32        }
33        else if(!strcmp(argv[1], "Valencia"))
34        {
35            printf("<CENTER><IMG SRC=\"sol.gif\"></CENTER>\n");
36            printf("Cielo despejado. ");
37            printf("Vientos de aire caliente procedentes de levante.");
38        }
39        else if(!strcmp(argv[1], "Castellón"))
40        {
41            printf("<CENTER><IMG SRC=\"lluvias.gif\"></CENTER>\n");
```

```
42     printf("Lluvias durante todo el día. ");
43     printf("Riesgo alto de granizo y nieve.");
44 }
45 else
46     printf("El nombre de ciudad <I>%s</I> no es correcto", argv[1]);
47     printf("</BODY>\n</HTML>\n");
48 }
49
50 return 0;
51 }
```

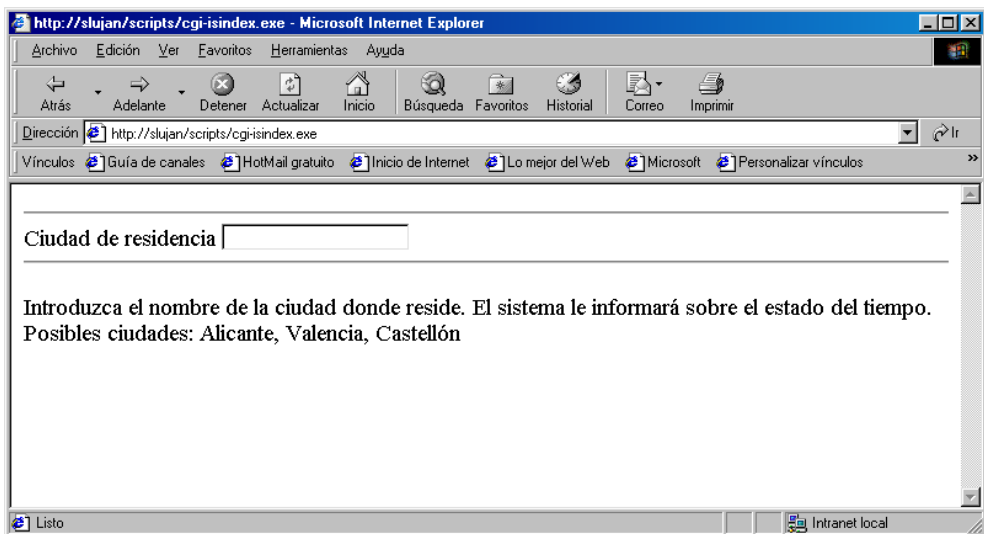


Figura 1.4: Página con cuadro de texto ISINDEX para realizar una búsqueda

Otra forma de realizar consultas ISINDEX es directamente a través de la **URL** (de forma manual). Cuando se llama a un programa **CGI** (por ejemplo, en un enlace) se pueden añadir términos de búsqueda a continuación del nombre del **CGI**: separado por el signo interrogación (?), se escriben los términos de búsqueda. Si hay más de uno, se tienen que separar por un signo más (+)¹⁵.

¹⁵Más adelante veremos que el signo más se emplea para codificar los espacios en blanco en la **URL**.

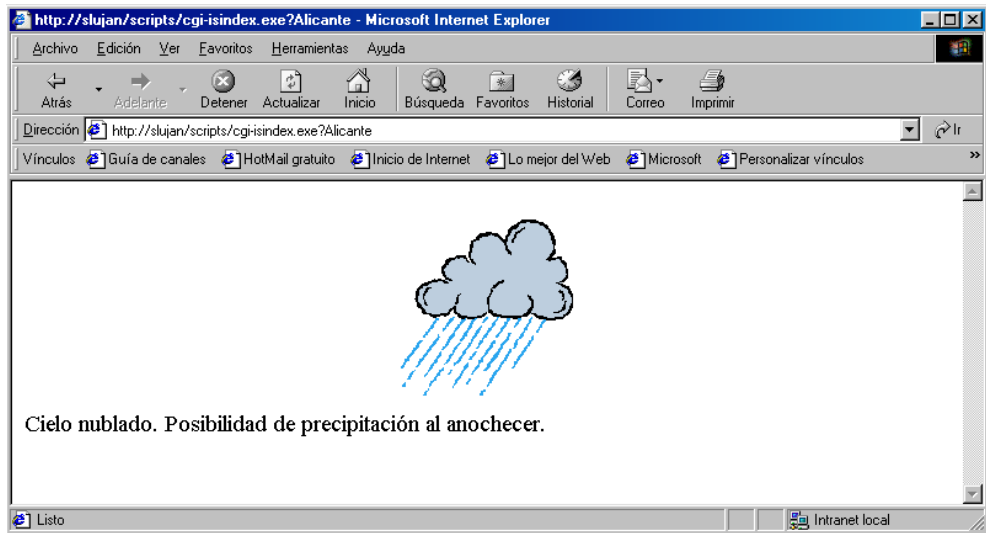


Figura 1.5: Página de respuesta a una búsqueda ISINDEX

Muy importante: si los términos de búsqueda contienen un signo igual (=), entonces no se realizará una consulta ISINDEX y la información no se pasará por la línea de comandos. Esto no ocurre si el signo igual se escribe en el cuadro de texto de una etiqueta ISINDEX, ya que el navegador se encarga de codificarlo¹⁶.

1.10.2. Cómo tratar los formularios

Los dos siguientes métodos (a través de la **URL** y a través de la entrada estándar) permiten que un programa **CGI** reciba los datos introducidos por el usuario en los controles de un formulario. Pero antes de estudiar esos dos métodos, hay que saber que el navegador codifica automáticamente la entrada del usuario cuando la envía al servidor web. Los datos introducidos en un formulario se envían al programa **CGI** con el siguiente formato:

Ejemplo 1.10

```
1 control1=valor1&control2=valor2&...&controln=valorn
```

¹⁶El código del signo igual es %3D.

donde `control1`, `control2`, ..., `controln` son los distintos nombres de los controles que forman el formulario y `valor1`, `valor2`, ..., `valorn` son los distintos valores que ha introducido o seleccionado el usuario y las distintas parejas `control=valor` se separan mediante *ampersand* (&). Por ejemplo, a partir de un formulario con tres controles se puede obtener una entrada como la siguiente:

Ejemplo 1.11

```
1 nombre=Jose&universidad=UA&carrera=Derecho
```

Si el usuario no ha especificado un determinado valor en algún control, aparecerá de todas formas la correspondiente cadena `control=`, sin ningún valor asociado.

Otro aspecto importante es que en los valores introducidos por el usuario, los espacios en blanco se sustituyen por el signo + y si aparecen caracteres especiales, como por ejemplo “&”, “%”, “\$” o “ñ”, se codifican usando el símbolo “%” seguido de dos dígitos que expresan, en hexadecimal, su código **ASCII**¹⁷. Esta codificación se conoce como “codificación URL” (*URL encoding* o *escaping*). Por ejemplo, la cadena “&%\$ñ” se codificaría como “%26%25%24%F1”. Se emplea esta codificación de los datos de entrada para evitar una interpretación accidental de caracteres especiales por parte del sistema operativo, lo que podría originar un agujero de seguridad.

Por tanto, el programa **CGI** debe realizar la descodificación de la entrada “antes de poder hacer nada”:

1. Tiene que separar las distintas parejas `control=valor`. Para ello, hay que dividir los datos recibidos cada vez que se encuentre un *ampersand* (&). No hay peligro de confundirse con la entrada del usuario, ya que si un usuario introduce un *ampersand*, se envía codificado (%26).
2. Una vez que se tienen las distintas parejas, se separan en nombre de control y valor de control usando para ello el signo igual (=). No hay peligro de confundirse con la entrada del usuario, ya que si un usuario introduce un signo igual, se envía codificado (%3D).

¹⁷Se codifican los caracteres con un código **ASCII** menor de 33 (21 hexadecimal) o mayor que 127 (7F hexadecimal). El espacio en blanco podría codificarse como %20, pero como el espacio en blanco es tan común, se ahorra espacio y es “más elegante emplear” el signo más (+).

- Los distintos valores se descodifican. Se substituyen los signos más por espacios en blanco y se buscan cadenas de la forma `%##`, donde `##` son códigos hexadecimales. No hay peligro de confundirse con la entrada del usuario, ya que si un usuario introduce un signo porcentaje, se envía codificado (`%25`).

En el Cuadro 1.5 se han resumido los caracteres especiales que se emplean en la codificación **URL**.

Nombre	Carácter	Propósito
Ampersand	&	Separa pares <code>control=valor</code>
Equal	=	Separa el nombre del control del valor del control
Percent	%	Marca el inicio de un carácter codificado
Plus	+	Substituye espacios en blanco

Cuadro 1.5: Caracteres especiales en la codificación URL

1.10.3. A través de la URL

Este método se emplea cuando se usa un formulario¹⁸ con el método de envío **GET** o directamente a través de la **URL**. El programa **CGI** recibe la información codificada a través de la **QUERY_STRING**. El navegador se encarga de codificar la información que introduce el usuario en el formulario. Por tanto, si usamos el método directo (directamente escrito en una **URL**), tenemos que codificar manualmente los datos.

Cuando se usa este método directamente a través de la **URL**, los datos que se quieren enviar se añaden al final de la **URL**, separados del nombre del programa **CGI** mediante un signo de interrogación (?). Por ejemplo, si queremos que al pulsar sobre un enlace se llame al programa `cgi.exe` y se le pase la palabra `subtotal`, pondremos:

Ejemplo 1.12

```
1 <A HREF="cgi.exe?subtotal">Ver subtotal</A>
```

¹⁸En la mayoría de los navegadores, el método de envío por defecto es **GET**: si en un formulario no se indica el método con el atributo **METHOD**, se asume el método **GET**.

También existe el método **HEAD**, similar al método **GET**, excepto que con el método **HEAD** sólo las cabeceras **HTTP** (y no el cuerpo del mensaje) se envían desde el servidor web hacia el navegador.

1.10.4. A través de la entrada estándar

Este método se emplea cuando se usa un formulario con el método de envío **POST**. El programa **CGI** recibe la información codificada a través de la entrada estándar (**stdin**) (el navegador se encarga de codificar la información que introduce el usuario en el formulario).

El servidor web no tiene la obligación de enviar una marca de final de fichero (**EOF**) al final de los datos. Para saber cuántos datos hay que leer de la entrada, se tiene que consultar la variable de entorno **CONTENT_LENGTH**, que proporciona el número de bytes que se pueden leer. El servidor web también informa sobre el tipo de datos que va a recibir el programa **CGI** mediante la variable de entorno **CONTENT_TYPE**. La codificación estándar para los datos de un formulario es **application/x-www-form-urlencoded**.

Cuando se emplea este método, la variable de entorno **QUERY_STRING** está vacía, a no ser que después del nombre del programa **CGI** aparezca un signo de interrogación (?) y algo más. Por ejemplo, en el siguiente formulario, se envía la entrada del usuario mediante **POST**, pero también se pasa información a través de la **URL**:

Ejemplo 1.13

```
1 <FORM ACTION="cgi.exe?id=es" METHOD="POST">
2 Nombre: <INPUT TYPE="TEXT" NAME="nombre">
3 </FORM>
```

La ventaja principal del método **POST** sobre el método **GET** es que el primero no tiene ninguna limitación sobre el número de bytes que se pueden enviar, mientras que el segundo, como los datos se envían en la **URL** y la información se almacena en la variable de entorno **QUERY_STRING**, puede verse limitado por el tamaño máximo que pueda tener una **URL** (1024 bytes normalmente) o por el tamaño máximo de una variable de entorno en el sistema operativo.

Un programa **CGI** puede saber si se le han enviado los datos mediante **GET** o **POST** consultando la variable de entorno **REQUEST_METHOD**.

1.10.5. A través de información de ruta

También existe otra forma de enviar datos al programa **CGI** desde el cliente a través de la **URL**, incluyendo información extra en la vía de acceso al programa **CGI**. Esta información adicional no se codifica de ninguna manera. En este caso, el programa **CGI** recibe la información extra en la variable de entorno **PATH_INFO**.

Esta forma de enviar información se emplea normalmente para transmitir la localización de ficheros al programa **CGI**, aunque se puede emplear para otros usos. Por ejemplo, imaginemos que tenemos un **CGI** llamado `cgi-orden.exe` que es capaz de ordenar las líneas de un fichero y mostrar el resultado en una página **HTML**. Si queremos que procese el fichero `lista.txt` que se encuentra en el directorio `ficheros` que pertenece al directorio principal del sitio web, se tiene que realizar la llamada al programa **CGI** de esta forma:

Ejemplo 1.14

```
1 cgi-orden.exe/ficheros/lista.txt.
```

1.11. Variables de entorno CGI

Además de las variables de entorno que hemos visto (**CONTENT_LENGTH**, **CONTENT_TYPE**, **PATH_INFO**, **QUERY_STRING** y **REQUEST_METHOD**), el servidor web asigna valor a otras variables cuando ejecuta el programa **CGI**. A continuación se muestran las variables más importantes agrupadas en tres grupos: específicas del servidor, específicas del cliente y específicas de la petición. Algunas variables puede ser que no estén disponibles en algunos servidores web. Por otro lado, además de a todas estas variables de entorno específicas de **CGI**, también podemos acceder a las pertenecientes al sistema operativo, como **PATH**, **TEMP**, etc.

Cada programa **CGI** recibe sus propias variables de entorno con sus propios valores. Se pueden ejecutar concurrentemente varios programas sin problemas, ya que cada uno recibirá su propia copia de las variables de entorno.

1.11.1. Específicas del servidor

Estas variables comunican al programa **CGI** características sobre el servidor web en que se está ejecutando. Normalmente, se sabe en que servidor se

está ejecutando un programa **CGI**, así que estas variables se suelen usar poco.

- **GATEWAY_INTERFACE**. El nombre y la versión de la especificación **CGI** utilizada por el servidor. El formato es `CGI/versión`. Ejemplo: `CGI/1.1`.
- **SERVER_NAME**. El nombre del servidor, el alias *Domain Name System* (**DNS**) o la dirección **IP** tal como aparecería en las direcciones **URL** que hacen referencia a sí mismas. Ejemplo: `www.ua.es`.
- **SERVER_PORT**. El número de puerto en el que el servidor ha recibido la petición **HTTP**. Ejemplo: `80`¹⁹.
- **SERVER_PROTOCOL**. El nombre y la versión del protocolo empleado por el servidor para procesar las peticiones. El formato es `protocolo/versión`. Ejemplo: `HTTP/1.1`.
- **SERVER_SOFTWARE**. El nombre y la versión del software del servidor que responde a la petición y que ejecuta el **CGI**. El formato es `nombre/versión`. Ejemplo: `Microsoft-IIS/4.0`.

1.11.2. Especificas del cliente

Mediante estas variables, el servidor web informa al programa **CGI** sobre el cliente web (navegador). El servidor web obtiene la información a partir de las cabeceras que envía un cliente web en una petición (por ello, todas las variables comienzan por **HTTP**, ya que el contenido de estas variables se recibe con cada petición **HTTP**²⁰). No todos los clientes web proporcionan toda la información posible.

- **HTTP_ACCEPT**. Enumera los tipos de respuesta que acepta el cliente. El formato es `tipo/subtipo, tipo/subtipo, ...`. Ejemplo: `image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, /*/*`.
- **HTTP_ACCEPT_ENCODING**. Identifica los tipos de esquemas de codificación que acepta el cliente. Ejemplo: `gzip, deflate`.

¹⁹El puerto 80 es el puerto por defecto para comunicaciones **HTTP**, pero puede ser cambiado.

²⁰En la cabecera **HTTP**, el signo de subrayado `_` de los nombres de las variables de entorno específicas del cliente aparece realmente como un guión `-`. Además, todos los caracteres se han pasado a mayúsculas en el nombre de la variable de entorno.

- `HTTP_ACCEPT_LANGUAGE`. Enumera los códigos *International Organization for Standards (ISO)* de los lenguajes que el cliente entiende y espera recibir. Ejemplo: `es-ES, en, pdf`.
- `HTTP_REFERER`. Identifica la URL del documento que contiene el enlace que apunta al documento actual. Ejemplo: `http://www.ua.es/-index.html`.
- `HTTP_USER_AGENT`. Identifica el software del cliente web. Ejemplo: para Netscape Communicator 4.7 se obtiene `Mozilla/4.7 [en] (Win98; I)` y para Microsoft Internet Explorer 5.5 la cadena `Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)`.

1.11.3. Específicas de la petición

- `AUTH_TYPE`. El método de autenticación que el servidor utiliza para validar a los usuarios cuando intentan tener acceso a un programa **CGI** protegido. Normalmente la autenticación se realiza mediante un nombre de usuario y una contraseña. Ejemplo: `BASIC`.
- `AUTH_USER`. Nombre de usuario autenticado.
- `CONTENT_LENGTH`. Número de bytes enviados a la entrada estándar (`stdin`) de un programa **CGI** debido a una petición `POST`. Esta variable está vacía cuando el método empleado es `GET`.
- `CONTENT_TYPE`. El tipo **MIME** de los datos enviados por el cliente web mediante el método `POST`. Esta variable está vacía cuando el método empleado es `GET`. Ejemplo: `application/x-www-form-urlencoded`.
- `PATH_INFO`. Información adicional de ruta para el programa **CGI** pasada como parte de la **URL**, a continuación del nombre del programa. Ejemplo: `/myhome`.
- `PATH_TRANSLATED`. La versión traducida de `PATH_INFO`. La ruta virtual se convierte en ruta física. Ejemplo: `D:\Inetpub\wwwroot\myhome`.
- `QUERY_STRING`. Información de consulta almacenada en la cadena que sigue al signo de interrogación (?) en la **URL**.

- **REMOTE_ADDR**. La dirección **IP** del cliente web que hace la petición. Ejemplo: 156.78.65.9.
- **REMOTE_HOST**. El nombre de host del cliente que realiza la petición. Si el servidor no posee esta información, debe fijar el valor de **REMOTE_ADDR** y dejar esta variable en blanco.
- **REMOTE_USER**. Nombre del usuario remoto, si el usuario se ha autenticado correctamente.
- **REQUEST_METHOD**. El método que se utiliza para hacer la petición. Los más usuales son **HEAD**, **GET** y **POST**.
- **SCRIPT_NAME**. La ruta virtual al programa **CGI** que se está ejecutando. Esta variable es útil en los programas **CGI** que se llaman a sí mismos²¹. Ejemplo: /scripts/cgivar.exe.

En la Figura 1.6 se muestra el valor de algunas de las variables de entorno **CGI** en un servidor Microsoft Personal Web Server 4.0 ejecutándose en Microsoft Windows 98 y cuando recibe una petición de un cliente Microsoft Internet Explorer 5.5.

1.11.4. Cómo acceder a las variables desde C

Para acceder a las variables de entorno desde *C* se puede emplear la función `getenv()` que se encuentra en la librería `stdlib.h`. El prototipo de la función es:

Ejemplo 1.15

```
1 char *getenv(const char *name);
```

Por ejemplo, el siguiente código muestra el valor de las variables específicas del servidor mostradas en la Figura 1.6:

²¹Por ejemplo, en los programas **CGI** que generan un formulario y también lo procesan cuando se envía. Mediante la variable **REQUEST_METHOD** se puede distinguir el primer caso (**GET**) del segundo (**POST**).

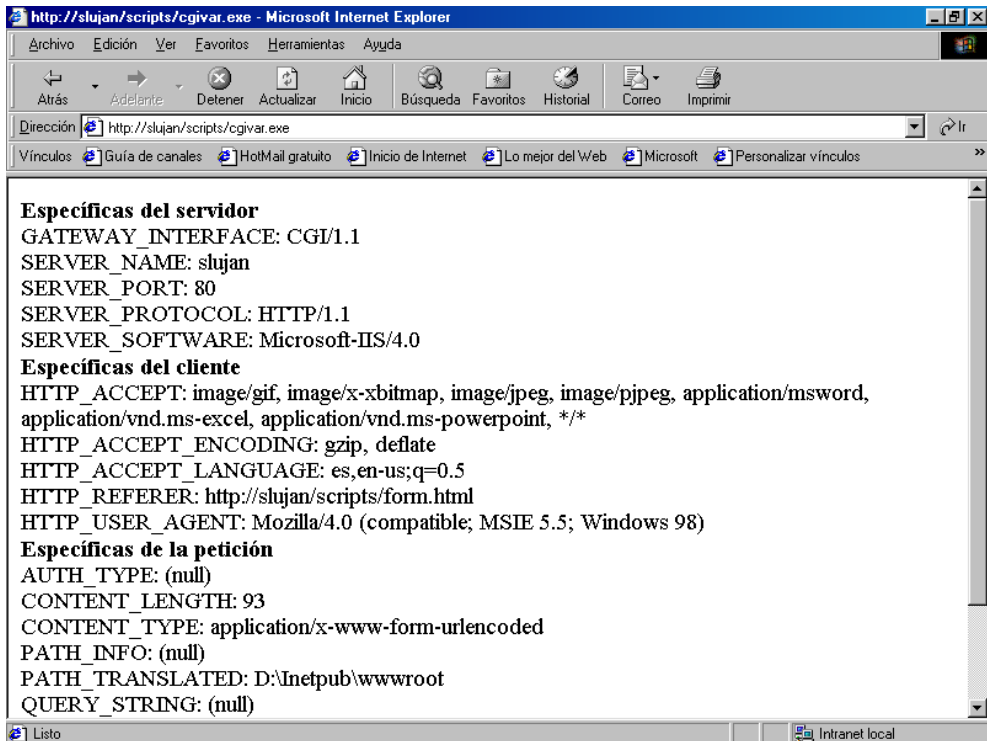


Figura 1.6: Ejemplo de variables de entorno

Ejemplo 1.16

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {
6     char *variable;
7
8     fprintf(stdout, "Content-type: text/html\n\n");
9     fprintf(stdout, "<HTML>\n<BODY>\n");
10
11     /* SERVIDOR */
12     printf("<B>Específicas del servidor</B><BR>");
13
14     variable = getenv("GATEWAY_INTERFACE");
15     fprintf(stdout, "GATEWAY_INTERFACE: %s<BR>", variable);
16
17     variable = getenv("SERVER_NAME");
18     fprintf(stdout, "SERVER_NAME: %s<BR>", variable);
19
20     variable = getenv("SERVER_PORT");
21     fprintf(stdout, "SERVER_PORT: %s<BR>", variable);
22
23     variable = getenv("SERVER_PROTOCOL");
24     fprintf(stdout, "SERVER_PROTOCOL: %s<BR>", variable);
25
26     variable = getenv("SERVER_SOFTWARE");
27     fprintf(stdout, "SERVER_SOFTWARE: %s<BR>", variable);
28
29     fprintf(stdout, "</BODY>\n</HTML>\n");
30     return 0;
31 }
```

1.12. Un ejemplo más complejo

El siguiente ejemplo muestra un programa **CGI** más complejo. Se compone de dos ficheros: `cgi-select.c` que contiene el código del programa y `cgi.data` que contiene la información que emplea el programa para construir la página.

El programa genera dos páginas web. En la primera (Figura 1.7), se mues-

tra una lista desplegable que contiene una serie de valores leídos del fichero `cgi.data`. Una vez que se elige un valor de la lista, se vuelve a ejecutar el programa **CGI**, pero en la segunda página (Figura 1.8) se muestran dos listas: la que se mostraba antes y otra cuyos valores dependen del valor elegido en la primera lista.

cgi-select.c

Este fichero contiene el código en *C* del programa **CGI**.

Ejemplo 1.17

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 /* Construye las opciones de la primera lista */
6 void
7 optionsA(FILE *f)
8 {
9     char centinela = 1;
10    char linea[100];
11
12    while(centinela)
13    {
14        /* Fin si se llega al final del fichero */
15        if(fgets(linea, 100, f) == NULL)
16            centinela = 0;
17        else
18            {
19                if(linea[0] == '*' && linea[strlen(linea) - 2] == '*')
20                    {
21                        /* Elimina el salto de linea */
22                        linea[strlen(linea) - 1] = '\0';
23                        printf("<OPTION VALUE=\"%s\">", linea);
24                        /* Elimina el ultimo asterisco */
25                        linea[strlen(linea) - 1] = '\0';
26                        /* +1: elimina el primer asterisco */
27                        printf("%s</OPTION>\n", linea + 1);
28                    }
29            }
30    }

```

```
31 }
32
33 /* Construye las opciones de la segunda lista */
34 void
35 optionsB(FILE *f, char *s)
36 {
37     char centinela = 1;
38     char linea[100];
39
40     while(centinela)
41     {
42         /* Fin si se llega al final del fichero */
43         if(fgets(linea, 100, f) == NULL)
44             centinela = 0;
45         else
46         {
47             if(!strncmp(s, linea, strlen(s)))
48             {
49                 /* Lee todos los productos mientras que no
50                  se llegue al final del fichero a una linea
51                  con un salto de linea (fin de producto) */
52                 while(centinela)
53                 {
54                     /* Fin si se llega al final del fichero */
55                     if(fgets(linea, 100, f) == NULL)
56                         centinela = 0;
57                     else if(linea[0] == '\n')
58                         centinela = 0;
59                     else
60                     {
61                         /* Elimina el salto de linea */
62                         linea[strlen(linea) - 1] = '\0';
63                         printf("<OPTION>%s</OPTION>\n", linea);
64                     }
65                 }
66             }
67         }
68     }
69 }
70
71 int
72 main(void)
```



```
73 {
74     char *var;
75     char entrada[100];
76     int i, lon;
77     FILE *f;
78
79     f = fopen("cgi.data", "r");
80     if(f == NULL)
81     {
82         printf("Content-type: text/html\n\n");
83         printf("<HTML>\n");
84         printf("<HEAD>\n");
85         printf("<TITLE>CGI con listas desplegables - Error</TITLE>\n");
86         printf("</HEAD>\n");
87         printf("<BODY>\n");
88         printf("Error: no encuentro el fichero cgi.data\n");
89         printf("</BODY>\n</HTML>");
90
91         return 0;
92     }
93
94     var = getenv("REQUEST_METHOD");
95     if(!strcmp(var, "GET"))
96     {
97         printf("Content-type: text/html\n\n");
98         printf("<HTML>\n");
99         printf("<HEAD>\n");
100        printf("<TITLE>CGI con listas desplegables - Página 1</TITLE>\n");
101        printf("</HEAD>\n");
102        printf("<BODY>\n");
103        /* El formulario llama al propio CGI */
104        var = getenv("SCRIPT_NAME");
105        printf("<FORM ACTION=\"%s\" METHOD=\"POST\">\n", var);
106        printf("Seleccione sistema operativo:<BR>\n");
107        printf("<SELECT NAME=\"sistema\" ONCHANGE=\"submit();\">\n");
108        printf("<OPTION SELECTED></OPTION>\n");
109        optionsA(f);
110        printf("</SELECT>\n");
111        printf("</FORM>\n");
112        printf("</BODY>\n</HTML>");
113    }
114    else
```

```
115  {
116      /* Lee los datos recibidos por la entrada estándar */
117      var = getenv("CONTENT_LENGTH");
118      lon = atoi(var);
119      for(i = 0; i < 100 & i < lon; i++)
120          entrada[i] = fgetc(stdin);
121      entrada[i] = '\0';
122      printf("Content-type: text/html\n\n");
123      printf("<HTML>\n");
124      printf("<HEAD>\n");
125      printf("<TITLE>CGI con listas desplegables - Página 2</TITLE>\n");
126      printf("</HEAD>\n");
127      printf("<BODY>\n");
128      /* El formulario llama al propio CGI */
129      var = getenv("SCRIPT_NAME");
130      printf("<FORM ACTION=\"%s\" METHOD=\"POST\">\n", var);
131      printf("Seleccione sistema operativo:<BR>\n");
132      printf("<SELECT NAME=\"sistema\" ONCHANGE=\"submit();\">\n");
133      printf("<OPTION SELECTED></OPTION>\n");
134      optionsA(f);
135      printf("</SELECT>\n");
136      printf("<BR><BR>\n");
137      /* +8: elimina sistema= */
138      printf("Seleccione producto para %s:<BR>\n", entrada + 8);
139      printf("<SELECT>\n");
140      /* El puntero se posiciona en el principio del fichero */
141      fseek(f, 0, SEEK_SET);
142      /* +8: elimina sistema= */
143      optionsB(f, entrada + 8);
144      printf("</SELECT>\n");
145      printf("</FORM>\n");
146      printf("</BODY>\n</HTML>");
147  }
148
149  fclose(f);
150  return 0;
151 }
```

cgi.data

Este fichero contiene la información que se quiere mostrar en las listas. Cada valor tiene que escribirse en una línea independiente; los valores principales que se quieran mostrar en la primera lista tienen que aparecer encerrados entre asteriscos (*), a continuación se escriben los valores (segunda lista) correspondientes al valor principal. Muy importante: los valores principales no pueden contener espacios en blanco ni caracteres especiales. A continuación se muestra el fichero empleado en la Figura 1.7 y 1.8.

Ejemplo 1.18

```
1 *Windows95*
2 Actualización USB
3 Parche efecto 2000
4
5 *Windows98*
6 Parche problemas de apagado
7 Parche agujero de seguridad
8
9 *Windows2000*
10 Actualización Office 2000
11 Parche seguridad IIS
12 Internet Explorer 6.0
```

1.13. Seguridad

El estándar **CGI** no es inseguro por sí mismo: simplemente define un interfaz para que un servidor web se comunique con aplicaciones externas. Pero como un **CGI** es un programa ejecutable, al usarlo en nuestra web estamos permitiendo que “extraños” ejecuten un programa en nuestro servidor, lo cual no es lo más seguro del mundo. Por tanto, existen una serie de precauciones que hay que tener en cuenta a la hora de programar un **CGI**.

1.13.1. Permisos de ejecución

Lo primero que hay que saber es que para que se ejecute un **CGI**, éste tiene que residir en un directorio especial, de forma que el servidor web sepa que

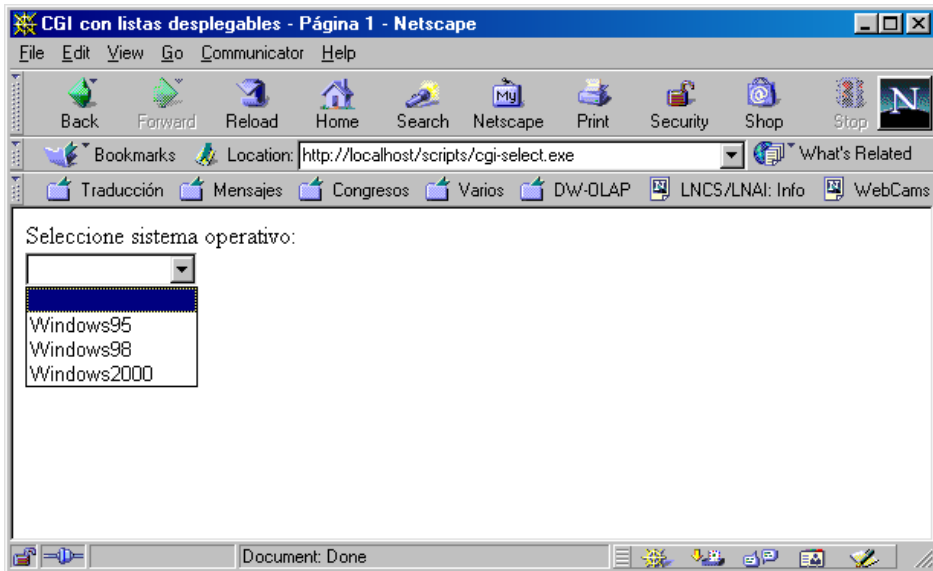


Figura 1.7: cgi-select: página 1

tiene que ejecutar el programa en vez de mostrarlo²². Además, de este modo, un usuario particular no puede colocar un programa **CGI** en su directorio particular sin que el administrador del sistema lo sepa y lo permita.

Normalmente, en los servidores web que se ejecutan en sistemas Unix, el directorio se llama `/cgi-bin` o `/cgibin`. En los dos servidores web de MICROSOFT, Microsoft Personal Web Server y Microsoft Internet Information Server, el directorio suele llamarse `/Scripts`. En la Figura 1.9 podemos ver los permisos que posee por defecto el directorio `D:\Inetpub\scripts` en el servidor Microsoft Personal Web Server 4.0. Como se puede apreciar, están activos los permisos **Ejecución** y **Archivos de comandos**. Para que se ejecute un **CGI**, sólo hace falta tener activado el permiso **Ejecución**.

²²Si no fuera así, los usuarios podrían acceder y descargarse el **CGI**, lo que plantea un grave problema de seguridad.

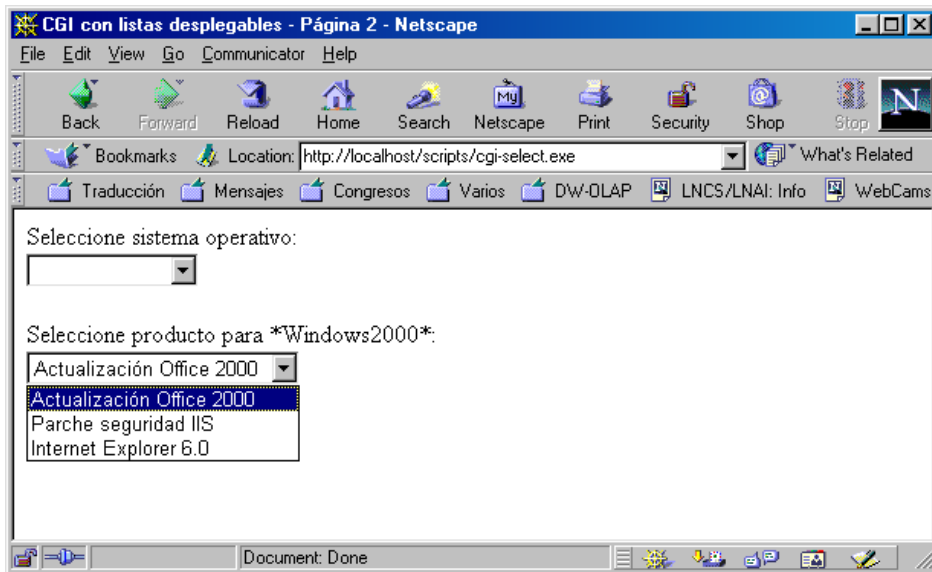


Figura 1.8: cgi-select: página 2

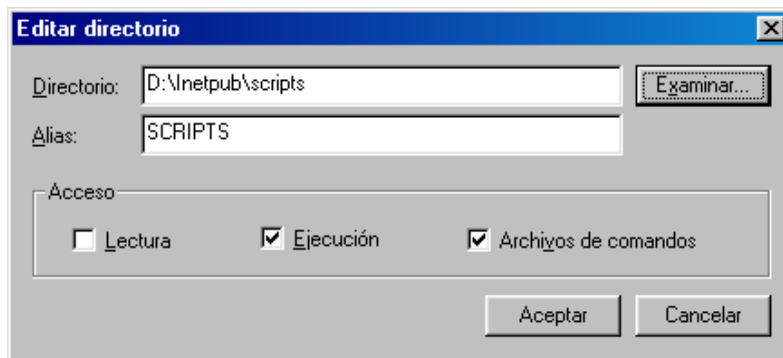


Figura 1.9: Permisos de ejecución en Microsoft Personal Web Server

1.13.2. Examina el código

Si se usa un **CGI** programado por otra persona, es conveniente revisar el código para comprobar qué hace y cómo lo hace realmente. No hay que fiarse de los programas **CGI** ya compilados: pueden esconder un “caballo de Troya”²³ o una puerta trasera (*backdoor*) de entrada a nuestro sistema. Por ejemplo, un programa **CGI** puede realizar una función dada “benigna”, pero además, sin que nadie se entere, puede enviar a su creador el fichero `/etc/passwd` cada vez que se ejecute.

Este problema también se puede dar cuando usamos librerías de código de origen desconocido. Aunque sea una librería conocida, hay que descargarla de un sitio de confianza, para evitar que haya sido manipulada previamente.

1.13.3. Versiones estables

Siempre que sea posible, hay que emplear las últimas versiones estables de los programas empleados. Nada de versiones “beta”, ya que suelen tener muchos problemas de vulnerabilidad, que además son publicados en Internet y conoce todo el mundo rápidamente.

1.13.4. Las presunciones son peligrosas

A la hora de tratar los datos de entrada del usuario, las presunciones son muy peligrosas:

- Suponer que los datos que se reciben provienen de nuestro formulario es un error. Cualquiera puede apuntar un formulario cualquiera a nuestro programa **CGI** o generar una petición **HTTP**²⁴ que parezca el resultado de un formulario, pero que contenga datos peligrosos.
- Es peligroso asumir que los datos que recibe el **CGI** se pueden almacenar correctamente. Cualquier limitación que se imponga en un formulario²⁵, se puede saltar fácilmente con un formulario distinto o con una

²³Un caballo de Troya es una aplicación “maligna” que se camufla como un programa que realiza una función “benigna”, pero que realmente realiza una serie de tareas ocultas sin que el usuario se de cuenta. Al contrario que los virus, los caballos de Troya no se replican ni infectan otros ficheros, pero pueden ser tan destructivos como ellos.

²⁴Se puede crear un programa que haga cosas que no puede hacer un navegador, como enviar cientos de megabytes a un **CGI**.

²⁵Por ejemplo, `<INPUT TYPE="TEXT" MAXLENGTH="10">`.

petición **HTTP** directa. El exceso de datos produce desbordamientos de buffers (*buffer overrun*), que pueden bloquear el sistema o permitir el acceso en modo superusuario (administrador) al sistema. Si se reciben los datos mediante **POST**, es conveniente verificar su longitud a través de **CONTENT_LENGTH**.

- También es un error suponer que los caracteres especiales en los datos han sido codificados por el navegador mediante las secuencias **%xx**.

En definitiva, un programa **CGI** tiene que estar preparado para esperar datos de entrada que contienen basura, están vacíos, son aleatorios o superan el tamaño máximo esperado. Evidentemente, tanta prevención tiene un inconveniente: el código del programa aumenta considerablemente y el mantenimiento futuro es más complicado.

1.13.5. Programa defensivamente

A la hora de tratar los datos recibidos, hay que elegir un criterio. Por ejemplo, si un campo de un formulario tiene que contener una dirección de correo electrónico, el programa **CGI** tiene que rechazar aquellos datos que no se ajusten a una dirección de correo electrónico.

Si un cuadro de texto tiene un límite de longitud (**MAXLENGTH**), los datos recibidos pueden superar dicho valor. El programa **CGI** debe verificar la longitud de los datos.

Si en un campo se espera una única línea de texto, el programa **CGI** tiene que rechazar aquellos datos que contengan un salto de línea (más de una línea).

Si un formulario incluye listas desplegables, cuadros de verificación o botones de radio, el programa **CGI** tiene que rechazar cualquier dato que no coincida con los presentados al usuario en el formulario.

1.13.6. Limpia los datos antes de usarlos

Los datos que introduce un usuario hay que limpiarlos o validarlos antes de emplearlos. Por ejemplo, si en un cuadro de texto se espera el nombre de un fichero, antes de realizar cualquier operación que suponga trabajar con un fichero hay que verificar que se trata de un nombre de fichero válido. Por ejemplo, verificar que²⁶:

²⁶Las verificaciones dependen del sistema de archivos de cada sistema operativo.

- No comienza por un punto.
- No contiene separadores de ruta (/ o \).
- No contiene dos puntos (:), subrayado (_) o cualquier otro carácter especial.
- Tiene una longitud máxima.

Para verificar que unos datos son válidos, se pueden tomar dos soluciones: verificar que contiene caracteres válidos o verificar que no contiene caracteres no válidos. Es preferible emplear la primera alternativa, ya que si se emplea la segunda es probable que se olvide comprobar algún carácter no válido.

El siguiente código en *C* permite validar el contenido de una cadena. Tiene como argumentos dos punteros a cadenas: la primera cadena contiene los datos de entrada que se quieren validar y la segunda los caracteres aceptados. La función devuelve un puntero a la primera cadena con los caracteres no aceptados eliminados.

Ejemplo 1.19

```
1 char *stripchars(char *cadena, const char *acepta)
2 {
3     char flags[256], *chr, *pos;
4     int n;
5
6     /* Tabla de flags que indica si un caracter es aceptado */
7     for(n = 0; n < 256; n++)
8         flags[n] = 0;
9     for(chr = acepta; *chr != '\0'; chr++)
10        flags[*chr] = 1;
11    /* Sobre la propia cadena, copia unicamente los caracteres
12       validos */
13    for(chr = cadena, pos = cadena; *chr != '\0'; chr++)
14        {
15        *pos = *chr;
16        pos += flags[*chr];
17        }
18    *pos = '\0';
19
20    return cadena;
21 }
```

1.13.7. Limpia los datos antes de pasarlos a otro programa

Es conveniente evitar el tener que pasar datos a otros programas. Si no hay más remedio porque el programa **CGI** simplemente actúa como pasarela y pasa los datos a otro programa, es conveniente que el **CGI** verifique que los datos no contienen ningún carácter especial que pueda producir un error: no se sabe como va a responder el programa externo ante datos inadecuados.

Cuando se empleen programas externos hay que indicar de forma explícita la ruta al programa, y no confiar en que se encuentran en el **PATH**. Si no se lleva cuidado, puede ser que se ejecute el programa que no es (incluso un programa que haya colocado una persona maliciosa en nuestro servidor).

1.13.8. Cuidado con HTML

Otra posible fuente de problemas es recibir código **HTML** cuando se espera texto plano. Suponed que tenemos un libro de firmas (*guestbook*); si el usuario introduce en alguno de los campos (por ejemplo, el nombre) códigos **HTML**, cuando se visualice su entrada en el libro de firmas, no se mostrará como el programador espera, si no que se aplicarán los formatos que el usuario haya introducido. De este modo tan sencillo se pueden insertar enlaces o imágenes de cualquier naturaleza en una web de otra persona.

Mucho peor es que inserte algún comando que permita realizar alguna operación. Por ejemplo, si el servidor web sabe procesar *Server Side Include (SSI)*, un usuario puede incluir una instrucción como `<!-- #include file="/etc/passwd" -->` para visualizar el fichero de contraseñas o `<!-- #exec cmd="rm -rf /" -->` para borrar todo el sistema de archivos.

Existen dos soluciones para evitar este problema:

1. Impedir que el usuario pueda introducir los caracteres `<` y `>`. Si el usuario los introduce, se le muestra un mensaje de error para que vuelva a introducir su entrada o automáticamente se eliminan.
2. Traducir los dos caracteres a sus respectivos códigos de escape: `<` para `<` y `>` para `>`.

1.13.9. Nivel de privilegio

Si el sistema operativo lo permite, es recomendable ejecutar los programas **CGI** como un usuario no privilegiado, preferiblemente como un usuario espe-

cífico al que se le pueda asignar privilegios concretos (normalmente, para ello el servidor web se tiene que ejecutar con ese usuario).

1.13.10. Nivel de prioridad

Hay que evitar que un **CGI** nunca termine: varios programas **CGI** en un bucle infinito pueden colapsar y bloquear un servidor web. Para evitar estas situaciones, es conveniente asignar a los programas **CGI** una prioridad menor que el resto de procesos: de este modo, aunque nunca termine de ejecutarse un programa, no bloqueará el sistema.

1.13.11. Usa un ordenador para los CGI

La mejor solución para evitar la mayoría de los problemas que se han comentado es elegir un ordenador para que sea servidor de **CGI**. Este ordenador no contendrá información importante y además no poseerá permisos de acceso a los otros ordenadores de la red. De este modo, aunque se produzca un agujero de seguridad, el atacante no podrá obtener mucha información. Además, es preferible que este ordenador se encuentre fuera del cortafuegos (*firewall*).

1.13.12. Consulta listas de correo y grupos de noticias

En los distintos medios de comunicación de Internet se puede encontrar información sobre agujeros de seguridad, nuevas versiones de software, etc. Toda esa información nos puede ayudar a tener un sitio web seguro.

1.13.13. Nunca olvides el código fuente

Nunca hay que dejar el código fuente de un programa **CGI** en el mismo directorio donde reside el ejecutable. Si se dispone del código fuente es más fácil localizar posibles agujeros de seguridad. Desgraciadamente, en aquellos casos en los que se emplea algún lenguaje interpretado (*Perl*, *shell* de Unix) no es posible evitar este problema.

Además, si sabes que tu programa **CGI** posee alguna vulnerabilidad, no lo indiques con un comentario en el código fuente. Si cae en manos ajenas, el atacante sólo necesita seguir las instrucciones que has dejado.

1.14. WinCGI

En 1994, Bob Denny creó el primer servidor web específico para Microsoft Windows 3.1. En aquella época, el lenguaje de programación más empleado en ese sistema operativo era Microsoft Visual Basic. Para facilitar la programación de **CGI** mediante Microsoft Visual Basic²⁷, Bob Denny creó un interfaz de programación similar a **CGI**, al que bautizó como WinCGI²⁸.

Poco después, el servidor de Bob Denny fue comprado por O'REILLY & ASSOCIATES y se ha comercializado desde entonces con el nombre de O'Reilly WebSite Professional²⁹. Este servidor web se encuentra disponible para todos los sistemas operativos de MICROSOFT.

Mientras que en el estándar **CGI** el servidor web pasa al programa **CGI** la información a través de variables de entorno, en WinCGI la información se pasa mediante los típicos ficheros `.ini` de Microsoft Windows. El intercambio de información en ambos sentidos (del servidor al programa **CGI** y viceversa) se realiza a través de ficheros (*file spooling*), lo que supone una merma en la velocidad de procesamiento frente a **CGI**.

Cuando el servidor web ejecuta un programa WinCGI, le pasa un único parámetro que indica la localización del fichero `.ini`. La especificación WinCGI define que en un fichero `.ini` existen ocho secciones y cada sección se compone de parejas `clave = valor`:

- **[CGI]**. Contiene las variables **CGI** usuales, como `Request Protocol`, `Request Method` o `Query String`.
- **[Accept]**. Indica los tipos **MIME** que el cliente comunica que acepta en el encabezado **HTTP**.
- **[System]**. Contiene variables que son específicas del estándar WinCGI. Las más importantes son `Content File` que indica el fichero que contiene los datos de la petición enviada por el cliente y `Output File` que indica el fichero en el que el programa tiene que almacenar su salida.

²⁷ Este lenguaje presentaba en sus primeras versiones varios inconvenientes que impedían su uso a la hora de programar **CGI**, como la dificultad que planteaba a la hora de acceder a las variables de entorno.

²⁸ WinCGI no se encuentra estandarizado como **CGI**, sólo existe una especificación informal.

²⁹ Desde el 20 de agosto de 2001, O'REILLY & ASSOCIATES ha cedido los derechos de su servidor web a DEERFIELD.COM.

-
- [Extra Headers]. Encabezados adicionales que se han encontrado en la petición del cliente.
 - [Form Literal]. Si la petición se ha enviado mediante POST, el servidor descodifica los datos recibidos y los coloca en esta sección en forma de parejas `campo=valor`.
 - [Form External]. Si alguno de los valores recibidos supera los 254 caracteres o contiene caracteres de control, se almacena en un fichero temporal y en esta sección se indica la localización del fichero.
 - [Form File]. Si el formulario contiene controles para enviar ficheros³⁰, en esta sección se indica la localización de los ficheros recibidos.
 - [Form Huge]. Si los datos recibidos superan los 65 535 bytes, el servidor web no los descodifica, pero en esta sección indica la localización de cada valor en el fichero indicado por `Content File`.

³⁰`<INPUT TYPE="FILE">`.

Capítulo 2

SSI

SSI es una tecnología que permite crear páginas web dinámicas. Se basa en el uso de una serie de comandos que permiten, por ejemplo, incluir en una página web el contenido de un fichero o el resultado producido por un programa al ejecutarse. Desgraciadamente, no existe un estándar sobre SSI, por lo que cada servidor web puede definir los comandos SSI de diferente forma. Sin embargo, algunos comandos se han convertido en estándar de facto y su comportamiento es idéntico (o muy similar) en la mayoría de los servidores web.

Índice General

2.1. Introducción	48
2.2. Qué necesito para programar mediante SSI	48
2.3. Procesamiento de los archivos	49
2.4. Comentarios HTML y comandos SSI	50
2.5. Comandos SSI más comunes	51
2.5.1. config	51
2.5.2. echo	54
2.5.3. exec	58
2.5.4. flastmod	59
2.5.5. fsize	61
2.5.6. include	61

2.6. Ejemplo de programa SSI 62

2.1. Introducción

SSI es una técnica que permite que una página **HTML** sea interpretada por el servidor web antes de ser enviada al cliente. Desgraciadamente, **SSI** no se encuentra estandarizado por ningún organismo, así que cada fabricante de servidores web es libre de incluir e interpretar estas directivas como mejor le parezca. Por tanto, lo más recomendable es consultar la documentación del servidor web para averiguar qué directivas reconoce y con qué sintaxis.

Al principio, **SSI** sólo permitía incluir en una página **HTML** el contenido de otro archivo (de ahí su nombre). Pero **SSI** se ha ido ampliado con más comandos: se puede incluir la fecha y la hora actuales, la fecha de la última modificación de la página o el tamaño de un fichero cualquiera. El comando más “potente” permite ejecutar un programa externo e incluir su salida en la página **HTML**.

El uso de archivos incluidos es una forma cómoda de almacenar en un único archivo información que se utiliza en varias páginas. Por ejemplo, se pueden utilizar las directivas **SSI** para insertar un aviso de *copyright* o un pie de página en todos los documentos de un sitio web.

Suponed que tenemos 300 páginas en un sitio web, y en todas ellas aparece un pie de página con la dirección de correo del administrador del sitio web (*webmaster*) y con un enlace a una página de ayuda. Si queremos modificar el pie de página (por ejemplo, cambiar la dirección de correo del administrador), habrá que ir una por una cambiando todas las páginas. Si se emplea **SSI** para incluir el pie de página desde un fichero externo, sólo habrá que realizar las modificaciones en un único fichero, y todas las páginas estarán automáticamente actualizadas.

Pero no todo son ventajas: el uso de **SSI** puede crear un agujero de seguridad y además puede reducir el rendimiento del servidor web.

2.2. Qué necesito para programar mediante SSI

Para poder programar mediante **SSI** y probarlo hacen falta los siguientes programas:

- Un editor de textos como Bloc de notas de Microsoft Windows o joe de Linux para crear las páginas **HTML**.
- Si se va a crear un programa externo que se va a ejecutar mediante las directivas **SSI**, hace falta el correspondiente compilador (*C*, *C++*, *Pascal*, etc.) o intérprete (*Perl*, *shell* de Unix, etc.).
- Un servidor web (ya sea local o remoto) que acepte las directivas **SSI**. Por ejemplo, Microsoft Personal Web Server, Microsoft Internet Information Server o Apache.
- Por último, un navegador como Netscape Communicator o Microsoft Internet Explorer para poder comprobar las páginas **HTML** y el funcionamiento de las directivas **SSI**.

No es necesario disponer de una conexión a Internet, ya que se puede comprobar localmente el código creado.

Lo que sí que es recomendable es utilizar un buen editor de textos, que sea cómodo, configurable, soporte macros, etc. y que sea *syntax highlight*. Esta última característica significa que el editor es capaz de comprender el lenguaje en el que se programa, y colorea las palabras diferenciándolas según sean variables, palabras reservadas, comentarios, etc.

2.3. Procesamiento de los archivos

Normalmente, el procesamiento de archivos **SSI** se encuentra desactivado, ya que si no se tiene cuidado, puede suponer un agujero de seguridad. Los archivos que contienen **SSI** hay que tratarlos como si fueran programas **CGI**: al igual que estos, es conveniente que se encuentren almacenados en un único directorio, que sea el único que posea permisos de ejecución.

Además, es conveniente emplear una extensión especial para aquellas páginas **HTML** que contienen directivas **SSI**. No es conveniente activar el intérprete **SSI** para todas las páginas, ya que entonces todas las páginas se van a interpretar (incluidas las que no contienen **SSI**) y el tiempo de respuesta del servidor web aumentará. Las extensiones predeterminadas normalmente son **.stm**, **.shtm** y **.shtml**, aunque en la mayoría de los servidores web se pueden añadir otras.

El servidor web procesa las directivas **SSI** mientras procesa la página **HTML**. Cuando llega a una directiva **SSI**, inserta directamente en la página **HTML** el resultado que produce el comando, como por ejemplo el contenido de un archivo incluido. Si el archivo incluido contiene a su vez directivas **SSI**, también se inserta dicho archivo.

En Microsoft Personal Web Server, para que se interpreten los archivos **SSI** es necesario que el directorio en el que residen posea permisos de **Ejecución** o de **Archivos de comandos**, tal como vemos en la Figura 2.1. Además, tienen que estar vinculadas las extensiones de los archivos que contengan **SSI** con el intérprete `ssinc.dll`.

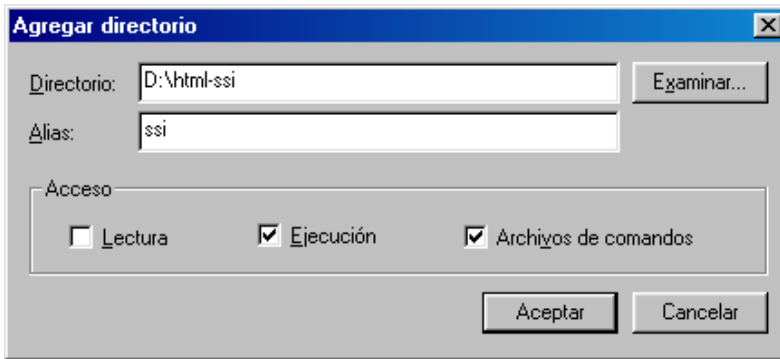


Figura 2.1: Permisos de ejecución en Microsoft Personal Web Server

2.4. Comentarios HTML y comandos SSI

Las directivas **SSI** se incluyen en una página **HTML** como comentarios de **HTML**. Esto facilita la portabilidad de las páginas: un servidor que no entienda **SSI**, enviará al cliente web (navegador) las páginas tal cual, sin interpretarlas; el cliente ignorará las directivas **SSI**, ya que se encuentran en comentarios. Por otra parte, el servidor web que sí sepa interpretar **SSI**, interpretará los comandos y sustituirá los comentarios por el resultado obtenido.

Los comentarios en **HTML** van encerrados entre los símbolos `<!--` y `-->`. Por ejemplo:

```
1 <!-- Formulario de entrada: nombre y apellidos -->
```

Cuando **SSI** está activo, el servidor web analiza los comentarios **HTML** y busca comandos **SSI**. El servidor web puede distinguir un comentario normal de un comando porque los comandos tienen que empezar con el símbolo almohadilla (**#**). La sintaxis general de los comandos **SSI** es:

Ejemplo 2.2

```
1 <!-- #comando parametro="valor" -->
```

donde **comando** es la palabra clave que indica lo que el servidor tiene que hacer, **parametro** indica el tipo de parámetro que se le pasa al comando, y **valor** es el valor asignado a dicho parámetro. Esta sintaxis es muy importante seguirla escrupulosamente, ya que muchos servidores no son flexibles: por ejemplo, si se deja un espacio en blanco entre **#** y el nombre del comando, la mayoría de los servidores no ejecutarán nada (lo tomarán como un comentario normal de **HTML**).

2.5. Comandos SSI más comunes

Como se ha comentado previamente, no existe un estándar **SSI**. Por tanto, los nombres de los comandos y su funcionalidad puede variar de un servidor web a otro. A continuación comentamos los comandos más comunes que se encuentran en la mayoría de los servidores: **config**, **echo**, **exec**, **flastmod**, **fsize** e **include**.

2.5.1. config

Esta directiva especifica el formato utilizado para presentar los mensajes de error, fechas y tamaños de archivo devueltos por otros comandos. La sintaxis de este comando es:

Ejemplo 2.3

```
1 <!-- #config parametro="valor" -->
```

donde **parametro** indica el resultado al que se le va a asignar formato. Los posibles parámetros que acepta el comando **config** son: **errmsg**, **timefmt** y **sizefmt**. Cada uno de estos parámetros posee unos valores específicos que se van a explicar a continuación.

errmsg

Controla el mensaje devuelto al navegador del cliente cuando se produce un error durante el proceso de una directiva **SSI**. De forma predeterminada, el mensaje de error proporciona información de depuración que detalla exactamente el error. Para suprimir dichos detalles, se puede indicar un mensaje de error corto y sencillo.

Por ejemplo, la siguiente página web contiene una etiqueta **SSI** con un error. En la Figura 2.2 se puede ver como se visualiza la página en un navegador y el mensaje de error que produce el servidor web.

Ejemplo 2.4

```

1 <HTML>
2 <BODY>
3 Esta página contiene una etiqueta SSI con un error
4 <BR><BR>
5 <!-- #config sizefmt="nada" -->
6 </BODY>
7 </HTML>

```

El siguiente código es la misma página que la del ejemplo anterior, pero el mensaje de error se ha personalizado. En la Figura 2.3 se visualiza la página en un navegador; se puede observar como ha cambiado el mensaje de error.

Ejemplo 2.5

```

1 <HTML>
2 <BODY>
3 Esta página contiene una etiqueta SSI con un error
4 <BR><BR>
5 <!-- #config ERRMSG="<I>Se ha producido un error :-)</I><BR>" -->
6 <!-- #config sizefmt="nada" -->
7 </BODY>
8 </HTML>

```

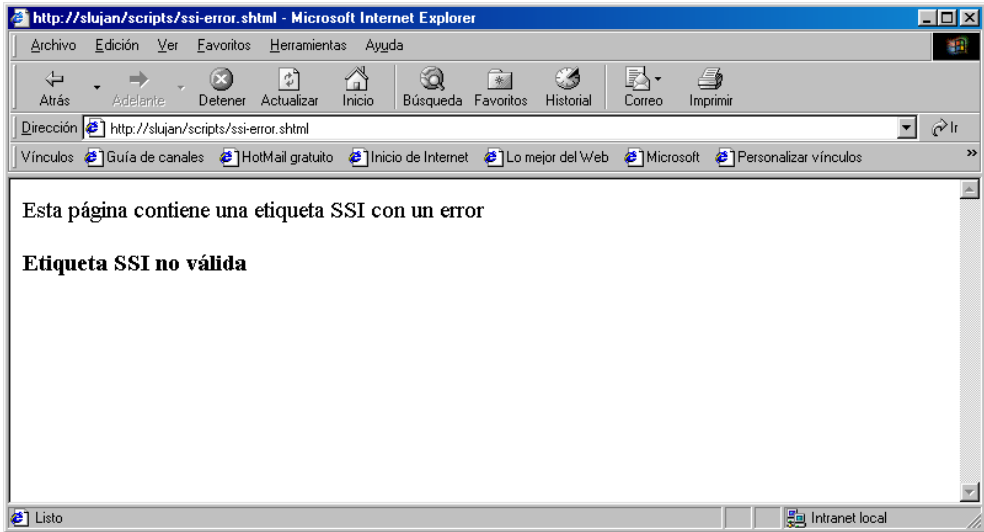


Figura 2.2: Mensaje de error por defecto

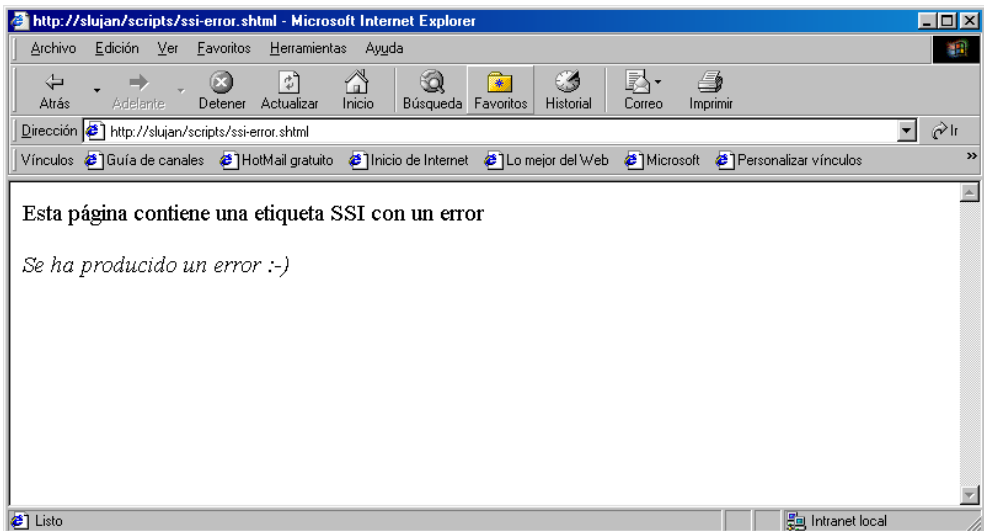


Figura 2.3: Mensaje de error personalizado

timefmt

Especifica el formato que se debe emplear para mostrar la fecha y la hora. El valor del parámetro especifica el formato. Se pueden extraer partes individuales de la fecha, como el día de la semana o el mes. El formato que se emplea es el mismo que se usa con la función `strftime()` de *C* y del sistema operativo Unix. El formato de la fecha y la hora modifica el formato de salida del comando `#echo` y `#flastmod`. Los modificadores de formato que se pueden emplear aparecen en el Cuadro 2.1.

En el Cuadro 2.2 se muestran algunos ejemplos de uso de los distintos formatos de la fecha. Además, en la Figura 2.4 se puede ver como se muestra la hora con el formato `%H:%M:%S %p`.

sizefmt

Presenta el tamaño del archivo en kilobytes o en bytes. Modifica el formato de salida del comando `#fsize`. Los modificadores de formato que se pueden emplear aparecen en el Cuadro 2.3.

2.5.2. echo

Esta directiva indica al servidor web que inserte el valor de una variable de entorno en una página **HTML**. La sintaxis de este comando es:

Ejemplo 2.6

```
1 <!-- #echo var="NombreVariable" -->
```

donde `NombreVariable` indica el nombre de una variable de entorno normal (`PATH`, `TEMP`), una variable de entorno **CGI** (`QUERY_STRING`, `REQUEST_METHOD`) o una variable de entorno **SSI** (`DATE_GMT`, `DATE_LOCAL`, `DOCUMENT_NAME`, `LAST_MODIFIED`, `QUERY_STRING_UNESCAPED`).

Por ejemplo, el resultado de la siguiente página se puede ver en la Figura 2.4 como se muestra en un navegador:

Ejemplo 2.7

```
1 <HTML>
2 <BODY>
3 <BR>DATE_LOCAL:
4 <!-- #config timefmt="%H:%M:%S %p" -->
```

Modificador	Descripción
%a	Nombre abreviado del día de la semana (por ejemplo, Lun).
%A	Nombre completo del día de la semana (por ejemplo, Lunes).
%b	Nombre abreviado del mes (por ejemplo, Feb).
%B	Nombre completo del mes (por ejemplo, Febrero).
%c	Representación de la fecha y la hora según la configuración regional (por ejemplo, 05/06/91 12:51:32).
%d	Día del mes como número decimal (01-31)
%H	Hora en formato de 24 horas (00-23).
%I	Hora en formato de 12 horas (01-12).
%j	Día del año como número decimal (001-366).
%m	Mes como número decimal (01-12).
%M	Minutos como número decimal (00-59).
%p	Indicador de A.M. o P.M. según la configuración regional actual para el formato de 12 horas (por ejemplo, p.m.).
%S	Segundos como número decimal (00-59).
%U	Semana del año como número decimal, siendo el domingo el primer día de la semana (00-51).
%w	Día de la semana como número decimal, siendo el domingo el primer día (0-6).
%W	Semana del año como número decimal, siendo el lunes el primer día de la semana (00-51).
%x	Representación de la fecha según la configuración regional actual (por ejemplo, 05/06/91).
%X	Representación de la hora según la configuración regional actual (por ejemplo, 12:51:32).
%y	Año sin siglo como número decimal (por ejemplo, 69).
%Y	Año con siglo como número decimal (por ejemplo, 1969).
%z, %Z	Nombre o abreviatura de la zona horaria; en blanco si la zona horaria es desconocida.
%%	Signo de porcentaje.

Cuadro 2.1: Modificadores de timefmt

Formato	Descripción
%d-%m-%y	Muestra el día, el mes y el año. Por ejemplo: 21-08-01
%H:%M:%S %p	Muestra las horas, los minutos y segundos con el indicador de 12 horas. Por ejemplo: 17:03:00 PM
%a %d %b %y	Muestra la fecha con el día de la semana y el nombre del mes. Por ejemplo: Tue 21 Aug 01

Cuadro 2.2: Ejemplos de distinto formato fecha

Modificador	Descripción
ABBREV	Presenta los tamaños de los archivos en kilobytes.
BYTES	Presenta los tamaños de los archivos en bytes.

Cuadro 2.3: Modificadores de sizefmt

```

5 <!-- #echo var="DATE_LOCAL" -->
6 <BR>Esta página es <!-- #echo var="DOCUMENT_NAME" -->
7 <BR>Tu dirección IP es <!-- #echo var="REMOTE_ADDR" -->
8 </BODY>
9 </HTML>

```

Otros ejemplos del comando `#echo` (en cursiva aparece el comando **SSI** y a continuación el resultado que produce):

```

La fecha y hora de hoy es <!-- #echo var="DATE_LOCAL" -->
La fecha y hora de hoy es Thursday, 23-Aug-2001 12:57:24 EDT

```

```

Has llegado aquí desde la página <!-- #echo var="HTTP_REFERER"
-->
Has llegado aquí desde la página http://www.ua.es/index.html

```

```

La hora en el meridiano de Greenwich es <!-- #echo var="DATE_GMT"
-->
La hora en el meridiano de Greenwich es Thursday, 23-Aug-2001
16:57:24 GMT

```

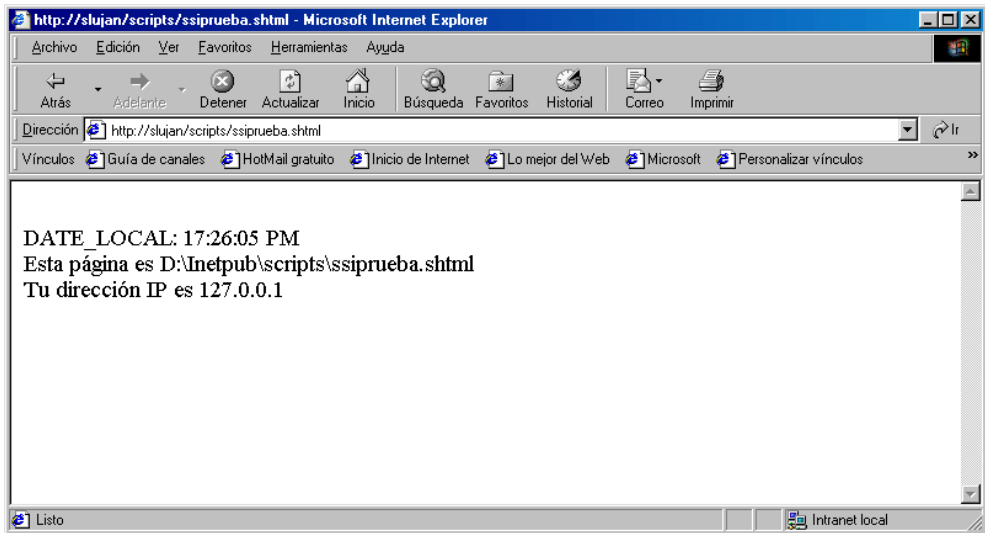


Figura 2.4: Ejemplo de comando echo

```
Tu IP es <!-- #echo var="REMOTE_ADDR" -->
Tu IP es 193.145.234.59
```

```
El nombre de esta página es <!-- #echo var="DOCUMENT_NAME" -->
El nombre de esta página es ssi-prueba.shtml
```

```
Tu navegador y sistema operativo es <!-- #echo
var="HTTP_USER_AGENT" -->
Tu navegador y sistema operativo es Mozilla/4.78 [en] (Windows NT
5.0; U)
```

También podemos crear un enlace o botón de **Volver** fácilmente con **SSI**. Como se ha visto en los ejemplos anteriores, existe la variable de entorno **CGI** **HTTP_REFERER** que indica la página anterior en la navegación. El siguiente código crea un enlace que permite volver atrás (si se sustituye el texto **Volver** por una imagen, se obtiene un botón):

Ejemplo 2.8

```
1 <A HREF="<!-- #echo var="HTTP_REFERER" -->">Volver</A>
```


2.5.3. exec

Esta directiva permite ejecutar un programa o comando del sistema operativo e incluir su resultado en una página **HTML**. Permitir que los archivos **HTML** ejecuten aplicaciones externas plantea ciertos riesgos de seguridad. Normalmente, en la mayoría de los servidores web se puede desactivar la directiva **#exec** y permitir que las páginas **HTML** sigan utilizando las demás directivas **SSI**.

La sintaxis de esta directiva es:

Ejemplo 2.9

```
1 <!-- #exec parametro="valor" -->
```

donde **parametro** indica el tipo de comando o programa que se va a ejecutar y **valor** es la ruta (absoluta o relativa) al comando o programa que se desea ejecutar. El tipo de comando suele ser **cgi**, pero también existe **cmd**, **exe** y **script**, aunque no están disponibles en todos los servidores. Por ejemplo, la siguiente instrucción ejecuta un programa y le pasa dos parámetros¹:

Ejemplo 2.10

```
1 <!-- #exec cgi="/scripts/guestbook.exe?Nombre+Apellidos" -->
```

En Microsoft Personal Web Server 4.0 se reconocen los tipos de comandos **cgi** y **cmd**:

cgi

Ejecuta una aplicación, como puede ser un programa **CGI**², un página **ASP** o una aplicación *Internet Server Application Program Interface* (**ISAPI**). El valor del parámetro es una cadena que contiene la ruta de acceso virtual de la aplicación, seguida de un signo de interrogación (?) y de los parámetros pasados a la aplicación. Los parámetros se separan por signos más (+).

¹Para pasar los parámetros, se emplea el mismo formato que con los programas **CGI**.

²En la documentación de Microsoft Personal Web Server figura que se puede ejecutar un programa **CGI**. Se puede hacer, pero las cabeceras que produzca el programa **CGI** (por ejemplo, **Content-type: text/html**) no se eliminan, por lo que se incluyen en la página **HTML**. Por tanto, realmente no es conveniente ejecutar programas **CGI** de esta forma.

cmd

Ejecuta un comando del intérprete de comandos. El valor del parámetro es una cadena que contiene la ruta física completa del comando, seguida de los parámetros separados por espacios en blanco. Si no se especifica la ruta completa, el servidor web busca en la ruta del sistema.

Por ejemplo, el siguiente código produce la página que se muestra en la Figura 2.5:

Ejemplo 2.11

```
1 <HTML>
2 <BODY>
3 <PRE>
4 <!-- #exec cmd="mem.exe" -->
5 </PRE>
6 </BODY>
7 </HTML>
```

2.5.4. flastmod

Esta directiva indica al servidor web que inserte en una página **HTML** el instante (fecha y hora) en que se modificó por última vez el archivo especificado. La sintaxis de esta directiva es:

Ejemplo 2.12

```
1 <!-- #flastmod parametro="valor" -->
```

donde **parametro** puede ser **file** o **virtual** y **valor** contiene la ruta al archivo. En el Cuadro 2.4 se muestran los distintos parámetros del comando **flastmod**.

De forma predeterminada, el instante de modificación se expresa como una fecha con el formato **NombreDía NombreMes NúmeroDía NúmeroAño** (por ejemplo, **Martes Junio 3 1997**). Se puede modificar el formato mediante la directiva **#config** con la opción **timefmt**.

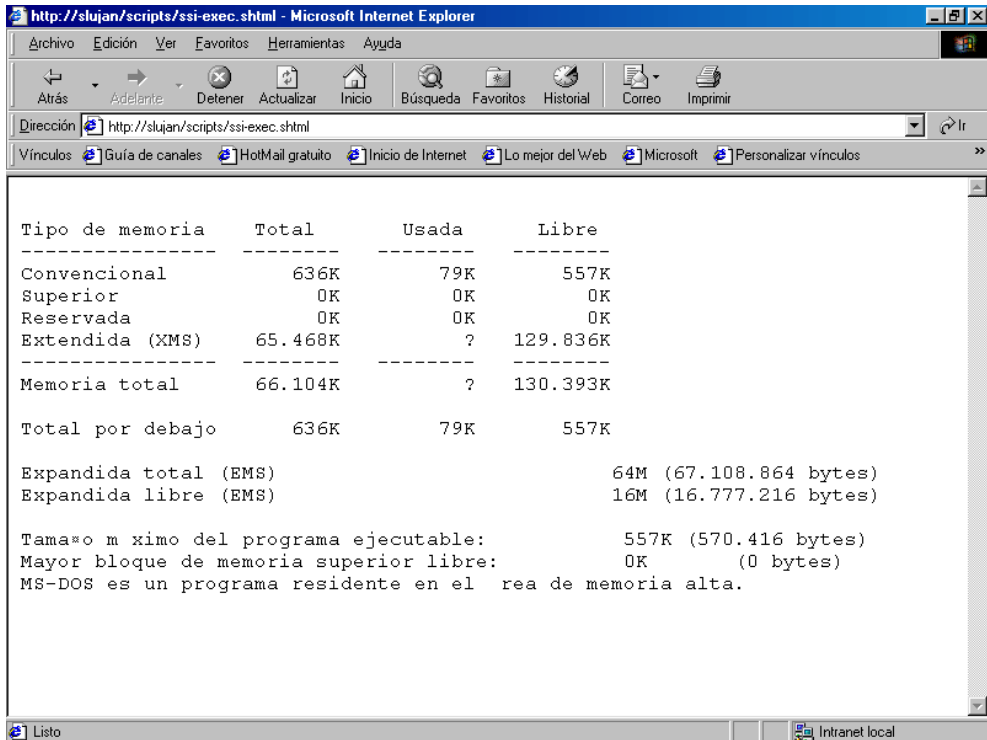


Figura 2.5: Ejemplo de comando exec

Parámetro	Descripción
file	El valor del parámetro indica una ruta relativa al archivo desde el directorio que contiene el documento con esta directiva.
virtual	El valor del parámetro contiene una ruta virtual completa al archivo desde el directorio principal del sitio web.

Cuadro 2.4: Parámetros del comando flastmod, fsize e include

2.5.5. fsize

Esta directiva indica al servidor web que inserte en una página **HTML** el tamaño del archivo especificado. La sintaxis de esta directiva es:

Ejemplo 2.13

```
1 <!-- #fsize parametro="valor" -->
```

donde **parametro** puede ser **file** o **virtual** y **valor** contiene la ruta al archivo. En el Cuadro 2.4 se muestran los distintos parámetros del comando **fsize**.

De forma predeterminada, el tamaño se expresa en kilobytes. Se puede modificar la unidad de medida mediante la directiva **#config** con la opción **sizefmt**.

2.5.6. include

Esta directiva indica al servidor web que inserte en una página **HTML** el contenido de un archivo. El archivo incluido puede tener cualquier contenido válido en los documentos **HTML**. Los archivos incluidos pueden tener cualquier extensión, pero se recomienda que tengan la extensión **.inc**.

La sintaxis de esta directiva es:

Ejemplo 2.14

```
1 <!-- #include parametro="valor" -->
```

donde **parametro** puede ser **file** o **virtual** y **valor** indica la ruta de acceso al archivo que se quiere incluir. En el Cuadro 2.4 se muestran los distintos parámetros del comando **include**.

Evidentemente, el directorio donde se encuentre el archivo que se quiere incluir debe poseer permisos de lectura para que el servidor web lo pueda leer.

Por ejemplo, suponed que tenemos un pie de página que lo queremos incluir en todas las páginas de nuestro sitio web:

Ejemplo 2.15

```
1 Correo electrónico:  
2 <A HREF="mailto:slujan@dlsi.ua.es">slujan@dlsi.ua.es</A>
```

La línea anterior la almacenamos en un archivo, por ejemplo `pie.inc`. A partir de entonces, en todas nuestras páginas podemos poner una directiva `#include` que inserte ese archivo. Por ejemplo, suponed que las siguientes instrucciones pertenecen a la página principal del sitio web:

Ejemplo 2.16

```

1 <HTML>
2 <BODY>
3 <H1>Bienvenido a la web de SLM</H1>
4 Información: ...
5 <BR>
6 Enlaces: ...
7 <BR>
8 <!-- #include file="pie.inc" -->
9 </BODY>
10 </HTML>

```

Cuando el cliente web (navegador) reciba la página, el servidor web habrá procesado la directiva **SSI** y lo que realmente recibirá será:

Ejemplo 2.17

```

1 <HTML>
2 <BODY>
3 <H1>Bienvenido a la web de SLM</H1>
4 Información: ...
5 <BR>
6 Enlaces: ...
7 <BR>
8 Correo electrónico:
9 <A HREF="mailto:slujan@dlsi.ua.es">slujan@dlsi.ua.es</A>
10 </BODY>
11 </HTML>

```

2.6. Ejemplo de programa SSI

El siguiente ejemplo se compone de una página **HTML** con una directiva `#exec` y un programa ejecutable en *C*. El programa calcula cuantos días faltan hasta el *Día de Navidad* (el código se puede adaptar o hacer genérico para que

calcule cuantos días faltan hasta una fecha dada). La página **HTML** ejecuta el programa mediante la directiva **#exec** y muestra el resultado en la propia página. El código de la página **HTML** es:

Ejemplo 2.18

```

1 <HTML>
2 <BODY>
3 <!-- #exec cgi="/scripts/ssi-navidad.exe" -->
4 </BODY>
5 </HTML>

```

El código del programa ejecutable en *C* es:

Ejemplo 2.19

```

1 #include <stdio.h>
2 #include <time.h>
3
4 void main(int argc, char *argv[])
5 {
6     struct tm hoy;
7     time_t ahora;
8     int dias;
9
10    /* Obtiene la fecha actual como UCT */
11    time(&ahora);
12    /* Convierte la fecha actual a la hora local */
13    hoy = *localtime(&ahora);
14    mktime(&hoy);
15
16    /* Los meses comienzan desde 0, pero los dias y
17       años desde 1 */
18    if((hoy.tm_mon == 11) && (hoy.tm_mday == 24))
19        printf("Hoy es Nochebuena, y mañana Navidad");
20    else if((hoy.tm_mon == 11) && (hoy.tm_mday == 25))
21        printf(";Hoy es Navidad!");
22    else
23    {
24        /* Calcula cuantos dias faltan hasta Navidad */
25        dias = 0;
26        while((hoy.tm_mon != 11) || (hoy.tm_mday != 25))
27            {

```

```
28     dias++;
29     hoy.tm_mday = hoy.tm_mday + 1;
30     mktime(&hoy);
31 }
32 printf("Faltan <B>%d</B> dias hasta Navidad", dias);
33 printf("<BR>\n");
34 if(dias < 30)
35 {
36     printf(";Menos de un mes! ");
37     printf("Navidad está ahí.");
38 }
39 else if(dias < 60)
40 {
41     printf("Menos de dos meses. ");
42     printf("Hay que comprar los regalos.");
43 }
44 else
45 {
46     printf("Aún falta un poco. ");
47     printf("Pero hay que empezar a prepararse.");
48 }
49 }
50
51 /* Vacía la salida estándar */
52 fflush(stdout);
53 return;
54 }
```

En la Figura 2.6 vemos como se visualiza la página en un navegador. En el código se puede ver que la hora se obtiene mediante la función `time()` y se convierte a la hora local por medio de la función `localtime()`. La hora que se obtiene es la del servidor web, no la del cliente.

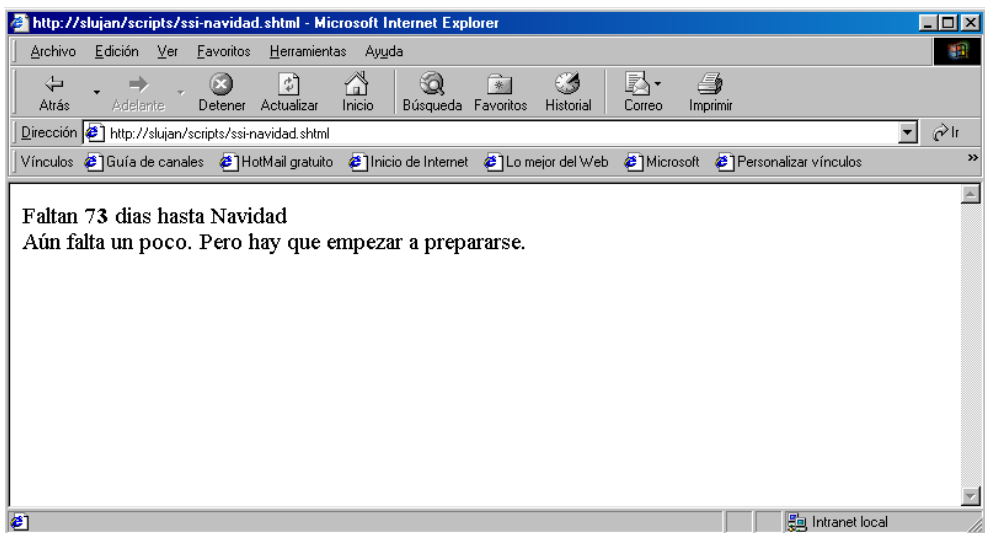


Figura 2.6: Ejemplo de programa ejecutado mediante exec

Capítulo 3

IDC

IDC es una tecnología desarrollada por Microsoft que permite integrar el contenido de una base de datos dentro de una página web. Gracias a esta tecnología, se pueden crear páginas web dinámicas. Una aplicación IDC consta de dos páginas: una contiene la información necesaria para realizar la consulta a la base de datos (qué información se tiene que recuperar), mientras que la otra es una plantilla que indica como se tiene que formatear el resultado devuelto por la base de datos (cómo se tiene que presentar la información). Con IDC, se puede ejecutar cualquier comando SQL (actualización, inserción, borrado, etc.) en una base de datos.

Índice General

3.1. Introducción	68
3.2. Cómo funciona	68
3.3. Qué necesito para programar mediante IDC	69
3.4. Un IDC sencillo	72
3.5. El archivo .idc	74
3.5.1. Campos obligatorios	74
3.5.2. Campos opcionales	75
3.5.3. Campos opcionales avanzados de ODBC	77
3.6. El archivo .htx	77
3.6.1. Valor de un campo en un formulario	78
3.6.2. Variables integradas	79

3.7. Cómo procesar los campos de un formulario	79
3.8. Un IDC más complejo	80
3.8.1. Ejemplo 1	80
3.8.2. Ejemplo 2	81
3.8.3. Ejemplo 3	84

3.1. Introducción

Internet Database Connector (**IDC**) es una tecnología desarrollada por MICROSOFT para su servidor web Microsoft Internet Information Server. Mediante esta tecnología, se pueden crear páginas web dinámicas: permite extraer información de una base de datos y devolverla en forma de página web a través de Internet. Básicamente, **IDC** permite:

- Consultar una base de datos y mostrar el resultado en una página web.
- Actualizar o insertar datos nuevos en una base de datos a partir de los datos introducidos por el usuario en un formulario.
- Eliminar registros en una base de datos según el usuario haya seleccionado en una página web.

La conexión con la base de datos se realiza mediante *Open Database Connectivity* (**ODBC**) de 32 bits. Por tanto, se puede emplear como base de datos cualquier sistema que disponga de su correspondiente controlador **ODBC**.

IDC se ha visto superada por otras tecnologías, como **ASP** de la propia MICROSOFT o **JSP** de SUN MICROSYSTEMS. Pero aún quedan sitios web que emplean esta tecnología, es muy interesante saber cómo se programaba en los primeros años de la web y su sencillez facilita el posterior estudio de tecnologías más avanzadas.

3.2. Cómo funciona

En la Figura 3.1 está representado el funcionamiento básico de un programa **IDC**:

1. El cliente web (el navegador) lanza una petición nueva¹ de un archivo `.idc` mediante **HTTP**. Esta petición puede ir acompañada de datos codificados por el navegador (por ejemplo, información introducida por el usuario en un formulario).
2. El servidor web recibe la petición, analiza la **URL** y detecta que se trata de un archivo `.idc`. Le pasa la petición a **IDC**².
3. **IDC** recibe la petición. Lee el archivo `.idc` indicado en la **URL**. Estos archivos contienen la información necesaria para conectar con el origen de datos **ODBC** apropiado y ejecutar la instrucción *Structured Query Language* (**SQL**). Los archivos `.idc` también contienen el nombre y la ubicación del archivo de extensión de **HTML** correspondiente (`.htx`).
4. Realiza la petición a la base de datos y obtiene los datos de la operación.
5. Lee el archivo `.htx`. Este archivo es la plantilla del documento **HTML** real que se va a devolver al cliente después de que **IDC** haya combinado en ella la información de la base de datos.
6. El programa **IDC** genera su resultado, una página **HTML**, y lo envía al servidor.
7. El servidor web procesa la información recibida del **IDC**: le añade el código necesario para formar un encabezado **HTTP** correcto.
8. El servidor web reenvía el resultado del **IDC** al cliente web.
9. El cliente web muestra la página generada de forma dinámica.

3.3. Qué necesito para programar mediante IDC

Para poder programar mediante **IDC** y probarlo hacen falta los siguientes programas:

¹Una petición nueva porque el usuario ha pulsado sobre un enlace o porque ha escrito una dirección nueva (**URL**).

²**IDC** es una *Dynamic Link Library* (**DLL**) **ISAPI**, que se encuentra en el fichero `Httpodbc.dll`.

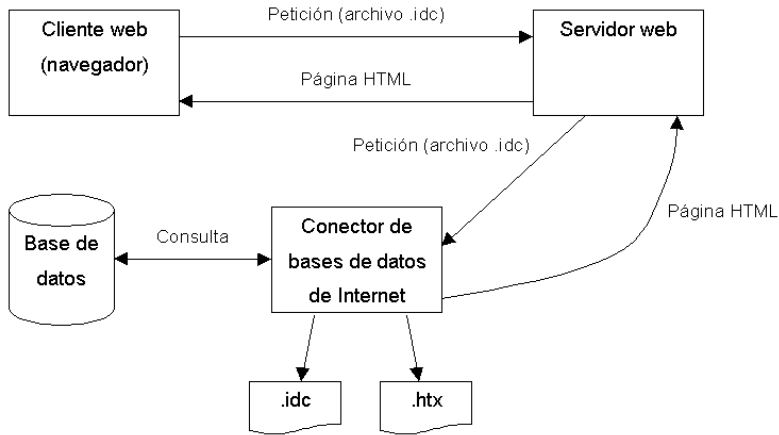


Figura 3.1: Esquema básico de una aplicación web basada en IDC

- Un editor de textos como Bloc de notas de Microsoft Windows o joe de Linux para crear las páginas **HTML** y los ficheros **.idc** y **.htx**.
- Un servidor web (ya sea local o remoto) que acepte **IDC**. Por ejemplo, Microsoft Personal Web Server o Microsoft Internet Information Server.
- Si se desea acceder a una base de datos³, el sistema gestor de bases de datos apropiado (por ejemplo, Microsoft SQL Server o Informix) y el controlador (*driver*) **ODBC** correspondiente⁴.
- Por último, un navegador como Netscape Communicator o Microsoft Internet Explorer para poder comprobar las páginas **HTML** y el funcionamiento de **IDC**.

No es necesario disponer de una conexión a Internet, ya que se puede comprobar localmente el código creado.

Para utilizar **IDC** se tienen que seguir los siguiente pasos:

1. Instalar el controlador **ODBC** necesario para acceder a la base de datos.

³Se puede emplear cualquier base de datos que sea accesible mediante **ODBC**.

⁴Aunque Microsoft Access no se puede considerar un gestor de bases de datos, como dispone de driver **ODBC** también se puede emplear.

2. Crear un *Data Source Name* (**DSN**) de sistema para acceder a la base de datos.
3. Crear el archivo `.idc` con instrucciones de consulta **SQL** a la base de datos (se hace referencia al **DSN** creado). Además, este fichero también contiene el nombre del archivo `.htx` que se va a emplear como plantilla.
4. Crear el archivo `.htx` con las etiquetas **HTML** que dan formato a los datos devueltos por la base de datos.

Para poder ejecutar los ficheros `.idc`, el directorio donde se encuentran alojados debe de poseer permisos de ejecución. En caso contrario, al intentar ejecutar un **IDC** se muestra el mensaje de error que se ve en la Figura 3.2.

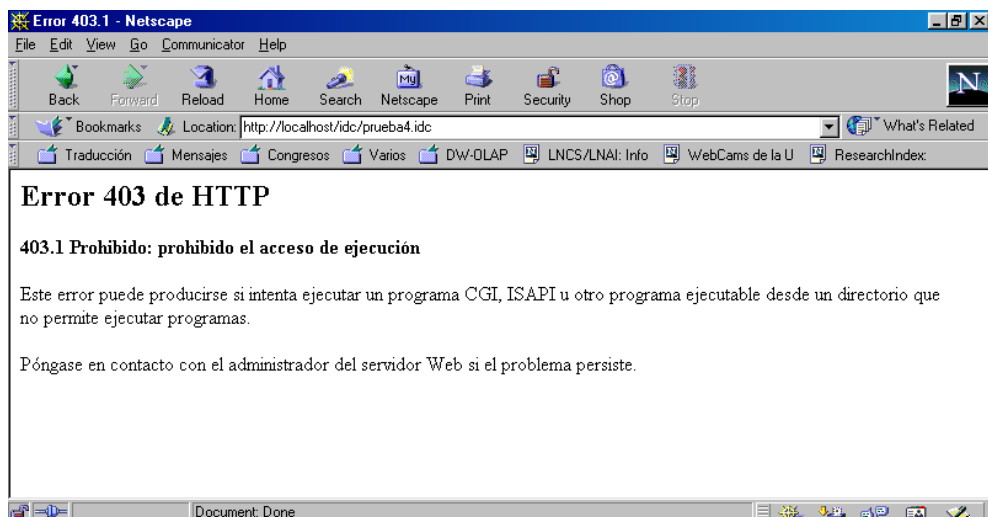


Figura 3.2: Mensaje de error porque no hay permisos de ejecución

Si se emplea Microsoft Personal Web Server, el directorio que almacena estos ficheros tiene que poseer los permisos de **Ejecución** o **Archivos de comandos**, tal como se puede apreciar en la Figura 3.3.

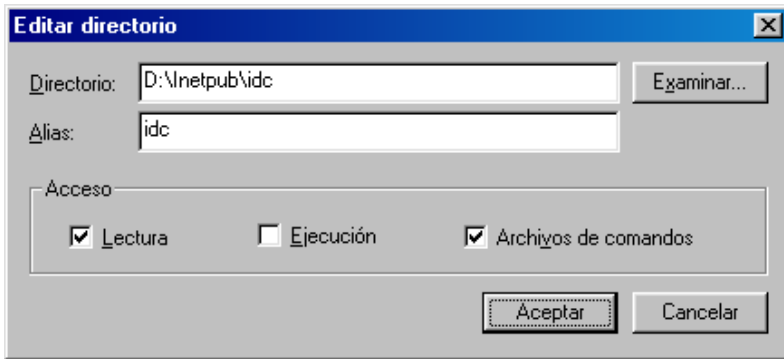


Figura 3.3: Permisos de ejecución en Microsoft Personal Web Server

3.4. Un IDC sencillo

Veamos con un ejemplo sencillo como funciona **IDC**. Suponemos que tenemos creada una base de datos (por ejemplo, en Microsoft Access) y hemos creado un **DSN** llamado **prueba**. La base de datos contiene una tabla llamada **Articulos** con dos campos: **Codigo** y **Descripcion**. El código mínimo para el archivo **.idc** (**prueba1.idc**) es:

Ejemplo 3.1

```

1 Datasource: prueba1
2 Template: prueba1.htx
3 SQLStatement: SELECT * FROM Articulos

```

donde **Datasource** indica el **DSN**, **Template** el nombre del archivo de extensión de **HTML** asociado y **SQLStatement** la sentencia **SQL** que se desea ejecutar.

El código del archivo **prueba1.htx** puede ser el siguiente:

Ejemplo 3.2

```

1 <HTML>
2 <BODY>
3 <HR>
4 <%begindetail%>
5 Codigo: <B><%Codigo%></B><BR>

```

```
6  Descripcion: <I><%Descripcion%></I><BR>
7  <HR>
8  <%enddetail%>
9  </BODY>
10 </HTML>
```

Como se ve, el archivo `.htx` es una página **HTML** normal con algunas etiquetas especiales que van encerradas entre `<% ... %>`. Las secciones `<%begindetail%>` y `<%enddetail%>` delimitan la zona del documento en la que aparecerán las filas devueltas por la base de datos. Las columnas devueltas por la consulta están enmarcadas por `<% ... %>`, como `<%Codigo%>` y `<%Descripcion%>` en este ejemplo.

En la Figura 3.4 se puede ver como se muestra la página en el navegador Microsoft Internet Explorer.

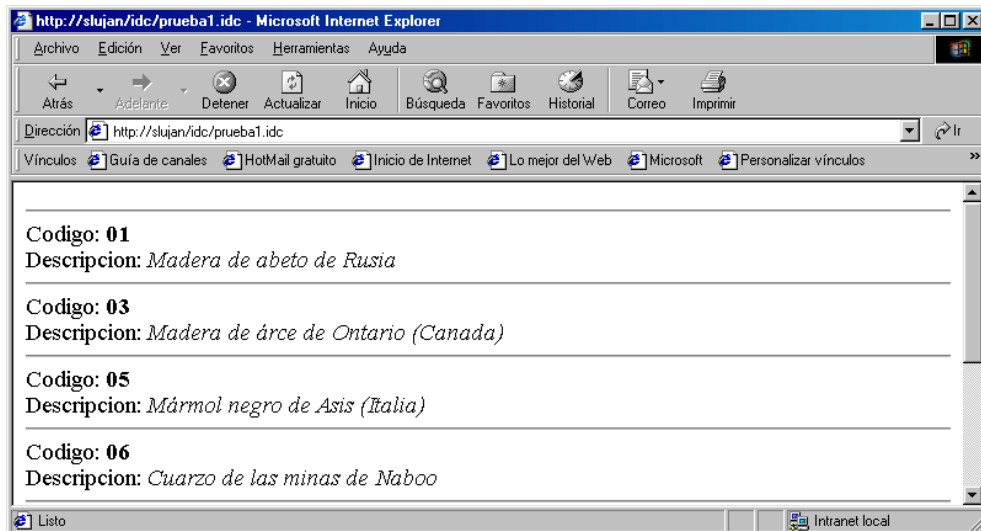


Figura 3.4: Ejemplo de un IDC sencillo

3.5. El archivo .idc

Los archivos `.idc` contienen la información necesaria para conectarse a una base de datos y ejecutar una sentencia **SQL**. Además, también contienen una referencia al archivo `.htx`.

Los campos que se emplean en un archivo `.idc` se clasifican en tres grupos: campos obligatorios (siempre tienen que estar), campos opcionales y campos opcionales avanzados de **ODBC** (especifican opciones para el controlador **ODBC**).

3.5.1. Campos obligatorios

Los campos obligatorios siempre tienen que aparecer en un archivo `.idc`: es lo mínimo para que funcione. Estos campos son: **Datasource**, **Template** y **SQLStatement**.

Datasource

Nombre del origen de datos (**DSN** del sistema) que se emplea para conectar con la base de datos. En la Figura 3.5 se muestra el mensaje de error que produce **IDC** cuando no se encuentra el **DSN** indicado.

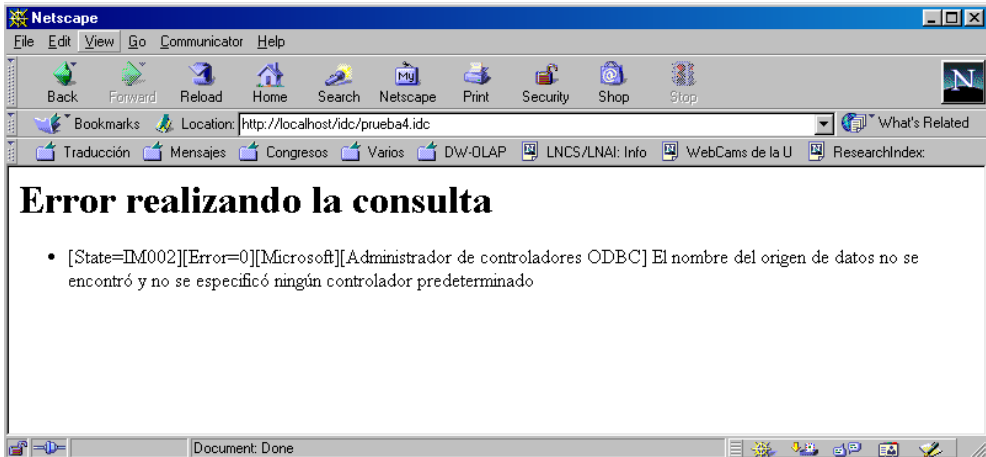


Figura 3.5: Mensaje de error porque no existe DNS

Template

Nombre del archivo de extensión de **HTML** que da formato a los datos devueltos por la consulta.

SQLStatement

Instrucción **SQL** que se desea ejecutar. Puede ocupar varias líneas: a continuación de este campo, las líneas que comiencen con un signo más (+) se consideran parte del campo.

3.5.2. Campos opcionales

Content-Type

Tipo **MIME** válido que describe lo que se va a devolver al cliente. Casi siempre será `text/html` si el archivo `.htx` contiene **HTML**.

DefaultParameters

Asigna valor a aquellos parámetros que el usuario no haya asignado valor. Por ejemplo:

Ejemplo 3.3

```
1 DefaultParameters=ciudad=Alicante,provincia=Alicante
```

Expires

Número de segundos que hay que esperar antes de actualizar una página almacenada en memoria caché. De forma predeterminada, **IDC** no almacena en memoria caché las páginas; sólo las almacena cuando se utiliza este campo.

MaxFieldSize

Tamaño máximo asignado a cada campo devuelto por una base de datos. El resto de caracteres se trunca. El valor predeterminado es 8192 bytes.

MaxRecords

Número máximo de registros que se devuelven desde cualquier consulta. Este campo no tiene un valor predeterminado, lo que significa que una consulta puede devolver hasta 4000 millones de registros.

ODBCConnection

Permite indicar si se desea agregar la conexión al conjunto de conexiones que se mantienen abiertas para futuras consultas (`Pool`) o no (`NoPool`). Esta opción permite mejorar el rendimiento en el acceso a las bases de datos, pero el número de conexiones simultáneas puede tener un límite.

Password

Contraseña necesaria para el nombre de usuario empleado para conectarse a la base de datos. Se emplea junto con el campo `Username`.

RequiredParameters

Nombres de los parámetros, si hay alguno, que `Httpodbc.dll` comprueba que se reciben del cliente; de lo contrario, se devuelve un error. Los nombres de los parámetros están separados por comas.

Translationfile

Ruta del archivo que asigna caracteres no ingleses (como ñ, à, ô o é) de forma que los navegadores puedan presentarlos correctamente en formato **HTML**. Si el archivo de traducción no está en el mismo directorio que el archivo `.idc`, se debe escribir la ruta completa al archivo de traducción. Por ejemplo:

Ejemplo 3.4

```
1 Translationfile: C:\Ficheros\traduccion.txt
```

El archivo de traducción es un archivo de texto que asigna a cada carácter especial una traducción mediante el formato siguiente: `valor=cadena<CR>`, donde `valor` es un carácter especial y `cadena` es el correspondiente código **HTML**.

Username

Nombre de usuario empleado para conectarse a la base de datos. Se emplea junto con el campo `Password`.

3.5.3. Campos opcionales avanzados de ODBC

Las opciones avanzadas de **ODBC** permiten depurar y ajustar el controlador **ODBC**. El formato que se emplea en el archivo **IDC** es el siguiente:

Ejemplo 3.5

```
1 ODBCOptions: Opcion=Valor[,Opcion=Valor...]
```

No vamos a comentar cada opción, pero incluimos la lista de opciones disponibles: `SQL_ACCESS_MODE`, `SQL_LOGIN_TIMEOUT`, `SQL_OPT_TRACE`, `SQL_OPT_TRACEFILE`, `SQL_PACKET_SIZE`, `SQL_TRANSLATE_DLL`, `SQL_TRANSLATE_OPTION`, `SQL_TXN_ISOLATION`, `SQL_MAX_LENGTH`, `SQL_MAX_ROWS`, `SQL_NOSCAN` y `SQL_QUERY_TIMEOUT`.

3.6. El archivo .htx

Los archivos `.htx` (*HTML Extension Template*) son unas plantillas que se emplean para formatear los resultados de una consulta. En realidad, la extensión de estos archivos puede ser cualquiera (no es obligatorio que sea `.htx`).

Estos archivos contienen una serie de palabras clave que controlan cómo se genera el documento **HTML** de resultado: `<%begindetail%>`, `<%enddetail%>`, `<%if %>`, `<%else%>` y `<%endif%>`.

Las palabras clave `<%begindetail%>` y `<%enddetail%>` enmarcan una sección del archivo `.htx` en la que se van a combinar datos procedentes de una base de datos. Para marcar la posición de los datos devueltos por la base de datos, los nombres de las columnas van encerrados entre `<% ... %>`. Si la consulta no devuelve registros, se saltará la sección `<%begindetail%>`.

Los archivos `.htx` pueden contener lógica condicional con una instrucción `<%if ... then ... else ... %>` para controlar cómo se genera la página. La sintaxis de esta instrucción es:

Ejemplo 3.6

```

1 <%if condición %>
2 texto HTML
3 [<%else%>
4 texto HTML]
5 <%endif%>

```

donde la parte `<%else%>` es opcional. La condición tiene el siguiente formato: `valor1 operador valor2`. Los operadores contemplados se muestran en el Cuadro 3.1, y aunque aparecen en mayúsculas, también se pueden emplear en minúsculas.

Operador	Descripción
EQ	Cierto si <code>valor1</code> es igual a <code>valor2</code>
LT	Cierto si <code>valor1</code> es menor que <code>valor2</code>
GT	Cierto si <code>valor1</code> es mayor que <code>valor2</code>
CONTAINS	Cierto si cualquier parte de <code>valor1</code> contiene la cadena <code>valor2</code>

Cuadro 3.1: Operadores de las expresiones lógicas

Como se ve, los operadores disponibles son muy pocos, lo que dificulta la programación de expresiones complejas (hay que emplear muchas sentencias condicionales en cascada).

3.6.1. Valor de un campo en un formulario

Si se quiere mostrar el valor que se ha introducido o seleccionado en un campo de un formulario, se tiene que emplear la sintaxis `<%idc.nombreCampo%>`, donde `nombreCampo` es el nombre asignado al campo del formulario mediante el atributo `NAME`⁵. Es muy importante no dejar un espacio en blanco entre `%` e `idc`.

⁵Por ejemplo, `<INPUT TYPE="TEXT" NAME="apellidos">`.

3.6.2. Variables integradas

Existen dos variables integradas que se pueden emplear en las páginas `.htx`: `CurrentRecord` y `MaxRecords`.

La variable integrada `CurrentRecord` contiene el número de veces que se ha procesado la sección `<%begindetail%>`. La primera vez que se pasa por la sección `<%begindetail%>`, el valor es cero. Después, el valor de `CurrentRecord` se incrementa por cada registro devuelto por la base de datos.

La variable integrada `MaxRecords` contiene el valor del campo `MaxRecords` del archivo `.idc` asociado a la página.

Muy importante: tanto `MaxRecords` como `CurrentRecord` sólo se pueden utilizar en instrucciones `<%if ... then ... else ...%>`.

3.7. Cómo procesar los campos de un formulario

Se pueden realizar consultas a una base de datos en función de los datos de entrada del usuario en un formulario **HTML**. Para ello hay que emplear los valores que se envían con una petición desde un formulario.

El navegador web envía los datos de un formulario en forma de parejas `parametro=valor`, donde `parametro` es el nombre del control en el formulario y `valor` es la entrada del usuario en el control. Cuando se quiera emplear el valor de un parámetro en una consulta, se tiene que encerrar el nombre del control (parámetro) entre signos de porcentaje (%). Por ejemplo, si tenemos un formulario con un cuadro de texto llamado `nombre`, una posible consulta a una base de datos puede ser:

Ejemplo 3.7

```
1 SQLStatement: SELECT * FROM Clientes WHERE
2 +NombreCliente='%nombre%'
```

La cadena `%nombre%` aparece encerrada entre comillas porque el campo `NombreCliente` de la base de datos es de tipo cadena (`char`). Si fuera de tipo numérico (por ejemplo, `int`), no tendría que aparecer entrecorillado.

El carácter de porcentaje (%) también es un carácter comodín en **SQL**. Los comodines se utilizan en las consultas **SQL** para buscar elementos de una tabla (búsquedas mediante un patrón). Como el signo de porcentaje tiene un significado especial cuando se emplea **IDC**, para insertar un único carácter de porcentaje como comodín de **SQL**, se tiene que escribir dos veces (%%).

3.8. Un IDC más complejo

A continuación presentamos tres ejemplos de páginas web basadas en **IDC**. El primero realiza una consulta que depende del valor seleccionado por el usuario en un formulario. El segundo ejemplo inserta en una base de datos los datos introducidos por el usuario en un formulario **HTML**. El último ejemplo muestra como se puede incorporar validación de usuarios a una aplicación web.

3.8.1. Ejemplo 1

El siguiente ejemplo se compone de tres ficheros: `prueba2.html`, `prueba2.idc` y `prueba2.htx`. Además, hace falta una base de datos que contenga una tabla llamada `Cientes` con los campos `Nombre`, `Direccion` y `Poblacion` (todos ellos de tipo texto o cadena), y crear un origen de datos para la base de datos llamado `prueba2`.

`prueba2.html`

Este fichero crea una página **HTML** con un formulario que posee una lista desplegable con dos valores (`Alicante` y `Valencia`). Al pulsar el botón de consulta, se solicita la página `prueba2.idc`.

Ejemplo 3.8

```
1 <HTML>
2 <BODY>
3 <FORM ACTION="prueba2.idc" METHOD="POST">
4 Seleccione una población:
5 <SELECT NAME="poblacion">
6 <OPTION VALUE="Alicante">Alicante</OPTION>
7 <OPTION VALUE="Valencia">Valencia</OPTION>
8 </SELECT>
9 <INPUT TYPE="SUBMIT" VALUE="Consultar">
10 </FORM>
11 </BODY>
12 </HTML>
```

`prueba2.idc`

Contiene el origen de datos, el nombre del fichero `.htx` asociado y la consulta **SQL** correspondiente. El nombre del campo del formulario que se emplea

para realizar la consulta va encerrado entre signos de porcentaje (%).

Ejemplo 3.9

```

1 Datasource: prueba2
2 Template: prueba2.htx
3 SQLStatement: SELECT * FROM Clientes WHERE
4 +Poblacion='%poblacion%'

```

prueba2.htx

Contiene el código **HTML** necesario para dar formato a los datos que se recuperan de la base de datos.

Ejemplo 3.10

```

1 <HTML>
2 <BODY>
3 <HR>
4 <%begindetail%>
5 Nombre: <B><%Nombre%></B><BR>
6 Direccion: <I><%Direccion%></I><BR>
7 Poblacion: <U><%Poblacion%></U><BR>
8 <HR>
9 <%enddetail%>
10 </BODY>
11 </HTML>

```

3.8.2. Ejemplo 2

El siguiente ejemplo se compone de tres ficheros: `prueba3.html`, `prueba3.idc` y `prueba3.htx`. Además, hace falta una base de datos que contenga una tabla llamada `Articulos` con los campos `Codigo` de tipo texto, `Cantidad` de tipo numérico y `Descripción` de tipo texto, y crear un origen de datos para la base de datos llamado `prueba3`.

prueba3.html

Este fichero crea una página **HTML** con un formulario que posee tres cuadros de texto: `Codigo`, `Cantidad` y `Descripcion`. Al pulsar el botón `Insertar`, se envían los datos al servidor y se insertan en la base de datos.

Ejemplo 3.11

```

1 <HTML>
2 <BODY>
3 <CENTER>
4 <FORM ACTION="prueba3.idc" METHOD="POST">
5 Datos del artículo:
6 <BR>
7 <TABLE BORDER="0">
8 <TR>
9   <TD>Código: </TD>
10  <TD><INPUT TYPE="TEXT" NAME="codigo"></TD>
11 </TR>
12 <TR>
13   <TD>Cantidad: </TD>
14   <TD><INPUT TYPE="TEXT" NAME="cantidad"></TD>
15 </TR>
16 <TR>
17   <TD>Descripción: </TD>
18   <TD><TEXTAREA NAME="descripcion" ROWS="5" COLS="40"></TEXTAREA></TD>
19 </TR>
20 </TABLE>
21 <INPUT TYPE="SUBMIT" VALUE="Insertar">
22 </FORM>
23 </CENTER>
24 </BODY>
25 </HTML>

```

En la Figura 3.6 se muestra como se visualiza el código anterior en el navegador Microsoft Internet Explorer.

prueba3.idc

Contiene el origen de datos, el nombre del fichero `.htx` asociado y la instrucción de inserción en la base de datos mediante **SQL**. El nombre del campo del formulario que se emplea para realizar la consulta va encerrado entre signos de porcentaje (%).

Darse cuenta de que los campos `codigo` y `descripcion` van encerrados entre comillas simples porque los campos correspondientes esperan cadenas de texto. Sin embargo, el campo correspondiente a `cantidad` es numérico, por lo que no hace falta que aparezca entre comillas simples.

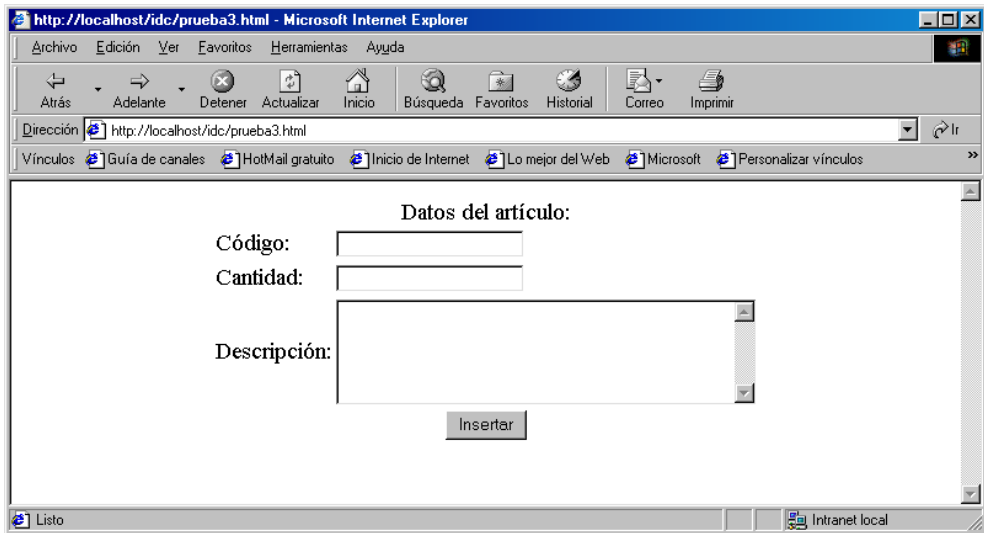


Figura 3.6: Formulario de toma de datos para inserción

Ejemplo 3.12

```

1 Datasource: prueba3
2 Template: prueba3.htx
3 SQLStatement: INSERT INTO Articulos(Codigo, Cantidad, Descripcion)
4 +VALUES ('%codigo%', %cantidad%, '%descripcion%')
```

prueba3.htx

Si la inserción ha sido realizada, se muestra la siguiente página.

Ejemplo 3.13

```

1 <HTML>
2 <BODY>
3 Insercción realizada
4 </BODY>
5 </HTML>
```

3.8.3. Ejemplo 3

El siguiente ejemplo se compone de tres ficheros: `prueba4.html`, `prueba4.idc` y `prueba4.htx`. Además, hace falta una base de datos que contenga una tabla llamada `Usuarios` con los campos `Nombre` y `Contraseña` (todos ellos de tipo texto), y crear un origen de datos para la base de datos llamado `prueba4`.

Este ejemplo muestra como se puede controlar el acceso a una parte privada de una web mediante la solicitud de un nombre de usuario y una contraseña.

`prueba4.html`

Este fichero crea una página **HTML** con un formulario con dos campos de texto que solicitan un `Nombre` y una `Contraseña`. Al pulsar el botón `Entrar`, se envían los datos al servidor y se comprueba la existencia de un nombre de usuario con la contraseña indicada. En la Figura 3.7 se puede ver como se muestra el siguiente código en un navegador.

Ejemplo 3.14

```
1 <HTML>
2 <BODY>
3 <CENTER>
4 <FORM ACTION="prueba4.idc" METHOD="POST">
5 Acceso a la parte privada:
6 <BR><BR>
7 <TABLE BORDER="0">
8 <TR>
9   <TD>Usuario: </TD>
10  <TD><INPUT TYPE="TEXT" NAME="nombre"></TD>
11 </TR>
12 <TR>
13   <TD>Clave: </TD>
14   <TD><INPUT TYPE="PASSWORD" NAME="contra"></TD>
15 </TR>
16 <TR>
17   <TD>&nbsp;</TD>
18   <TD ALIGN="CENTER">
19     <INPUT TYPE="SUBMIT" VALUE="Entrar">
20     <INPUT TYPE="RESET" VALUE="Borrar">
21   </TD>
22 </TR>
```

```
23 </TABLE>
24 </FORM>
25 </CENTER>
26 </BODY>
27 </HTML>
```

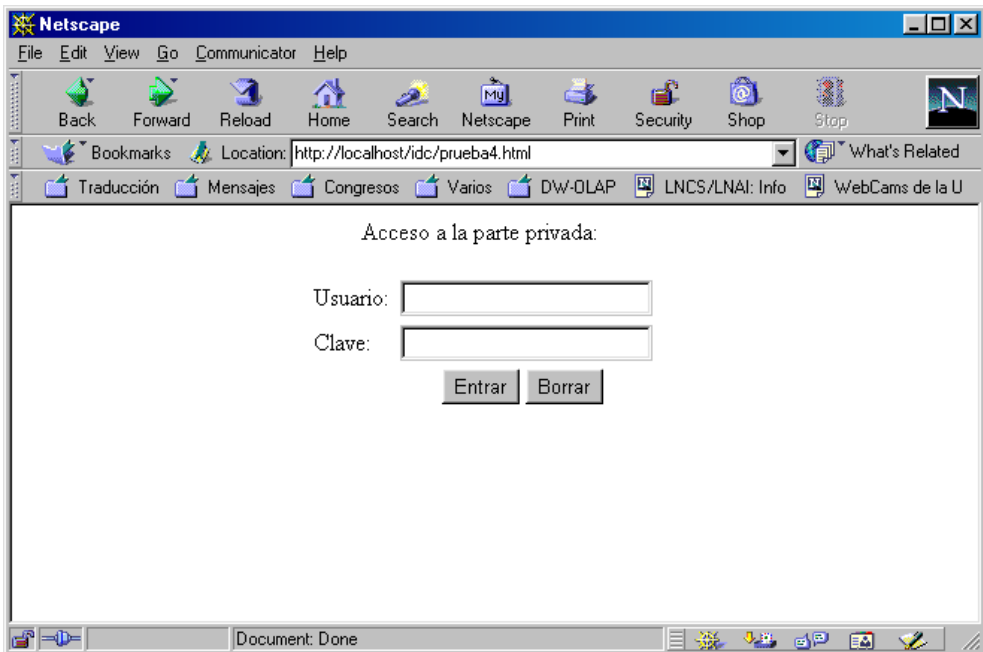


Figura 3.7: Formulario de acceso a la parte privada

prueba4.idc

Contiene el origen de datos, el nombre del fichero `.htx` asociado y la instrucción de consulta en la base de datos mediante **SQL**. El nombre del campo del formulario que se emplea para realizar la consulta va encerrado entre signos de porcentaje (%).

La sentencia **SQL** busca en la base de datos una tupla (registro) con el nombre de usuario y la contraseña introducidos en el formulario.

Ejemplo 3.15

```
1 Datasource: prueba4
2 Template: prueba4.htx
3 SQLStatement: SELECT * FROM Usuarios
4 +WHERE Nombre = '%nombre%' AND Contraseña = '%contra%'
```

prueba4.htx

Esta página muestra un mensaje de error si el nombre de usuario o la contraseña no son correctos, o el menú principal de la parte privada si la validación ha sido correcta. Para comprobar si la validación ha sido correcta, consulta el valor de la variable **CurrentRecord**: si vale 0, la sentencia **SQL** no ha devuelto ninguna tupla, lo que implica que el usuario no existe.

Ejemplo 3.16

```
1 <HTML>
2 <BODY>
3 <CENTER>
4 <%begindetail%>
5 <%enddetail%>
6 <%if CurrentRecord EQ 0%>
7 <FONT SIZE="4" COLOR="red">Acceso no permitido</FONT>
8 <%else%>
9 <FONT SIZE="4" COLOR="blue">Acceso permitido</FONT>
10 <BR>
11 Menú principal de la aplicación...
12 <%endif%>
13 </CENTER>
14 </BODY>
15 </HTML>
```

Apéndice A

Cómo crear un DSN

En este apéndice se explica en qué consiste ODBC y cómo se puede crear un origen de datos (DSN) en un sistema operativo Microsoft Windows 98 para acceder a una base de datos a través de ODBC.

Índice General

A.1. ODBC	87
A.2. Creación de un DSN	91

A.1. ODBC

Cada **SGBD** posee su propio *Application Program Interface* (**API**): su propio conjunto de funciones, su propia secuencia de llamadas necesarias para realizar una operación y su propia forma de devolver la información¹. De cara al programador, esta situación origina graves inconvenientes:

- Si queremos que una aplicación acceda simultáneamente a varias bases de datos alojadas en distintos **SGBD**, necesitaremos conocer y usar los distintos **API** de cada sistema.

¹El estándar **SQL** define las sentencias que se pueden emplear para almacenar o recuperar información en una base de datos, pero no especifica los pasos necesarios para conectarse a una base de datos, enviar o recuperar los datos, etc.

- Si tenemos una aplicación que accede a una base de datos en un **SGBD** y queremos almacenar la base de datos en otro sistema totalmente distinto, tendremos que reprogramar la aplicación para que haga uso del nuevo **API**.

Para evitar estos y otros problemas, se han definido distintas capas de abstracción que evitan estos problemas, ya que la aplicación no tiene que tratar directamente con el **SGBD**. Una de las más conocidas y empleadas es **ODBC**. **ODBC** ofrece una capa intermedia (*middleware*) entre las aplicaciones y las bases de datos, lo que se traduce en una mayor flexibilidad a la hora de sustituir un **SGBD** por otro.

En la Figura A.1 se han resumido los tres mecanismos de acceso a una base de datos más comunes: mediante una solución a medida (*ad-hoc*), mediante un protocolo de comunicación facilitado por el fabricante del **SGBD** (solución propietaria) o mediante **ODBC**.

En el primer método, mediante una solución a medida, el programador tiene que crear el protocolo de comunicación que permita conectar su aplicación cliente con el servidor que aloja la base de datos. Para enviar y recibir los mensajes, se puede hacer uso de un protocolo de transporte ya existente, como *Transmission Control Protocol/Internet Protocol (TCP/IP)* o NetBEUI. El programador tiene que establecer la forma de mantener un “diálogo” y tiene que definir cada uno de los posibles mensajes que pueden emplearse. Este método suele ser el más rápido y el que menos recursos necesita, pero es el más laborioso y muy propenso a cometer errores.

En el segundo método, mediante un protocolo de comunicación facilitado por el fabricante, el programador hace uso de un **API** específico que le facilita el fabricante del **SGBD** por medio de una capa específica que se establece entre la aplicación y la base de datos. Esta capa intermedia se encarga de toda la comunicación entre el cliente y el servidor.

Por último, mediante **ODBC**, el programador hace uso de un **API** estándar que le facilita **ODBC**, y es este último el que se encarga de traducir las llamadas al **API** estándar en llamadas al **API** específico de cada **SGBD**. Desde un punto de vista conceptual, **ODBC** es similar al sistema de impresión en los sistemas operativos Microsoft Windows².

²Para imprimir, una aplicación emplea un **API** de una impresora genérica (virtual), y un controlador (*driver*) específico para cada impresora traduce las órdenes enviadas a la impresora genérica en órdenes específicas para la impresora destino.

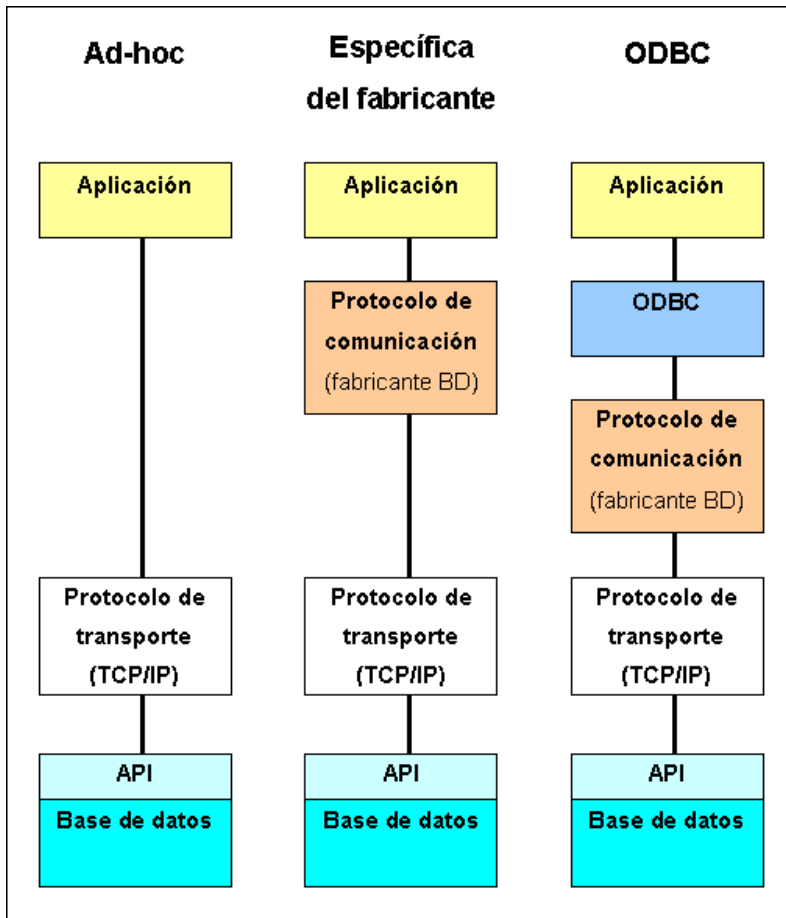


Figura A.1: Mecanismos de acceso a bases de datos

En la Figura A.2 se ha esquematizado la arquitectura de **ODBC**: un gestor de controladores **ODBC** gestiona todos los controladores instalados en un sistema; para que una aplicación pueda acceder a una base de datos, hace falta disponer del controlador apropiado; cada controlador **ODBC** específico para un **SGBD** transforma las llamadas al **API ODBC** en llamadas al **API** específico de cada sistema.

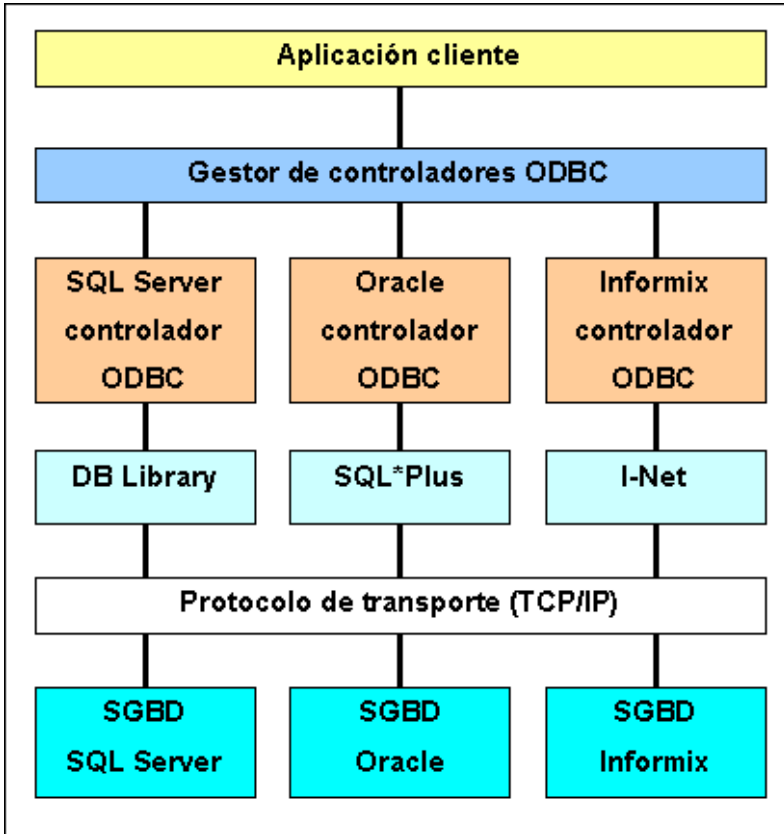


Figura A.2: Arquitectura de ODBC

ODBC se puede considerar el interfaz de programación estándar *de facto* para acceder a bases de datos en los sistemas operativos de MICROSOFT, aunque también se encuentra disponible en otros sistemas operativos (IBM OS/2, algunos Unix, etc.).

MICROSOFT creó **ODBC** en 1992 a partir de SAG CLI³. La última versión, la 3.0⁴, apareció en 1996, y cumple prácticamente la especificación de ANSI/ISO SQL-92 CLI, el estándar *de jure* (oficial).

ODBC define un **API** con cientos de funciones y constantes. Las funciones se pueden clasificar en distintas categorías:

- Manipulación de datos: `SQLExecute`, `SQLFetch`, `SQLGetData`, `SQLRowCount`, etc.
- Información sobre la base de datos (metadatos)⁵: `SQLTables`, `SQLColumns`, `SQLProcedures`, `SQLPrimaryKeys`, etc.
- Información sobre el controlador: `SQLGetFunctions`, `SQLGetInfo`, `SQLGetTypeInfo`, etc.

La principal ventaja de **ODBC** es a su vez su peor inconveniente: al ofrecer un **API** estándar, se compone de aquellas funciones que se podrían considerar el “mínimo común denominador” de todos los **API** existentes, por lo que muchas funciones que existen en los **API** de algunos **SGBD** no se encuentran disponibles en **ODBC**.

ODBC define distintos niveles de “conformidad” con el estándar. Por un lado, existen una serie de niveles respecto a las sentencias **SQL** (gramática **SQL**) aceptadas: mínimo (*minimum*), núcleo (*core*) y extendido (*extended*). Por otro lado, existen distintos niveles respecto al **API** proporcionado: niveles 0, 1 y 2. Ambos niveles son independientes, por lo que un controlador puede presentar cualquier combinación posible.

El uso de **ODBC** se encuentra muy extendido, de modo que los propios fabricantes de **SGBD** ofrecen controladores **ODBC** para sus propios sistemas y terceras compañías también venden controladores para la mayoría de las bases de datos.

A.2. Creación de un DSN

Mediante **ODBC**, una aplicación puede acceder a distintas bases de datos, controladas por distintos **SGBD**, sin tener que modificar el código de la

³SQL Access Group Call-level Interface.

⁴La última revisión a principios de 2001 es la 3.5.

⁵Información sobre los nombres de las tablas, de las columnas, privilegios, etc.

aplicación para cada sistema. **ODBC** añade una capa intermedia entre una aplicación y un **SGBD**, de forma que se logra un nivel de abstracción (independencia) adicional.

Para que una aplicación compatible con **ODBC** pueda emplear una base de datos almacenada en un **SGBD** nuevo, sólo hace falta disponer del controlador (*driver*) apropiado.

En entornos Microsoft Windows es posible lograr un nivel de independencia adicional, gracias al empleo de **DSN**. Un **DSN** contiene toda la información necesaria para conectarse a un origen de datos: ubicación, sistema de la base de datos, usuario, contraseña, etc. En Microsoft Windows 98⁶, existe la aplicación **Fuentes de datos ODBC (32 bits)**⁷ (Figura A.3), que permite crear un **DSN**. Esta aplicación es accesible a través de **Inicio** → **Configuración** → **Panel de control**.



Figura A.3: Fuentes de datos ODBC

En la Figura A.4 se muestra la pantalla principal de la aplicación **Administrador de orígenes de datos ODBC**. Como se puede observar, se pueden crear tres tipos de **DSN**: **DSN** de usuario, de sistema y de archivo.

Cuando se emplea **DSN** de usuario, los orígenes de datos son locales para un equipo y sólo pueden ser utilizados por el usuario actual.

Por otro lado, los **DSN** de sistema son orígenes de datos locales para un equipo, más que dedicados a un usuario. El sistema, o cualquier usuario que tenga privilegios, puede utilizar un origen de datos configurado con un **DSN** de sistema.

Por último, los **DSN** de archivo son orígenes de datos basados en archivos que pueden ser compartidos entre todos los usuarios que tengan instalados los

⁶La explicación se refiere a Microsoft Windows 98, pero se puede aplicar a la mayoría de los sistemas operativos de MICROSOFT.

⁷Las “fuentes de datos” también se conocen como “orígenes de datos”.

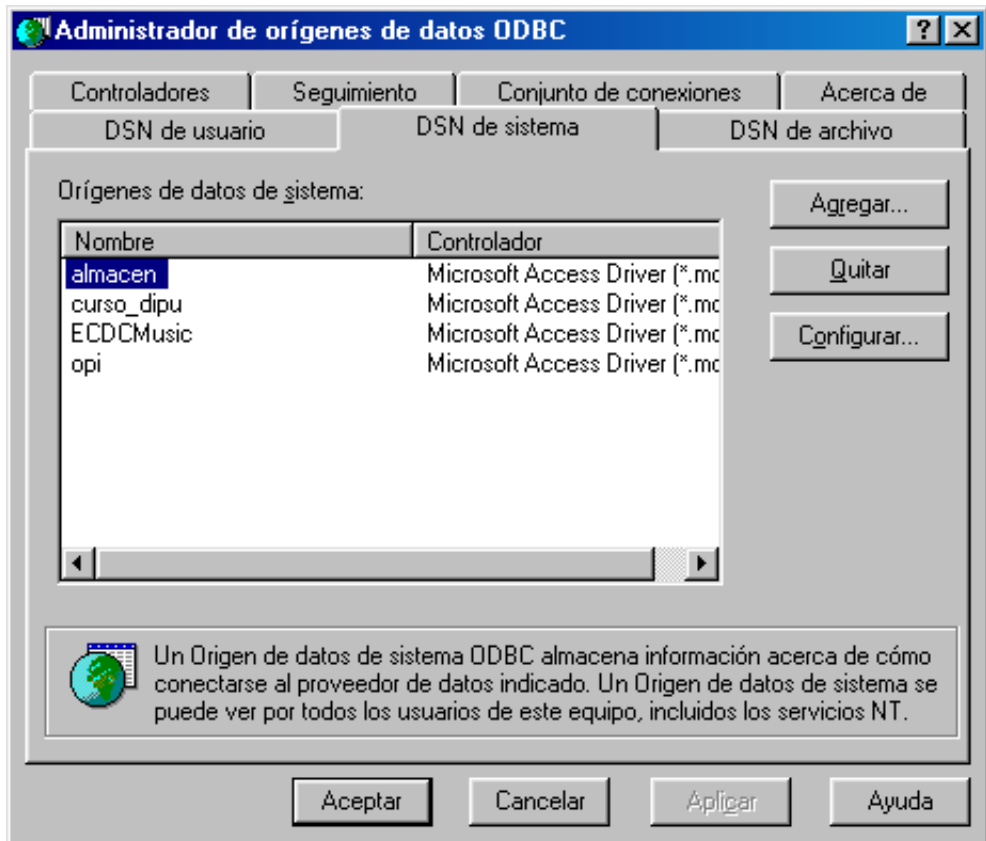


Figura A.4: Pantalla principal de Fuentes de datos ODBC

mismos controladores y, por tanto, tengan acceso a la base de datos. Estos orígenes de datos no necesitan estar dedicados a un usuario ni ser locales en un equipo.

Dependiendo del controlador seleccionado se sucederán diferentes ventanas para la creación del **DSN**, ya que la información necesaria en cada caso varía considerablemente⁸. En este libro vamos a estudiar un único caso: la creación de un **DSN** de sistema para Microsoft Access.

Para crear un nuevo origen de datos de sistema, se tiene que pulsar el botón **Agregar** en la ventana mostrada en la Figura A.4. Al pulsar este botón, se presenta el cuadro de diálogo **Crear nuevo origen de datos** (Figura A.5) con una lista que muestra todos los controladores **ODBC** instalados en el equipo.

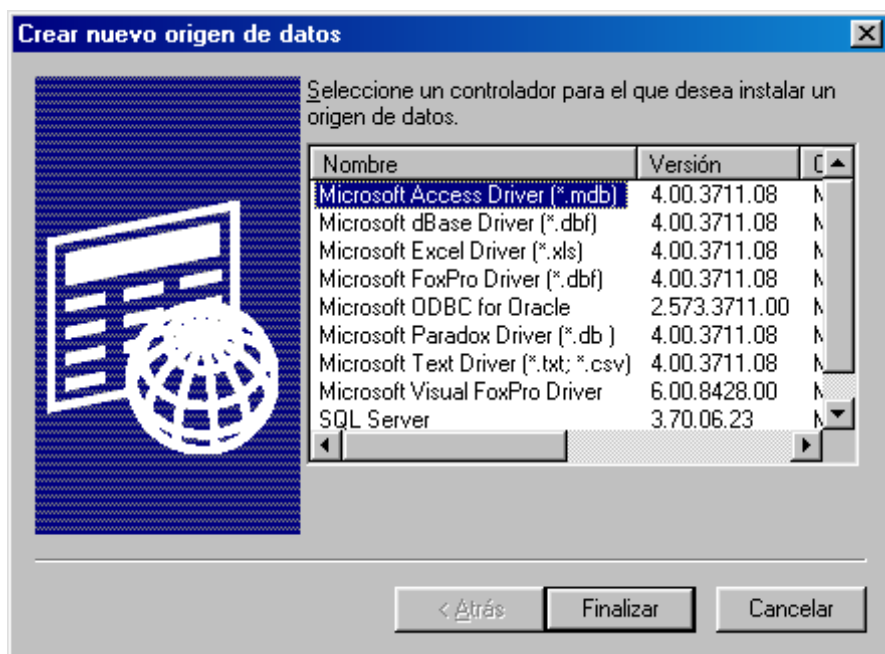


Figura A.5: Selección del controlador

Una vez elegido el controlador para el que se desea agregar un **DSN**, se

⁸Por ejemplo, para un origen de datos en un fichero de texto sólo hace falta la ruta del fichero, mientras que para acceder a un servidor remoto puede hacer falta la dirección IP, el nombre de usuario, la contraseña, etc.

pulsa el botón **Finalizar** y aparece un cuadro de diálogo de configuración específico del controlador. En la Figura A.6 se muestra el cuadro de diálogo que posee el controlador para Microsoft Access.

El cuadro de diálogo **Instalación de ODBC para Microsoft Access** (Figura A.6) posee muchas opciones, pero sólo hace falta pulsar un par de botones y rellenar un par de campos para crear un **DSN**. El resto de opciones permite un control mayor de la conexión **ODBC**, pero sólo es conveniente modificarlas si se posee conocimientos avanzados.

El cuadro de texto **Nombre del origen de datos** es el campo más importante: en él indicamos el nombre (o alias) que se va a emplear en los programas para acceder a la base de datos. El siguiente cuadro de texto, **Descripción**, es opcional. A continuación, podemos elegir la base de datos si ya existe o crear una nueva.

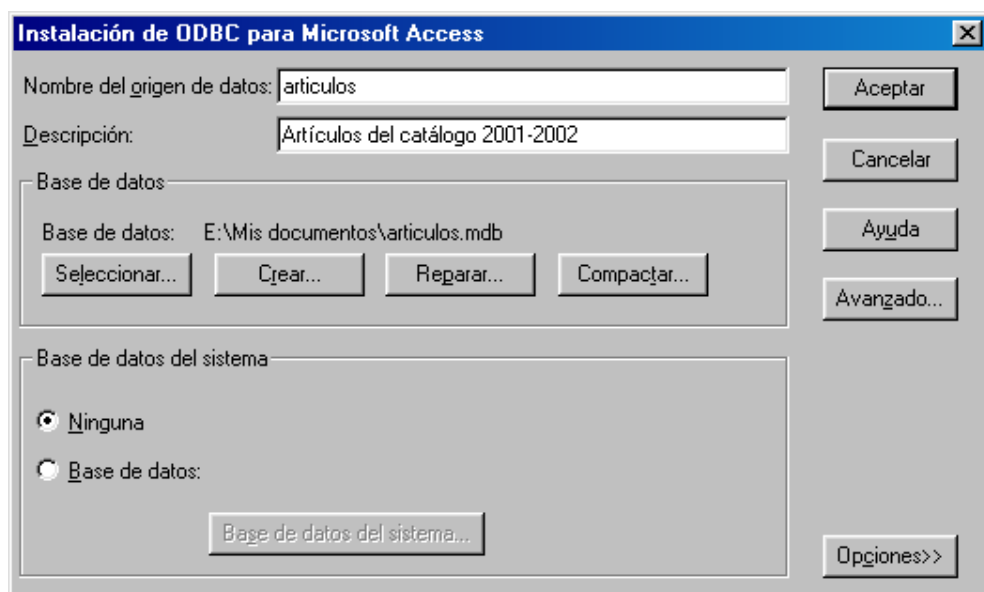


Figura A.6: Creación de un DSN para Microsoft Access

Si ya tenemos creada la base de datos, pulsamos el botón **Seleccionar...** y se abre el cuadro de diálogo de la Figura A.7. Desde aquí podemos localizar el fichero **.mdb** que contiene la base de datos en formato **Microsoft Access** que

queremos emplear.

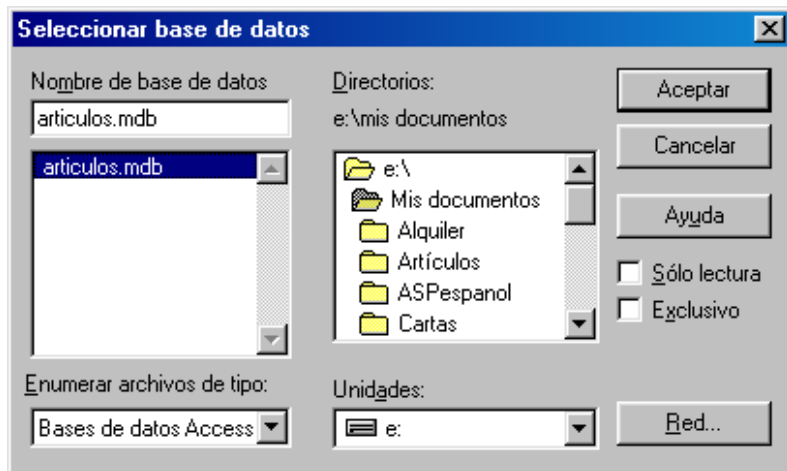


Figura A.7: Seleccionar una base de datos

Como se puede ver en la Figura A.7, el cuadro de diálogo posee un botón etiquetado **Red...** Este botón permite seleccionar una base de datos situada en otro equipo. Esta situación resulta muy interesante cuando se publica información en Internet, ya que al situar la base de datos en un equipo distinto al que ofrece el servicio web se está añadiendo un nivel de seguridad adicional.

Si no tenemos creada la base de datos, pulsamos el botón **Crear...** y aparece el cuadro de diálogo de la Figura A.8. En esta ventana tenemos que indicar las características de la base de datos que deseamos crear:

- **Nombre de base de:** indicamos el nombre que queremos asignar a la base de datos (nombre del fichero).
- **Directorios:** seleccionamos el directorio donde queremos crear la base de datos.
- **Local:** escogemos el idioma que emplea nuestro sistema operativo. Para evitar posibles problemas, escogemos **Español moderno** o **Español tradicional**.

- **Formato:** indicamos la versión de Microsoft Access que queremos que posea la base de datos. Si queremos que la versión sea la misma que la empleada en Microsoft Access 97, seleccionamos **Versión 3.x**.

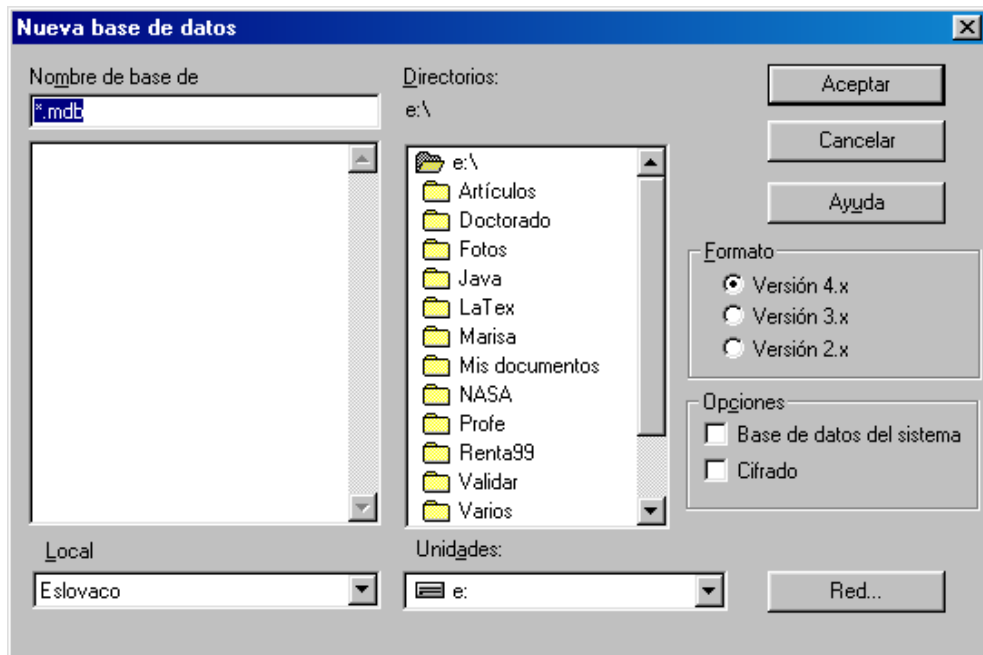


Figura A.8: Crear una base de datos

Al pulsar el botón **Aceptar** se crea una base de datos de Microsoft Access vacía (en blanco): sin tablas, sin consultas, sin formularios, etc.

Una vez creado un **DSN**, cualquier programa que necesite acceder a la base de datos sólo necesita conocer el nombre que posee el **DSN**. La base de datos se puede mover de sitio (incluso a otro equipo accesible a través de la red) o se puede convertir a otro formato (a otro **SGBD**) y en ambos casos, los programas que acceden a ella no se tienen que modificar.

Bibliografía

- [1] Eduardo Parra Murga. *Diccionario de Internet*. PC World, IDG Communications, Madrid.
- [2] Eric Ladd, Jim O'Donnell. *Platinum Edition Using HTML 3.2, Java 1.1, and CGI*. Que, Macmillan Computer Publishing, 1996.
- [3] Eugene Eric Kim. *CGI Programming Unleashed*. Sams.net Publishing, 1996.
- [4] H. M. Deitel, P. J. Deitel, T. R. Nieto. *Internet and World Wide Web. How to program*. Prentice Hall, New Jersey, 2000.
- [5] John December, Mark Ginsburg. *HTML 3.2 and CGI. Professional Reference Edition*. Sams.net Publishing, 1996.
- [6] Krishna Sankar. *Internet Explorer and ActiveX Companion*. Que, Macmillan Computer Publishing, 1997.
- [7] Molly Holzschlag. *Special Edition Using HTML 4*. Que, Macmillan Computer Publishing, 1999.
- [8] Sergio Ríos Aguilar. *Lenguajes HTML, Java y CGI. El diseño de páginas Web para Internet a su alcance*. Abeto Editorial, Madrid, 1996.

Índice alfabético

+, 23, 58
.htx, 69, 71, 74, 77
.idc, 69, 71, 74
.inc, 61
.shtm, 49
.shtml, 49
.stm, 49
<!-- -->, 50
<HEAD>, 18
<% ... %>, 73, 77
<%begindetail%>, 73, 77, 79
<%else%>, 77
<%enddetail%>, 73, 77
<%endif%>, 77
<%idc.%>, 78
<%if ... then ... else ... %>, 77,
79
<%if%>, 77
=, 23
?, 21, 24, 25, 28, 58
#, 51
%, 23, 24, 79
%%, 79
&, 23

A

Active Server Pages, *véase* ASP
AHTML, 8
Altavista, 5
American Standard Code for Information Interchange, *véase* ASCII
Apache, 5, 6, 49
API, XIII, 87, 88, 90, 91
applets, 10
Application Program Interface, *véase* API
ASCII, XIII, 11, 23
ASP, XIII, 9, 10, 58, 68

B

Bloc de notas, 6, 49, 70

C

C, XIV, 6–8, 13, 14, 19, 29, 32, 41,
49, 54, 62, 63
C++, 6, 8, 49
códigos HTTP, 16
cabecera HTTP, 11, 13

CGI, XIII, 2–11, 13–15, 17–19, 21–29, 31, 32, 36, 37, 39, 40, 42–44, 49, 54, 57, 58
 cgi, 58
 CGI-HTML, 8
 cgi-lib, 8
 cmd, 58, 59
 codificación URL, 23
 ColdFusion, 10
 comentario en HTML, 50
 Common Gateway Interface, *véase* CGI
 config, 51
 Content-Type, 75
 Content-type, 13
 CONTENT_LENGTH, 25, 26, 28, 40
 CONTENT_TYPE, 25, 26, 28
 CurrentRecord, 79

D

Data Source Name, *véase* DSN
 Datasource, 72, 74
 Deerfield.com, 44
 DefaultParameters, 75
 DLL, XIV, 69
 DNS, XIV, 27
 Domain Name System, *véase* DNS
 DSN, XIV, 71, 72, 74, 92, 94, 95, 97
 Dynamic Link Library, *véase* DLL

E

echo, 51, 54
 encabezado HTTP, *véase* cabecera HTTP

errmsg, 52
 exe, 58
 exec, 51, 58
 Expires, 75
 Extensible HyperText Markup Language, *véase* XHTML
 Extensible Markup Language, *véase* XML

F

flastmod, 51, 59
 Fortran, 7
 fsize, 51, 61
 fuentes de datos, 92

G

GET, 17, 24, 25, 28, 29
 getenv(), 29
 Google, 5

H

HEAD, 25, 29
 HTML, XIV, 3–6, 10, 11, 13, 14, 18, 19, 26, 42, 48–51, 54, 58, 59, 61–63, 69–73, 75–77, 79–81, 84
 HTTP, XIV, 3, 10, 11, 17, 25, 27, 39, 40, 44, 69
 HyperText Markup Language, *véase* HTML
 HyperText Transfer Protocol, *véase* HTTP

I

IBM, xvi
IBM OS/2, 90
IDC, xv, 68–72, 74, 75, 77, 79, 80
include, 51, 61
Informix, 70
International Organization for Standards, *véase* ISO
Internet, 68
Internet Database Connector, *véase* IDC
Internet Protocol, *véase* IP
Internet Server Application Program Interface, *véase* ISAPI
IP, xv, 5, 9, 27, 29
ISAPI, xv, 58, 69
ISINDEX, 18, 21
ISO, xv, 28

J

Java, xv
Java Server Pages, *véase* JSP
joe, 6, 49, 70
JSP, xv, 9, 10, 68

L

Linux, 6, 49, 70
Location, 14

M

MaxFieldSize, 75
MaxRecords, 76, 79

Microsoft, xiii, xv, 37, 44, 68, 90–92
Microsoft Access, 70, 72, 94, 95, 97
Microsoft Internet Explorer, 5, 6, 14, 15, 28, 29, 49, 70, 73, 82
Microsoft Internet Information Server, xv, 5, 6, 37, 49, 68, 70
Microsoft Personal Web Server, 6, 29, 37, 49, 50, 58, 70, 71
Microsoft SQL Server, 70
Microsoft Visual Basic, 44
Microsoft Windows, xiv, 6, 29, 44, 49, 70, 88, 92
Microsoft Word, 11
middleware, 88
MIME, xv, 11, 13, 28, 44, 75
MS-DOS, 19
Multipurpose Internet Mail Extensions, *véase* MIME

N

Netscape Communicator, 5, 6, 14, 28, 49, 70
nph-, 17

O

O'Reilly & Associates, 44
O'Reilly WebSite Professional, 44
ODBC, xv, 68–70, 74, 77, 88, 90–92, 94, 95
ODBCConnection, 76
ODBCOptions, 77
Open Database Connectivity, *véase* ODBC

operadores de IDC, 78
 orígenes de datos, 92

P

Pascal, 6, 7, 49
 Password, 76
 PATH_INFO, 17, 26, 28
 Perl, XIV, 6–8, 43, 49
 PHP, 9, 10
 plug-ins, 11
 POST, 17, 25, 28, 29, 40, 45
 Python, 7

Q

QUERY_STRING, 17, 18, 24–26,
 28

R

Request for Comments, *véase* RFC
 REQUEST_METHOD, 25, 26, 29
 RequiredParameters, 76
 RFC, XVI

S

script, 58
 sendmail, 9
 Server Side Include, *véase* SSI
 servlets, 10
 SGBD, XVI, 3, 87, 88, 90–92, 97
 SGML, XVI
 Sistema Gestor de Bases de Datos,
véase SGBD
 sizefmt, 52, 54, 61

SQL, XVI, 69, 71, 72, 74, 75, 79, 80,
 82, 85–87, 91

SQLStatement, 72, 74, 75

SSI, XVI, 42, 48–52, 54, 56–58, 62

Standard Generalized Markup Lan-
 guage, *véase* SGML

Status, 15

Structured Query Language, *véase*
 SQL

Sun Microsystems, XV, 68

syntax highlight, 7, 49

T

TCL, 7

TCP/IP, XVI, 88

Template, 72, 74, 75

timefmt, 52, 54, 59

tipos MIME, *véase* MIME

Translationfile, 76

Transmission Control Protocol/Internet
 Protocol, *véase* TCP/IP

U

Universal Resource Locator, *véase*
 URL

Unix, 6, 9, 37, 43, 49, 54, 90

URL, XVI, 3, 17, 18, 21, 22, 24–28,
 69

Username, 77

V

variable de entorno, 54

variable de entorno CGI, 26, 54

variable de entorno SSI, 54

variables específicas de la petición,
28
variables específicas del cliente, 27
variables específicas del servidor, 26
VBScript, XIII

W

W3C, XVI
WinCGI, 44
World Wide Web, *véase* WWW
World Wide Web Consortium, *véa-
se* W3C
WWW, XVII

X

XHTML, XVII
XML, XVII

Y

Yahoo, 5