Este texto está pensado tanto para aquellos que desean pasar de la tecnología ASP al nuevo entorno de desarrollo de aplicaciones Web que propone Microsoft, es decir, ASP .NET, como para todos aquellos que desconocen la tecnología ASP (Active Server Pages) y desean introducirse en el mundo del desarrollo de aplicaciones de negocio en Internet dentro del nuevo entorno del .NET Framework de Microsoft.

Se abordan diversos temas relacionados con la tecnología ASP .NET como pueden ser los controles Web, el lenguaje C#, Visual Studio .NET, Internet Information Server 5, desarrollo de controles ASP .NET y componentes .NET, ADO .NET, construcción y utilización de servicios Web, etc.

Se requiere conocer los fundamentos de Internet/Intranet, estar familiarizado con la

navegación por la web, conocer HTML y conocer el sistema operativo Windows a nivel de usuario, así como algunos conocimientos de programación orientada a objetos.

DESARROLLO DE APLICACIONES PARA INTERNET CON ASP .NET

ANGEL ESTEBAN

GRUPO EIDOS

ISBN 84-88457-52-9

ASP .NET

Angel Esteban Núñez

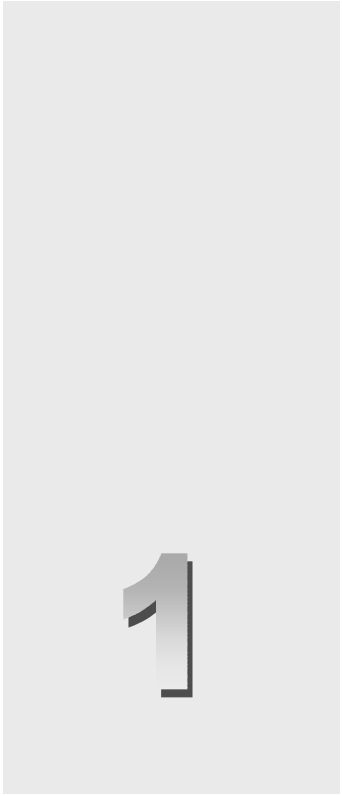
Responsable editorial
Paco Marín (fmarin@eidos.es)

Coordinación de la edición
Antonio Quirós (aquiros@eidos.es)

Autoedición
Magdalena Marín (mmarin@eidos.es)
Angel Esteban (aesteban@eidos.es)

Grupo EIDOS
C/ Téllez 30 Oficina 2
28007-Madrid (España)
Tel: 91 5013234 Fax: 91 (34) 5017824
www.grupoeidos.com/www.eidos.es
www.LaLibreriaDigital.com

Índice



Introducción al .NET Framework

Introducción

En este primer capítulo se va a tratar de exponer una serie de generalidades sobre la plataforma .NET, denominada también .NET Framework. Por lo tanto en este capítulo no vamos a ver ningún aspecto particular de ASP .NET sino que veremos conceptos globales de .NET. En el siguiente capítulo si que comenzaremos a tratar temas específicos de ASP .NET.

Algo está cambiando

El mundo del desarrollo de aplicaciones se encuentra sumido en una nueva etapa de transformación y evolución hacia nuevos esquemas de trabajo.

Los factores determinantes de dicho cambio los podemos encontrar en la necesidad de utilizar Internet como vehículo de intercambio por parte de diversos sectores de la economía.

Las empresas requieren establecer relaciones comerciales más dinámicas con sus clientes, de modo que su volumen de negocio se incremente a través del canal de ventas electrónico (el denominado comercio electrónico o e-commerce). Por otro lado también necesitan unas relaciones empresariales más ágiles en este mismo marco del ciberespacio (el llamado B2B o Bussiness to bussiness).

Aparte de todos estos elementos, nos encontramos con que el usuario de este medio, Internet, dispone de dispositivos cada vez más sofisticados para desplazarse por la Red, no sólo el PC; y además, exige que todos ellos permitan un acceso rápido y sencillo, a múltiples aplicaciones simultáneamente, con un mayor grado de interacción, y obteniendo información de un amplio conjunto de fuentes de datos; todo

esto, naturalmente, sin los tradicionales esfuerzos de configuración que requieren algunas aplicaciones.

Con el paso del tiempo, Internet se ha convertido en el principal entorno de trabajo para el desarrollo de aplicaciones que gestionan información, haciendo que su alcance sea mayor que ningún otro medio hasta el momento. Baste pensar, que con un simple dispositivo que tenga acceso a Internet (léase un PC) y un programa navegador, es posible acceder a infinidad de sitios web basados en este paradigma.

Sin embargo, actualmente, la comunicación entre servidores es complicada (sobre todo si residen en plataformas distintas), y la integración de aplicaciones en dispositivos que no sean el típico PC, es limitada con las herramientas disponibles hasta la fecha. Pero no desesperemos, nos encontramos en un momento crucial, en el que todos esos inconvenientes pueden ser salvados gracias a un nuevo avance tecnológico: Microsoft .NET.

¿Qué es .NET?

.NET es toda una nueva arquitectura tecnológica, desarrollada por Microsoft para la creación y distribución del software como un servicio. Esto quiere decir, que mediante las herramientas de desarrollo proporcionadas por esta nueva tecnología, los programadores podrán crear aplicaciones basadas en servicios para la web.

Las características principales que conforman .NET son las siguientes:

- La plataforma .NET Framework, que proporciona la infraestructura para crear aplicaciones y el entorno de ejecución para las mismas.

- Los productos de Microsoft enfocados hacia .NET, entre los que se encuentran Windows .NET Server, como sistema operativo que incluirá de forma nativa la plataforma .NET Framework; Visual Studio .NET, como herramienta integrada para el desarrollo de aplicaciones; Office .NET; b.Central para .NET, etc.

- Servicios para .NET desarrollados por terceros fabricantes, que podrán ser utilizados por otras aplicaciones que se ejecuten en Internet.

Existen adicionalmente un conjunto de productos, que bajo la etiqueta de Servidores Empresariales para .NET (.NET Enterprise Servers) se incluyen dentro de la estrategia .NET. Entre estos productos podemos encontrar a SQL Server 2000, BizTalk Server, Commerce Server 2000, etc. Sin embargo, hemos de hacer una puntualización importante: estos productos no están basados en .NET Framework, pueden funcionar dentro del entorno de ejecución de .NET Framework, pero el único producto actualmente desarrollado bajo el nuevo entorno es Visual Studio .NET.

Gracias a .NET y a su modelo de desarrollo basado en servicios, se flexibiliza y enriquece el modo en el que hasta ahora se construían aplicaciones para Internet. La idea que subyace bajo esta tecnología, es la de poblar Internet con un extenso número de aplicaciones, que basadas en servicios para la web (Web Services), formen un marco de intercambio global, gracias a que dichos servicios están fundamentados en los estándares SOAP y XML, para el intercambio de información.

En este sentido, un programador puede crear Web Services para que sean utilizados por sus propias aplicaciones a modo de componentes (pero de una forma mucho más avanzada que empleando el modelo COM clásico), siguiendo una estructura de programación ya conocida. Ver Figura 1.

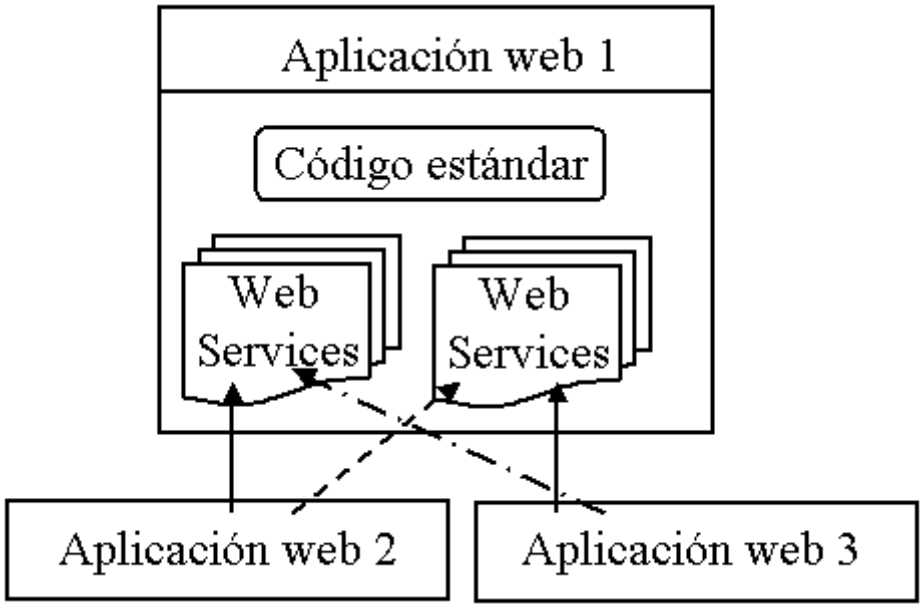


Figura 1. Esquema de funcionamiento de aplicación web incluyendo Web Services.

Sin embargo, los Web Services traen de la mano un nuevo modelo de distribución del software; el basado en el desarrollo y publicación de Web Services y en la suscripción a los mismos por parte de otras aplicaciones, potenciales usuarios de tales servicios. Ver Figura 2.

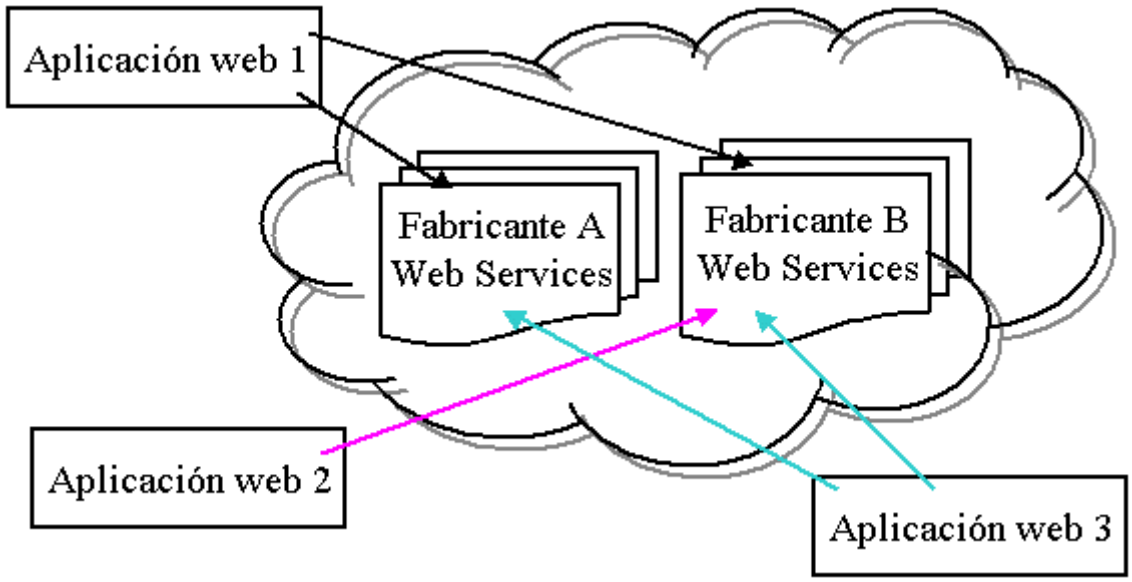


Figura 2. Interacción de aplicaciones con Web Services publicados en Internet.

Los fabricantes de software, pueden de esta manera, dedicarse a la creación de servicios web y a su alquiler. Nace de esta manera, la figura del proveedor de servicios web.

Dado el esquema anterior, el programador puede construir sus aplicaciones a base de Web Services, reduciendo significativamente el tiempo y esfuerzo en el desarrollo.

.NET Framework

.NET Framework constituye la plataforma y elemento principal sobre el que se asienta Microsoft .NET. De cara al programador, es la pieza fundamental de todo este nuevo modelo de trabajo, ya que proporciona las herramientas y servicios que necesitará en su labor habitual de desarrollo.

.NET Framework permite el desarrollo de aplicaciones a través del uso de un conjunto de herramientas y servicios que proporciona, y que pueden agruparse en tres bloques principales: el Entorno de Ejecución Común o Common Language Runtime (CLR a partir de ahora); la jerarquía de clases básicas de la plataforma o .NET Framework Base Classes; y el motor de generación de interfaz de usuario, que permite crear interfaces para la web o para el tradicional entorno Windows, así como servicios para ambos entornos operativos. La Figura 3 muestra un diagrama con la distribución de elementos dentro del entorno de .NET Framework.

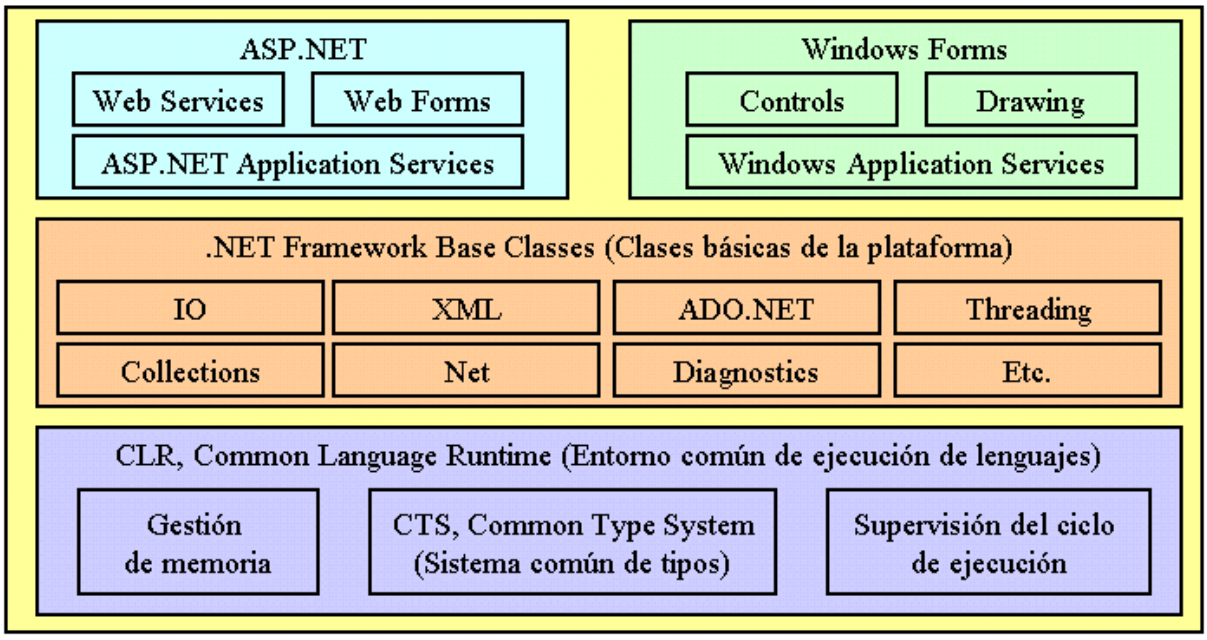


Figura 3. Esquema de componentes dentro de la plataforma .NET Framework.

En la base del entorno de ejecución, se encuentra el CLR, que constituye el núcleo de .NET Framework, encargándose de la gestión del código en cuanto a su carga, ejecución, manipulación de memoria, seguridad, etc.

En el nivel intermedio, se sitúa la jerarquía de clases básicas del entorno de ejecución, que constituyen un sólido API de servicios a disposición del programador, para multitud de tareas como, gestión del sistema de ficheros, manipulación multihebra, acceso a datos, etc.

Finalmente, en el nivel superior, encontramos las clases que permiten el diseño del interfaz de usuario de nuestras aplicaciones. Si necesitamos desarrollar aplicaciones para Internet, utilizaremos ASP.NET, que nos provee de todo lo necesario para crear aplicaciones para la Red: web forms, web services, etc.

Y no piense el programador tradicional de Windows, que todo en .NET Framework es programación para Internet. La plataforma no se ha olvidado de este colectivo de programadores, que necesitan desarrollar programas para este sistema operativo, y pone a su disposición los denominados Windows Forms, la nueva generación de formularios, con características avanzadas y muy superiores a las del motor de generación de formularios de VB6.

Adicionalmente, existe la posibilidad de que necesitemos servicios del sistema que no requieran interfaz de usuario en absoluto. Este aspecto también está contemplado por la plataforma, permitiéndonos, por ejemplo, la creación de servicios para Windows 2000 y NT.

El CLR, Common Language Runtime

El Entorno de Ejecución Común de Lenguajes o CLR (Common Language Runtime), representa el alma de .NET Framework y es el encargado de la ejecución del código de las aplicaciones.

A continuación se enumeran algunas de las características de este componente de la plataforma:

- Proporciona un desarrollo de aplicaciones más sencillo y rápido gracias a que gran parte de las funcionalidades que tradicionalmente debía de crear el programador, vienen implementadas en el entorno de ejecución.
- Administra el código en tiempo de ejecución, en todo lo referente a su carga, disposición en memoria, recuperación de memoria no utilizada a través de un recolector de memoria, etc.
- Implementa características de gestión a bajo nivel (administración de memoria, por ejemplo), que en ciertos lenguajes, eran labor del programador.
- Proporciona un sistema común de tipos para todos los lenguajes del entorno.
- Gestiona la seguridad del código que es ejecutado.
- Dispone de un diseño abierto a lenguajes y herramientas de desarrollo creadas por terceros fabricantes.
- Facilita enormemente la distribución e instalación de aplicaciones, ya que en teoría, es posible instalar una aplicación simplemente copiando los ficheros que la componen en uno de los directorios del equipo en el que se vaya a ejecutar, eliminando los temibles conflictos de versiones entre librerías, problema conocido también con el nombre de *Infierno de las DLL* o *DLL Hell*.

La Figura 4 muestra un esquema de la organización interna del CLR.

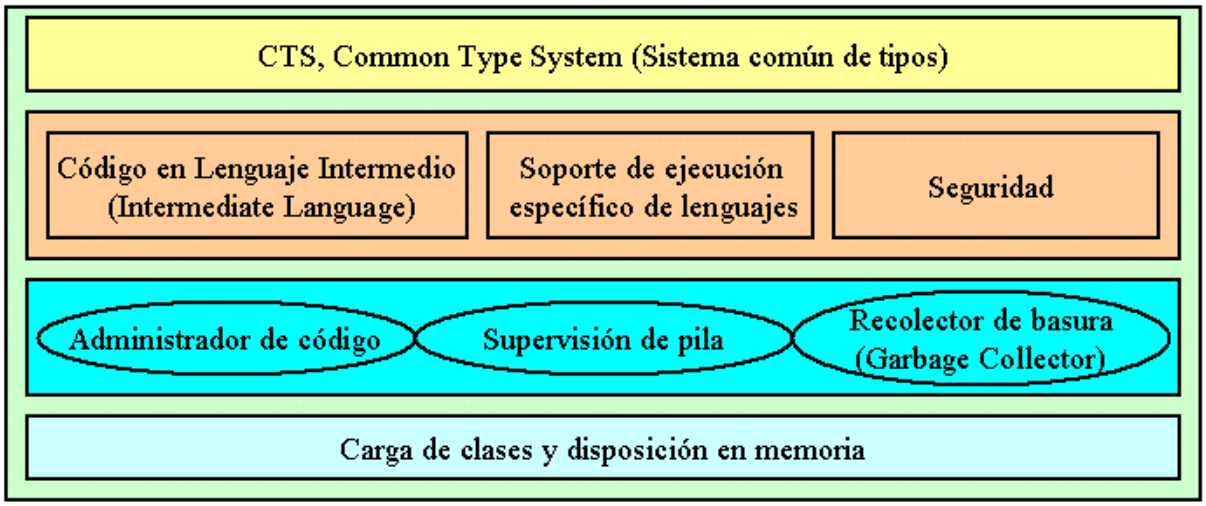


Figura 4. Esquema de elementos dentro del CLR.

En los siguientes apartados, haremos una descripción de los elementos y características más destacables del CLR, que permitan al lector obtener una visión global del mismo, y de las ventajas de escribir programas para este entorno de ejecución.

El CTS, Common Type System

El Sistema Común de Tipos o CTS (Common Type System), es el mecanismo del CLR que permite definir el modo en que los tipos serán creados y manipulados por el entorno de ejecución de .NET Framework.

Entre las funcionalidades que comprende, podemos destacar la integración de código escrito en diferentes lenguajes; optimización del código en ejecución; un modelo de tipos orientado a objeto, que soporta múltiples lenguajes; y una serie de normas que aseguran la intercomunicación entre objetos.

El sistema común de tipos (CTS a partir de ahora), como hemos indicado, permite definir o diseñar el modo en cómo el código de la aplicación será ejecutado, pero no se encarga directamente de su ejecución; dicho de otro modo, el CTS le dice al CLR cómo quiere que sea ejecutado el código.

Un ejemplo de las ventajas del CTS, consiste en que desde un lenguaje como VB.NET, podemos instanciar un objeto de una clase escrita en otro lenguaje como C#; y al hacer una llamada a uno de los métodos del objeto, no es necesario realizar conversiones de tipos en los parámetros del método, funcionando todo de forma transparente.

Metadata (metadatos)

Durante el diseño de .NET Framework, se hizo evidente que una aplicación necesitaba, además de su propio código ejecutable, información adicional sobre la propia aplicación, que pudiera ser utilizada por el entorno de ejecución para funcionalidades diversas.

Para resolver este problema, se optó por incluir toda esta información complementaria dentro de la propia aplicación. A la información que va incluida en la aplicación pero que no forma parte del código ejecutable se le denomina metadatos, y con esta técnica obtenemos aplicaciones o componentes auto-descritos.

Los metadatos son creados por el compilador del lenguaje utilizado en cada caso y grabados dentro del fichero resultante (EXE o DLL) en formato binario, siendo el CLR el encargado de recuperarlos en el momento que los necesite.

Algunos de los datos proporcionados por los metadatos de una aplicación son la descripción del ensamblado (trataremos los ensamblados posteriormente) junto a su versión, clave y tipos que lo componen (clases, interfaces, etc.).

Soporte multi-lenguaje

Uno de los puntos clave del CLR es que está diseñado para soportar múltiples lenguajes, permitiendo así unos elevados niveles de integración entre los mismos. Con tal motivo, .NET Framework proporciona los siguientes lenguajes con sus correspondientes compiladores para la escritura de aplicaciones:

- VB.NET.

- C#.
- C++ con Extensiones Administradas.
- JScript.NET.

Por integración de lenguajes podemos definir algo tan poderoso como el hecho de escribir una clase en C#, y heredar de dicha clase desde VB.NET. Esto permite formar grupos de trabajo heterogéneos, en los que cada integrante del grupo, puede escribir el código de una aplicación en el lenguaje de su preferencia. Gracias a que el entorno de ejecución es común, y el código compilado no pasa directamente a código ejecutable puro, sino a un código intermedio (lo veremos más adelante), podemos crear nuestros programas en el lenguaje con el que nos sintamos más cómodos en cuanto a sintaxis y prestaciones, por ejemplo VB.NET; con la ventaja de que la velocidad de ejecución será muy parecida a la obtenida habiendo escrito el código en otro lenguaje en principio más rápido como C++ o C#.

El CLS (Common Language Specification)

La integración entre lenguajes mencionada en el anterior apartado, puede llevar a preguntarnos cómo es posible conseguir que lenguajes de distinta naturaleza y sintaxis *se entiendan*.

La respuesta la hallamos en la Especificación Común de Lenguajes o CLS (Common Language Specification), que consiste en un conjunto de características comunes, que deben cumplir todos los lenguajes de la plataforma, para poder integrarse entre sí.

Esto tiene varias finalidades, que describimos a continuación:

- **Independencia del lenguaje**. En muchas ocasiones el programador se ve obligado a escribir el código en un lenguaje que no es de su agrado; la causa de ello es que dicho lenguaje le provee de funcionalidades de las cuales carece su lenguaje preferido. Con .NET, esto no ocurre, puesto que es la propia plataforma la que proporciona la funcionalidad de modo independiente al lenguaje, por lo que podemos escribir nuestras aplicaciones utilizando el lenguaje con el que nos sintamos más cómodos, ya que el resultado será el mismo.

- **Integración entre lenguajes**. Es posible escribir, por ejemplo, una librería de clases en un lenguaje, y utilizarla desde otro lenguaje distinto (siempre que ambos lenguajes cumplan con las normas del CLS). Este concepto no es nuevo, hasta ahora también podíamos escribir una librería en C++ y utilizarla desde VB, pero gracias al CLS, se extiende y se potencia este modo de trabajo, ya que al basarse los lenguajes en un conjunto de reglas comunes, el acceso en el caso antes mencionado, a una librería de clases, se facilita enormemente desde cualquier otro lenguaje creado en base al CLS.

- **Apertura a nuevos lenguajes**. Finalmente, al ser esta, una especificación abierta, es posible incorporar a .NET Framework nuevos lenguajes, aparte de los actualmente disponibles, y no sólo creados por Microsoft, sino por cualquier otro fabricante. Mediante el CLS, un fabricante de software sabe qué requisitos debe observar un nuevo lenguaje que él desarrolle, para poder integrase en el entorno de .NET Framework.

  Terceros fabricantes ya han anunciado en este sentido, su intención de proporcionar nuevos lenguajes para .NET; de esta forma aparecerán progresivamente versiones para esta plataforma de Cobol, Perl, Smalltalk, etc., en una lista en la que actualmente figuran más de veinte lenguajes candidatos.

Ejecución administrada

La ejecución administrada se trata de un conjunto de elementos existentes en .NET Framework, que supervisan el código del programa durante su ejecución dentro del CLR, asegurándose de que el código cumple todos los requisitos para poder hacer uso de los servicios proporcionados por el entorno de ejecución, y beneficiarse de sus ventajas.

Código administrado

El código que escribamos orientado a utilizar todas las cualidades del CLR se denomina código administrado. Por defecto el código escrito en VB.NET, C# y JScript.NET es administrado, con lo que el programador no debe preocuparse en configurar de manera especial su proyecto.

Por el contrario, el código escrito en C++ no es administrado por defecto, lo que significa que el entorno no lo supervisa y no garantiza su fiabilidad al ser ejecutado por el CLR. Si el programador de C++ quiere que su código sea administrado debe utilizar las extensiones administradas que la plataforma proporciona para este lenguaje y activarlas a través de una opción del compilador.

El hecho de que el entorno realice labores de comprobación sobre el código, supone evidentemente, una labor extra que repercute sobre el rendimiento final a la hora de ejecutar el programa. Sin embargo, las pruebas realizadas ofrecen como resultado una pérdida de un 10% en el rendimiento del código administrado con respecto al código no administrado.

Teniendo en cuenta los niveles de seguridad que nos ofrece el código administrado y dado que la velocidad de los procesadores evoluciona, esta pérdida de rendimiento que supone la ejecución administrada posiblemente no sea significativa en un corto plazo de tiempo.

La ejecución de código dentro del CLR

El proceso de ejecución del código en el entorno de .NET Framework, varía notablemente respecto al modo de ejecución tradicional de programas. En este apartado, realizaremos un repaso de los elementos y técnicas que intervienen en dicho proceso, de modo que el lector tenga un conocimiento más detallado de lo que sucede con el código, desde que termina de escribirlo, y hasta el resultado obtenido tras su ejecución.

El IL, Intermediate Language

Durante el proceso de compilación, el código fuente es tomado por el compilador del lenguaje utilizado para su escritura, y convertido, no directamente a código binario, sino a un lenguaje intermedio, que recibe el nombre de Microsoft Intermediate Languaje (MSIL o IL).

Este lenguaje o código intermedio (IL a partir de ahora), generado por el compilador, consiste en un conjunto de instrucciones que son independientes del sistema operativo o procesador en el que vaya a ejecutarse el programa, y que se ocupan de la manipulación de objetos, accesos a memoria, manejo de excepciones, etc.

La Figura 5 muestra un diagrama con el proceso de generación de lenguaje intermedio a partir del código fuente.

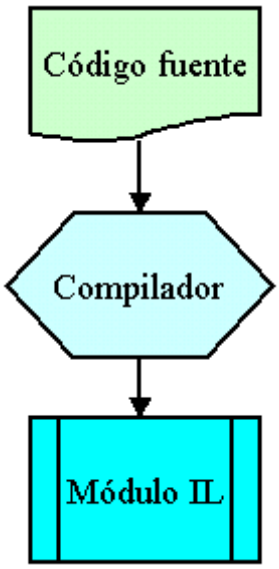


Figura 5. Obtención de lenguaje intermedio a partir de código fuente.

Además del código en IL, el compilador genera también metadatos, que como se ha explicado en un apartado anterior, contienen información adicional, incluida en la propia aplicación, y que serán utilizados por el CLR al ejecutar el programa.

Tanto el código en IL, como los metadatos generados, se guardan en un fichero de tipo EXE o DLL, basado en la especificación tradicional de Microsoft para ficheros con formato de ejecutable transportable (Portable Executable o PE) y objeto común (Common Object File Format o COFF). Con el desarrollo de la tecnología .NET, esta especificación ha sido ampliada para dar cabida, además de código binario, código IL y metadatos.

Ya que el código obtenido en IL es independiente del procesador, en su estado actual no es posible todavía ejecutarlo, debido a que el IL no ha sido diseñado para conocer las instrucciones específicas del procesador en el que se va a ejecutar. La ejecución se lleva a cabo, realizando el paso final de compilación que se detalla seguidamente.

Compilación instantánea del IL y ejecución

Como acabamos de indicar, el código en lenguaje intermedio no es directamente ejecutable, ya que desconoce la arquitectura del procesador sobre la cual va a funcionar.

Antes de realizar la ejecución, el código en IL debe ser convertido a código máquina, utilizando lo que se denomina un compilador instantáneo o compilador Just-In-Time (JIT compiler), que es el encargado de generar el código binario específico para el procesador en el que el programa será ejecutado. La Figura 6 muestra un esquema con el proceso de compilación llevado a cabo por el compilador Just-In-Time (JIT a partir de ahora).

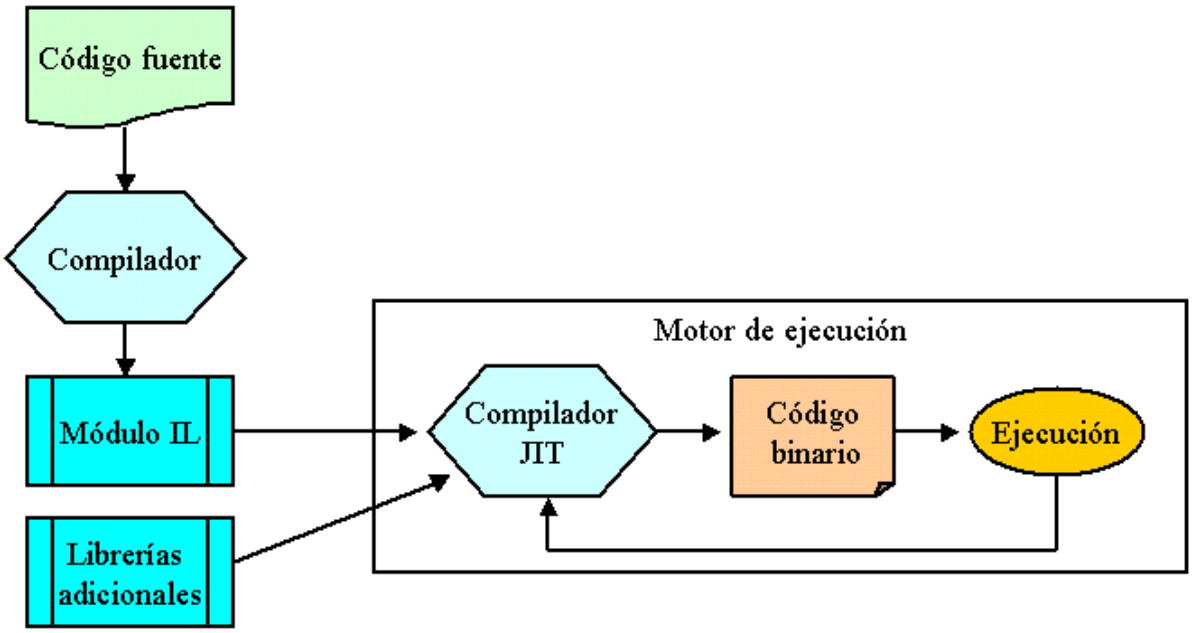


Figura 6. Proceso de compilación instantánea de código IL.

Compilación bajo demanda

Para optimizar la ejecución y mejorar su velocidad, el compilador JIT se basa en el hecho de que es posible que ciertas partes del código que compone la aplicación nunca sean ejecutadas. Por este motivo, al ejecutar la aplicación, no se toma todo su IL y se compila, sino que sólo se compila el código según se va necesitando y se almacena el código máquina resultante de modo que esté accesible en las siguientes llamadas. Veamos con un poco más de detalle este proceso.

Durante la carga de la aplicación, el cargador de código del CLR, toma cada tipo incluido en el programa, y para cada uno de los métodos que componen el tipo, crea y pega una etiqueta indicativa de su estado.

En la primera llamada a un método, se comprueba su estado de compilación a través de la etiqueta de estado; como aún no está compilado, se pasa el control al JIT, que compila el código IL a código máquina. A continuación se modifica la etiqueta de estado, de modo que en las próximas llamadas a ese método, la etiqueta de estado informa que el método ya ha sido compilado, por lo que se evita repetir el proceso de compilación, ejecutando directamente el código máquina creado con anterioridad. Esta técnica optimiza notablemente la velocidad de ejecución. Ver Figura 7.

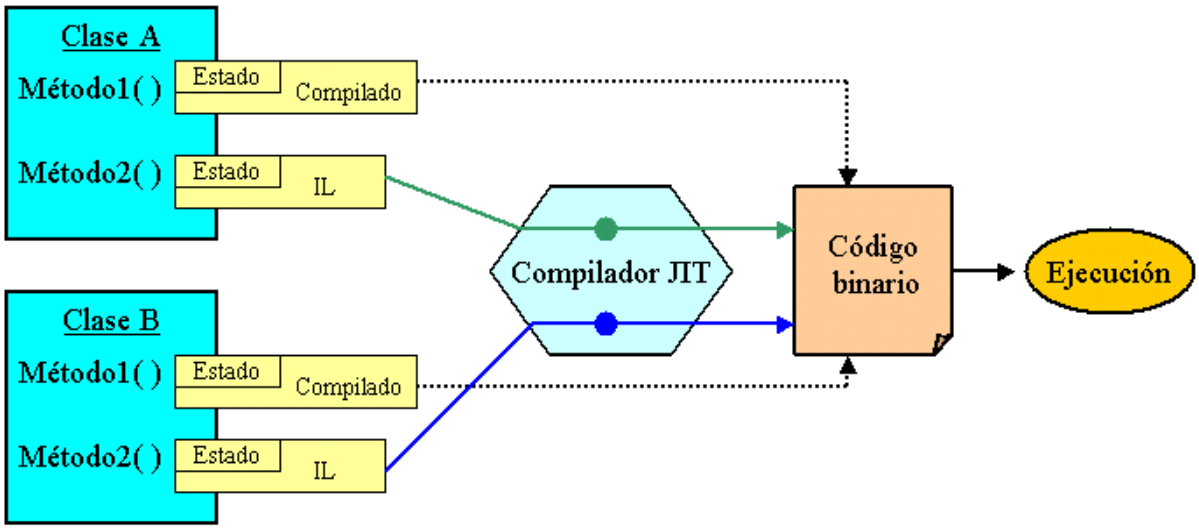


Figura 7. Compilación bajo demanda en .NET Framework.

Independencia de plataforma

Ya que el código máquina ejecutable, es obtenido a través de un compilador JIT, con las instrucciones adecuadas para un procesador determinado, .NET Framework proporciona varios compiladores JIT para cada una de las plataformas que soporta, consiguiendo así que la aplicación, una vez escrita, pueda funcionar en distintos sistemas operativos, y haciendo realidad el objetivo de que nuestro código sea independiente de la plataforma en la que se vaya a ejecutar, actuando .NET Framework como una capa intermedia, que aísla el código del sistema operativo. Ver Figura 8.

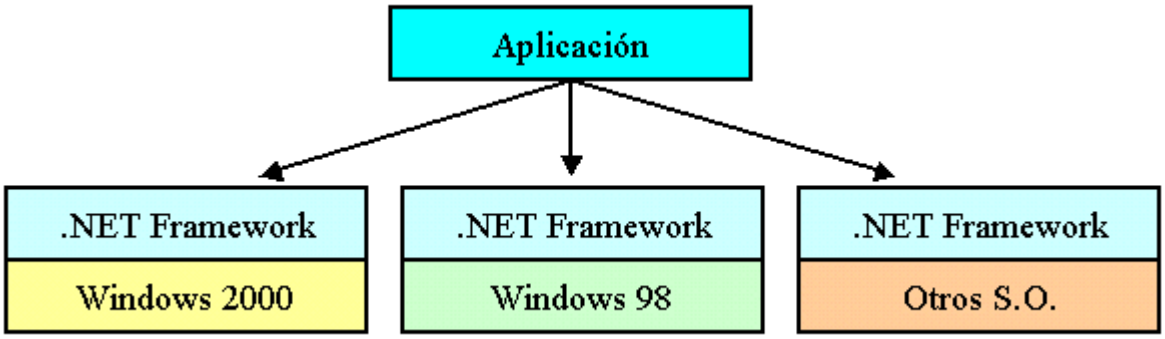


Figura 8. Una misma aplicación se ejecuta en distintos sistemas a través de .NET Framework.

Dominios de aplicación

En .NET Framework se han reforzado las características de seguridad y aislamiento hasta un nivel que permite la ejecución de múltiples aplicaciones en un mismo proceso. A este contexto de ejecución de un programa se le denomina *dominio de aplicación* (Application Domain).

La técnica utilizada tradicionalmente para conseguir aislar las aplicaciones, de modo que no se produzcan colisiones entre las mismas, ha sido a través de procesos. Cada aplicación se carga en un proceso separado, que proporciona el adecuado nivel de aislamiento; de este modo, se evitan posibles conflictos entre las direcciones de memoria utilizadas por cada programa. Sin embargo, esto supone un

gran consumo de recursos, cuando las aplicaciones deben hacer llamadas a otras aplicaciones que residan en procesos distintos, debido a que se debe de realizar un traspaso de procesos entre la aplicación que realiza la llamada y la aplicación destino. Esta técnica ha sido mejorada en .NET, de modo que se consigue tener en un mismo proceso, varias aplicaciones en ejecución.

El código administrado en .NET Framework, para poder ser considerado como seguro, debe pasar en primer lugar una fase de comprobación, efectuada por el CLR, que asegure el hecho de que no realice ningún acceso no permitido a direcciones de memoria u otras operaciones que puedan provocar un fallo del sistema. Una vez superada dicha comprobación, el código es marcado como seguro a nivel de tipos (type-safe), y la aplicación ejecutada.

Superada esta fase de verificación, el programa se ejecutará en un dominio de aplicación, que como hemos comentado antes, consiste en una técnica que permite ejecutar varias aplicaciones en un único proceso, con el mismo nivel de aislamiento que si se estuvieran ejecutando en procesos separados, y la ventaja de eliminar la sobrecarga producida cuando distintas aplicaciones están situadas en diferentes procesos y deben hacerse llamadas entre sí. Cada aplicación se ejecuta en su propio dominio de aplicación

Los dominios de aplicación incrementan notablemente la capacidad de crecimiento de los servidores al ejecutar múltiples aplicaciones en un mismo proceso. La Figura 9 muestra un esquema del proceso de carga y ejecución de aplicaciones en sus correspondientes dominios de aplicación.

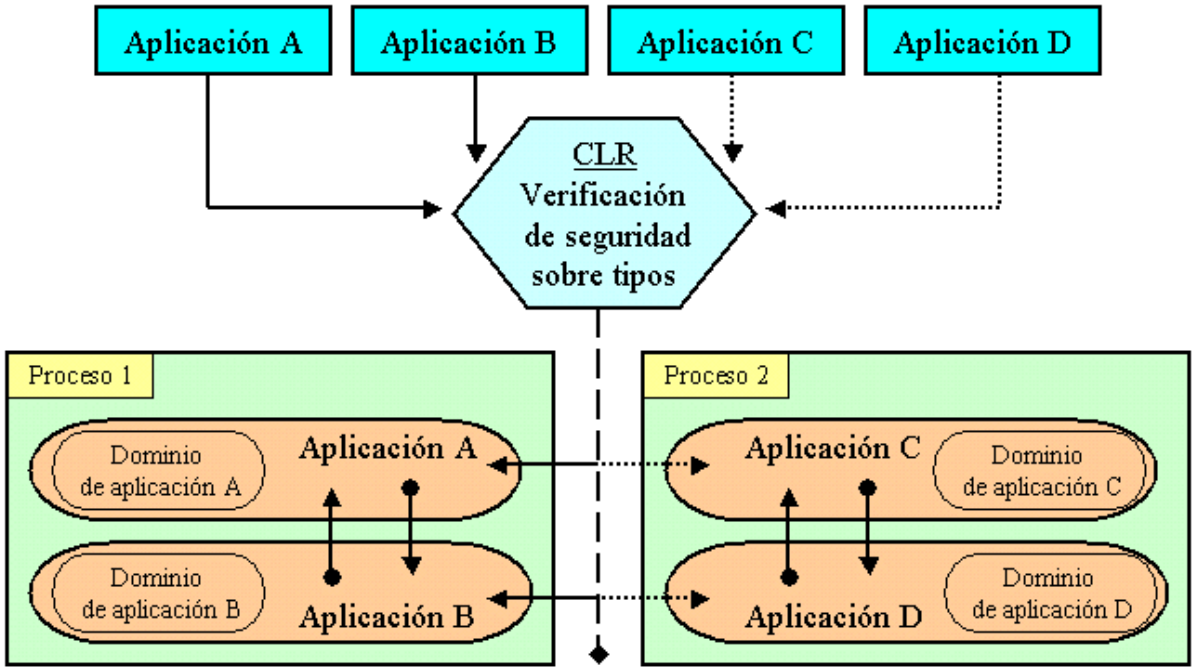


Figura 9. Carga de aplicaciones y creación de dominios de aplicación en procesos.

Servidores de entorno

Un servidor de entorno o Runtime Host es el encargado de ejecutar un dominio de aplicación dentro del CLR, aprovechando las ventajas proporcionadas por este último.

Cuando el CLR se dispone a ejecutar una aplicación, un servidor de entorno crea el entorno de ejecución o shell para dicha aplicación, y lo carga en un proceso; a continuación, crea un dominio de aplicación en ese proceso y por último carga la aplicación en el dominio.

.NET Framework dispone entre otros, de los servidores de entorno relacionados a continuación:

- **ASP.NET**. Carga el entorno en un proceso preparado para gestionarse en la web; creando también, un dominio de aplicación para cada aplicación de Internet ejecutada en un servidor web.
- **Internet Explorer**. Crea un dominio de aplicación por cada sitio web visitado, en el que se ejecutan controles administrados basados en el navegador.
- **Windows Shell**. Crea un dominio de aplicación con interfaz Windows, para cada programa que es ejecutado.

La Figura 10 muestra un diagrama del trabajo realizado por un servidor de entorno.

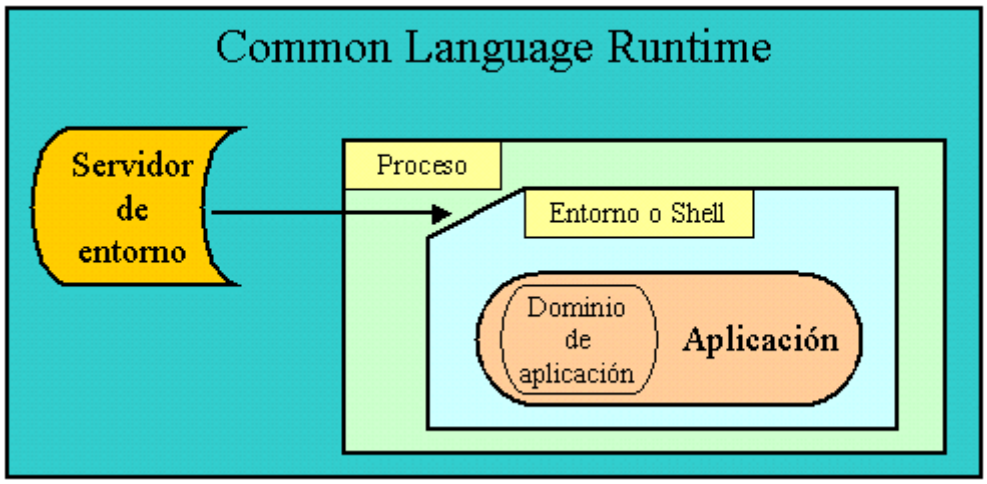


Figura 10. Servidor de entorno creando un entorno para un dominio de aplicación.

Namespaces

Otro de los pilares que forman los cimientos de .NET Framework es el concepto de *espacio de nombres* o *namespaces*.

Un namespace o espacio de nombres, también denominado nombre calificado, es el medio proporcionado por la plataforma para organizar las clases dentro del entorno, agrupándolas de un modo más lógico y jerárquico. Para comprender mejor este concepto veamos un ejemplo:

Estamos desarrollando un conjunto de clases para las operaciones de gestión contable y facturas de una empresa. Podemos ir escribiendo todas las clases y situarlas dentro de una misma aplicación o DLL. Actualmente tenemos dos clases para operaciones contables, denominadas Balance y LibroIVA, y otras dos clases para operaciones con facturas, denominadas Albaran y Factura.

Pero necesitamos añadir una clase más para las facturas que registre el libro de IVA de las facturas emitidas. El nombre más idóneo sería LibroIVA, pero ya está siendo utilizado, así que para evitar problemas de duplicidad de nombres, debemos elegir otro que puede no se ajuste a definir la funcionalidad de la clase.

Mediante el uso de espacios de nombre este problema sería solucionado, con el añadido de poder organizar mejor cada clase, asignándole un nombre jerárquico para la funcionalidad que desempeña.

Para ello, deberíamos crear un namespace con el nombre Gestion, que contuviera otros dos namespaces llamados Contabilidad y Facturación, para finalmente incluir en cada uno de ellos las clases correspondientes. La Figura 11 muestra un diagrama organizativo de las clases de este ejemplo utilizando espacios de nombre.

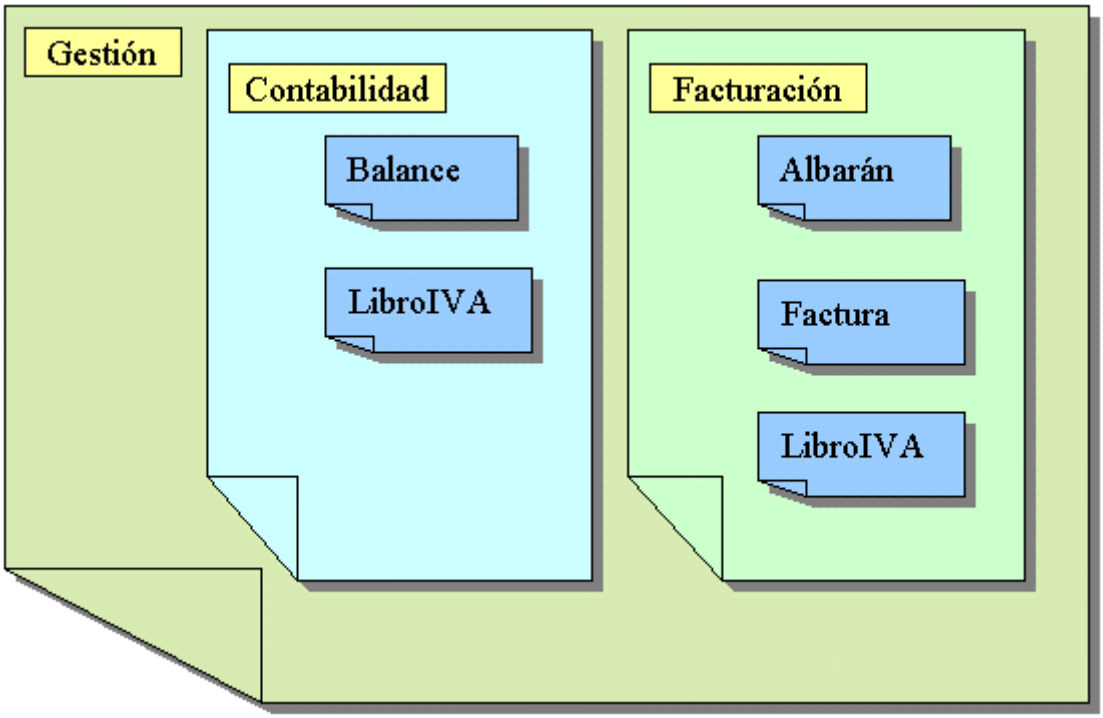


Figura 11. Estructura de un namespace con namespaces contenidos y clases dependientes.

Para acceder desde el código de una aplicación, a una clase contenida dentro de un espacio de nombre, debemos indicarlo en la aplicación realizando una operación que se denomina *Importar*

La convención sintáctica para hacer referencia a una clase contenida en un espacio de nombre es el espacio de nombre y la clase separados por un punto. En el caso de acceder a una clase que se encuentra con varios espacios de nombre de profundidad, especificaremos dichos espacios de nombre separados por un punto, e igualmente al final, la clase. La inclusión al final del nombre de la clase, depende de si instanciamos directamente el objeto usando la lista de espacios de nombre o importamos dicha lista.

En el caso de instanciar un objeto directamente en el código, escribiremos los espacios de nombre y al final, el nombre de la clase. Si importamos los espacios de nombre, no debemos poner el nombre de la clase, sino que debemos terminar con el espacio de nombres que contiene la clase que necesitamos.

Todas las clases de la plataforma .NET están contenidas dentro de espacios de nombre, por lo que siempre que necesitemos instanciar un objeto, deberemos hacerlo usando la convención de espacios de nombre y puntos explicada anteriormente.

La jerarquía de clases de .NET Framework

El entorno de ejecución integra toda la funcionalidad y servicios necesarios a través de la jerarquía de clases base de la plataforma. La mayor parte de las necesidades básicas del programador están

cubiertas por este amplio conjunto de clases, que permiten dotar a las aplicaciones de todas las características necesarias.

El desarrollador experimentado puede estar preguntándose la necesidad de implementar una nueva jerarquía de clases si las actuales ya cumplen con su cometido. Entre las posibles razones, queremos destacar las siguientes:

- El nuevo sistema de clases está mucho mejor organizado, y provee al programador de una potencia y versatilidad para sus aplicaciones nunca antes lograda en versiones anteriores de Visual Studio.

- Podemos crear una nueva clase, heredando de una clase propia de la plataforma, para extender su funcionalidad.

- Desplazando la funcionalidad de las clases fuera de los lenguajes, y haciéndolas por lo tanto, independientes de los mismos, simplifica el proceso de desarrollo.

Al ser las clases de .NET Framework, comunes a todos los lenguajes, se eliminan las barreras tradicionales que impedían a los programadores abordar ciertos proyectos por el hecho de usar un lenguaje que no disponía de cierta funcionalidad que sí tenía otro lenguaje. Ahora cualquier programador, con independencia del lenguaje que elija, tiene pleno acceso a todas las funcionalidades que le brinda la plataforma .NET.

Ensamblados

Un *ensamblado* o *assembly*, consiste en un conjunto de tipos y recursos, reunidos para formar la unidad más elemental de código que puede ejecutar el entorno de .NET Framework.

De igual forma que los edificios se crean a base de la unión de un conjunto de materiales, dentro de la tecnología .NET, los ensamblados se presentan como los *bloques de construcción* software, que se unen o ensamblan para crear aplicaciones. Una aplicación desarrollada para .NET Framework debe estar compuesta por uno o varios ensamblados, ver Figura 12.

Podemos establecer una analogía entre un ensamblado y una DLL, ya que ambos contienen clases, que se exponen a otras aplicaciones. Por dicho motivo, a un ensamblado también se le da el nombre de *DLL lógica*; el término *DLL* se emplea porque tiene un comportamiento similar al de las DLL's tradicionales, y el término *lógica* porque un ensamblado es un concepto abstracto, ya que se trata de una lista de ficheros que se referencian en tiempo de ejecución, pero que no se compilan para producir un fichero físico, a diferencia de lo que ocurre con las DLL's tradicionales.

Sin embargo, un ensamblado extiende sus funcionalidades a un horizonte mucho más amplio, ya que puede contener otros elementos aparte de clases, como son recursos, imágenes, etc.

Por otro lado, simplifican los tradicionales problemas de instalación y control de versiones sobre los programas, uno de los objetivos de la tecnología .NET, en la que en teoría, para instalar una aplicación, sólo sería necesario copiar los ficheros que la componen en un directorio de la máquina que la vaya a ejecutar.

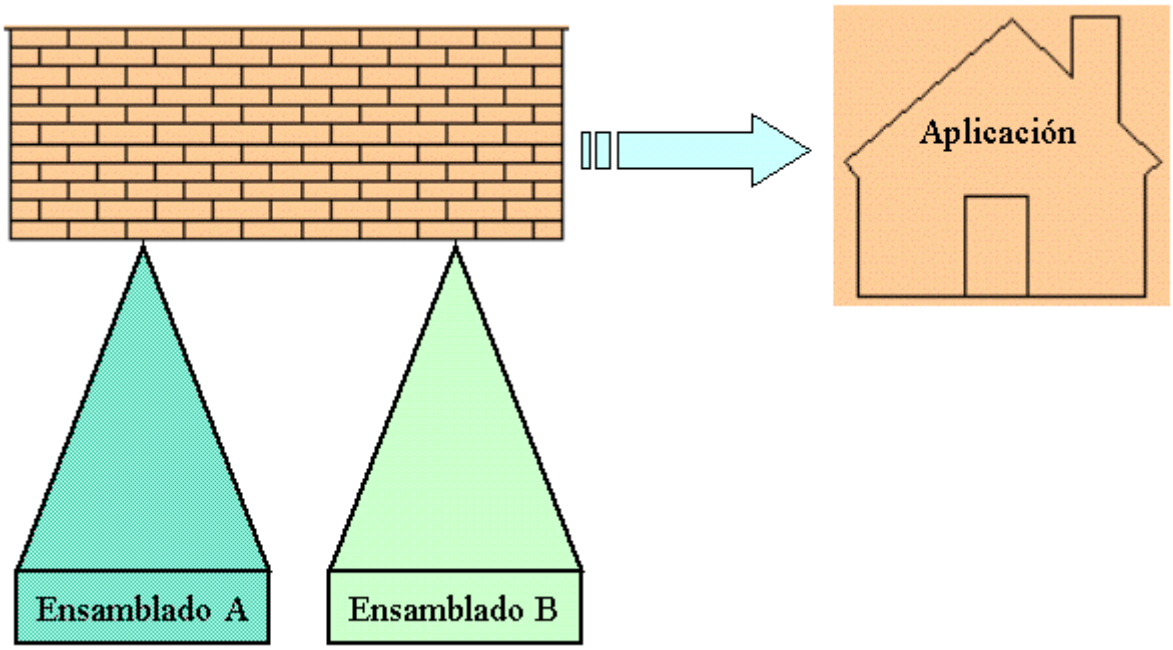


Figura 12. Los ensamblados forman bloques de construcción de aplicaciones.

2

Introducción a ASP .NET

Introducción

Este texto pretende ofrecer una visión detallada de la tecnología ASP .NET perteneciente a la nueva plataforma de Microsoft, denominada .NET Framework o plataforma .NET. ASP .NET es la nueva versión de las páginas activas de servidor, más conocidas como Active Server Pages (ASP).

ASP .NET ofrece toda una nueva forma de desarrollar aplicaciones basadas en el entorno de Internet/Intranet, esta forma nueva de trabajar incluye una serie de novedades que no sólo son las correspondientes a una siguiente versión de ASP, sino que son las que se desprenden también de la nueva plataforma ofrecida por Microsoft, es decir, la plataforma .NET.

Este texto es válido para aquellos lectores que ya conozcan alguna de las versiones anteriores de ASP, y también para aquellos que no conocen ASP o que tienen algunas nociones básicas.

Nuestro objetivo va a ser centrarnos en la tecnología ASP .NET, aunque comentaremos de brevemente, y cuando sea necesario, algunos conceptos generales de la plataforma .NET (.NET Framework), ya que no debemos olvidar que es la plataforma sobre la que se van a ejecutar las aplicaciones Web desarrolladas con la nueva tecnología ASP .NET, es decir, las páginas ASP .NET no se van a ejecutar directamente sobre un sistema operativo determinado, sino que lo van a hacer sobre la nueva plataforma que ofrece Microsoft.

La plataforma .NET ofrece una serie de herramientas y tecnologías necesarias para construir y desarrollar aplicaciones Web, así pues, las páginas ASP .NET se van a ejecutar dentro del entorno de ejecución que nos facilita el .NET Framework. Podríamos decir que ASP .NET es una parte de la plataforma .NET, y es esta parte la que se va a tratar en el presente texto.

ASP .NET se diferencia bastante de ASP 3.0, ya que ofrece un entorno de trabajo distinto al que teníamos en las versiones anteriores de ASP, esto es debido a que ASP .NET no es únicamente una nueva versión de ASP, sino que es el desarrollo de aplicaciones Web dentro del entorno de ejecución ofrecido por la plataforma .NET. Para adelantar algún aspecto y para ilustrar esta afirmación una de las novedades de ASP .NET es que cada página es compilada a un código intermedio para su posterior ejecución.

El código intermedio es una característica común que poseen todas las tecnologías englobadas en la estrategia .NET de Microsoft, a la que lógicamente pertenece ASP .NET. Las páginas ASP .NET cuando reciben la primera petición se compilan automáticamente a un lenguaje intermedio que es conocido como Common Language Runtime, es decir, es un lenguaje común al que compilan todos los lenguajes que utilicemos en nuestras páginas ASP .NET, generando el mismo código, ya sea Visual Basic .NET, C# o JScript. Gracias a esta característica podemos obtener grandes ventajas en lo que a rendimiento en tiempo de ejecución se refiere, ya que la compilación de las páginas sólo se produce en la primera petición que se realiza sobre la página, o bien cuando el código fuente de la misma se ha modificado y necesita por lo tanto actualizarse, además el resultado de esta compilación permanece en caché para poder ser reutilizada.

Pero no adelantemos los acontecimientos, todo esto y más lo veremos más adelante cuando se realice una comparativa de ASP .NET con ASP 3.0.

El lector que ya conozca ASP en cualquiera de sus versiones anteriores no debe desanimarse ni asustarse ante las novedades que se avecinan, ya que ASP .NET conserva gran parte de la filosofía que nos ofrecían las anteriores versiones de la tecnología ASP.

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

El anterior párrafo está extraído completamente del anterior texto dedicado a ASP, es decir, el titulado "Programación para aplicaciones para Internet con ASP 3.0", y es perfectamente aplicable a ASP .NET, pero ahora esta nueva versión de ASP incluida en la plataforma .NET, va más allá presentando un entorno de desarrollo mucho más elaborado y completo, con lo que la afirmación de párrafo anterior se quedaría muy corta, ya que únicamente define uno de los aspectos de las páginas ASP .NET.

El cambio que aporta ASP .NET es sobre todo en la forma de desarrollar aplicaciones Web, es decir, la forma en la que vamos a utilizar los distintos componentes y servicios que nos ofrece ASP .NET, y también cambia la forma de programar, como muestra diré que podemos tratar los eventos de cliente desde código de servidor, pudiendo crear formularios Web al estilo de las aplicaciones típicas de Visual Basic.

Pero volvamos a la definición de ASP .NET, de forma más genérica podemos definir ASP .NET como el entorno de desarrollo incluido en la plataforma .NET de Microsoft, que nos permite desarrollar completas aplicaciones Web que se ejecutarán sobre el entorno ofrecido por el .NET Framework. A lo largo de este texto iremos descubriendo paso a paso las distintas posibilidades (que son bastante amplias) que nos ofrece ASP .NET, como adelanto diremos que una de las novedades más importantes que aporta ASP .NET son los Web Forms (formularios Web). Esta es una nueva característica de ASP .NET que ofrece un nuevo modelo de programación que nos va a permitir generar contenidos dinámicos de una forma más sencilla.

Desde ASP .NET también vamos a poder utilizar el nuevo acceso a datos ofrecido por la plataforma .NET, se trata de ADO (ActiveX Data Objects) .NET, es decir, la versión del modelo de acceso a datos de ADO dentro del .NET Framework.

Y ya que estamos adelantando algunos de los aspectos que veremos dentro de este texto dedicado a ASP .NET, también voy a adelantar que como lenguaje de programación para la tecnología ASP .NET, vamos a utilizar el nuevo lenguaje llamado C# (C Sharp). Este lenguaje se puede considerar como una mezcla entre Java, C y C++, de hecho para aquellos lectores que conozcan alguno de estos lenguajes les será muy sencillo aprender C#, siendo la curva de aprendizaje muy suave, aunque debo advertir que este texto no trata en profundidad el lenguaje C#, sino que lo vamos a utilizar como una herramienta desde ASP .NET.

ASP .NET soporta diversos lenguajes de programación, entre los que se encuentra C#, los otros lenguajes que soporta directamente ASP .NET son Visual Basic .NET (VB7) y JScript, como ya hemos dicho en este texto vamos a emplear C#, ya que es el nuevo lenguaje que ofrece Microsoft y pretende ser el lenguaje están para el .NET Framework. Apunto como dato curioso que el 90% de la herramienta de desarrollo Visual Studio .NET, es decir, la versión que nos ofrece Microsoft de Visual Studio para desarrollar en la plataforma .NET, ha sido desarrollado con el lenguaje C#, además este nuevo lenguaje esta siendo sometido a estandarización por parte de ECMA, la misma entidad de normalización que llevó a cabo la estandarización de JavaScript.

Debido a estas razones todos los ejemplos del texto se encuentra escritos utilizando el lenguaje C#, de todas formas los aspectos específicos de ASP .NET son iguales en todos los lenguajes, es decir, se pueden rescribir los ejemplos en VB .NET o JScript sin demasiada dificultad.

Los lectores que ya conozcan ASP y alguno de los siguientes lenguajes: Java, C o C++, lo van a tener algo más fácil que aquellos que no tengan estas nociones, pero que nadie se asuste, este texto está pensado para ambos tipos de audiencia.

Hasta aquí hemos realizado una pequeña introducción a las páginas ASP .NET, en los siguientes apartados vamos a comentar otros aspectos introductorios que tiene que ver con la tecnología ASP .NET e incluso desarrollaremos nuestra primera página ASP .NET.

En este primer capítulo se pretende introducir algunos de los aspectos más destacables de ASP .NET, como pueden ser los Web Forms junto con los controles de servidor o el mecanismo de Data Binding y el nuevo acceso a datos con ADO .NET. En este capítulo se ofrece una primera toma de contacto con ASP .NET, a lo largo de los siguientes capítulos iremos ampliando debidamente cada uno de los temas relacionados con ASP .NET.

Recorrido por las distintas versiones de ASP

Hasta la fecha nos podemos encontrar con cuatro versiones de la tecnología de las páginas activas de servidor de Microsoft, es decir, de ASP (Active Server Pages). Vamos a realizar un recorrido histórico a través de ASP.

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

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

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

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

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

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

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

ASP 3.0 se encuentra disponible dentro de Windows 2000 en cualquiera de sus versiones (Professional, Server, Advanced Server). Dentro de Windows 2000 encontramos el componente Internet Information Services o Internet Information Server, que como todos los lectores sospechan es la nueva versión del servidor Web de Microsoft Internet Information Server. Al instalar Internet Information Services 5.0 (IIS 5.0) dotamos a nuestro servidor de todas las funcionalidades de un potente servidor Web, y dentro de estas funcionalidades se encuentra el protagonista de nuestro artículo, ASP 3.0.

ASP 3.0 podemos decir que es la evolución lógica de ASP 2.0, no supone ningún cambio radical, ofrece una serie de mejoras y novedades. Se añade un nuevo objeto integrado llamado ASPError, este nuevo objeto es utilizado para el tratamiento de errores.

Y por fin llegamos a la nueva versión de ASP, ASP .NET, en este caso los cambios respecto a ASP 3.0 si que son notables, tanto que en muchos casos podemos considerar que se parecen en poco, ni siquiera podríamos considerar que se trata de ASP 4.0, ya que como veremos a lo largo del presente texto los cambios van más allá. ASP .NET plantea una nueva forma de desarrollar aplicaciones Web dentro del entorno ofrecido por el .NET Framework. Veremos que ofrece una nueva forma de programar aplicaciones Web.

ASP .NET es completamente compatible con ASP, podemos tener aplicaciones Web basadas en ASP y ASP .NET funcionando en el mismo servidor Web, que será Internet Information Server 5.0, las páginas ASP tienen la extensión .ASP y son procesadas por la DLL ASP.DLL y sin embargo las páginas ASP .NET poseen la extensión .ASPX y son procesadas por el entorno de ejecución .NET Framework, que las transforma en el código intermedio.

Como curiosidad comentaré a los lectores que en las primeros pasos de la plataforma .NET (beta 1 y versiones preliminares previas) ASP .NET se denominaba ASP+ y la propia plataforma poseía la denominación de NGWS (New Generation Windows Services, Nueva Generación de Servicios de Windows).

Material necesario

En este apartado vamos a comentar los materiales mínimos de los que debemos disponer para poder desarrollar aplicaciones ASP .NET, Para seguir el texto de forma satisfactoria, y poder probar todos los ejemplos que se muestran en el mismo es necesario disponer del siguiente software:

- Microsoft .NET Framework SDK (Software Development Kit): es la implementación de la plataforma .NET sobre la que se ejecutarán las páginas ASP .NET. El .NET Framework SDK contiene toda las clases que componen la plataforma .NET. Para poder instalar el .NET Framework SDK es necesario tener instalado Internet Explorer 6.0 y el Service Pack 2 de Windows 2000. Este producto es gratuito y se puede obtener en el sitio Web de Microsoft.

- SQL Server 7/2000: va a ser el servidor de base de datos utilizado para los distintos ejemplos del texto, sobre todo en los ejemplos de la parte dedicada a ADO .NET.

- Sistema operativo Windows 2000 Server/Professional. ASP .NET no es soportado por Windows NT ni por Windows 9x.

- El servidor Web Internet Information Server 5.0 (IIS 5.0). Incluido como parte del sistema operativo Windows 2000.

También es recomendable, aunque no obligatorio, disponer de la nueva versión de Visual Studio, que no es otra que Visual Studio .NET. Visual Studio .NET ofrece el primer entorno de desarrollo para la plataforma .NET. Ofrece un entorno integrado desde el que podemos desarrollar nuestra aplicaciones con ASP .NET.

En un principio para desarrollar páginas ASP .NET no es necesaria ninguna herramienta especial, podemos escribir nuestras páginas mediante un procesador de textos sencillo, como puede ser el bloc de notas de Windows. Pero recomiendo al lector que obtenga el entorno de desarrollo de Microsoft Visual Studio, que se denomina Visual Studio .NET, ya que ofrece una serie de ayudas y utilidades muy interesantes a la hora de desarrollar aplicaciones ASP .NET, además de valiosa documentación.

El siguiente apartado está dedicado sobre todo a los lectores que ya conocían versiones previas de ASP, ya que consiste en realizar una comparativa de ASP .NET con ASP 3.0. Este apartado también puede ser recomendable para los lectores que no conozcan ASP, ya que se adelantan algunas de las características de ASP .NET.

Comparativa de ASP .NET con ASP 3.0

En este apartado se van a adelantar una serie de conceptos y de características de ASP .NET, que más adelante se retomarán para comentarlas en más detalle.

A continuación vamos a comentar las diferencias principales que presenta ASP .NET respecto a su predecesor ASP 3.0, este apartado nos puede ayudar a realizar una migración de nuestras aplicaciones ASP a aplicaciones ASP .NET. de todas formas si instalamos el .NET Framework SDK, nuestras aplicaciones ASP seguirán funcionando, ya que las páginas ASP tienen la extensión .ASP y son procesadas por la Dll ASP.DLL y sin embargo las páginas ASP .NET poseen la extensión .ASPX y son procesadas por el entorno de ejecución .NET Framework, que las transforma en el código intermedio, ya que ahora las páginas se compilan previamente a su ejecución. Las páginas ASP .NET en un principio eran denominadas también páginas ASPX.

Código compilado

Esta es la primera diferencia que encontramos entre las páginas ASP y las páginas ASP .NET, las páginas ASP eran interpretadas línea a línea por la DLL ASP.DLL, pero ahora las páginas ASP .NET cuando reciben la primera petición se compilan automáticamente a un lenguaje intermedio que es conocido como Common Language Runtime (CLR), es decir, es un lenguaje común al que compilan todos los lenguajes que utilicemos en nuestras páginas ASP .NET, generando el mismo código, ya sea Visual Basic .NET, C# o JScript. Gracias a esta característica podemos obtener grandes ventajas en lo que a rendimiento en tiempo de ejecución se refiere, ya que la compilación de las páginas sólo se produce en la primera petición que se realiza sobre la página, o bien cuando el código fuente de la misma se ha modificado y necesita por lo tanto actualizarse, además el resultado de esta compilación permanece en caché para poder ser reutilizada.

La compilación a un lenguaje intermedio, Intermediate Language (IL), es una característica común de la plataforma .NET. El código intermedio o lenguaje intermedio posteriormente deberá ser interpretado por el entorno de ejecución de la plataforma .NET. Este código intermedio mantiene una idea similar al bytecode que se genera en el lenguaje Java. Pero la plataforma .NET con su Intermediate Language va más allá del concepto inicial introducido por Java, se puede decir que es una extensión de este concepto, ya que con el .NET Framework en lugar de un único lenguaje para muchas plataformas, se pretende un entorno común multiplataforma, que soporte muchos lenguajes, basándose en que todos ellos compilen a un mismo código intermedio.

En general se puede decir que en ASP .NET se ha mejorado el rendimiento en la ejecución respecto a ASP 3.0.

Los lectores que conozcan la tecnología de las páginas de servidor de Java, es decir, JSP (Java Server Pages), comprobarán que la característica de la compilación de ASP .NET es muy similar a la solución que ofrece Sun. A lo largo del texto veremos que existen algunas otras similitudes entre ASP .NET y JSP, esto puede servir de ayuda para comprender mejor la tecnología ASP .NET a los lectores que ya conocen JSP.

Bloques de código

Otro aspecto diferente de ASP .NET, es que los delimitadores de script de servidor, es decir los delimitadores <% %> han pasado a utilizarse en un segundo plano, siendo más común la utilización de los bloques <script></script>.

Los delimitadores <% %> se puede seguir utilizando, pero no se podrán utilizar a la hora de definir un procedimiento a una función dentro de una página, para esa labor debemos utilizar los bloques <script language="C#"runat="server"></script>.

Por lo tanto ahora disponemos de tres delimitadores de código de servidor, los delimitadores <script></script> para declarar métodos, los delimitadores <% %> para ejecutar sentencias y los delimitadores <%= %> para mostrar el resultado de una expresión. Más ade2lante veremos con más detenimiento la sintaxis básica de las páginas ASP .NET.

Lo más común en ASP .NET es utilizar los delimitadores <script></script>, ya que nos van a permitir definir todos los métodos para el tratamiento de eventos, y por lo tanto gracias al nuevo tratamiento de eventos que nos ofrece ASP .NET podremos utilizar un código mucho más estructurado en nuestras páginas, sin tener que mezclar continuamente el código de servidor en C# con el código HTML.

Directivas

En versiones anteriores de ASP únicamente podíamos utilizar una única directiva en la página, por ejemplo para indicar el lenguaje de programación utilizado en la página, sin embargo en ASP .NET podemos utilizar varias sentencias con directivas en la página. Además ASP introduce nuevas directivas. Las directivas que presenta ASP .NET son Page, Control, Import, Register, Assembly y OutputCache, a continuación vamos a pasar a comentar brevemente cada una de estas directivas.

La directiva mas compleja de todas, en lo que a número de atributos se refiere, y también la más utilizada es la directiva Page. La directiva Page define una serie de atributos específicos para cada página y que serán utilizados por el compilador a la hora de generar el código intermedio (IL) al que se compila cada página. Esta directiva soporta todos los atributos existentes para la única directiva que se podía utilizar en ASP, es decir, ofrece los atributos Language, Transaction, etc. La directiva Page es la directiva por defecto, así por ejemplo la directiva <@Language="VBScript"%> de ASP 3.0, tendría el equivalente <@Page Language="VB"> en ASP .NET. Otros atributos o propiedades de la directiva son: Trace, Inherits, Src, etc.

La directiva Control, es menos común que la anterior y se utiliza para definir atributos específicos de controles definidos por el usuario.

Otra directiva algo más común es la directiva Import, que permite importar un espacio con nombre (namespace) en la página actual. Los espacios con nombre o espacios calificados es una forma de organizar las clases que forman parte del .NET Framework. Para los lectores que conozcan Java, podemos decir que se trata de un mecanismo similar al de los paquetes, para los cuales utilizamos en Java una sentencia llamada import. El espacio con nombre puede contener clases de la librería de clases del .NET Framework, o bien contener clases definidas por el usuario, que indicará su propio namespace. Al igual que muchos de los conceptos que estamos adelantando en este apartado, lo veremos con más detalle más adelante en el texto.

La nueva directiva Register asocia un alias con un namespace, este alias será el prefijo que se utilice en las etiquetas que hagan uso del namespace correspondiente.

La directiva Assembly nos permite establecer una referencia dentro de una página a un assembly, pudiendo utilizar todas las clases e interfaces definidos en el mismo. Un assembly es similar a un componente, físicamente es un fichero .DLL.

Y al última de las directivas de ASP .NET es la directiva OutputCache, que permite especificar la política del caché de salida de la página, pudiendo indicar el número de segundos que el resultado de la ejecución de una página se va a mantener en caché.

Cada una de estas directivas se verán en detalle más adelante cuando su uso sea necesario dentro de las páginas ASP .NET.

Acceso a datos

En esta nueva versión de ASP se utiliza el acceso a datos común para toda la plataforma .NET, es decir, la nueva versión de ADO (ActiveX Data Objects) denominada ADO .NET, en los capítulos correspondientes realizaremos una introducción al acceso a datos con ADO .NET y al modelo de objetos que presenta.

Lenguajes

ASP .NET no soporta el lenguaje de script VBScript, sino que soporta lenguajes más completos y potentes como pueden ser Visual Basic .NET, C# y JScript, C++ o incluso también Perl y COBOL.

De forma predeterminada ASP .NET soporta los lenguajes Visual Basic .NET, C# y Jscript.

Microsoft está apostando fuerte por el lenguaje C#, que parece que se va a convertir en el lenguaje insignia de la plataforma .NET. Por este motivo y otros ya comentados anteriormente, este texto va a utilizar como lenguaje para las páginas ASP .NET el lenguaje C#.

Orientación a objetos

ASP .NET se encuentra dentro de un modelo de programación orientado a objetos. La plataforma .NET se basa en un conjunto bastante amplio de clases organizadas mediante los espacios con nombre (NameSpaces), y ASP .NET al estar incluido dentro de la plataforma .NET utiliza también este conjunto de clases.

En ASP .NET cada página es un objeto, será una instancia de la clase Page, y cada página tendrá sus atributos, así tenemos un atributo llamado Request que es un objeto de la clase HttpRequest y representa una petición que se ha realizado sobre una página ASP .NET, lo mismo ocurre con el atributo (o miembro) Response, que es una instancia de la clase HttpResponse que representa la salida que la página envía al navegador Web del usuario. Ambas clases se encuentran dentro del espacio con nombre System.Web. Es decir, en ASP .NET los objetos integrados o intrínsecos de ASP 3.0, se obtiene a través de miembros de la clase Page.

La orientación a objetos también se encuentra muy patente a la hora de utilizar los controles de servidor. Estos controles los veremos más adelante, pero cada uno de ellos posee su clase correspondiente.

Por lo tanto para poder comprender ASP .NET y desarrollar aplicaciones Web basadas en ASP .NET es necesario conocer los conceptos de la programación orientada a objetos, como puede ser: el concepto de clase, objeto, mecanismo de herencia, etc.

Otros cambios

En este subapartado que finaliza el presente punto se han agrupado una serie de cambios que presenta ASP .NET respecto de ASP 3.0. El primero de ellos es que se abandona el método Server.CreateObject para la creación de objetos, ahora se debe declarar el objeto del tipo específico y utilizar además la palabra reservada New, así por ejemplo la sentencia Set obj=Server.CreateObjec("progID") se debe sustituir por la sentencia Dim obj As New progID. Esto le será muy familiar a los lectores conocedores de Visual Basic.

Ya no es necesario obtener una referencia al objeto ObjectContext, que se introducía en ASP 2.0, sino que ASP .NET ofrece el nuevo objeto Context que permite acceder al entorno de ejecución de las páginas ASP .NET.

El fichero GLOBAL.ASA a pasado a denominarse GLOBAL.ASAX y permite capturar nuevos eventos como pueden ser: Application_Error, Application_Begin_Request, Application_Authorize Request, etc., hasta un total de 16 eventos frente a los 4 de las versiones anteriores de ASP.

La configuración de nuestra aplicación Web (aplicación ASP .NET) ya no es necesario (ni recomendable) realizarla a través de la consola de administración del servidor Web Internet Information Server, sino que disponemos de un fichero especial de configuración en formato XML denominado WEB.CONFIG. Para los lectores conocedores de JSP (Java Server Pages) les comento que la finalidad de este fichero es muy similar al fichero de configuración WEB.XML de la tecnología de las páginas de servidor de Java.

Sintaxis de ASP .NET

En este apartado vamos a realizar una primera aproximación a la sintaxis que ofrecen las páginas ASP .NET.

ASP .NET utiliza tres delimitadores distintos para indicar el código de servidor, es decir, el código que se ejecutará en el servidor y que estará escrito en el lenguaje C#-

Encerrado dentro de los delimitadores <%%> se va a encontrar todo el código de script de servidor, de esta forma el comando <%nombre="Pepe"%> asigna el valor Pepe a la variable nombre.

Dentro del segundo tipo de delimitadores <%=%> se encuentran expresiones que devuelve algún valor para ser mostrado en la salida de la página, así por ejemplo la expresión <%=nombre%> enviará al navegador el valor Pepe, es decir, el valor actual de la variable, más adelante se verá una equivalencia de estos delimitadores con un método de un objeto integrado de ASP .NET.

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

El tercer tipo de delimitador es una etiqueta de la siguiente forma <script language=”C#” runat=”server”></script>. Esta etiqueta es utilizada únicamente para declarar métodos (procedimientos y funciones) de la página, y el delimitador <%%> sólo para escribir código que se ejecute dentro de la página fuera de procedimientos y funciones, es decir,. El código que se ejecuta en línea.

A continuación vamos a ofrecer unos sencillos fragmentos de código en los que se utilizan cada uno de los delimitadores comentados.

En el Código fuente 1 se puede ver la utilización de los delimitadores <%%> para definir un objeto de la clase DateTime con la fecha y hora actual. Y también se utilizan los delimitadores <%=%> para mostrar la fecha y hora actual en el navegador del usuario.

```
<%DateTime ahora=DateTime.Now;%>
<%=ahora.ToString()%>
```

Código fuente 1

Dentro de los delimitadores de código de servidor se pueden encontrar también instrucciones del lenguaje de correspondiente, así por ejemplo puede aparecer una instrucción if...else del lenguaje C# como se puede apreciar en el Código fuente 2.

```
<%String nombre="", variable="";
if(nombre==""){
      variable="Nombre desconocido";
}else{
      variable="Hola amigo "+nombre;
}%>
<font color="green"><%=variable%></font>
```

Código fuente 2

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

También se puede incluir código HTML entre las instrucciones del lenguaje de servidor, aunque no es recomendable de cara a una lectura del código más sencilla. Un ejemplo equivalente al código anterior lo podemos ver en el Código fuente 3.

```
<font color="green">
<%String nombre="";
if(nombre==""){%>
      Nombre desconocido
<%}else{%>
      Hola amigo <%=nombre%>
<%}%>
</font>
```

Código fuente 3

También para poder realizar una lectura más sencilla del código ASP .NET se recomienda utilizar los delimitadores del código de servidor encerrando varias líneas de código en lugar de un par de delimitadores por cada línea. Así, en lugar de escribir el código que se muestra en el Código fuente 4, es recomendable utilizar el Código fuente 5.

```
<%String strNombre="";%>
<%Session.Add("nombre","Angel");%>
<%strNombre=Session["nombre"].ToString();%>
<%Response.Write(strNombre);%>
```

Código fuente 4

```
<%String strNombre="";
Session.Add("nombre","Angel");
strNombre=Session["nombre"].ToString();
Response.Write(strNombre);%>
```

Código fuente 5

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

```
<%strNombre=Session["nombre"].ToString();%>
```

Código fuente 6

Si tenemos que definir un método (procedimiento o función) utilizaremos las etiquetas <script> como se puede ver en el Código fuente 7.

```
<script language="c#" runat="server">
void Pulsado(Object sebder, EventArgs args){
      etiqueta.Text="¡Hola Mundo!";
}
</script>
```

Código fuente 7

La sintaxis de ASP .NET también está muy relacionada con la sintaxis XML. Los controles Web de servidor que utilizamos desde ASP se instancian utilizando una sintaxis basada en XML. En el fragmento de código que se ofrece en el Código fuente 8, se puede ver la sintaxis que se utiliza para instanciar dos controles Web de servidor, en este caso se crea una etiqueta y un botón, dentro de lo que se denomina un Web Form o formulario Web. Los conceptos de controles Web y Web Form los trataremos más adelante.

```
<form id="formulario" method="post" runat="server">
      <asp:label  id="etiqueta" runat="Server"></asp:label>
      <asp:button id="boton"   onclick="Pulsado" runat="server"
                                                  text="Pulsa"></asp:button>
</form>
```

Código fuente 8

Sin embargo la sintaxis utilizada para crear controles HTML de servidor desde las páginas ASP .NET es distinta. En este caso se utiliza una sintaxis prácticamente idéntica a la del lenguaje HTML.

```
<form id="controlesHTML" method="post" runat="server">
<input type="button" id="boton" runat="server"  value="Pulsar"
onserverclick="Pulsado">
<div id="etiqueta" runat="server"></div>
</form>
```

Código fuente 9

Esta es la sintaxis básica de las páginas ASP .NET, según vayamos avanzado en el texto veremos algunos aspectos en más detalle, y también veremos la sintaxis del lenguaje C#. A continuación se ofrece un apartado en el que se introduce una nueva y muy importante característica de las páginas ASP .NET, los formularios Web (Web Forms).

Web Forms

Los Web Forms (formularios Web) es una nueva característica de ASP .NET que ofrece un nuevo modelo de programación que nos va a permitir generar contenidos dinámicos de una forma más sencilla.

Los Web Forms vienen a sustituir e integrar los formularios HTML dentro del entorno de programación de ASP .NET. Un Web Form en última instancia va a generar el código HTML correspondiente al formulario que estamos diseñando.

Una de las grandes ventajas que ofrecen los Web Forms es que ofrecen un modelo de programación muy similar al de los tradicionales formularios de Visual Basic, incluso podemos realizar el tratamiento de eventos del cliente desde código del servidor.

Los Web Forms tienen tanta importancia dentro de ASP .NET que incluso en el entorno de desarrollo Visual Studio .NET cuando queremos añadir una nueva página, no parece la opción de añadir nueva página ASP .NET, sino que se muestra la posibilidad de añadir un nuevo Web Form, es decir, en algunos lugares las páginas ASP .NET también se denominan Web Forms.

A continuación se van a describir las ventajas que nos ofrecen los Web Forms, en los siguientes capítulos se verán en detalle el modelo de programación ofrecidos por los Web Forms junto con los controles ASP .NET de servidor. Las ventajas que nos ofrecen los Web Forms son las siguientes:

- Nos permiten reutilizar controles de interfaz de usuario (UI) que encapsulan su funcionalidad, reduciendo de esta forma el código que el desarrollador debe escribir.

- Ofrece un código más limpio y claro en la página, ya que no aparece mezclado con código HTML como ocurría en la versiones anteriores de ASP, en las que era bastante usual intercalar código HTML con código ASP, aunque en ASP .NET esto también se permite por compatibilidad, pero si nos ceñimos al modelo de programación que ofrecen los Web Forms no es necesario.

- Podemos construir todo el interfaz del Web Form a través de herramientas o entornos de desarrollo, al igual que construimos los formularios con Visual Basic arrastrando y soltando los controles del interfaz de usuario. Visual Studio .NET nos ofrece una vista de diseño para los Web Forms, que incluso nos permite diseñar los Web Forms situando sus controles según las posiciones x e y de la pantalla.

- Integra el tratamiento de eventos del cliente con el código correspondiente del servidor. El tratamiento de eventos es muy similar al que ya se utilizaba en los formularios de las anteriores versiones de Visual Basic.

- Realizan un mantenimiento del estado del formulario (Web Form) de forma automática entre distintas peticiones, es decir, si seleccionamos una serie de valores en el Web Form, al enviarlo de forma automática se van a mantener estos valores en el caso de que se vuelva a recargar el formulario.

- Ofrece una mayor separación entre la lógica de la aplicación y la presentación de la misma.

- Permite utilizar un gran conjunto de controles de servidor que generarán el código HTML necesario para mostrar el Web Form en la página cargada por el cliente en su navegador. Después veremos que dentro de los Web Forms podemos encontrar varios tipos de controles.

A continuación se ofrece un código muy sencillo (Código fuente 10) que muestra la utilización un Web Form dentro de una página ASP .NET. El ejemplo consiste en un Web Form que ofrece una lista de destinos en una lista desplegable con sus precios, y al pulsar el botón del formulario se indica en una etiqueta el nombre y precio del destino seleccionado.

```
<html><head>
<script language="C#" runat="server">
void Pulsado(Object sender, EventArgs args){
      etiqueta.Text="<br>El precio para el destino " + lista.SelectedItem.Text +
            " es de " + lista.SelectedItem.Value + " euros";
}
</script>
</head>
<body>
<form id="WebForm" method="post" runat="server">
<asp:DropDownList id="lista" runat="server">
      <asp:listitem value="1000" text="Isla de Pascua"></asp:listitem>
      <asp:listitem value="900" text="Morea"></asp:listitem>
      <asp:listitem value="750" text="Papeete"></asp:listitem>
      <asp:listitem value="850" text="Bora-Bora"></asp:listitem>
</asp:DropDownList>
<asp:Button id="boton" runat="server" Text="Ver precio"
onclick="Pulsado"></asp:Button>
<asp:Label id="etiqueta" runat="server"></asp:Label>
</form>
</body></html>
```

Código fuente 10

En la Figura 13 se puede ver un ejemplo de ejecución de esta página ASP .NET.

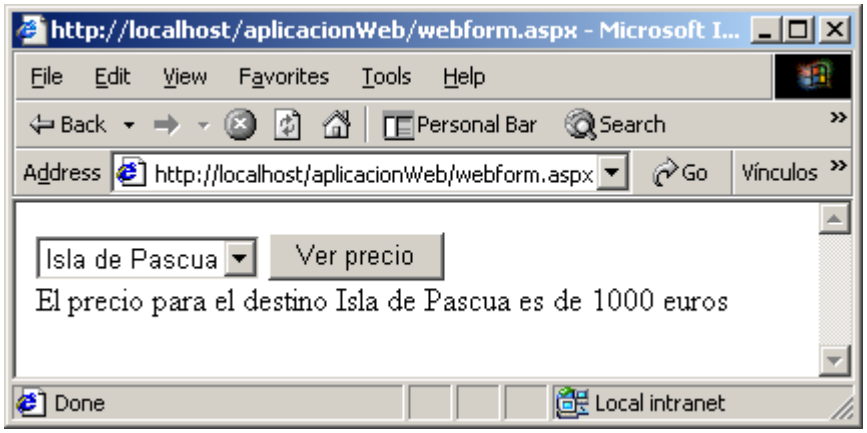


Figura 13

A la vista del código del ejemplo se pueden realizar los siguientes comentarios acerca de los Web Forms y su utilización desde las páginas ASP .NET. Los controles (controles ASP .NET) del Web Form tienen una notación en formato XML, a excepción de la etiqueta que se corresponde con el propio formulario, y como se puede comprobar todos ellos poseen la propiedad runat con el valor server, para indicar que son controles de servidor.

Si ejecutamos la página se puede ver que en distintas peticiones se va conservando el valor que se seleccione en la lista de destinos, no hemos escrito ningún código específico para realizar este

comportamiento (que sería necesario en versiones anteriores de ASP), el propio Web Form lo hace por nosotros generando el código HTML necesario, que consiste en un campo oculto llamado _VIEWSTATE. El código HTML que genera la página del ejemplo se puede ver a continuación (Código fuente 11).

```
<html><head>

</head>
<body>
<form name="WebForm" method="post" action="webform.aspx" id="WebForm">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTY2NzM0NjY1Njt0PDtsPGk8Mj47PjtsPHQ8O2w8aTw1Pjs+O2w8dDxwPHA8bDxUZXh0Oz47
bDxcPGJyXD5FbCBwcmVjaW8gcGFyYSBlbCBkZXN0aW5vIElzbGEgZGUgUGFzY3VhIGVzIGRlIDEwMDAgZXV
yb3M7Pj47Pjs7Pjs+Pjs+" />

<select name="lista" id="lista">
      <option selected="selected" value="1000">Isla de Pascua</option>
      <option value="900">Morea</option>
      <option value="750">Papeete</option>
      <option value="850">Bora-Bora</option>

</select>
<input type="submit" name="boton" value="Ver precio" id="boton" />
<span id="etiqueta"><br>El precio para el destino Isla de Pascua es de 1000
euros</span>
</form>
</body></html>
```

Código fuente 11

Otro punto importante es que, como se puede comprobar, no hemos utilizado la colección Form del objeto Request para obtener los valores de los elementos enviados en el formulario, sino que simplemente hemos accedido a las propiedades de los controles a través del nombre con el que los hemos declarado, es decir, al estilo de los formularios de Visual Basic.

Esto es posible gracias a que cada control del formulario es un objeto (objetos de la página ASP .NET), es decir una instancia de una clase determinada, y por lo tanto cada uno de ellos poseerá sus propiedades y métodos. Así por ejemplo la lista desplegable que hemos indicado en el Web Form se corresponde con un objeto de la clase DropDownList que se encuentra dentro del espacio con nombre (NameSpace) System.Web.UI.WebControls presente en el .NET Framework. Lo mismo ocurre con el objeto boton que es de la clase Button y el objeto etiqueta que es una instancia de la clase Label. Existe un gran número de controles de servidor y cada uno de ellos pertenece a una clase del .NET Framework.

En el siguiente apartado vamos a comentar los controles de servidor que nos ofrece ASP .NET.

Controles ASP .NET

Directamente relacionados con los Web Forms tenemos los distintos controles de servidor que se puede utilizar dentro de los Web Forms y en las páginas ASP .NET en general. Todos estos controles generarán el correspondiente código HTML para que el usuario pueda utilizarlos en la página cargada en su navegador. Los numerosos controles de servidor incluidos en ASP .NET se pueden agrupar en cuatro categorías o familias:

- Controles intrínsecos, que a su vez se subdividen en dos grupos, y que representan elementos HTML.
- Controles de lista, utilizados para distribuir y mostrar datos en una página.
- Controles ricos, ofrecen funcionalidades avanzadas de interfaz de usuario.
- Controles de validación, ofrecen distintos tipos de validación de entrada de datos.

A lo largo de los siguientes subapartados vamos a ir comentando brevemente los distintos tipos de controles que ofrecen los Web Forms.

Controles intrínsecos

Estos controles son los encargados de generar elementos HTML en el cliente, son los que hemos utilizado en el ejemplo del apartado anterior, cada uno de estos controles tiene su etiqueta HTML correspondiente, esto lo podemos comprobar en el código HTML que se genera en el ejemplo anterior. Dentro de este grupo de controles podemos distinguir dos tipos, los llamados controles HTML y los controles Web.

Controles HTML

Los controles HTML son muy similares a las etiquetas HTML que representan y sus propiedades son casi idénticas, estos controles se diferencian de las etiquetas HTML en que poseen la propiedad runat con el valor server, por lo demás el aspecto que presentan es idéntico.

En el siguiente código de ejemplo (Código fuente 12) se muestra una página que posee un Web Form y dos controles HTML, un botón (HtmlButton) y una etiqueta (HtmlGenericControl) que contiene un texto, al pulsar el botón parecerá la fecha y hora actual en la etiqueta . Estos controles se encuentran dentro del espacio con nombre System.Web.UI.HtmlControls.

```
<html>
<body>
<script language="C#" runat="server">
void Pulsado(Object sender, EventArgs args){
      DateTime ahora;
      ahora = DateTime.Now;
      etiqueta.InnerText ="El momento actual es: " + ahora.ToString();
}
</script>
<form id="controlesHTML" method="post" runat="server">
<input type="button" id="boton" runat="server"  value="Pulsar"
                                        onserverclick="Pulsado" NAME="boton">
<div id="etiqueta" runat="server"></div>
</form>
</body>
</html>
```

Código fuente 12

Estos controles permiten mantener el estado de forma automática entre distintas llamadas a una página ASP .NET. Para poder acceder al control dentro de la página ASP .NET se debe identificar mediante el atributo id de la etiqueta HTML correspondiente.

Estos controles se ofrecen para mantener la compatibilidad con versiones anteriores de ASP, así si queremos que una etiqueta HTML pase a ser un control HTML simplemente debemos indicar la propiedad runat con el valor server, normalmente se suelen utilizar el siguiente tipo de controles intrínsecos, los controles Web.

Controles Web

Estos controles ASP .NET cumplen la misma funcionalidad que los anteriores, en este caso muestran una sintaxis basada en XML, y no existe una correlación directa entre los controles y las etiquetas HTML que se generan.

Los controles Web pertenecen al espacio con nombre (espacio calificado) System.Web.UI. WebControls, y a diferencia que el espacio de nombre de los controles HTML, el espacio de nombres que contiene los controles Web, contiene al resto de familias de controles ASP .NET, que seguiremos viendo a continuación, es decir, contiene un mayor número de controles.

Si rescribimos el ejemplo anterior para utilizar controles Web tendremos el siguiente código fuente en nuestra página ASP .NET (Código fuente 13).

```
<html>
<body>
<script language="C#" runat="server">
void Pulsado(Object sender, EventArgs args){
      DateTime ahora;
      ahora = DateTime.Now;
      etiqueta.Text ="El momento actual es: " + ahora.ToString();
}
</script>
<form id="controlesWeb" method="post" runat="server">
<asp:button id="boton" text="Pulsar" onclick="Pulsado" runat="Server"></asp:button>
<asp:label id="etiqueta" runat="server"></asp:label>
</form>
</body>
</html>
```

Código fuente 13

Como se puede comprobar existen algunas diferencias a la hora de utilizar controles HTML y controles Web, además de las más evidentes relativas a la distinta sintaxis a la hora de declarar los controles.

Algunos nombres de las propiedades de los controles de servidor cambian, el control HTML HtmlButton posee la propiedad value, y sin embargo el control Web equivalente (Button) posee la propiedad text. A la hora de indicar los métodos para el tratamiento de eventos de la pulsación de un botón del ratón en los controles HTML utilizamos la propiedad onserverclick, y en los controles Web utilizamos la propiedad onclick. En los controles HTML se debe distinguir si se trata de un evento de cliente o de servidor, ya que podemos tratar ambos tipos, sin embargo desde los controles Web sólo tenemos la posibilidad de utilizar eventos de servidor, es decir, eventos que son tratados por rutinas en el servidor, es decir, por métodos de la página para el tratamiento de eventos.

Los controles Web ofrecen un mayor nivel de abstracción que los controles HTML, y su modelo de objetos no refleja la sintaxis HTML necesariamente. Cuando la página se carga en el navegador, el control Web determina el tipo de navegador que ha realizado la petición, y de acuerdo con esta información genera el código HTML apropiado, podemos decir que en este aspecto se trata de controles inteligentes.

En la siguiente tabla (Tabla 1) se muestra la correspondencia existente entre los controles Web y las etiquetas HTML que se generan.

Control Web	**Código HTML**
<asp:textbox>	<input type=text>
<asp:button>	<input type=submit>
<asp:imagebutton>	<input type=image>
<asp:checkbox>	<input type=checkbox>
<asp:radiobutton>	<input type=radiobutton>
<asp:listbox>	<select size="5"></select>
<asp:dropdownlist>	<select></select>
<asp:hyperlink>	
<asp:image>	<img src="...">
<asp:label>	
<asp:panel>	
<asp:table>	<table></table>

Tabla 1

En algunos casos nos puede interesar utilizar controles Web, y en otros controles HTML, los controles Web deberán ser usados en las siguientes situaciones:

- Preferimos un modelo de programación similar a Visual Basic.
- Estamos desarrollando Web Forms que deben ser mostrados por varios tipos de navegadores.
- Se necesita una funcionalidad específica, como puede ser un calendario o un rotador de anuncios.

Y los controles HTML es preferible utilizarlos en las siguientes situaciones:

- Preferimos un modelo de objetos similar al lenguaje HTML.

- Estamos trabajando con páginas Web existentes y las queremos migrar a Web Forms.
- El control debe interactuar con script de cliente y de servidor.

A continuación vamos a pasar a comentar las restantes familias de controles ASP .NET.

Controles de lista

Estos controles se encuentran especializados en mostrar listados de datos en las páginas ASP .NET, tarea que suele ser bastante común en la mayoría de las aplicaciones Web, gracias a estos controles, que pertenecen al espacio con nombre System.Web.UI.WebControls, las tareas de relacionadas con los listados de información son mucho más sencillas, estos controles ASP .NET generan de forma automática el interfaz de usuario que muestra la información permitiendo la paginación, filtrado y otras operaciones sobre los contenidos correspondientes.

Los controles de tipo lista se suelen utilizar en combinación con otra nueva característica ofrecida por el entorno de los Web Forms, se trata de Data Binding, es decir, la conexión de los datos con los controles en los que se van a mostrar.

Existen tres controles estándar dentro de esta familia: Repeater, DataList y DataGrid. Relacionado con los controles de lista, más estrechamente con los controles Repeater y DataList, nos encontramos un interesante mecanismo que nos ofrece ASP .NET denominado plantillas (templates), que nos va a permitir personalizar el aspecto de la información que vamos a mostrar, el control es absoluto, existiendo para ello cinco tipos distintos de plantillas, que los comentaremos en detalle en el capítulo correspondiente. Estas plantillas se utilizan en combinación con los controles ASP .NET DataList y Repeater.

El control Repeater es el más simple de todos estos controles, y es utilizado para mostrar la salida del contenido un número determinado de veces. Mediante las plantillas podemos definir el estilo de la cabecera, elementos, pies y separadores utilizados por el control. En realidad este componente no tiene ningún resultado visual, para obtener un interfaz de usuario debemos aplicar las plantillas que consideremos necesarias.

El control DataList también nos ofrece una funcionalidad parecida al control Repeater, pero en este caso permite personalizar aun más el resultado mostrado en pantalla correspondiente a la información que contiene. En el control DataList podemos indicar si los elementos se van a distribuir de manera horizontal o vertical, esto nos permite tener columnas de elementos de información. además podemos indicar el número de columnas en las que deseamos que se distribuyan los elementos.

El último de los controles de lista, y el más complejo, es el control DataGrid, este control muestra la información en forma de tabla con celdas y columnas, generando el código HTML equivalente a una tabla. Se puede utilizar de forma muy sencilla, o podemos complicar su uso para llegar a altos niveles de personalización a la hora de mostrar la información.

Controles ricos

En este grupo de controles (rich controls) se encuentran una serie de controles avanzados que ofrecen una gran funcionalidad, pero en la versión beta 2 de ASP .NET únicamente nos encontramos con dos controles: el control rotador de anuncios y el control calendario, se espera que se añadan nuevos controles en siguientes versiones de la tecnología. Pertenecen también al espacio de nombres System.Web.UI.WebControls.

El control rotador de anuncios (AdRotator), no es ninguna novedad de ASP .NET, ya que disponíamos de este control en versiones anteriores de ASP a través de un componente de servidor. Este control permite mostrar una serie de anuncios dentro de una página ASP, alternando la aparición de los mismos. Para indicar las imágenes y enlaces de los anuncios utilizamos un fichero especial en formato XML.

El otro control avanzado que encontramos en la beta 2 de ASP .NET es el control calendario (Calendar), que ofrece un completo calendario. Es muy sencillo de utilizar aunque ofrece un alto grado de personalización.

En el siguiente subapartado se comenta el último tipo de controles ASP .NET.

Controles de validación

Este grupo de controles nos permiten validar la entrada dada por el usuario a través de los controles de un Web Form, esta validación se puede realizar en el cliente o en el servidor. Si el navegador lo soporta (navegador de alto nivel) se realizará en el cliente a través de script de cliente, pero si el navegador no lo soporta (navegador de bajo nivel), la validación se llevará a cabo en el servidor, lo que supone una conexión más con el servidor. Es posible forzar que la validación se realice en el servidor.

ASP .NET ofrece seis controles de validación que pertenecen también al NameSpace System.Web.UI.WebControls, y son los siguientes:

- RequiredFieldValidator: este control valida si se ha rellenado un campo con un valor, es decir, valida los campos requeridos.

- CompareValidator: comprueba si un control contiene un valor determinado o coincide con el contenido de otro control, se utiliza con un operador de comparación.

- RangeValidator: valida si el valor indicado por el usuario se encuentra dentro de un rango determinado.

- RegularExpresionValidator: es el más versátil de los controles de validación, valida si el valor indicado por el usuario se corresponde con una expresión.

- CustomValidator: este control permite especificar una función de validación personalizada.

- ValidationSummary: este control contiene todos los mensajes de error que se han producido como resultado de las validaciones realizadas y muestra una lista de ellos en la página.

La validación se realiza sobre los controles de un Web Form, y un mismo control puede tener distintos tipos de validación, es decir, se pueden utilizar distintos controles de validación sobre un mismo control a validar.

En el capítulo correspondiente veremos en detalle la utilización de los controles ASP .NET más importantes, como ya se ha indicado el presenta capítulo pretende únicamente ofrecer una pequeña muestra de lo que podemos encontrar dentro de ASP .NET.

ADO .NET

Esta es la nueva versión del modelo de objetos ADO (ActiveX Data Objects), es decir, la estrategia que ofrece Microsoft para el acceso a datos dentro de la plataforma .NET.

ADO .NET. al igual que sucedía con ASP .NET, en las primeras versiones de la plataforma .NET se denominaba ADO+.

ADO .NET ha sido ampliado para cubrir todas las necesidades que ADO no ofrecía, ADO .NET está diseñado para trabajar con conjuntos de datos desconectados, lo que permite reducir el tráfico de red. ADO .NET utiliza XML como formato universal de transmisión de los datos.

ADO .NET posee una serie de objetos que son los mismos que aparecen en la versión anterior de ADO, como pueden ser el objeto Connection o Command, e introduce nuevos objetos tales como el objeto DataReader, DataSet o DataView. A continuación vamos a comentar brevemente los objetos principales que posee ADO .NET.

Los espacios con nombre que utiliza ADO .NET principalmente son System.Data y System.Data.OleDb o System.Data.SqlClient. System.Data ofrece las facilidades de codificación para el acceso y manejo de datos, y System.Data.OleDb y System.Data.SqlClient contienen los proveedores, en el primer caso los proveedores genéricos de OLE DB y en el segundo los proveedores nativos de SQL Server que ofrece la plataforma .NET. Para los lectores que han seguido la evolución de la plataforma .NET, apunto que estos espacios con nombre se denominaban System.Data.ADO y System.Data.SQL en la beta 1 de la plataforma .NET.

El objeto Connection define como se establece la conexión con el almacén de datos, el .NET Framework ofrece dos objetos Connection: SqlConnection y OleDbConnection, que se corresponde con las dos posibilidades de proveedores que disponemos.

Otro objeto importante dentro del modelo de objetos de ADO .NET es el objeto System.Data.DataSet (conjunto de datos), este nuevo objeto representa un conjunto de datos de manera completa, pudiendo incluir múltiples tablas junto con sus relaciones. No debemos confundir el nuevo objeto DataSet con el antiguo objeto Recordset, el objeto DataSet contiene un conjunto de tablas y las relaciones entre las mismas, sin embargo el objeto Recordset contiene únicamente una tabla. Para acceder a una tabla determinada el objeto DataSet ofrece la colección Tables.

Para poder crear e inicializar las tablas del DataSet debemos hacer uso del objeto DataAdapter, que posee las dos versiones, es decir, el objeto SqlDataAdapter para SQL Server y OleDbDataAdapter genérico de OLE DB. En la beta 1 de la plataforma .NET el objeto DataAdapter se denominaba DataSetCommand.

Al objeto DataAdapter le pasaremos por parámetro una cadena que representa la consulta que se va a ejecutar y que va a rellenar de datos el DataSet. Del objeto DataAdapter utilizaremos el método Fill(), que posee dos parámetros, el primero es una cadena que identifica el objeto DataTable (tabla) que se va a crear dentro del objeto DataSet como resultado de la ejecución de la consulta y el segundo parámetro es el objeto DataSet en el que vamos a recoger los datos.

Un DataSet puede contener diversas tablas, que se representan mediante objetos DataTable. Para mostrar el contenido de un DataSet, mediante Data Binding, por ejemplo, necesitamos el objeto DataView. Un objeto DataView nos permite obtener un subconjunto personalizado de los datos contenidos en un objeto DataTable. Cada objeto DataTable de un DataSet posee la propiedad DefaultView que devuelve la vista de los datos por defecto de la tabla. Todo esto y más lo veremos con ejemplos y detallado en los capítulos correspondientes.

Otro objeto de ADO .NET es el objeto DataReader, que representa un cursor de sólo lectura y que sólo permite movernos hacia adelante (read-only/forward-only), cada vez existe un único registro en memoria, el objeto DataReader mantiene abierta la conexión con el origen de los datos hasta que es cerrado. Al igual que ocurría con otros objetos de ADO .NET, de este objeto tenemos dos versiones, que como el lector sospechará se trata de los objetos SqlDataReader y OleDbDataReader.

Data Binding

Relacionado con el apartado anterior, en el que se realizaba una pequeña introducción al acceso a datos con ADO .NET encontramos una nueva característica de ASP .NET denominada Data Binding. El mecanismo de Data Binding (conexión con los datos o enlace con los datos) es una nueva facilidad que ofrece ASP .NET, pero que debe ser familiar para los lectores que conozcan el entorno de desarrollo de Visual Basic. Mediante este mecanismo podemos asociar unos datos con un control del interfaz de usuario correspondiente (control ASP .NET). ASP .NET permite establecer el origen de los datos de los controles ASP .NET ofreciendo compatibilidad con todos los navegadores.

La idea del mecanismo de Data Binding es muy sencilla, se conectan controles Web a una fuente de datos y ASP .NET muestra la información de forma automática, es una forma muy sencilla de organizar la información en listas, cajas de texto, grids, etc.

La forma más sencilla de establecer el origen de los datos de un control que muestra un conjunto de datos (ListBox, Repeater, DataGrid, etc.) es utilizar la propiedad DataSource del control correspondiente para indicar la fuente de los datos y a continuación llamar al método DataBind() del control. También es posible llamar al método DataBind() de la página ASP .NET, con lo que se establecen todos lo enlaces con los datos existentes en todos los controles se servidor de la página.

En la propiedad DataSource del control Web podemos especificar un amplio tipo de fuentes de datos: un array, un objeto DataView de ADO .NET, una expresión, etc. A continuación vamos a mostrar el ejemplo más sencillo de Data Binding (Código fuente 14), que consiste en utilizar como origen de los datos un array, en esta página deseamos mostrar en una lista desplegable (control Web DropDownList) el contenido de un array.

```
<html>
<%
       ArrayList datos=new ArrayList();
       datos.Add("Bora-bora");
       datos.Add("Morea");
       datos.Add("Pascua");
       datos.Add("Papeete");
       datos.Add("Santiago de Chile");
       lista.DataSource=datos;
       lista.DataBind();
%>
<body>
<form id="DataBindingSencillo" method="post" runat="server">
Destinos: <asp:DropDownList id="lista" runat="server"></asp:DropDownList>
</form>
</body></html>
```

Código fuente 14

El array se crea y se enlaza con el control cuando se carga la página ASP .NET, el resultado es el que aparece en la Figura 14.

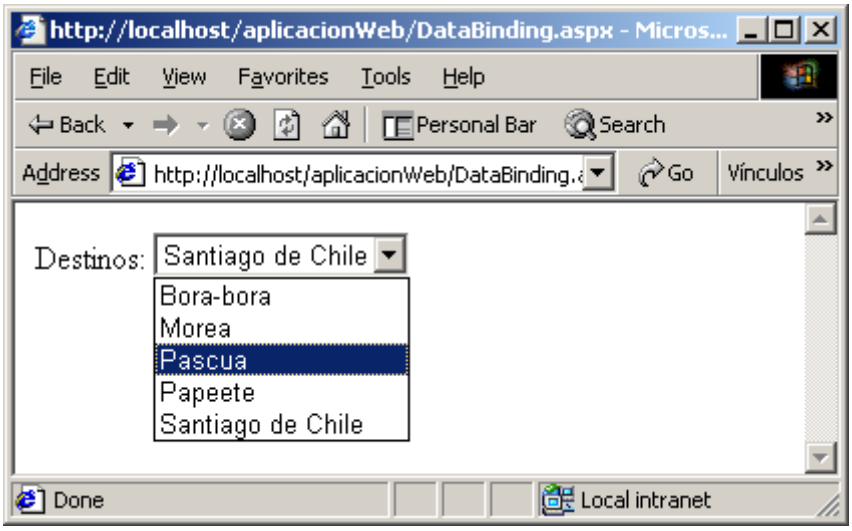


Figura 14

Este mecanismo de enlace con los datos es especialmente útil cuando estamos desarrollando páginas ASP .NET que utilizan bases de datos, es decir, páginas que hagan uso de ADO .NET.

Eventos de la página

ASP .NET ofrecen una serie de eventos que nos permiten estructurar el código en las páginas, estos eventos no existían en versiones anteriores de ASP. Los eventos se producen en un orden determinado, el primero de estos eventos es el evento Page_Init, al que se sigue el evento Page_Load, este evento se produce siempre que se carga la página. A continuación se producirán todos los eventos que pertenecen a los controles (controles de servidor ASP .NET) de los Web Forms, y cuyo orden dependen de la interacción del usuario con los mismos.

Cuando se finaliza la ejecución de la página se lanza el método Page_UnLoad, este será siempre el último evento de una página.

Relacionada con los eventos de las páginas nos encontramos la propiedad IsPostBack del objeto Page, esta propiedad tendrá el valor verdadero si la página ha sido enviada al servidor alguna vez, por ejemplo mediante un botón de submit. Así si deseamos indicar si una página ha sido ya enviada al servidor o no, utilizaremos el Código fuente 15.

```
<script language="C#" runat="server">
void Page_Load(Object fuente,EventArgs evento){
      if(Page.IsPostBack)
            mensaje.Text="Página enviada al servidor";
      else
            mensaje.Text="La página no se ha enviado al servidor";
}
</script>

<html>

<body>

<form method="POST" id="formulario" runat="server">
<asp:label id="mensaje" runat="Server"></asp:label>
<asp:button id="boton" runat="Server" text="Enviar"/>
```

```
</form>
</body>
</html>
```

Código fuente 15

Si nos remontamos al ejemplo del apartado anterior, podemos modificar la página ASP .NET para que únicamente se realice la conexión con los datos (Data Binding) la primera vez que se cargue la página, es decir, se va a utilizar el método Page_Load. Esto se puede ver en el.

En el próximo apartado se va a comentar otra característica de ASP .NET que permite acometer el desarrollo de aplicaciones Web de forma más sencilla, se trata de la posibilidad de utilizar la separación de código, para que la página ASP :NET sea lo más sencilla posible y se ocupe de mostrar el interfaz al usuario, estando separada de la lógica de la aplicación.

Separación de código

ASP .NET permite realizar una clara separación entre la lógica de las páginas y el interfaz, para ello se pueden utilizar varios mecanismos: Code-Behind, controles de usuario y componentes. Cualquiera de estos mecanismos nos ofrecen las siguientes ventajas:

- La posibilidad de realizar una clara división del trabajo entre los programadores y los diseñadores.
- Permite a los desarrolladores o programadores utilizar sus entornos preferidos.
- Los autores de código HTML pueden utilizar sus herramientas de diseño de páginas Web.

La más sencilla de las posibilidades es Code-Behind (código por detrás o código oculto). Este nuevo mecanismo nos permite separar el interfaz de usuario de la lógica de la aplicación, de esta forma el código fuente puede residir en un fichero separado, que puede ser una clase de C#, este código es el que se denomina Code-Behind, ya que permanece oculto.

Para indicar que una página ASP .NET va a utilizar una clase determinada (como Code-Behind), se utilizan los atributos Inherits y Src de la directiva Page. En el atributo Inherits debemos indicar el nombre de la clase y en el atributo Src la ruta del fichero de la clase. Si no indicamos la ruta ASP .NET buscará la clase en el directorio bin de la aplicación. Una vez indicado en la página ASP .NET la clase que se va a utilizar, únicamente nos queda definir e implementar dicha clase.

La otra forma de separar el código de la presentación es mediante los controles de usuario. Los controles de usuario permiten de una manera sencilla reutilizar y separar funcionalidades comunes a través de una aplicación Web completa. Los controles de usuario se utilizan dentro de las páginas ASP .NET de la misma que forma que se utilizan los controles de servidor. Para incluir y utilizar un control de usuario en una página se debe hacer uso de la directiva Control.

Los controles de usuario vienen a sustituir a los ficheros include que se utilizaban en versiones anteriores de ASP. La diferencia más notable entre los controles de usuario y el código oculto (Code-Behind) es que los controles permiten generar parte del interfaz de usuario y el código oculto únicamente permite la utilización de métodos a través del mecanismo de herencia.

Y la última de las posibilidades de separación de código es mediante el uso de componentes. Un componente básicamente es una clase que puede ser instanciada desde una página ASP .NET que

debe importar el componente. Por lo tanto para utilizar un componente se debe utilizar la directiva Import.

Un componente físicamente es un fichero .DLL. En la plataforma .NET los componentes reciben también el nombre de assembly. Un assembly es una colección de recursos que trabajan juntos y que poseen una funcionalidad determinada atendiendo a unas reglas, es la unidad básica a la hora de distribuir un componente. Un assembly se puede traducir como unidad de ensamblaje o unidad simple. Por lo tanto a la hora de generar un componente se dará lugar a un assembly.

En el capítulo correspondiente profundizaremos en las tres posibilidades de separación de código que ofrece ASP .NET.

Los servicios de caché

ASP .NET implementa una serie de servicios de caché a dos niveles, uno se denomina caché de salida y otro caché ASP .NET. El primero de estos servicios de caché nos permite configurar la forma en la que generan la salida las páginas ASP .NET dinámicas que son las mismas para distintos usuarios. Este mecanismo de caché almacena la salida generada por las páginas, y utiliza esta copia almacenada para las siguientes peticiones de la página.

Este tipo caché únicamente será útil si la salida de la página para distintos usuarios es la misma, si se debe generar una salida diferente por cada usuario distinto que acceda a la página no se debe utilizar este mecanismo de caché de salida. ASP .NET puede detectar la cadena de consulta (query string) que se añade a la URL de la página que se demanda, y utilizará una copia almacenada en la caché únicamente si coinciden exactamente los valores del query string.

Para poder utilizar la caché de salida se debe indicar mediante la directiva OutputCache, esta directiva posee un atributo llamado Duration, en el que indicamos en segundos el tiempo que va a permanecer en la caché una página ASP .NET determinada.

El valor por defecto del atributo Duration de la directiva OutputCache es de cero segundos, es decir, por defecto no se utiliza caché de salida.

El otro tipo de caché, denominado caché ASP .NET, que ofrece ASP .NET es la que permite almacenar nuestros propios valores y objetos para ser reutilizados entre distintas páginas ASP .NET. La caché es global a toda la aplicación, por lo tanto la información se puede acceder desde todas las páginas. Los métodos utilizados para acceder a la caché son seguros en lo que a accesos concurrentes se refiere, no es necesario utilizar bloqueos a la hora de modificar o recuperar valores.

Para utilizar este tipo de caché ASP .NET nos ofrece el nuevo objeto Cache, este objeto lo utilizaremos de forma muy similar a los objetos Application y Session de las anteriores versiones de ASP.

Al igual que el resto de objetos integrados de las anteriores versiones de ASP (Request, Response, Session, Application y Server) obtendremos la referencia al nuevo objeto Cache a través de una propiedad de la página, es decir, a través de una propiedad de la clase Page perteneciente al NameSpace System.Web.UI.

Este apartado finaliza el conjunto de apartados dedicado a mostrar de forma breve las nuevas características que ofrece ASP .NET. En el siguiente y último apartado veremos como crear nuestra primera página ASP .NET, que mostrará el típico mensaje “Hola Mundo”.

Hola Mundo con ASP .NET

Una vez realizada la introducción a ASP .NET y a algunas de las características que ofrece, vamos a centrarnos en realizar una página ASP .NET que va a mostrar el famoso mensaje “Hola Mundo” cuando pulsemos sobre el botón incluido en la página.

El Código fuente 16 se corresponde con el código de la página ASP .NET, primero se muestra el código completo y a continuación lo vamos a comentar detenidamente.

```
<html>
<body>
<script language="c#" runat="server">
void Pulsado(Object fuente, EventArgs args){
      etiqueta.Style["font-size"]="20";
      etiqueta.Text="¡Hola Mundo!";
}
</script>
<form id="formulario" method="post" runat="server">
      <asp:label  id="etiqueta" runat="Server"></asp:label>
      <asp:button id="boton" onclick="Pulsado" runat="server"
text="Pulsa"></asp:button>
</form>
</body>
</html>
```

Código fuente 16

Si ejecutamos la página ASP .NET del ejemplo y pulsamos sobre el botón mostrará el aspecto de la Figura 15.

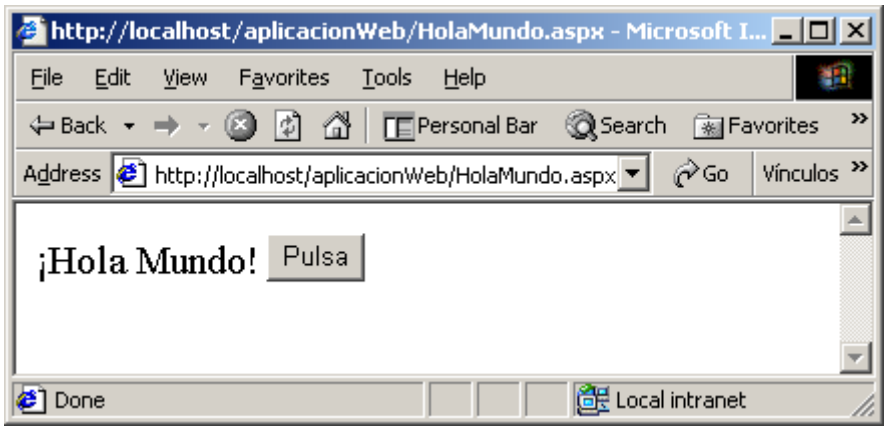


Figura 15

Lo primero que nos puede llamar la atención del Código fuente 16 es el fragmento de código que parece entre las etiquetas <script>. Este fragmento de código escrito en el lenguaje C# define un método que se ejecutará cuando el usuario pulse sobre el botón que pertenece al Web Form contenido en la página ASP .NET. El Web Form lo trataremos más tarde, ahora nos vamos a centrar en la definición del método Pulsado(), que es le método de tratamiento el evento de pulsación del botón.

Una vez que indicamos el lenguaje que se va a utilizar en la etiqueta <script> se pasa a definir la cabecera del método Pulsado(). En este caso este método no devuelve ningún valor (void) y posee dos

parámetros, que en este caso nos ofrecen información acerca del evento que se ha producido, no se debe olvidar que el método que estamos definiendo es un método utilizado para el tratamiento de eventos.

Este método posee dos sentencias. La primera de ellas accede al estilo de la etiqueta del Web Form para darle un tamaño determinado, y la segunda establece el valor de la propiedad Text de la etiqueta con el mensaje "¡Hola mundo!".

El método Pulsado() es muy breve, pero dentro de él se pueden apreciar algunas de las características del lenguaje C#, como el delimitador de sentencias, el punto y coma(;), y el delimitador de bloques de código, los corchetes ({}).

Una vez finalizada la definición del método y del cierre de las etiquetas <script>, nos encontramos con la definición de un Web Form (formulario Web), que será el que va a contener el botón. La sintaxis es muy similar a la utilizada para definir un formulario típico de HTML, la diferencia se encuentra en la propiedad runat que posee el valor server, es decir, indica que es un formulario de servidor, un Web Form.

Entre las etiquetas <form> que definen el formulario Web, se encuentran las etiquetas con sintaxis XML que instancian dos controles ASP .NET. Se trata de dos controles Web que pertenecen a las clases Label y Button del espacio calificado System.Web.UI.WebControls.

El control Label representa a la etiqueta que mostrará el mensaje en pantalla y el objeto Button es el botón que invocará el método Pulsado() cuando el botón sea pulsado por el usuario. Los dos controles Web poseen un atributo id, este atributo es el nombre que se utilizará para acceder a ellos, en el método Pulsado() utilizamos el identificador "etiqueta" para acceder al objeto Label.

El objeto Button posee la propiedad onclick en la que indicamos el método responsable de tratar el evento de pulsación del botón, en este caso se trata del ya conocido método Pulsado().

Una vez creados los controles ASP .NET se cierra el Web Form con la etiqueta correspondiente.

Como se puede observar en el método Pulsado() no se utiliza el objeto Request ni su colección Form para acceder a los elementos del formulario, ya que mediante los Web Forms tenemos acceso a todos los controles, ya que forman parte de la página comportándose como propiedades de la misma.

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

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

Con este último apartado damos por finalizado este capítulo de introducción a la tecnología ASP .NET. En el siguiente capítulo veremos una serie de conceptos generales sobre la orientación a objetos y los lenguajes orientados a objetos, y en un tercer capítulo trataremos el lenguaje que vamos a utilizar en las páginas ASP .NET, es decir, el nuevo lenguaje de Microsoft C# (C Sharp).

3

Introducción a la POO

Introducción

Este capítulo se ha incluido en este texto para tratar el paradigma de la programación a objetos, sobre el que se apoya el lenguaje C#. En este capítulo vamos a tratar aspectos genéricos de la programación orientada a objetos, es decir, son independientes del lenguaje de programación utilizado, pero necesarios para poder utilizar correctamente el lenguaje que vamos a utilizar en nuestras páginas ASP .NET, es decir, el lenguaje C#.

Este capítulo va a ser sobre todo teórico, pero necesario para después poder desarrollar nuestras páginas ASP .NET.

¿Qué es la POO?

Las siglas POO se corresponden con Programación Orientada a Objetos, aunque muchas veces las podemos encontrar escritas en inglés OOP (Object Oriented Programming). En este capítulo vamos a tratar de explicar de forma sencilla los principales conceptos y términos que se utilizan dentro de este tipo de programación, es decir, dentro de la Programación Orientada a Objetos. No vamos a entrar en sesudas divagaciones filosóficas respecto a la POO, sino que vamos a definir lo más claramente posible cada uno de los elementos clave que parecen en la POO para después poder aplicarlos al lenguaje que vamos a utilizar en ASP .NET, es decir, al lenguaje C#.

Este tema es muy necesario debido a que el lenguaje C# es un lenguaje que se basa en la Programación Orientada a Objetos, por lo tanto para conocer el lenguaje C#, y en general ASP .NET y la plataforma .NET, es necesario conocer la Programación Orientada a Objetos.

La Programación Orientada a Objetos trata de utilizar una visión real del mundo dentro de nuestros programas. La visión que se tiene del mundo dentro de la POO es que se encuentra formado por objetos.

Para comprender bien la POO debemos olvidar un poco la Programación Estructurada, que si nos fijamos bien es algo artificial, la POO es una forma de abordar los problemas más natural. Aquí natural significa más en contacto con el mundo real que nos rodea, de esta forma si queremos resolver un problema determinado, debemos identificar cada una de las partes del problema con objetos presentes en el mundo real.

En esta definición de POO ya estamos haciendo referencia al elemento clave de la misma: el objeto. El objeto va a ser la modelización de los objetos que nos encontramos en el mundo real, estos objetos los vamos a utilizar en nuestros programas para dar la solución al problema que nos ocupe en cada caso.

Objetos

Como ya hemos adelantado un objeto es la pieza básica de la POO, es una representación o modelización de un objeto real perteneciente a nuestro mundo, por ejemplo, podemos tener un objeto perro que represente a un perro dentro de nuestra realidad, o bien un objeto factura, cliente o pedido. Los objetos en la vida real tienen todos, dos características: estado y comportamiento. El estado de un objeto viene definido por una serie de parámetros que lo definen y que lo diferencian de objetos del mismo tipo. En el caso de tener un objeto perro, su estado estaría definido por su raza, color de pelo, tamaño, etc. Y el comportamiento viene definido por las acciones que pueden realizar los objetos, por ejemplo, en el caso del perro su comportamiento sería: saltar, correr, ladrar, etc. El comportamiento permite distinguir a objetos de distinto tipo, así por ejemplo el objeto perro tendrá un comportamiento distinto a un objeto gato.

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

Los parámetros o variables que definen el estado de un objeto se denominan atributos o variables miembro y las acciones que pueden realizar los objetos se denominan métodos o funciones miembro, y para indicar variables miembro y funciones miembro se utiliza el término general miembro.

Si lo comparamos con la programación estructurada podríamos hacer la siguiente aproximación: los atributos, propiedades o variables miembro serían variables y los métodos o funciones miembro procedimientos y funciones.

A partir de ahora y a lo largo de todo este capítulo vamos a utilizar únicamente la nomenclatura de atributos y métodos. Los atributos de un objeto deben encontrarse ocultos al resto de los objetos, es decir, no se va a poder acceder directamente a los atributos de un objeto para modificar su estado o consultarlo. Para acceder a los atributos de un objeto se deben utilizar métodos. Es decir, los métodos exponen toda la funcionalidad del objeto, mientras que los detalles del estado interno del objeto permanecen ocultos. Incluso algunos métodos también pueden permanecer ocultos..

En el caso del lenguaje C# se ofrecen dos métodos especiales de acceso (get() y set()) que permiten definir propiedades que hacen referencia a los atributos internos del objeto, algunos autores denominan a estos atributos que permanecen ocultos campos de propiedad, ya uqe internamente van a representar el valor de las propiedades del objeto.

El hecho de ocultar la implementación interna de un objeto, es decir, como está construido y de que se compone se denomina encapsulación. La encapsulación es uno de los beneficios y particularidades del paradigma de la Programación Orientada a Objetos.

Normalmente un objeto ofrece una parte pública que será utilizada por otros objetos para interactuar entre sí, pero también permanece una parte oculta para encapsular los detalles de la implementación del objeto.

Ya se ha dicho que un objeto está compuesto de atributos y métodos. Como la caja negra de un avión, el objeto recubre la información que almacena y solamente podemos obtener la información e indicarle que realiza acciones por medio de lo que comúnmente se denomina interfaz del objeto, que estará constituido por los métodos públicos (y las propiedades en el caso del lenguaje C#).

Los datos y la implementación queda oculta a los demás objetos que interaccionan en el programa, lo que favorece enormemente la protección de los datos y las estructuras internas contra las modificaciones externas al objeto. De este modo es mucho más sencillo localizar errores en los programas puesto que cada objeto está altamente especializado, y sólo se encarga de su tarea. Como se puede observar, esto consigue una mayor modularidad, que facilita además el diseño en equipo de programas y la reutilización de clases (componentes) creados por otros desarrolladores.

Hemos indicado que la encapsulación es un beneficio que nos aporta la POO, es un beneficio porque permite abstraernos de la utilización de los objetos, es decir, a mí me interesa realizar una determinada tarea con un objeto (por ejemplo imprimir una pantalla o rellenar una factura), pero a mí no me interesa como realiza este proceso internamente el objeto que estoy utilizando. En este momento entra en juego otro concepto importante de la POO y que es la abstracción.

La abstracción indica la capacidad de ignorar determinados aspectos de la realidad con el fin de facilitar la realización de una tarea. Nos permite ignorar aquellos aspectos de la realidad que no intervienen en el problema que deseamos abordar, y también nos permite ignorar los aspectos de implementación de los objetos en los pasos iniciales, con lo cual sólo necesitamos conocer qué es lo que hace un objeto, y no cómo lo hace, para definir un objeto y establecer las relaciones de éste con otros objetos.

Un objeto lo podríamos representar como dos circunferencias, una interna que permanece oculta al mundo exterior y que contendría todos los detalles de la implementación del objeto, y otra circunferencia concéntrica externa, que representa lo que el objeto muestra al mundo exterior y le permite utilizar para interactuar con él. En la Figura 16 se puede ver dicha representación.

La encapsulación ofrecida a través de objetos tienen varios beneficios, entre los que destacan la modularidad y la ocultación de la información. Mediante la modularidad podemos escribir código de manera independiente de cómo se encuentren construidos los diferentes objetos que vamos a utilizar. Y ocultando la información se permite realizar cambios en el código interno de los objetos sin que afecte a otros objetos que los utilicen o dependan de ellos. No es necesario entender la implementación interna de un objeto para poder utilizarlo.

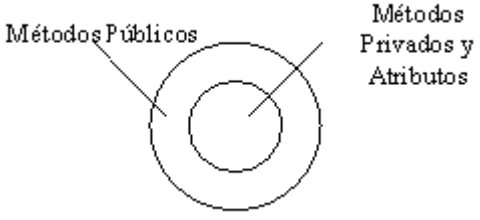


Figura 16

Mensajes

Los mensajes son la forma que tienen de comunicarse distintos objetos entre sí. Un objeto por sí sólo no es demasiado útil, sino que se suele utilizar dentro de una aplicación o programa que utiliza otros objetos.

El comportamiento de un objeto está reflejado en los mensajes a los que dicho objeto puede responder. Representan las acciones que un determinado objeto puede realizar.

Un mensaje enviado a un objeto representa la invocación de un determinado método sobre dicho objeto, es decir, la ejecución de una operación sobre el objeto. Es la manera en la que un objeto utiliza a otro, el modo en el que dos objetos se comunican, ya que la ejecución de ese método retornará el estado del objeto invocado o lo modificará.

Los mensajes se utilizan para que distintos objetos puedan interactuar entre sí y den lugar a una funcionalidad más compleja que la que ofrecen por separado. Un objeto lanzará o enviará un mensaje a otro objeto si necesita utilizar un método del segundo objeto. De esta forma si el objeto A quiere utilizar un método del objeto B, le enviará un mensaje al objeto A.

Para enviar un mensaje se necesitan tres elementos: el objeto al que se le va a enviar el mensaje, el nombre del método que se debe ejecutar y los parámetros necesarios para el método en cuestión.

Clases

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

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

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

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

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

Las clases ofrecen el beneficio de la reutilización, utilizaremos la misma clase para crear distintos objetos. Y luego veremos que una vez que tenemos una clase podremos aprovecharla heredando de ella para complicarla o especializarla para una labor concreta.

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

Herencia

La herencia es un mecanismo mediante el cual podemos reutilizar clases ya definidas. Es decir, si tenemos una clase botón que define un tipo de objeto que se corresponde con un botón que tiene un texto, que se puede pulsar, etc., si queremos definir una nueva clase llamada botón de color, no tenemos que rescribir todo el código y crear una clase completamente nueva, sino lo que haremos será heredar de la clase botón, utilizar lo que nos ofrezca esta clase y añadirle lo que sea necesario para la nueva funcionalidad deseada.

La herencia dentro de la POO es un mecanismo fundamental que se puede definir también como una transmisión de las características de padres a hijos. Entendiendo aquí características como métodos y atributos de una clase. La clase hija puede añadir atributos, métodos y redefinir los métodos de la clase padre.

Podemos ver la herencia como una sucesiva especialización de las clases. La clase de la que se hereda se suele denominar clase padre o superclase, y la clase que hereda se denomina clase hija o subclase. El mecanismo de herencia es muy potente, puesto que nos permite agregar funcionalidades nuevas a una clase ya existente, reutilizando todo el código que ya se tenga disponible de la clase padre, es decir, se heredarán sus atributos y métodos, como ya habíamos indicado con anterioridad. En las clases hijas podemos redefinir el comportamiento de la clase padre.

Para verificar que la herencia entre dos clases es correcta y coherente, debemos hacernos la pregunta de *"¿es un*?" o *"¿es un tipo de*?". Por ejemplo, en el caso que comentábamos del botón, tenemos una clase BotonColor que hereda de la clase Boton. Esta herencia contesta perfectamente a la pregunta definida antes: *¿un botón de color es un botón*?, evidentemente sí.

Mediante el mecanismo de herencia podemos definir superclases denominadas clases abstractas que definen comportamientos genéricos. De esta clase pueden heredar otras clases que ya implementan de forma más concreta estos comportamientos. De esta forma podremos crear jerarquías de clases.

Así por ejemplo podemos tener una clase bicicleta, que será más o menos genérica. De esta clase pueden heredar la clase bicicleta de montaña, bicicleta de carreras y tandem, que ya ofrecen una clase más especializada que la clase padre.

Los métodos que se heredan de la clase padre no tienen porqué utilizarse sin realizar ningún cambio, se puede llevar a cabo lo que se denomina la sobrescritura de métodos. Podemos heredar un método de la clase padre, pero en la clase hija le podemos dar una implementación diferente para que se adecue a la nueva clase.

La herencia puede ser simple si la clase hija hereda de una única clase padre o múltiple si hereda de varias clases padre, más adelante veremos que el tipo de herencia que soporta C# es una herencia simple. Algunas veces la herencia múltiple puede llegar a ser confusa.

Métodos

Como ya hemos dicho anteriormente los métodos son las acciones que se pueden realizar con los objetos. También se podría definir un método como la implementación de un mensaje, al fin y al cabo, un mensaje es la llamada o invocación de un método de un objeto.

Existen dos métodos especiales dentro de la POO que se denominan constructor y destructor. El método constructor se ejecuta automáticamente cada vez que se crea un objeto de la clase en cuestión, sobre el objeto que acaba de crearse, inmediatamente después de haberse asignado memoria a dicho objeto. Algunos lenguajes proporcionan un constructor por defecto, pero generalmente lo correcto es que lo defina el diseñador de la clase y que en él se lleven a cabo las inicializaciones y todas aquellas operaciones que se necesiten para poder usar el objeto.

El método destructor se invoca automáticamente inmediatamente antes de liberar la memoria del objeto en cuestión, o lo que es lo mismo, antes de que se salga del ámbito de la declaración del objeto, por lo que se ha de emplear para que la destrucción del objeto se efectúe correctamente y contendrá operaciones tales como liberación de memoria asignada dinámicamente dependiente del objeto, grabación de todos o parte de los atributos del objeto en un fichero o base de datos y operaciones similares.

Polimorfismo

El término polimorfismo expresa la posibilidad de que el mismo mensaje, enviado a objetos distintos, ejecute métodos distintos. Esto significa que podemos definir dentro de dos clases distintas dos operaciones con el mismo nombre y aspecto externo, pero con distintas implementaciones para cada clase.

En el momento de emplear estas operaciones, el lenguaje es capaz de ejecutar el código correspondiente dependiendo de la clase del objeto sobre el que se ejecuta la operación. Esto permite definir un interfaz común, un aspecto externo idéntico, para una serie de clases.

De esta forma podemos definir un método suma() para la clase Enteros y otro método suma() para la clase Matrices. El mensaje será el mismo, pero la implementación de los métodos será distinta ya que no es lo mismo sumar enteros que matrices.

Sobrecarga

La sobrecarga de métodos se produce cuando una clase tiene métodos con el mismo nombre pero que difieren o bien en el número o en el tipo de los parámetros que reciben dichos métodos.

Un ejemplo de sobrecarga los podemos tener en el método imprimir(), mediante este método vamos a imprimir en pantalla y realizar un salto de línea. Este método acepta como parámetro una variable de tipo entero o una variable de tipo cadena de caracteres, incluso si no indicamos ningún parámetro realizaría un salto de línea.

La ley de Demeter

En lo que respecta a la implementación de métodos de una clase, existe una ley que nos da una serie de pautas para realizar una Programación Orientada a Objetos correcta, esta ley es la Ley de Demeter.

La Ley de Demeter, enunciada por Karl Lieberherr, determina el acceso o visibilidad de los objetos en la implantación de un método, y se rige por el siguiente principio: en la implantación de un método se puede tener acceso a los siguientes objetos:

- Atributos de su clase.
- Los parámetros que se pasan al método.
- Objetos temporales creados dentro del método.

Existen dos formas de la Ley de Demeter, la flexible y la estricta, la flexible permite acceder a los atributos de la clase padre o superclase y la estricta indica que este acceso se debe lograr mediante la utilización de métodos de acceso de la clase padre.

Modelo de objetos

El modelo de objetos es un conjunto de principios que se deben dar para que se puedan modelar objetos computacionales (objetos dentro de nuestro código y programas) a partir de objetos de la realidad de manera que éstos reflejen los posibles comportamientos presentes en la realidad y compongan un modelo computacional válido.

Es decir, definido e identificado un problema se trata de realizar una representación lógica de los objetos que forman parte del problema en el mundo real, de esta forma podremos utilizar en nuestro programa estos objetos para dar la solución a través de nuestro código.

El modelo de objetos que definen un problema de la realidad se basa en los siguientes principios: abstracción, encapsulación, herencia y polimorfismo. Todos estos conceptos básicos de la POO ya han sido discutidos en este tema en mayor o menor medida.

Por lo tanto antes de empezar a construir un programa para una tarea específica, es necesario construir y diseñar su modelo de objetos. Existen diversas técnicas de modelización, pero el objetivo de este texto no es el de explicarlas, así que no entraremos en más detalles.

Relaciones entre clases

Una clase por sí sola no ofrece una funcionalidad demasiado interesante, como ya veremos a lo largo del curso, el lenguaje Java se encuentra formado por un gran número de clases que forman parte de una compleja y completa jerarquía. Cada una de las clases está especializada en una función o tarea específica, es decir, se da una gran modularidad, cada clase tiene su cometido.

Por lo tanto para ofrecer una mayor funcionalidad y realizar tareas más complejas es necesario que exista una relación entre distintas clases. Un ejemplo podría ser las piezas del motor de un coche. Las piezas por si solas no realizan ninguna tarea de importancia, cuando realmente se saca provecho de ellas es cuando se relacionan entre sí y se construye con ellas un motor. A su vez este motor puede ser incorporado dentro de un coche, que estará compuesto a su vez de más objetos o piezas. Como vemos la correspondencia entre los objetos del mundo real y los objetos del mundo computacional es bastante obvia en algunos casos.

Básicamente una clase se puede relacionar con otra de tres formas diferentes. Las relaciones que podemos distinguir son:

- Relación de composición: una clase puede estar compuesta de otras clases. Esto se consigue implementando los atributos de la clase como objetos de otra clase. Una clase hace uso de otra a través de sus atributos, una clase se encuentra formada por varias clases. Por ejemplo la clase Coche tiene atributos que son de la clase Puerta, Rueda, Motor, etc.

- Relación de uso: una clase se relaciona con otra a través de los mensajes que le envía. Esto se consigue pasándose una instancia de la clase como uno de los parámetros del método invocado por el mensaje. Es decir, si tenemos la clase Pantalla con el método dibujar() y queremos dibujar un objeto de la clase Rectangulo, deberíamos realizar lo siguiente objPantalla.dibujar(Rectangulo), es decir, el parámetro que se le pasa al método dibujar() de la clase Pantalla es un objeto de la clase Rectangulo.

- Relación de herencia: este tipo de relación entre clases ya la conocemos y consiste en que una clase hija hereda de una clase padre o superclase pudiendo utilizar así toda la funcionalidad ofrecida por la clase padre y añadir nuevas funcionalidades. En esta forma de relación se consigue la reutilización del código y una progresiva especialización a través de una jerarquía de clases.

Ventajas e inconvenientes de la POO

En este apartado vamos a comentar que ventajas nos ofrece el paradigma de la POO, también mostraremos algunas de sus desventajas. Como ventajas o aportaciones podemos destacar las siguientes:

- Facilita la reutilización del software. A través de la herencia se nos permite utilizar en un objeto las operaciones implementadas para otros sin esfuerzo adicional. Por otro lado la encapsulación nos facilita el uso de objetos que no hemos definido ni implementado. Se puede considerar que la encapsulación y el polimorfismo son las herramientas más potentes del paradigma de la POO.

- Facilita la construcción de programas portables. Es posible diseñar una capa de objetos que se comunique con la máquina o el sistema operativo, y sobre ésta los objetos que dan funcionalidad a la aplicación. Una migración a otra arquitectura sólo requerirá cambios en dicha capa, y la encapsulación nos garantiza que los cambios se van a limitar a esos objetos. C# y la plataforma .NET refuerza más esta característica de la portabilidad, ya que, el lenguaje C# es independiente de la plataforma en la que se ejecuta, generando el código intermedio, que comentaremos en el siguiente capítulo.

- Facilita el mantenimiento. El encapsulamiento nos garantiza que las modificaciones realizadas en un objeto tendrán un efecto limitado. Si un elemento de datos se accede directamente y en un momento dado cambia de tipo o formato, habrá que localizar todos los puntos en los que se accede a dicho elemento y modificarlos en consecuencia. Si este elemento de datos está encapsulado en un objeto y siempre es accedido mediante las operaciones disponibles, un cambio como el indicado se limitará al objeto, del que sólo habrá que cambiar el elemento de datos y las operaciones del objeto que lo utilizan. Lo recomendable es desconocer la implementación interna del objeto, nosotros necesitaremos utilizar un objeto que ofrece una serie de funcionalidades, pero no nos interesa de que forma nos ofrece las mismas.

- Provoca que las tareas de análisis, diseño e implementación sean más intuitivas, ya que se manejan objetos, concepto con el que todos estamos familiarizados, y estructuradas, ya que podemos asignar las tareas de diseño e implementación en el ámbito de objetos. Se debe

recordar que los objetos en nuestros diseños van a representar objetos presentes en el mundoreal.

No todos son beneficios, la POO ofrece también una serie de desventajas, aunque más que desventajas o inconvenientes podemos decir que presenta una serie de dificultades, las cuales se comentan a continuación:

- Curvas de aprendizaje largas, debido principalmente a que implica un cambio mentalidad, y no sólo el aprender un nuevo lenguaje.

- Dificultad en determinar las características de un objeto. Debido a que un objeto no se define aisladamente, sino que depende de las relaciones con otros objetos, el establecimiento de nuevas relaciones puede implicar un cambio en la definición del objeto y viceversa. Por todo ello es conveniente iterar, esto es, repetir el proceso de definición de objetos y de identificación de relaciones, tantas veces como sean necesarias para alcanzar un modelo estable. En muchos casos resulta complicado llegar a un modelo de objetos definitivo con el que se pueda abordar el problema planteado de forma completa.

- Jerarquías de herencia complejas: en muchos casos se tiende a utilizar de forma desmedida el mecanismo de herencia, ofreciendo en algunos casos unas jerarquías de herencia difíciles de seguir y abordar.

Un ejemplo sencillo

No hace falta decir que el capítulo que nos ocupa es un capítulo eminentemente teórico, pero aun así vamos a incluir un breve y sencillo ejemplo con el que se pretende mostrar de forma sencilla como se puede construir un modelo de objetos, identificando los objetos que intervienen en el problema a resolver, definiendo los atributos de la clase, los métodos, relaciones entre las diferentes clases, etc.

En este caso vamos a abandonar los típicos ejemplos de la clase Coche y similares y vamos a mostrar un ejemplo más lúdico. En nuestro ejemplo vamos a tener la clase VideoJuego, es decir, vamos a crear el juego de *matamarcianos* conocido por todos.

No vamos a profundizar demasiado en todos los atributos y métodos de las clases, sino que vamos a mostrar únicamente los más relevantes.

En el primer paso vamos a identificar las clases necesarias para abordar nuestro problema, en este caso, la creación de un juego de matamarcianos. Primero tengamos en cuenta las especificaciones:

- En el juego tenemos dos tipos de enemigos: marcianos y venusianos.

- El protagonista del juego es un terrícola.

- El terrícola dispone de un arma para defenderse de los enemigos, un lanza cohetes.

Como se puede observar no vamos a tener demasiadas complicaciones a la hora definir el modelo de objetos de nuestro programa.

Se pueden distinguir seis clases: VideoJuego, Enemigo, Marciano, Venusiano, Terricola y LanzaCohetes.

La clase VideoJuego sería la clase principal y contendría al resto de las clases. Por lo tanto entre la clase VideoJuego y el resto de las clases existe una relación de composición. Esta clase tendría como

atributos posibles: Enemigos, Heroes y Armas; y como métodos: comenzarPartida(), interrumpir Partida(), reanudarPartida() y finalizarPartida(), la finalidad de cada uno de ellos está bastante clara.

La clase Enemigo en realidad va a ser una clase abstracta, ya que no vamos a instanciar objetos de esta clase. La clase Enemigo va a ser la clase padre de la clase Marciano y Venusiano, y lo que se consigue con esta clase es agrupar todo el comportamiento que es común a los enemigos que aparecen en el juego. Esta clase tiene los siguientes atributos: Color, NumeroOjos y NumeroPiernas. Estos atributos serán comunes a las clases Marciano y Venusiano. Existe una relación de herencia entre la clase Enemigo y las clases Marciano y Venusiano.

La clase Enemigo podría tener los siguientes métodos: mover(), atacar() y disparar(). Estos métodos también serán heredados por la clase hijas mencionadas. Las clases hijas tienen dos opciones, si la implementación de los métodos anteriores que realiza la clase padre es la adecuada, los heredarán y utilizarán sin más, pero si las clases hijas quieren realizar modificaciones o ampliaciones sobre los métodos heredados deberán implementarlos y por lo tanto sobrescribirlos aportando un comportamiento diferente.

Las clases Marciano y Venusiano pueden además de sobrescribir los métodos heredados de la clase Enemigo, aportar nuevos métodos y nuevos atributos. La clase Marciano añade el atributo Visible, es decir, nuestro amigo tiene la capacidad de hacerse invisible, y la clase Venusiano añade el atributo NumeroCabezas.

La clase Terricola representa al héroe de nuestro juego y tiene los atributos: NumeroVidas y Municion, ambos requieren poca explicación. Esta clase implementa el método disparar(), que recibe como parámetro un objeto de la clase LanzaCohetes, la relación entre ambas clases es una relación de uso.

La clase LanzaCohetes que representa el arma que va a utilizar la clase Terricola y tienen el atributo NumeroCohetes, y posee el método lanzarCohete().

Podríamos entrar en más detalles e ir afinando más en la identificación de objetos, podríamos definir también la clase Cohete, que representaría a los cohetes lanzados por una instancia de la clase LanzaCohetes. Pero no vamos a entrar en más detalles y vamos a dejar de definir más clases, quedándonos con las vistas hasta ahora.

En la Figura 17 se trata de mostrar un esquema de todas las clases que implementan el problema propuesto y la relación existente entre cada una de ellas. Cada clase se representa por una circunferencia y las relaciones existentes mediante flechas.

Con este ejemplo damos por finalizado el capítulo dedicado a la introducción de la programación orientada a objetos. De todas formas si el lector tiene dudas, puede profundizar en el estudio de la POO a través de otros textos.

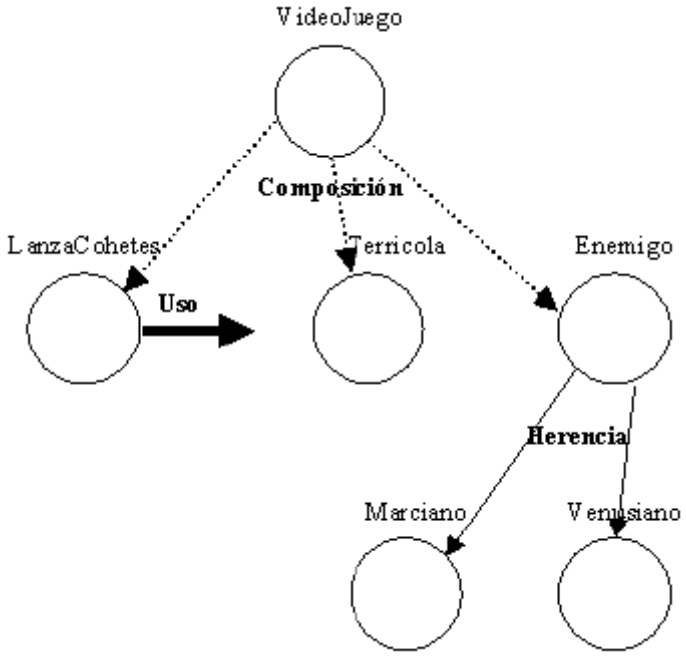


Figura 17

En el siguiente capítulo veremos la sintaxis básica del lenguaje C#, así con los fundamentos teóricos vistos en este capítulo y la sintaxis del lenguaje que veremos en el siguiente, podremos más adelante empezar a escribir nuestras primeras páginas ASP .NET, en las que podremos utilizar el modelo de objetos que ofrecen y los controles de servidor a través de Web Forms.

El lenguaje C#

Introducción

En este capítulo vamos a tratar la sintaxis básica del lenguaje C#, (C-Sharp) es decir, no pretende ser un tutorial completo del nuevo lenguaje de Microsoft, ya que no es el objetivo del presente texto, sino que vamos a tratar los aspectos del lenguaje que nos van a interesar para poder utilizarlo dentro de ASP .NET.

Ya se comentó en el capítulo anterior que ASP .NET ofrece soporte directamente para tres lenguajes distintos: C#, Visual Basic .NET y JScript. Se ha seleccionado el lenguaje C# por ser el más interesante desde el punto de vista de la programación orientada a objetos y por haber sido este lenguaje el utilizado para construir el 90% de la herramienta de desarrollo de Microsoft para la plataforma .NET, es decir, Visual Studio .NET, y también ha sido utilizado para implementar de forma completa la tecnología ASP .NET, por lo tanto se puede decir que C# es un lenguaje de mucho peso dentro de la plataforma .NET.

Vamos a tratar el lenguaje C#, pero no en profundidad, sino que nos centraremos en la sintaxis y otras características del lenguaje para que podamos utilizarlo sin ningún problema dentro de las páginas ASP .NET.

Primero se va a ofrecer una visión general del lenguaje en la que se va a realizar una introducción al mismo y se comentarán las características que presenta, y a continuación se tratará su sintaxis.

Historia reciente del lenguaje C#

El lenguaje C# se presenta como el último invento en materia de lenguajes de programación, y constituye también la más reciente y ambiciosa apuesta en este sentido por parte de Microsoft. Quizás, lo primero que habría que aclarar, es que, de todo el .NET Framework, fue la primera parte que se finalizó, hasta el punto de que el propio Visual Studio .NET ha sido construido al 90% en C# y el 10% restante en C++. Por otro lado, el lenguaje merece el calificativo de estándar, en el sentido de que (al igual que algunos otros aspectos del entorno) está siendo sometido a estandarización por parte de ECMA, la misma entidad de normalización que llevó a cabo la estandarización de JavaScript**.**

Es natural que, ante la creciente comunidad mundial de profesionales de las TI (Tecnologías de la Información), la aparición de un nuevo lenguaje tenga más impacto hoy que en etapas anteriores, más arcaicas. Los medios lo favorecen, y la difusión de Internet, también. Por eso, nos hemos paseado por algunas revistas en la Web donde los gurús mas conocidos dan sus opiniones.

La impresión de estos autores podríamos resumirla así: C# supone una mejora respecto a otros lenguajes existentes por dos razones básicas: primero, por que es el último, y por lo tanto, el más adaptado a las necesidades actuales del programador y el que más ha aprendido de los demás, heredando lo mejor de cada entorno, y añadiendo las cosas que los programadores solicitaban. En segundo término, por que su creador principal (el jefe del equipo de desarrollo, el danés Anders Hejlsberg), ha podido hacer un excelente trabajo basado su experiencia personal (es el diseñador de Turbo Pascal 5.5 y Delphi), con tiempo suficiente, y un pequeño pero impresionante equipo de colaboradores entre los que figuran Peter Kukol, Jeffrey Richter, etc.

De hecho, el propio nombre del lenguaje (se pronuncia CSharp) fue una decisión posterior, como se ha sabido, en el sentido de que era una extensión de C++: C++++ (con 4 +), para indicar su origen principal. Si bien esto es cierto, no lo es menos que en un famoso e-mail que se hizo público a raíz del contencioso Sun-Microsoft, dirigido por Hejlsberg a Bill Gates, se habla del proyecto de desarrollo del nuevo lenguaje con frases esclarecedoras sobre la intención con que se construía.

Así, podemos leer (la traducción es de Marino Posadas) "Peter Kukol y yo hemos estado trabajando en un conjunto de extensiones de Java que yo definiría como un cuidadoso maridaje entre Java y C". Y más adelante, en una significativa referencia al nombre del nuevo lenguaje, afirma "A falta de un nombre mejor, estamos llamando al lenguaje J++", en una clara alusión a sus fundamentos basados también en el lenguaje Java. Finalmente, otra frase entresacada de este e-mail, explica claramente el propósito del diseño al afirmar que "lo que muchos programadores quieren es una versión "limpia" de C++ que aporte los beneficios de productividad de Java combinados con las características de bajo nivel de C**".**

El lenguaje C# y el Entorno Común de Ejecución (CLR)

Una de las características principales de C#, y de ASP .NET, es que se trata de un lenguaje que compila (por defecto) a un formato intermedio, al estilo de Java, denominado Intermediate Language (IL), que posteriormente, debe de ser interpretado por un entorno de ejecución, una máquina JIT (just-in-time), también al estilo de Java. La gran diferencia respecto a Java es que, ése intérprete será común a todos los lenguajes soportados por el entorno de ejecución y mediante este mecanismo permitirá que los componentes realizados en cualquier lenguaje puedan comunicarse entre sí.

Se trata pues, de una extensión del concepto inicial que dio origen a Java: en lugar de un único lenguaje para muchas plataformas, se pretende un entorno común multiplataforma, que soporte muchos lenguajes, basándose en que todos ellos compilen a un mismo código intermedio. Para hacer viable esta idea, se ha optimizado considerablemente la velocidad, respecto a Java y ya se están

anunciando los primeros .NET Framework para otras plataformas: El pasado mes de Marzo, Steve Ballmer anunciaba la disponibilidad para Linux, y están en marcha las implementaciones para Unix, McIntosh System-8 y BEos.

Este lenguaje intermedio, es gestionado por un mecanismo llamado Entorno Común de Ejecución (Common Language Runtime), encargado, además, de la gestión de memoria, y en general, de las tareas más importantes, relacionadas con la ejecución de programas.

Principios básicos de C#

Las bases o principios básicos sobre los que se asienta la construcción de un programa escrito en C# son los siguientes:

- Todo programa compilado para el entorno común de ejecución (CLR) comparte un mismo código intermedio (Intermediate Language ó IL) que debe ser ejecutado por un intérprete JIT que resida en la plataforma de ejecución. Aunque pueden distinguirse entre componentes y ejecutables, en éste último caso, el formato resultante es independiente del lenguaje que se utilizó en su escritura. El formato de los ejecutables independientes se denomina PE (Portable Executable).

- El entorno común de ejecución (CLR) es invocado por cualquier programa escrito en el formato PE, y es quien se hace cargo del mantener la zona de memoria en la que se ejecuta el programa y suministra un sistema de tipos de datos común (Common Type System), que describe los tipos soportados por el intérprete y cómo estos tipos pueden interactuar unos con otros y cómo puede ser almacenados en metadatos.

- Los servicios del sistema operativo se suministran mediante librerías (DLL) dispuestas por el CLR cuando se instala .NET Framework en una plataforma (Windows, McIntosh, Linux, etc.). Dichas librerías son referenciadas en el código de C# mediante la sentencia using seguida del nombre de la librería deseada. Tales declaraciones reciben el nombre de Espacios Calificados o Espacios con Nombre (Namespaces).

- En C#, todo puede ser tratado como un objeto, aunque no obligatoriamente, y sin la penalización que supone esta característica en lenguajes orientados a objetos "puros", como SmallTalk. Por tanto cualquier código en C# estará incluido en una clase, ya sea esta instanciada o no. Además, se protege la inversión realizada previamente en desarrollo, ya que permite la llamada a componentes COM, a librerías nativas del API de Windows (DLL's), permite el acceso de bajo nivel cuando sea apropiado y los ejecutables y componentes producidos con él se integran perfectamente con otros ejecutables o componentes realizados en otros lenguajes del entorno.

- La sintaxis de C# recuerda bastante la de Java, si bien los conceptos implicados en la construcción de objetos, tipos, propiedades, métodos, y eventos no suelen ser idénticos. La mayor similitud se encuentra, como cabía esperar, en el tratamiento de las estructuras de control del programa (prácticamente idénticas a C++ y Java).

Características generales

Existen características comunes a todos los lenguajes del entorno .NET, que lógicamente, ha heredado C#, y otras fruto de la propia concepción del lenguaje, sus herencias de otros lenguajes y sus mejoras. Respecto a las características heredadas cabe destacar:

1) Unidad del sistema de tipos, con presencia de tipos string y bool para la representación de cadenas y valores booleanos respectivamente (C++ no disponía de tal posibilidad). El tipo currency desaparece en detrimento del tipo decimal, que permite operaciones con reales de 128 bits, supliendo con creces esa necesidad. También deja de existir el tipo variant. Todos los tipos tienen un tamaño fijo, independiente del compilador o plataforma y establecido en el Common Type System (CTS).

2) Exquisita implantación de la OOP, sin excepciones. En C#, todo son objetos, incluyendo los tipos predefinidos. Se admite la herencia simple de clases, y la múltiple de interfaces. Todos los objetos derivan de un objeto base: System.Object. La palabra reservada internal permite establecer ámbitos de acceso para los módulos de ensamblado.

3) Gestión de eventos segura y multipropósito, gracias a los delegates.

4) Tratamiento unificado de excepciones, mediante estructuras try-catch-finally**.**

5) Un conjunto completo de directivas para el compilador, posibilidad de establecer atributos que modifiquen comportamientos, describan tipos, controlen versiones, etc.

6) Gestión automatizada de memoria mediante el Garbage Collector (recolector de basura). Como características propias del lenguaje podríamos citar (aunque algunas se encuentren también en otros lenguajes):

7) Sintaxis muy similar a Java y C++, pero simplificada respecto a C++, con un juego de operadores abreviado. Nuevas estructuras de control, como el bucle foreach para recorrer colecciones. Incorpora el concepto de Property que existía en Visual Basic para normalizar el acceso a las propiedades de una clase.

8) Sobrecarga: C# (a diferencia de otros lenguajes del entorno, como Visual Basic, o de fuera del entorno, como Java), permite la sobrecarga de operadores. Dicho proceso es vigilado por el compilador para que los operadores con opuesto deban redefinirse por parejas (< y >, ó = = y !=). También pueden sobrecargarse otros operadores, como el [] de acceso a elementos de un array, permitiendo el trabajo con tipos que se comporten como colecciones de objetos, a través del concepto de indexer (indizador).

9) Comprobación de rango de acceso en Arrays, de conversión de tipos, de inicialización de variables, y de control de desbordamiento (overflow)

10) Código autocontenido, sin necesidad de ficheros adicionales de cabecera o extensiones complementarias, como el lenguaje IDL de definición de interfaces para COM.

11) Posibilidad de incluir en un bloque unsafe { } código fuente estándar (no administrado) de C++, donde pueden manejarse punteros, gestionar directamente memoria, etc.

12) Posibilidad de llamadas a funciones de API's nativas mediante PInvoke (Platform Invocation). Puede usarse esta característica para la comunicación entre objetos COM y .NET.

Espacios con nombre (NameSpaces)

Un espacio con nombre (espacio calificado o nominado, NameSpace) es una especie de firma digital con la que podemos hacer referencia a una librería, entendiendo como librería conjunto de clases relacionadas, o bien hacer que sirva de identificador de todos los módulos que pertenezcan a una

aplicación, o un fragmento de aplicación. System es el NameSpace básico para todas las aplicaciones que requieran algún servicio del sistema (que son prácticamente todas).

El CLR (Common Language Runtime, Entorno Común de Ejecución) pone a disposición de cualquier aplicación todos los servicios de sistema operativo a través de un conjunto jerárquico de clases con sólo hacer referencia a ellas. No obstante, y para facilitar la escritura del código utilizamos el concepto de NameSpace (espacio calificado ó espacio con nombre). Un NameSpace es una forma adecuada de organizar clases y otros tipos de datos en una jerarquía. Podríamos decir que sirven al mismo tiempo para 3 propósitos: organizar (dentro de la jerarquía), para firmar (o marcar de forma única) fragmentos de código como pertenecientes a ese NameSpace, evitando colisiones con otros fragmentos que puedan contener definiciones iguales, y como método de abreviación, ya que una vez declarado el NameSpace podemos referirnos a sus contenidos en el código sin necesidad de repetir toda la secuencia jerárquica.

El concepto de NameSpace es similar al concepto de paquete de Java, básicamente se podría decir que ofrece la misma funcionalidad y que además posee un uso y sintaxis muy parecidas. De hecho, si tomamos la definición de paquete de Java: “Los paquetes es una forma de clasificar y organizar clases e interfaces, para evitar conflictos de nombres, para localizar una clase de forma más sencilla y rápida y para controlar el acceso a las clases”, como se puede comprobar se puede aplicar también a los NameSpaces del lenguaje C#.

Sintaxis y elementos básicos del lenguaje C#

Este apartado pretende ser una referencia rápida de la sintaxis del lenguaje C#, comentaremos los distintos elementos del lenguaje que serán acompañados por su ejemplo o fragmento de código correspondiente. También veremos los mecanismos de orientación a objetos que nos ofrece el lenguaje C#.

Case-sensitive

El lenguaje C#, al igual que sucede con los lenguajes C o Java, distingue entre mayúsculas y minúsculas, es decir, es case.sensitive. Por lo tanto deberemos ser cuidadosos a la hora de hacer referencia a objetos, métodos, clases, etc. Esto no será ningún problema para los lectores familiarizados con los lenguajes C o Java.

En C# existe una regla a la hora de utilizar las mayúsculas y minúsculas, los métodos y las clases tienen la primera letra de cada una de sus partes en mayúscula, así por ejemplo tenemos el método DataBind() o la clase DropDownList. Los tipos simples del lenguaje C# se escriben con todas sus letras en minúscula.

Realizado este comentario vamos a pasar a describir la sintaxis básica del lenguaje C#.

Declaración de variables

Para declarar una variable, ya sea una referencia a un objeto o a un tipo simple, primero indicamos el tipo de la variable (tipo simple o clase) y a continuación aparece el nombre de la variable. Si la variable posee algún modificador de acceso o de otro tipo, este modificador aparecerá antes del tipo de la variable. En el apartado correspondiente trataremos los modificadores que presenta C#.

En el Código fuente 17 se puede ver la declaración de varias variables, algunas son variables de que pertenecen a tipos simples, y otras son variables que son de una clase de la plataforma .NET. Los tipos simples siempre se escriben con minúsculas y las clases la primera letra de cada una de las palabras que la forman en mayúscula.

```
<script language="C#" runat="server">
String Variable;
int TipoSimple;
public DateTime FechaHora;
bool Fallo;
</script>
```

Código fuente 17

A la hora de definir variables de debe utilizar el delimitador de código de servidor que utiliza las etiquetas <script>.

Delimitador de sentencias

Una sentencia se delimita por un punto y coma (;), esto ya lo hemos visto en el ejemplo del apartado anterior.

Comentarios

Los comentarios pueden ser de una única línea, en este caso se utiliza los caracteres // antes del texto del comentario, sin ningún indicador de fin de comentario. Pero los comentarios pueden agrupar también varias líneas, en este caso el comentario irá entre los delimitadores /* */. En el Código fuente 18 se puede ver un ejemplo de uso de estos comentarios.

```
<%//comentario de una línea
/*comentario de
varias líneas*/%>
```

Código fuente 18

Bloques

Al igual que C o Java, C# utiliza las llaves ({} la primera de inicio y la otra de fin de bloque) para determinar los bloques dentro de un programa. Todo lo que se encuentra entre estas dos llaves se considera un bloque. Los bloques son parte de la sintaxis del lenguaje.

Los bloques pueden y suelen anidarse; utilizándose la sangría para clarificar el contenido de un bloque. De este modo, el bloque más externo se sitúa al margen izquierdo del fichero de código fuente, y cada vez que se anida un bloque se indenta (sangra) el texto que lo contienen un número determinado de columnas.

Ámbitos

Los bloques además, definen los ámbitos de las variables. El ámbito se refiere a la longevidad de las variables. Una variable existe sólo dentro del bloque donde ha sido declarada, eliminándola el compilador una vez que se sale de dicho bloque. Esto es cierto para todos los tipos de datos de C#.

Cuando una variable sale de ámbito, es eliminada y la memoria que ésta ocupaba es liberada por el recolector de basura (garbage collector).

Operadores

Entendemos por operador un símbolo que indica que debe realizarse una operación sobre uno o más argumentos, que reciben el nombre de operandos. A los operadores que actúan sobre un único argumento, se les llama operadores unarios. En la Tabla 2 se ofrecen los operadores disponibles en el lenguaje C#, así como su precedencia.

Categoría del Operador	Operadores
Primarios	(x), x.y, f(x), a[x], x++, x--, new, typeof, sizeof, checked, unchecked
Unarios	+, -, !, ~, ++x, --x, (T)x
Multiplicativos	*, /, %
Aditivos	+, -
De desplazamiento	<<, >>
Relacionales	<, >, <=, >=, is
Igualdad	= =
AND Lógico	&
XOR Lógico	^
OR Lógico	|
AND Condicional	&&
OR Condicional	||
Condicional	?:
Asignación	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=

Tabla 2

Como podemos ver, un completo juego de operadores basado principalmente en los existentes para los lenguajes C++/Java. Aunque la mayor parte de ellos, siguen las reglas establecidas para otros

lenguajes, conviene establecer algunas de las reglas generales más importantes que se cumplen en su tratamiento e implantación, y que son las siguientes:

1) La precedencia de los operadores es la que se indica en la tabla adjunta.

2) Todos los operadores, excepto el de asignación (=), se evalúan de izquierda a derecha. Esto significa que se da propiedad transitiva en los operadores. O sea, que la expresión

   A+B+C es equivalente a (A+B)+C. Como siempre, se utilizan los paréntesis para establecer precedencia específica en cualquier expresión. Nótese, además, que en C#, se utilizan distintos símbolos para el operador de asignación (=) y el de comparación (= =), evitando errores de interpretación, como sucede en otros lenguajes, al estilo de Visual Basic.

3) El operador (.) se utiliza para especificar un miembro de una clase o estructura. El operando de la izquierda simboliza la clase y el de la derecha el miembro de esa clase. Se eliminan otros operadores presentes en el lenguaje C++, tales como (->,::), para simplificar la sintaxis.

4) El operador ([]) se utiliza para hacer referencia a los elementos de un array. Todos los arrays en .NET comienzan en 0.

5) Los operadores (++ y --) se utilizan al estilo de C++, como operadores de autoincremento y autodecremento. Pueden ser utilizados dentro de expresiones.

6) new es el operador usado para invocar métodos constructores a partir de definiciones de clases.

7) typeof es el operador utilizado mediante la librería Reflection para obtener información acerca de un tipo.

8) sizeof se utiliza para determinar el tamaño de un tipo estándar expresado en bytes. Solo puede ser utilizado con los llamados value types (tipos por valor), o miembros de clases, pero no con las propias clases. Además, sólo pueden usarse dentro de bloques unsafe (código C++ embebido en un bloque unsafe de C#).

9) checked y unchecked se utilizan para la gestión de errores, dentro de estructuras try-catch-finally. Lo veremos con más detalle, en el apartado de tratamiento de errores.

10) Los operadores de asignación compuesta se utilizan para simplificar expresiones en las que un operando aparece en los dos miembros de la asignación. Por ejemplo, podemos simplificar un contador expresándolo como x += 1, en lugar de x = x + 1.

Acceso a propiedades indexadas

Para acceder a un elemento determinado de una propiedad indexada se utilizarán los corchetes ([]). En el Código fuente 19 se puede ver como se accede a un elemento de la propiedad QueryString.

```
<%Page.Response.Write(Request.QueryString["prueba"]);%>
```

Código fuente 19

Declaración de propiedades simples

La forma de declarar propiedades (atributos o miembros de la clase) de C# es bastante peculiar, la sintaxis general se puede ver en el Código fuente 20.

```
public ClasePropiedad NombrePropiedad{
	get{
		return...;
	}

	set{
		...=value;
	}
}
```

Código fuente 20

La palabra reservada get se utilizará para realizar la lectura de la propiedad, es decir cuando se recupera el valor de la propiedad, y set cuando se va a realizar la escritura de la propiedad, es decir, cuando se modifica la propiedad.

A la hora de modificar (set) o recuperar una propiedad(get) podremos ejecutar las sentencias que sean necesarias, únicamente se deberán incluir estas sentencias en la implementación de los métodos set y get de la propiedad correspondiente.

En el Código fuente 21 se puede ver un ejemplo en el que se define la propiedad de sólo lectura Nombre, y la propiedad de lectura/escritura Apellidos.

```
private String nombre, apellidos;
public String Nombre{
	get{
		return nombre;
	}
}
public String Apellidos{
	get{
		return apellidos;
	}
	set{
		apellidos=value;
	}
}
```

Código fuente 21

Todos los aspectos relacionados con la implementación de clases y la sintaxis específica de la programación orientada a objetos los volveremos a retomar cuando sea necesario a lo largo del texto. En este capítulo veremos los temas relacionados con la creación de clases pero de forma básica, únicamente a nivel genérico.

Arrays

Los arrays, al igual que muchos aspectos de la sintaxis de C#, presentan el mismo aspecto y tratamiento que los del lenguaje Java. En el Código fuente 22 se muestra un ejemplo de uso de array de una y de dos dimensiones que van a contener objetos de la clase String.

```
String[] letras = new String[3];
letras[0]=”a”;
letras[1]=”b”;
letras[2]=”c”;
String[] numeros = new String[3];
numeros[0]=”1”;
numeros[1]=”2”;
numeros[2]=”3”;
```

Código fuente 22

Como se puede comprobar los índices de los arrays comienzan en cero.

Inicialización

Para inicializar objetos se utiliza el operador de asignación (=), y en el caso de utilizar un constructor de una clase se deberá utilizar también el operador new. En el Código fuente 23 se puede observar varios ejemplos de inicialización, tanto de objetos, arrays como de variables de tipos de datos simples.

```
String saludo = “Hola Mundo”;
int i=0;
String [] letras={“a”,”b”,”c”};
ArrayList datos=new ArrayList();
```

Código fuente 23

Sentencias condicionales

En C# disponemos de la sentencia condicional if ...else..., que será conocida por nuestros lectores de otros lenguajes de programación. La sintaxis general de esta sentencia se puede ver en el Código fuente 24.

```
if(condición){
      .....
}else{
      .....
}
```

Código fuente 24

En el Código fuente 25 se puede ver la utilización de la sentencia condicional.

```
if (Request.QueryString["nombre"]!=null){
      Response.Write("nombre="+Request.QueryString["nombre"]);
}else{
      Response.Write("Sin valor");
}
```

Código fuente 25

En el ejemplo anterior también se puede apreciar el operador de concatenación de cadenas de C#, el operador +.

También tenemos en C# como sentencia condicional la sentencia switch...case... Esta sentencia tiene la sintaxis general del Código fuente 26.

```
switch(expresión){
      case expresión_constante:
            ........
            break;
      [default:
            ...
            break;]
}
```

Código fuente 26

En cada condición deberemos indicar la sentencia break sino queremos que se sigan evaluando las siguientes condiciones. La rama default no es obligatoria, pero se ejecutará siempre que no se cumplan ninguna de las condiciones anteriores.

En el Código fuente 27 se puede ver un ejemplo de utilización de la sentencia case.

```
switch (Request.QueryString["numero"]){
      case "Uno":
            Response.Write("1");
            break;
      case "Dos":
            Response.Write("2");
            break;
      case "Tres":
            Response.Write("3");
            break;
      default:
            Response.Write("Otro número");
            break;
}
```

Código fuente 27

Bucles

En C# tenemos dos tipos de bucles for, el bucle for al estilo de Java o C, cuya sintaxis podemos ver en el Código fuente 28, y un bucle foreach similar al de Visual Basic.

```
for(inicialización;expresión;iteradores){
       ...
       sentencias;
       ...
}
```

Código fuente 28

En inicialización se inicializa la variable contador que vamos a utilizar en el bucle, expresión define la condición que se debe cumplir para que no finalice el bucle e iteradores es una sentencia o conjunto de ellas que incrementará o decrementará el contador del bucle.

En el Código fuente 29 se puede ver un ejemplo de utilización del bucle for, que se utilizará para ejecutar un bloque de sentencias un número determinado de veces.

```
for (int i=0; i<=10; i++){
       Response.Write(i+"<br>");
}
```

Código fuente 29

La sintaxis general del bucle foreach la podemos ver en el Código fuente 30.

```
foreach (tipo identificador in expresión){
       ...
       sentencias;
       ...
}
```

Código fuente 30

La expresión en el bucle foreach será el nombre de un array o de una colección. El bucle foreach se utilizará para recorrer los elementos de un array o de una colección. En el Código fuente 31 se puede ver como se utiliza este bucle para mostrar el contenido de un array.

```
String [] numeros = {"Uno","Dos","Tres","Cuatro"};
foreach(String elemento in numeros){
       Response.Write(elemento+"<br>");
}
```

Código fuente 31

Otro bucle que podemos encontrar en C# es el que podemos realizar con la palabra clave while. Mediante este bucle podemos ejecutar un conjunto de sentencias hasta que una determinada condición sea falsa. En el Código fuente 32 se puede observar la sintaxis general de este bucle.

```
while(expresión){
```

```
        ...
        sentencias;
        ...
}
```

Código fuente 32

En el Código fuente 33 se puede observar la utilización del bucle while.

```
int x=0;
while (x<=10){
        Response.Write(x+"<br>");
        x++;
}
```

Código fuente 33

El lector que ya conozca o tenga alguna noción anterior de ASP habrá podido observar que en algunos ejemplos, como ocurre en este último, se han utilizado objetos integrados de ASP como puede ser el objeto Response o Request. Esto es debido a que dentro de ASP .NET seguimos teniendo casi todos los objetos integrados que existían en las versiones anteriores de ASP.

Los únicos objetos integrados que ya no existen dentro de la clase Page, que va a representar a una página ASP .NET, son los objetos ASPError y ObjectContext. El objeto ASPError fue añadido en la versión 3.0 de ASP, este objeto ha desaparecido debido a que el tratamiento de errores en ASP .NET ha cambiado respecto a las versiones anteriores, y el objeto ObjectContext, que permitía manejar transacciones e integrar el Servidor Internet Information Server 5.0 con los servicios de componentes de Windows 2000, también ha desaparecido debido al nuevo entorno de ejecución que ofrece la plataforma .NET. En el capítulo correspondiente veremos estos objetos integrados que se presentan como atributos (miembros) de la página ASP .NET, es decir, de la clase Page.

Los objetos integrados han realizado su parición dentro de esta pequeña referencia al lenguaje C#, debido a que nosotros el entorno en el que vamos a utilizar el lenguaje va a ser en ASP :NET, y todos los ejemplos están enfocados a la utilización de C# dentro de páginas ASP .NET.

Manejadores de eventos

Estos son métodos especiales que se van a encargar del tratamiento de eventos en el cliente, pero que ejecutarán código de servidor. Estos métodos los veremos en detalle cuando veamos más adelante los Web Forms y los controles de servidor.

Los métodos para el tratamiento de eventos poseen dos parámetros, el primero de ellos es el control que generó el evento (la fuente u origen del evento), y el segundo parámetro serán los argumentos adicionales que necesite el método, normalmente estos argumentos dependen del tipo de evento a tratar.

En el Código fuente 34 se puede ver la sintaxis general de un método para el tratamiento de eventos.

```
void NombreMetodo (Object origen, EventArgs argumentos){
        ...
```

```
        sentencias;
        ...
}
```

Código fuente 34

A continuación se ofrece un código (Código fuente 35) en el que se puede ver un método que va a tratar el evento de pulsación de un botón dentro de un Web Form. Para que el código sea mas completo se ha incluido también el Web Form y el botón que genera el evento al ser pulsado, en este código se adelanta lo que podremos ver a partir del siguiente capítulo, los Web Forms y los controles de servidor de ASP .NET.

```
<html>
<body>
<script language="C#" runat="server">
void Pulsado(Object origen, EventArgs argumentos){
        etiqueta.Text ="El origen del evento es: "+origen.ToString();
}
</script>
<form id="formulario" method="post" runat="server">
<asp:button id="boton" text="Pulsar" onclick="Pulsado" runat="Server"></asp:button>
<asp:label id="etiqueta" runat="server"></asp:label>
</form>
</body>
</html>
```

Código fuente 35

El resultado de la pulsación del botón se puede observar en la Figura 18, el origen del evento es un objeto de la clase Button que pertenece al espacio con nombre System.Web.UI.WebControls.

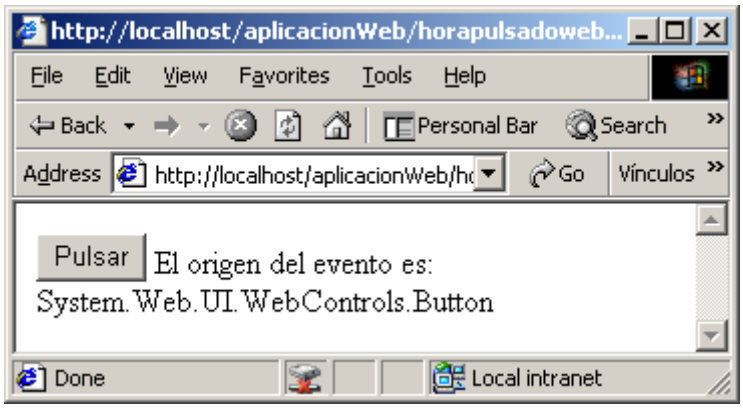


Figura 18

Conversión de tipos

El lenguaje C# es un lenguaje fuertemente tipado, es decir, es bastante estricto en lo que a conversión y compatibilidad de tipos de datos se refiere.

C# al igual que C y Java admite el moldeado de tipo (casting) o conversión de tipos, lo que nos permite convertir un tipo de dato en otro. Lo que se suele utilizar principalmente para cambiar el tipo

de dato devuelto por una función para que coincida con el tipo de dato esperado por otra a la que se lo vamos a pasar como parámetro.

Para realizar el moldeado de tipo (casting), se coloca el tipo desea entre paréntesis a la izquierda del valor que vamos a convertir.

En el Código fuente 36 se puede comprobar como se utiliza el mecanismo de casting, para convertir un objeto de la clase Object en un objeto de la clase String a la hora de recuperar una variable de sesión.

```
Session.Add("variable","valor");
String cadena=(String)Session["variable"];
Response.Write(cadena);
```

Código fuente 36

Además del mecanismo de casting, el lenguaje C# ofrece en las distintas clases una serie de métodos que permiten la conversión de tipos. Así por ejemplo si queremos convertir un objeto entero a una cadena de caracteres, utilizaremos el método ToString() de la clase de tipo simple int. También podemos tener el caso opuesto, en el que queremos convertir una cadena a un entero, para ello se utiliza el método Parse(). En el Código fuente 37 se pueden ver estas situaciones.

```
int i=3;
String s=i.ToString();
int numero=int.Parse("10");
```

Código fuente 37

Definición de clases

Para declarar una clase en C# se debe seguir la sintaxis general que se puede ver en el Código fuente 38.

```
using NameSpace;
namespace EspacioConNombre{
	Modificadores class NombreClase: ClasePadre{
		...
		implementación de la clase
		...
	}
}
```

Código fuente 38

Mediante la palabra clave using indicaremos si vamos hacer uno de algún espacio con nombre (NameSpace), por cada NameSpace que necesitemos dentro de nuestra clase incluiremos una sentencia con la directiva using.

Nuestra clase la podemos definir dentro de un NameSpace, para ello utilizamos la palabra clave namespace y a continuación indicamos el nombre deseado. No es obligatorio declarar la clase dentro de un NameSpace, aunque es lo más recomendable para tener organizadas de forma adecuada nuestras clases.

Entre las llaves del NameSpace se incluye la declaración de la clase, dónde se comienza indicando los modificadores o calificadores, que van a definir las características de la clase. A continuación aparece la palabra reservada class y el nombre de clase a definir.

Los modificadores que pueden utilizarse a la hora de definir una clase se verán puntualmente en los siguientes capítulos según sea necesario.

Si nuestra clase hereda de una clase padre o superclase deberemos utilizar los dos puntos (:) y a continuación el nombre de la clase de la que se hereda. Después entre las dos llaves se realizará la implementación de la clase, definiendo sus propiedades y métodos.

Todo lo relacionado con la declaración de clases lo veremos en detalle más adelante, cuando tengamos que definir una clase en C#.

Tratamiento de errores

El tratamiento de errores es muy similar al del lenguaje Java, desde C# vamos a utilizar las sentencias try{...}catch{...}finally{...}, que se utilizan para atrapar excepciones dentro de nuestro código.

Existen distintos tipos de errores que pueden causar excepciones, desde problemas en el hardware a problemas en la programación, al acceder por ejemplo a un elemento de un array que no existe.

En C#, cuando se produce un error en un método, el método crea un objeto excepción y se lo pasará al entorno de ejecución. Este objeto contendrá información acerca del error que se ha producido.

Los candidatos a manejar o tratar el error que se ha producido son los métodos que se encuentran en la pila de llamadas del método en el que se ha producido el error, es decir, los métodos que se han invocado antes. El entorno de ejecución va buscando hacia atrás en la pila de llamadas desde el método en el que se ha producido el error, hasta que encuentra el método que tiene el tratamiento de la excepción adecuado.

Un tratamiento de excepción adecuado se produce cuando la excepción que se trata es del mismo tipo que la que se ha lanzado al producirse el error. Cuando se trata la excepción en la terminología C# se dice que se ha atrapado la excepción.

Para tratar excepciones, el primer paso es declarar un bloque try que englobe a todas las sentencias susceptibles de producir una excepción. Si se produce alguna excepción en los métodos comprendidos dentro del bloque try, se ejecutará el manejador de excepciones asociado al bloque try.

El manejador de excepciones asociado a un bloque try se indica mediante uno o más bloques catch. En su forma más sencilla el bloque catch puede ir vacío, sin ninguna sentencia. Los parámetros de catch declaran una variable del tipo de excepción que se desea atrapar. Mediante la variable de la excepción se recogerá el objeto correspondiente a la excepción que se ha producido y se utilizará para consultar la información que ofrece.

Podemos utilizar para un mismo bloque try varios bloques catch, de esta forma podremos indicar diferentes tipos de excepciones a tratar. Se debe ir desde la excepción más particular a la más general,

para que no se oculten entre sí, cuando se produzca la excepción se ejecutará el bloque catch que corresponda al tipo de excepción en cada caso.

Como hemos visto a un bloque try le deben corresponder uno o más bloques catch, y hay veces en las que también le puede corresponder un bloque finally, aunque no de forma obligatoria.

En un bloque finally escribiremos todo el código de limpieza (liberar objetos, ficheros, etc.) en el caso de que se produzca una excepción, finally nos asegura que todas las sentencias de su bloque se ejecutarán cuando se produzca la excepción, una vez tratada por el bloque catch correspondiente.

En el Código fuente 39 ofrecemos un esquema de código que relaciona estos tres tipos de bloques, así un bloque try tendrá uno o más bloques catch y uno o ningún bloques finally.

```
try {
        ...
} catch (SubClaseTrowable nombreVariable) {
        ...
} [catch (SubClaseTrowable nombreVariable) {
        ...
}
[....]
[finally{
        .....
}]
```

Código fuente 39

Para lanzar una excepción desde nuestro código utilizaremos la palabra reservada throw, su sintaxis es muy sencilla únicamente debe ir seguida de un objeto que representa la excepción que se ha producido, que será un objeto que hereda de la clase System.Expection.

Más adelante veremos otra forma más de tratar los errores dentro de las páginas ASP .NET, se trata de las páginas de error, muy similares a las que podemos encontrar en las páginas activas de servidor de Java, es decir, las páginas JSP (Java Server Pages).

Con este subapartado finalizamos el apartado dedicado a la sintaxis básicas del lenguaje C#, en lo que queda de capítulo vamos a mostrar la creación de clases desde C# y la utilización práctica de otros mecanismos de orientación a objetos.

Campos

Un campo es una variable miembro de una clase, que sirve para almacenar un valor. Admiten varios modificadores, que definen el comportamiento del campo en tiempo de ejecución: private, public, protected, internal, static, readonly y const. A los cuatro primeros, se les denomina modificadores de acceso, teniendo el resto significados que veremos a continuación (Tabla 3).

Accesibilidad declarada	Significado
public	Acceso no restringido.
protected	Acceso limitado a la clase contenedora o a los tipos derivados de esta clase.

internal	Acceso limitado al proyecto actual.
protected internal	Acceso limitado al proyecto actual o a los tipos derivados de la clase contenedora.
private	Acceso limitado al tipo contenedor.

Tabla 3

La declaración private es la declaración predeterminada de un miembro en C#. Significa que dicho miembro sólo es visible desde la propia clase y sus clases anidadas (que no derivadas), pero no desde variables de instancia (desde otros objetos que instancien una variable de esa clase). Constituye el pilar fundamental del principio de la Encapsulación, y protege del acceso inadvertido a los miembros de una clase, permitiendo preservar la lógica interna definida en su creación.

La declaración public, Aunque existe en todos los lenguajes orientados a objetos, su utilización se justifica raramente, puesto que atenta contra el primer principio de la OOP: la Encapsulación. Un miembro public es accesible sin restricciones, por lo que no hay ningún tipo de control sobre los valores que son asignados y/o devueltos.

Entre estos dos límites C# proporciona métodos mixtos de acceso, permitiendo distintos niveles, que son los que vamos a tratar a continuación.

La declaración protected es idéntica a private, a excepción de que se permite que los miembros de las clases derivadas tengan acceso al mismo (al igual que las clases anidadas). Se presta bien para ofrecer funcionalidad a clases derivadas, sin permitir asignaciones inapropiadas, como en el caso de los miembros public. El siguiente código fuente (Código fuente 40) ilustra perfectamente esta situación.

```
class A
{
   protected int x = 123;
}

class B : A
{
   void F()
   {
      A a = new A();
      B b = new B();
      a.x = 10;   // Error
      b.x = 10;   // OK
   }
}
```

Código fuente 40

El fallo se produce debido a que la clase A no deriva de la clase B (sino al contrario), y el miembro x, está declarado en A, como protected.

Internal un modificador de acceso para tipos y miembros de tipos. Eso significa que el modificador es aplicable la declaración de clases, cuando queremos limitar su uso al proyecto en el que son declaradas. Los miembros internos sólo son accesibles dentro de archivos del mismo ensamblado.

Protected internal es el único modificador que admite más de una palabra reservada. El acceso se limita al proyecto actual o a los tipos derivados de la clase contenedora.

Static**,** aunque ya nos hemos referido a él anteriormente, recordemos que sirve para declarar un miembro que pertenece al propio tipo en vez de a un objeto específico. El modificador static se puede utilizar con campos, métodos, propiedades, operadores y constructores, pero no con indizadores, destructores o tipos.

Con ello, queremos decir que las referencias al tipo (y, por tanto, a sus miembros y métodos) nunca se harán a través de una instancia de la clase sino referenciando a la propia clase directamente, como si una instancia de ella con su propio nombre fuera de tipo publico.

Veamos el siguiente ejemplo típico del MSDN para ilustrar la situación (Código fuente 41).

```
public class A
{
   public static int X;
   internal static int Y;
   private static int Z;
}
internal class B
{
   public static int X;
   internal static int Y;
   private static int Z;
   public class C
   {
      public static int X;
      internal static int Y;
      private static int Z;
   }
   private class D
   {
      public static int X;
      internal static int Y;
      private static int Z;
   }
}
```

Código fuente 41

En esta situación, se dan las siguientes restricciones de accesibilidad:

El código anterior introduce dos clases de alto nivel (A y B), dándose la circunstancia de que B posee, a su vez, dos clases anidadas (C y D). Según sus modificadores de ámbito la situación es la siguiente:

- El dominio de accesibilidad de A y de A.X es ilimitado.
- El dominio de accesibilidad de A.Y, B, B.X, B.Y, B.C, B.C.X y B.C.Y es el texto del programa contenedor.
- El dominio de accesibilidad de A.Z es el texto del programa de A.
- El dominio de accesibilidad de B.Z y de B.D es el texto del programa de B, incluido el texto del programa de B.C y de B.D.

- El dominio de accesibilidad de B.C.Z es el texto del programa de B.C.
- El dominio de accesibilidad de B.D.X, B.D.Y y de B.D.Z es el texto del programa de B.D.

Observamos aquí algunas conclusiones de interés, como el hecho de que el dominio de accesibilidad de un miembro, nunca es mayor que el de un tipo contenedor, y que toda clase derivada hereda de su antecesora todos sus miembros, a excepción de sus constructores y destructores, lo que no significa que pueda acceder a los mismos. Por tanto, en dicha situación, una clase derivada, aunque herede los miembros privados, no puede acceder a ellos. El siguiente ejemplo aclara este concepto (Código fuente 42).

```
class A
{
   int x;
   static void F(B b) {
      b.x = 1;      // Correcto, aunque el parámetro sea un objeto de la clase B,
                    // el acceso se realiza desde el propio tipo A
   }
}
class B: A
{
   static void F(B b) {
      b.x = 1;      // Error, x no es accesible desde el tipo B.
   }
}
```

Código fuente 42

La encapsulación preserva el acceso al miembro x de la clase A, que es inaccesible para sus tipos derivados y para cualquier otro no derivado.

El modificador readonly, indica, obviamente, que un miembro es de sólo lectura. Se diferencian de las declaraciones const que veremos a continuación en que no pueden ser calculadas en tiempo de compilación, generalmente, debido a que se precisa una instancia de un objeto para hacerlo, y que deben de ser asignadas, o bien en su declaración, o en el método constructor de la clase.

El Código fuente 43 muestra la declaración y asignación de campos readonly, efectuando la inicialización, en el primer caso, dentro de la propia declaración, y en el segundo, en el método constructor predeterminado.

```
using System;
public class ClaseConCamposRO
{
   class ClaseAnidada
   {
      public int x;
      public readonly int y = 2; // Inicializa el campo readonly y
      public readonly int z;

      public ClaseAnidada()
      {
         z = 3;   // Inicializa el campo readonly z
      }
      public ClaseAnidada(int p1, int p2, int p3)
      {
         x = p1;
```

```
         y = p2;
         z = p3;
      }
   }
   public static void Main()
   {
      ClaseAnidada p1= new ClaseAnidada(4, 5, 6);   // Correcto
      Console.WriteLine("p1: x={0}, y={1}, z={2}" , p1.x, p1.y, p1.z);
      ClaseAnidada p2 = new ClaseAnidada();
      p2.x = 7;   // Correcto
      Console.WriteLine("p2: x={0}, y={1}, z={2}" , p2.x, p2.y, p2.z);
   }
}
```

Código fuente 43

Por tanto obtendríamos los siguientes resultados (Código fuente 44).

```
p1: x=11, y=21, z=32
p2: x=55, y=25, z=24
```

Código fuente 44

En el ejemplo anterior, si se utiliza una instrucción como la del Código fuente 45.

```
p2.y = 8;        // Error
```

Código fuente 45

Se obtendrá el siguiente mensaje de error del compilador.

```
The left-hand side of an assignment must be an l-value
```

Código fuente 46

Que es el mismo error que se obtiene al intentar asignar un valor a una constante.

Los campos const se definen como campos cuyo valor puede ser determinado por el compilador en tiempo de compilación y que no se pueden modificar en tiempo de ejecución (son, de forma implícita, readonly). Son, pues, más restrictivas que las variables readonly, pero pueden definirse también a partir de otras constantes (pero no variables aunque éstas hayan sido declaradas previamente).

```
const int x = 1;
const int y = x + 1;
int z = 2;
const w = z + y + 1; //Error z es una variable
```

Código fuente 47

Métodos

Los métodos de una clase constituyen su componente funcional, siendo aquello que las dota de un comportamiento. Siguiendo las definiciones formales que aporta el MSDN diremos que un método es "*un miembro que implementa un cálculo o una acción que pueden ser realizados por un objeto o una clase. Los métodos tienen una lista de parámetros formales (que puede estar vacía), un valor devuelto (salvo que el tipo de valor devuelto del método sea void), y son estáticos o no estáticos. El acceso a los métodos estáticos se obtiene a través de la clase. El acceso a los métodos no estáticos, también denominados métodos de instancias, se obtiene a través de instancias de la clase".*

Métodos constructores

Recordemos que un método constructor es aquel que se llama siempre que se crea una instancia de una clase. Debe de tener el mismo nombre de la clase, pero admite diferentes signaturas, dependiendo de los distintos conjuntos de parámetros que queramos pasar al constructor. En C# existen 3 tipos de constructores: de instancia, private y static.

Un constructor de instancia se utiliza para crear e inicializar instancias, y es –generalmente- el constructor predeterminado. La sintaxis completa de un constructor de este tipo es:

```
[atributos] [modificadores] identificador ([lista-de-parámetros-
formales]) [inicializador]
{ cuerpo-del-constructor }
```

donde los tres parámetros más significativos son:

- Los **modificadores** permitidos, que pueden ser ***extern*** ó cualquiera de los 4 modificadores de acceso
- **identificador,** que es el nombre de la clase, e
- **inicializador** que es una llamada opcional a la clase base (mediante **:base(<lista>)**) o una referencia a la propia clase, expresada mediante **this(<lista>)**, en caso de que deseemos llamar a otro constructor de la misma clase, antes de ejecutar el cuerpo del constructor que hace la llamada.

Dando por sentado que la sintaxis de instanciación de objetos en C# es:

```
<clase> <objeto> = new <clase> (argumentos del constructor),
```

Debemos tener en cuenta que en un método constructor se cumplen siempre los siguientes supuestos:

- Un método constructor siempre tiene el mismo nombre de la clase a la que pertenece.
- Un método constructor no devuelve ningún tipo de dato
- Un método constructor admite los modificadores: public, protected, internal, private, extern
- Un método constructor admite atributos.
- Se dice que la lista de parámetros formales opcional de un constructor de instancia está sujeta a las mismas reglas que la lista de parámetros formales de un método. Dicha lista de

parámetros constituye la firma del método y se encarga de discriminar la llamada correspondiente, cuando el constructor está sobrecargado.

- Todo método constructor (excepto System.Object) efectúa una llamada implícita previa a la clase base del constructor, inmediatamente antes de la ejecución de la primera línea del constructor. Se conoce estas llamadas como inicializadores de constructor (Constructor Initializers).
    - Estos inicializadores de constructor, si son explícitos (si se indican como tales en el código fuente), pueden presentarse en dos formas, dependiendo de la palabra reservada que usemos en la llamada: base o this.
        - Si utilizamos base estaremos llamando al constructor de la clase base de la que hereda nuestra propia clase.
        - Si usamos this, podemos efectuar una llamada interna desde un constructor a otro constructor dentro de la clase.

Veamos un ejemplo de esta situación (Código fuente 48).

```
using System;

class A
{
    public A()
  {
        Console.WriteLine("A");
    }

    public A(int x)
    {
        Console.WriteLine("A = {0}", x);
    }
}

class B : A
{
    public B(int x)
  {
        Console.WriteLine("B = {0}", x);
    }
}

class Pruebas
{
    public static void Main()
    {
        B b = new B(3);
    }
}
```

Código fuente 48

Podemos mejorar el ejemplo anterior para garantizar que la clase B llama al constructor deseado, (indicado con la línea más oscura), señalando explícitamente al compilador cuál es el constructor a llamar, mediante una modificación del constructor de B, que incluya una llamada explícita a la clase base (Código fuente 49).

```
public B(int x): base(x)
```

Código fuente 49

lo que nos garantiza que en la inicialización de B, el argumento recibido –x– es utilizado previamente en una llamada al constructor de su clase base, antes de ejecutarse.

Otra situación posible es aquella en la que deseamos que un determinado método constructor llame, no al constructor de la clase base, sino a otro constructor de la misma clase. Por ejemplo el Código fuente 50.

```
using System;

class A
{
    public A()
  {
        Console.WriteLine("Constructor predeterminado de A");
    }

    public A(int x):this() //segundo constructor de A, que llama al predeterminado
    {
        Console.WriteLine("A = {0}", x);
    }
}

class Pruebas
{
    public static void Main()
    {
        A a = new A(3);
    }
}
```

Código fuente 50

La salida en ejecución de este programa será:

```
Constructor predeterminado de A
A = 3
```

Código fuente 51

Los constructores private son un tipo especial de constructor de instancias que no es accesible desde fuera de la clase. Suelen tener el cuerpo vacío, y no es posible crear una instancia de una clase con un constructor private. Los constructores private son útiles para evitar la creación de una clase cuando no hay campos o métodos de instancia (son todos static como, por ejemplo, la clase Math), o cuando se llama a un método para obtener una instancia de una clase (Código fuente 52).

```
public class ConstructorEstatico
{
```

```
   private ConstructorEstatico() {} // Cuerpo vacío y miembros estáticos
   public static int contador;
   public static int IncrementarContador ()
   {
      return ++contador;
   }
}

class Pruebas
{
   static void Main()
   {
      // La llamada siguiente a ConstructorEstatico (comentada) genería un error
      // por que no es accesible
      // ConstructorEstatico cs = new ConstructorEstatico ();   // Error
      ConstructorEstatico.contador = 100;
      ConstructorEstatico.IncrementarContador ();
      Console.WriteLine("Valor: {0}", ConstructorEstatico.contador);
   }
}
```

```
Salida en ejecución:

Valor: 101
```

Código fuente 52

Los constructores static son un tipo especial de constructor que se utiliza para inicializar (no para crear una instancia) una clase. El MSDN nos recuerda que "se llama automáticamente para inicializar la clase antes de crear la primera instancia o de hacer referencia a cualquier miembro estático. Se declara de la siguiente forma:

```
[atributos] static identificador ( ) { cuerpo-del-constructor }
```

Tienen las siguientes características especiales:

Un constructor static no permite modificadores de acceso ni tiene parámetros.

- Es llamado automáticamente para inicializar la clase antes de crear la primera instancia o de hacer referencia a cualquier miembro estático.
- El constructor static no puede ser llamado directamente.
- El usuario no puede controlar cuando se ejecuta el constructor *static* en el programa.
- Los constructores *static* se utilizan normalmente cuando la clase hace uso de un archivo de registro y el constructor escribe entradas en dicho archivo"

Un ejemplo de su uso sería el Código fuente 53.

```
using System;
class A
{
   // Constructor estático
   static A ()
   {
      Console.WriteLine("Llamado el constructor estático");
   }
```

```
   public static void F()
   {
      Console.WriteLine("Llamado F().");
   }
}

class MainClass
{
   static void Main()
   {
      A.F();
   }
}
```

Código fuente 53

Con ello se consigue forzar la ejecución del constructor, sea cual sea el miembro de la clase que se llame.

Métodos estándar

Fuera de las categorías estudiadas, podemos ubicar los llamados métodos estándar, que tienen por objeto dotar al tipo de una funcionalidad específica para ese tipo. Admiten un conjunto de modificadores, que son: new, public, protected, internal, private, static, virtual, sealed, override, abstract y extern.

Modificadores de argumento

Además, existen dos palabras reservadas del lenguaje para alterar el modo en que los argumentos de un métodos son enviados y devueltos: out y ref. Vamos a comenzar con el estudio de estos dos modificadores de argumentos:

Básicamente, tienen el mismo propósito, pero una ligera diferencia de funcionamiento. Ambos permiten pasar argumentos por referencia, en vez de por valor. La diferencia es que los primeros necesitan ser inicializados previamente, y los segundos, no. En cierta forma, esto se relaciona con lo ya comentado acerca de los tipos por valor (*value types*) y los tipos por referencia. (*reference types*). Sólo que, en este caso, podrían darse hasta cuatro situaciones, en estas posibles combinaciones:

- **Paso por valor de un *value type***. (Sintaxis estándar, sin modificadores ni en la llamada, ni en la recepción).

- **Paso por referencia de un *value type***. (Se añade la palabra clave *ref*, tanto en la llamada, como en la lista de argumentos del método (Código fuente 54).

```
using System;
class PasaValorPorReferencia
{
    static void Doblar(ref int x)
    // x se pasa por referencia y los cambios afectan al original
    {
        x *= 2;
        Console.WriteLine("Valor dentro del método: {0}", x);
    }
```

```
    public static void Main()
    {
        int h = 5;
        Console.WriteLine("Valor antes de llamar al método: {0}", h);
        Doblar(ref h);   // pasa h por referencia.
        Console.WriteLine("Valor después de llamar al método: {0}", h);
    }
}
```

```
Salida:
```

```
Valor antes de llamar al método: 5
Valor dentro del método: 10
Valor después de llamar al método: 10
```

Código fuente 54

- **Paso por valor de un *reference type*** (Una situación a veces delicada, pues, dependiendo de la manipulación que el método haga de la referencia, ésta puede ser permanente o no). Veamos el caso en otro ejemplo (Código fuente 55).

```
using System;

class PasoDeReferenciaPorValor
{
   static void Cambiar(int[] arr)
   {
      arr[0]=888;   // El cambio a un elemento del array, afecta al original,
por
                    // que no se ha creado un nuevo espacio de almacenamiento
y
                    // es la primera instrucción. Si invertimos el orden de
las
                    // instrucciones, las asignaciones también varían.
      arr = new int[5] {-3, -1, -2, -3, -4}; // Pero este es local.
      Console.WriteLine("Dentro del método el primer elemento es: {0}",
arr[0]);
   }

   public static void Main()
   {
      int[] myArray = {1,4,5};
      Console.WriteLine("Dentro de Main, antes de llamar al método, el 1° es:
{0}", myArray [0]);
      Cambiar(myArray);
      Console.WriteLine("Dentro de Main, despues de llamar al método, el 1°
es: {0}", myArray [0]);
      Console.Read();
   }
}
```

```
Salida
```

```
Dentro de Main, antes de llamar al método, el 1° es: 1
Dentro del método el primer elemento es: -3
Dentro de Main, despues de llamar al método, el 1° es: 888
```

Código fuente 55

El código anterior puede no parecer (y no lo parece) evidente en un primer vistazo. myArray se pasa por valor al método Cambiar(), luego, aparentemente, tenemos una copia local y los cambios en la copia local no afectan al original. Pero, aquí sí que se ve afectado el original

(tras la ejecución de la primera línea). La razón es que el tipo pasado es –intrínsecamente– un objeto. En la primera instrucción, se cambia el dato en su ubicación original. Pero en la segunda, se crea un nuevo objeto y, con ello, asignamos un nuevo espacio de almacenamiento, por lo que los cambios realizados aquí no afectan al objeto original. Por eso, si se cambia el orden de las instrucciones variamos el resultado, ya que, al crear primero el nuevo objeto, asignamos un nuevo espacio de almacenamiento, y cualquier cambio a partir de ahí, es un cambio local.

Cuando se desea que el método pueda modificar el propio espacio de almacenamiento, se debe de recurrir al cuarto y último caso:

- Paso por referencia de un reference type (Es una situación similar a la del "paso por referencia de un value type"). Aquí utilizaremos las palabras ref tanto en la llamada como en la recepción, por lo que el comportamiento es el esperado inicialmente: podemos crear un nuevo array en el método Change() sabiendo que tales modificaciones sí se verán en el original.

  En ocasiones, necesitamos construir métodos que permitan recibir un número no determinado de argumentos (de 0 a n). En esos casos podemos declarar como argumento un array, y anteponerle el modificador params**.** Con ello, lo que hacemos es declarar el argumento como un array dinámico, que puede recibir cualquier número de argumentos – acorde con su tipo – en tiempo de ejecución. El funcionamiento es similar al de las declaraciones del método Main() cuando se recogen argumentos de la línea de comandos. La lista de argumentos es almacenada en la variable local, y puede ser recorrida posteriormente con un bucle, como en el siguiente ejemplo (Código fuente 56).

```
using System;

class EjemploParams
{
   static void LectorVariable(params int[] arr)
   {
      foreach (int x in arr) {
           Console.WriteLine( x );
      }
   }

   public static void Main()
   {
     //Esta llamada funciona
     LectorVariable( 3, 4, 5);
     // Y esta también
     LectorVariable( 3, 4, 5, 6, 7, 8, 9);
     // E incluso esta (aunque no imprima nada)
     LectorVariable();
     Console.Read();
   }
}
```

Código fuente 56

Modificadores de métodos

Son palabras reservadas que califican el comportamiento de un método tanto desde un punto de vista funcional como de arquitectura. Los principales modificadores de métodos aparecen explicados brevemente en la Tabla 4.

Modificador	Descripción
Static	El método pertenece al propio tipo (clase) en lugar de a un objeto específico. No pueden llamarse desde una variable de instancia.
Abstract	El método es un miembro de una clase abstracta. (Si la clase no se declarara como abstracta, esta opción no está disponible). No está permitido heredar de un tipo abstracto. Su función se limita a la definición de jerarquías de clases.
Virtual	La implementación del método puede cambiarse mediante un miembro de reemplazo (override) de una clase derivada. Pueden llamarse igual que los métodos estándar.
Extern	Sólo se establece su declaración formal. La implementación es externa. Se usa principalmente para declarar funciones de una librería externa a .NET Framework.
Override	Proporciona una nueva implementación de un miembro virtual heredado de una clase base.
New	Oculta explícitamente un miembro heredado de una clase base. Se diferencia de override en que la ocultación impide la llamada al mismo método en las clases antecesoras.
Modificadores Especiales	
Abstract	Aplicable a una clase y también a un método cuya clase haya sido declarada de igual forma, permite definir jerarquías de clases, sin proporcionar implementaciones de los métodos.
Sealed	Aplicable exclusivamente a las clases, establece que la clase no se puede heredar, solo se permite la instanciación de objetos de la clase.

Tabla 4

Hay que tener en cuenta que algunos modificadores se pueden combinar, pero otros son mutuamente excluyentes. Por ejemplo, se pueden establecer combinaciones como extern static, new virtual, y override abstract.

Revisaremos los significados principales de estos modificadores, comenzando con las diferencias entre virtual y override. En el primer caso se indica de forma explícita la posibilidad de que el método sea sustituido en una clase derivada por un miembro de reemplazo.

Téngase en cuenta que lo que se analiza aquí es la manera en que las jerarquías de clase pueden llegar o no a hacer uso de métodos en clases antecesoras, dependiendo de los modificadores utilizados en su declaración.

En el ejemplo (Código fuente 57), A.F() es sustituido en la jerarquía por B.F() que tiene la misma signatura, y se declara como new ocultando la implementación de A.F(). Hay que notar aquí que si en B declarásemos F() como override se produciría un error en tiempo de compilación, ya que para que un método pueda ser reemplazado (mediante override) debe de ser declarado como virtual, abstract u override, lo que no es el caso aquí. Hay una sutil diferencia entre reemplazo y sustitución.

```
using System;

class A
{
   public void F() { Console.WriteLine("A.F"); }
   public virtual void G() { Console.WriteLine("A.G"); }
}
class B: A
{
   new public void F() { Console.WriteLine("B.F"); }
   public override void G() { Console.WriteLine("B.G"); }
}
class Test
{
   static void Main() {
      B b = new B();
      A a = b;
      a.F();
      b.F();
      a.G();
      b.G();
   }
}
```

```
Salida:
```

```
A.F
B.F
B.G
B.G
```

Código fuente 57

El análisis de otro ejemplo del MSDN puede servirnos para ilustrar esta diferencia, cuando 4 clases definen sucesivamente un mismo método y luego éste es llamado en cadena. Como es posible que un método de una clase oculte otro método heredado, también lo es disponer de varios métodos virtuales con la misma firma (Código fuente 58).

```
class A
{
   public virtual void F() { Console.WriteLine("A.F"); }
}
class B: A
{
   public override void F() { Console.WriteLine("B.F"); }
}
class C: B
{
   new public virtual void F() { Console.WriteLine("C.F"); }
   // Si sustituyéramos esta declaración por
   // public override void F() { Console.WriteLine("C.F"); }
   // la salida final serían cuatro llamadas a D.F()
}
class D: C
{
   public override void F() { Console.WriteLine("D.F"); }
}
class Test
{
   static void Main() {
      D d = new D();
      A a = d;
      B b = d;
```

```
      C c = d;
      a.F();
      b.F();
      c.F();
      d.F();
   }
}
```

```
Salida
B.F
B.F
D.F
D.F
```

Código fuente 58

Aunque estamos llamando a cada uno de los métodos, sólo dos de ellos parecen ejecutarse. Esto es debido al fenómeno de la ocultación que comentamos.

Todo sucede de la siguiente forma: En tiempo de compilación se han declarado cuatro objetos de clases distintas, pero en tiempo de ejecución es el CLR el que decide cómo se resuelven las llamadas: eso significa que cuando A.F va a ser accedido, tal acción es ignorada debido a que su superior jerarquico ha reemplazado el método mediante override por B.F, que es al que se llama. En la siguiente intrucción, B.F accede sin problemas a su propio método, pero en la tercera (C.F), el método se declaro como new y virtual, siendo reemplazado en D, lo que provoca que las dos siguientes llamadas ejecuten D.F. Pruebe el lector, a modo de ejercicio, lo que sucede si en C, se sustituye la declaración actual por otra declaración idéntica a D, como se sugiere en el código comentado en el ejemplo.

El modificador extern en una declaración de método se usa para indicar que el método se implementa externamente, y suele utilizarse para declarar funciones de API's externas como las propias de Windows, si bien puede tratarse de cualquier otra implementación externa válida.

El ejemplo siguiente es ya un clásico en .NET y muestra como una aplicación de consola puede llamar a una función de Windows como MessageBox, mediante la declaración de la función externa (Código fuente 59).

```
using System;
using System.Runtime.InteropServices;
class LlamadaExterna
{
   [DllImport("User32.dll")]
   public static extern int MessageBox(int h, string m, string c, int type);

   public static int Main()
   {
      string cadena;
      Console.Write("Introduzca su nombre: ");
      cadena = Console.ReadLine();
      return MessageBox(0, "Hola, " + cadena, "Mensaje desde Windows", 0);
   }
}
```

Código fuente 59

El modificador abstract puede aplicarse tanto a clases, como a métodos y propiedades. No puede utilizarse en un método si la clase no ha sido declarada como como tal, y aunque las clases abstractas

pueden realizar declaraciones formales de métodos, éstos no pueden contener ninguna instrucción ejecutable, por lo que puede decirse que son –implícitamente—métodos virtuales.

Las clases de tipo abstract presentan las siguientes características:

- No se pueden crear instancias de una clase abstracta.
- Una clase abstracta puede contener descriptores de acceso y métodos y propiedades abstractas.
- No es posible modificar una clase abstracta con el modificador sealed, que significa que la clase no se puede heredar.
- Una clase no abstracta derivada de una clase abstracta debe incluir implementaciones reales de todos los descriptores de acceso y métodos abstractos heredados.

El siguiente código (Código fuente 60) devuelve un error de compilación del tipo "no se puede instanciar un objeto de la clase abstracta Abstracta"

```
using System;

abstract class Abstracta
{
     string nombre = "PP";
     public void Dime() { Console.WriteLine(nombre); }
}
class Prueba {

   public static void Main()
   {
       Abstracta c = new Abstracta();
       c.Dime();
   }
}
```

Código fuente 60

Podemos modificar el código para que sea una clase heredada la que haga la llamada (Código fuente 61).

```
using System;

abstract class Abstracta
{
     string nombre = "PP";
     public void Dime() { Console.WriteLine(nombre); }
}

class A : Abstracta {}

class Prueba {

   public static void Main()
   {
       A c = new A();
       c.Dime();
   }
}
```

Código fuente 61

No obstante, véase que la clase base declara una propiedad y un método NO abstractos. Si ese fuera el caso, el método Dime() no podría tener implementación, y la clase heredada debería de ejercer dicha función, como se puede ver en el Código fuente 62.

```
using System;

abstract class Abstracta
{
     abstract public string Nombre {
         get ;      // Adviértase que no se declara el cuerpo de get
     }
     abstract public void Dime();
}

class A : Abstracta {

     public override string Nombre {  //miembro de reemplazo de la propiedad
         get { return "Juancho"; }
     }
     public override void Dime() {  //miembro de reemplazo del método
     Console.WriteLine(Nombre);
     }
}

class Prueba {

   public static void Main()
   {
       A c = new A();
       c.Dime();
   }
}
```

Código fuente 62

Como vemos en el ejemplo final, la clase abstracta fuerza a la clase derivada a implementar, tanto la propiedad, como el método, usando el modificador override, y el método get de la propiedad abstracta, no incluye siquiera las llaves (puesto que definiría un cuerpo, aunque vacío) .

Para concluir con esta revisión operativa de los modificadores aplicables a los métodos, digamos que el modificador sealed sólo es aplicable a clases, e indica que la clase no puede heredarse, sino solamente instanciarse. Aparte de ser usada en jerarquías de clases de usuario, esta situación nos la encontraremos muy a menudo en clases de la jerarquía de .NET Framework, debido a su naturaleza. Por ejemplo, la propia clase Console, que hemos estado utilizando en nuestros ejemplos, viene definida en el Examinador de Objetos de Visual Studio de la siguiente forma ().

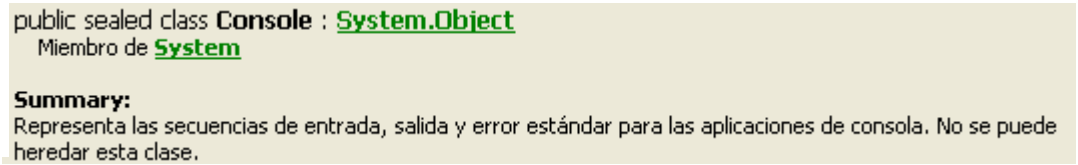


Figura 19

Con este apartado finalizamos el presente capítulo. Si a algún lector le han quedado dudas acerca del nuevo lenguaje de Microsoft, no debe preocuparse, ya que el lenguaje C# se va a utilizar a lo largo de

todo el texto, y en cada caso retomaremos con más detalle algunos de los aspectos esbozados en el presente capítulo.

5

Instalación de Visual Studio .NET

Preparación del entorno de trabajo

Antes de poder comenzar a escribir aplicaciones para .NET Framework, debemos instalar en nuestra máquina de trabajo las herramientas que nos permitirán el desarrollo de programas para este entorno de ejecución.

.NET Framework SDK

Se trata del kit de desarrollo de software para .NET Framework (Software Development Kit o SDK), que contiene la propia plataforma .NET y un conjunto de herramientas independientes, algunas funcionan en modo comando (en una ventana MS-DOS) y otras en modo gráfico. Los elementos imprescindibles para poder desarrollar aplicaciones para .NET están contenidos en este conjunto de herramientas.

Visual Studio .NET

Es la nueva versión de la familia de herramientas de desarrollo de software de Microsoft, naturalmente orientadas hacia su nuevo entorno de programación: .NET Framework.

Si bien es posible la escritura de programas empleando sólo el SDK de .NET Framework, este último, al estar compuesto de herramientas independientes, constituye un medio más incómodo de trabajo.

Visual Studio .NET (VS.NET a partir de ahora), al tratarse de un entorno de desarrollo integrado (Integrated Development Environment o IDE como también lo denominaremos a lo largo del texto), aúna todas las herramientas del SDK: compiladores, editores, ayuda, etc., facilitando en gran medida la creación de programas. Por este motivo, todas las explicaciones y ejemplos desarrollados a lo largo de este texto se harán basándose en este entorno de programación.

Requisitos hardware

La Tabla 5 muestra una lista con las características mínimas y recomendadas que debe tener el equipo en el que instalemos VS.NET.

	Mínimo	**Recomendado**
Procesador	Pentium II – 450 MHz	Pentium III – 733 MHz
Memoria	128 MB	256 MB
Espacio en disco duro	3 GB	

Tabla 5. Requerimientos hardware para instalar Visual Studio .NET.

Sistema operativo

VS.NET puede ser instalado en un equipo con uno los siguientes sistemas operativos:

- Windows 2000 (se requiere tener instalado el Service Pack 2).
- Windows NT 4.0. (se requiere tener instalado el Service Pack 5).
- Windows Me.
- Windows 98

Para aprovechar todo el potencial de desarrollo de la plataforma, es recomendable usar como sistema operativo Windows 2000, ya que ciertos aspectos del entorno (las características avanzadas de gestión gráfica por ejemplo) no están disponibles si instalamos .NET en otro sistema con menos prestaciones.

Recomendaciones previas

Es recomendable realizar la instalación sobre un equipo *limpio*, es decir, un equipo con el software mínimo para poder realizar pruebas con .NET Framework, o con otro tipo de aplicaciones sobre las que estemos seguros de que no se van a producir conflictos con el entorno.

En este sentido, una buena práctica consiste en crear en nuestro disco duro una partición que utilizaremos para el trabajo cotidiano con el ordenador, y otra partición en la que instalaremos VS.NET.

Para ayudar al lector a formarse una idea más aproximada en cuanto a configuraciones hardware y software, el equipo utilizado para realizar las pruebas mostradas en este texto ha sido un Pentium III a 933 MHz, con 256 MB de memoria y disco duro de 18 GB.

En cuanto a sistemas operativos, se han realizado dos particiones sobre el disco duro; en la partición primaria se ha asignado un tamaño de 2 GB y se instalado Windows 98. En el resto de espacio en disco se ha creado una unidad lógica sobre la que se ha instalado Windows 2000 Server y el Service Pack 2 para este sistema operativo.

Respecto a las aplicaciones utilizadas, aparte naturalmente de VS.NET, hemos instalado Visual Studio 6.0 que puede perfectamente convivir en el mismo equipo en el que esté instalado .NET Framework.

El orden más conveniente de instalación en el equipo del software antes mencionado, de forma que evitemos posibles conflictos ha sido el siguiente:

- Windows 2000 Server.
- Service Pack 2 para Windows 2000.
- Office 2000.
- Visual Studio 6.0.
- SQL Server 2000.
- Visual Studio .NET.

Y ya sin mas preámbulos, pasemos al proceso de instalación de .NET.

Instalación de Visual Studio .NET

En el momento de escribir este texto, se ha empleado Visual Studio .NET, Beta 2, versión española (número de versión 7.0.9254), que se compone de los tres CDs de instalación del producto más uno de actualización de componentes del sistema operativo (Windows Component Update)

Procederemos insertando el disco de instalación rotulado como CD1, el cuál detectará si es necesario actualizar algún componente a nivel del sistema operativo; en caso afirmativo, pulsaremos sobre el paso 1 *Windows Component Update*, en el que se nos pedirá el disco rotulado con el mismo nombre. Ver Figura 20.

Una vez insertado el disco de actualización de componentes para Windows, se mostrará la pantalla de la Figura 21. En caso de aceptar el contrato, haremos clic sobre *Continuar*, para que el instalador detecte qué componentes faltan por actualizar.

Figura 20. Selección de actualización de componentes de Windows.

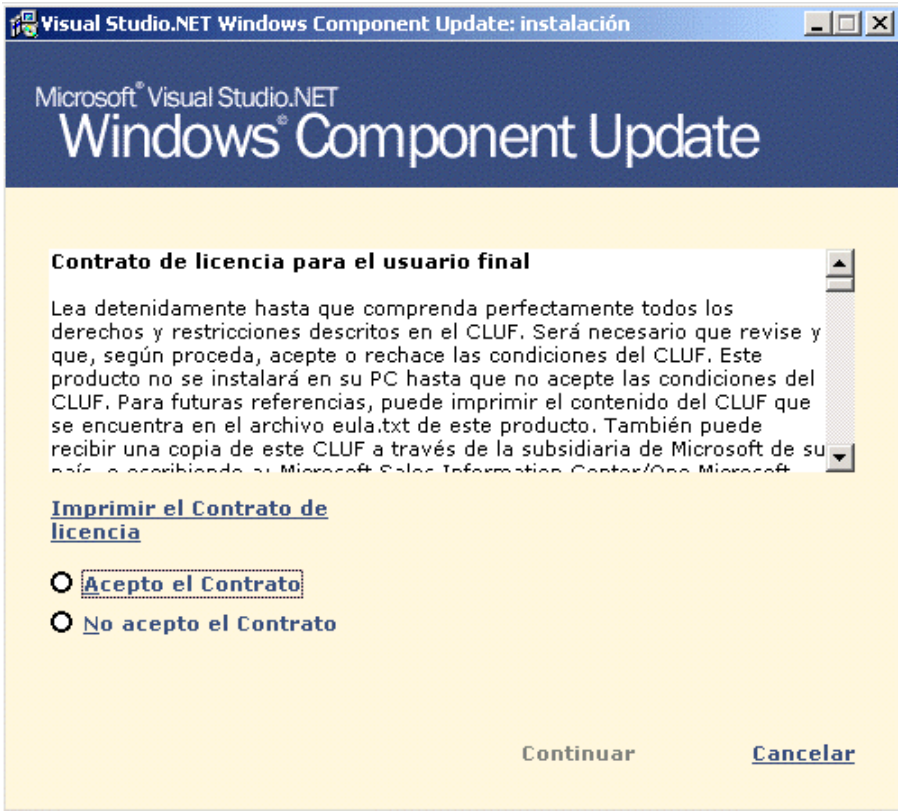


Figura 21. Contrato de instalación para Windows Component Update.

Una vez detectados los componentes que necesitan actualización, serán mostrados a continuación en la lista de la Figura 22, donde volveremos a pulsar sobre *Continuar*.

Ya que es posible que el programa de instalación reinicie el equipo una o más veces, a continuación estableceremos, en el caso de que existan en nuestro equipo, las claves de acceso al sistema, para que los reinicios sean automáticos. Ver Figura 23.

Figura 22. Lista de componentes que se necesita actualizar.

Pulsaremos a continuación sobre *Instalar ahora*, con lo que se procederá a la actualización de los componentes de la lista. Una vez terminada esta actualización, aceptaremos la ventana final de Windows Component Update y seguiremos con la instalación normal de VS.NET, lo que nos requerirá de nuevo la introducción del CD1.

Puesto que ya hemos actualizado los componentes del sistema, el siguiente paso será ya la instalación de VS.NET, que pondremos en marcha al hacer clic sobre el paso 2 de la instalación, que tiene el nombre de *Visual Studio .NET*. Ver Figura 24.

Se mostrará pues, la pantalla con los datos de licencia, producto y usuario. En el caso de estar de acuerdo con todos estos términos y aceptar el contrato, haremos clic sobre *Continuar*. Ver Figura 25.

A continuación debemos seleccionar aquellos elementos del producto que deseamos instalar, el entorno de ejecución, lenguajes, utilidades, ayuda, etc., y su ubicación en el disco duro, como muestra la Figura 26. Terminada la selección, pulsaremos sobre *Instalar ahora* para que comience el proceso.

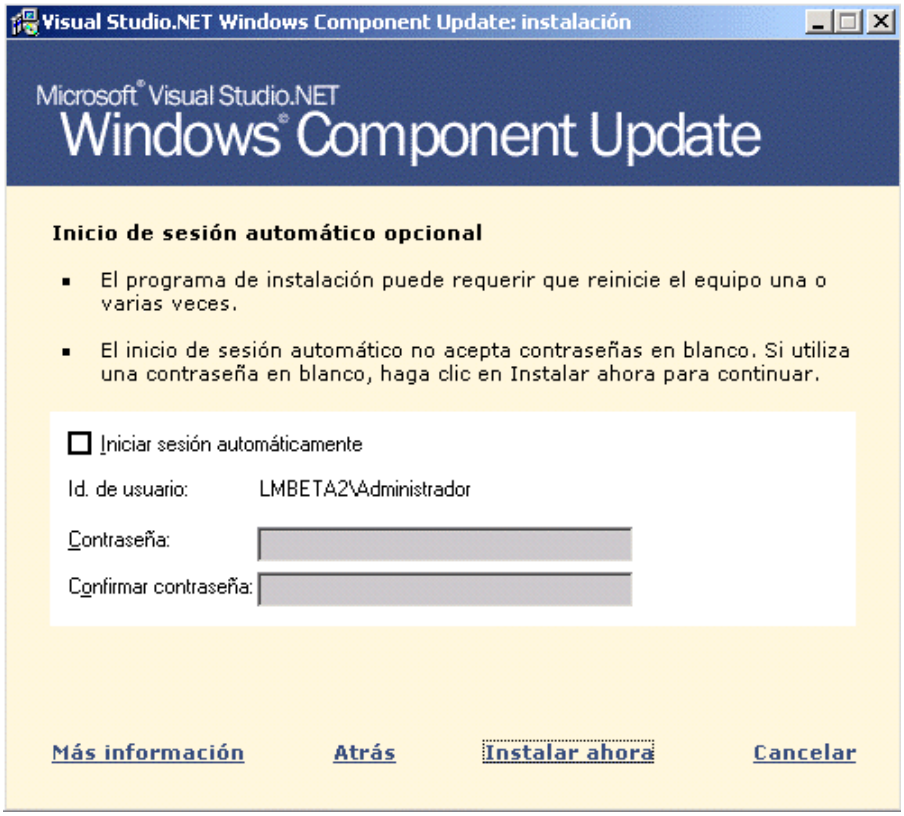


Figura 23. Valores para realizar reinicios automáticos del equipo.

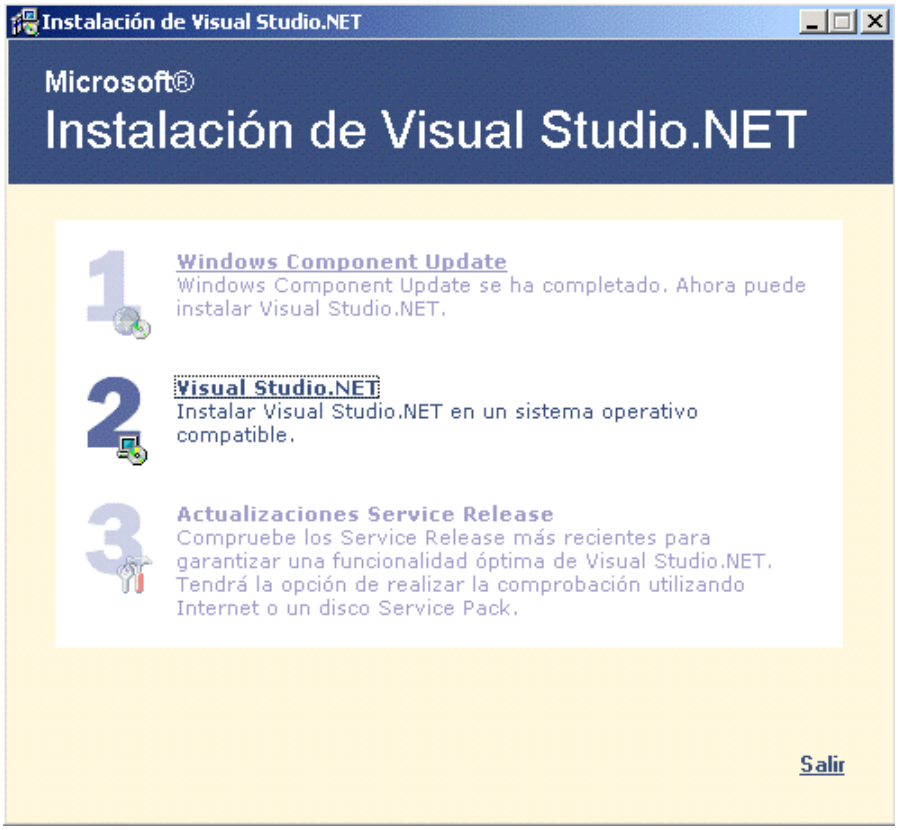


Figura 24. Instalación de Visual Studio .NET.

Figura 25. Información de licencia de Visual Studio .NET.

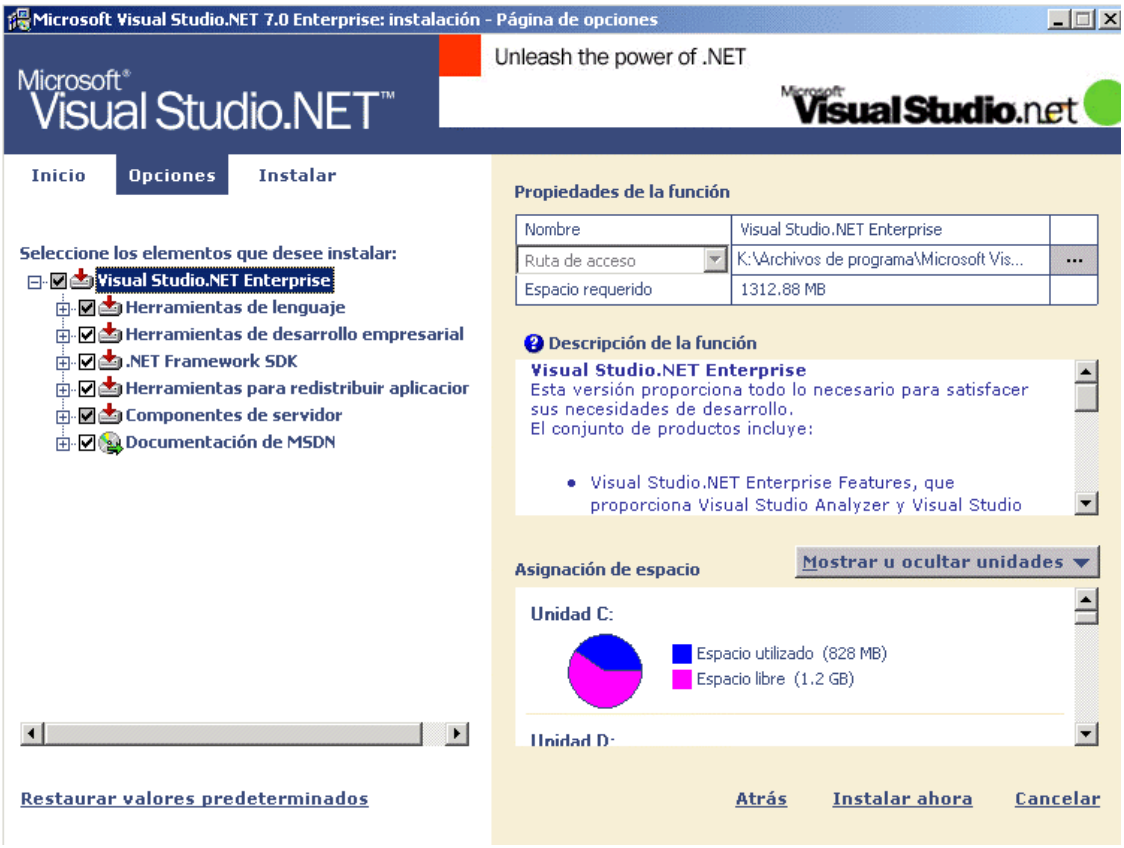


Figura 26. Selección de componentes a instalar de Visual Studio .NET.

Durante la instalación, el programa nos solicitará progresivamente los discos rotulados como CD2 y CD3.

Este proceso de instalación nos indica el archivo que se está instalando en cada momento, así como la información de su estado a través de una barra de progreso y el tiempo estimado restante, aunque por las pruebas realizadas, este último valor no es totalmente fiable. Para que el lector se forme una idea, en el equipo en el que se realizó la instalación, esta llevo un tiempo aproximado de dos horas. Ver Figura 27.

Concluida la instalación, el programa nos informará de si se produjo alguna incidencia. En caso de que no se hayan producido errores, finalizaremos haciendo clic sobre *Listo*, con lo que ya tendremos instalado Visual Studio .NET en nuestro ordenador. Ver Figura 28.

Figura 27. Información sobre el progreso de la instalación.

Figura 28. Final de la instalación.

6

Primeros pasos con ASP .NET

Objetivo de este tema

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

Además de comentar los distintos elementos que podemos utilizar en ASP .NET, también veremos como configurar nuestra máquina de trabajo, para que podamos ejecutar y desarrollar nuestras páginas ASP :NET

Este es un tema de enlace entre los tres primeros, que podrían resultar un poco más teóricos, y los siguientes temas que hacen uso de ASP .NET y de sus diferentes elementos y que resultan mucho más prácticos, comentando en más detalles los distintos aspectos de ASP .NET.

Materiales necesarios

En este apartado vamos a comentar que software y hardware necesitamos en nuestra máquina para poder seguir el texto de forma satisfactoria, y así poder ir probando y ampliando los ejemplos según avanzamos en el estudio del texto.

Empecemos con los requisitos que debe tener nuestro equipo:

- Procesador Pentium III a 800MHz o superior.
- 256 MB de memoria RAM.

- 1 GB de espacio libre en disco (después de las instalaciones).

Esto en cuanto a hardware, y en cuanto a software necesitaremos los siguientes elementos:

- Microsoft .NET Framework SDK (Software Development Kit): es la implementación de la plataforma .NET sobre la que se ejecutarán las páginas ASP .NET. El .NET Framework SDK contiene toda las clases que componen la plataforma .NET. Para poder instalar el .NET Framework SDK es necesario tener instalado Internet Explorer 6.0 y el Service Pack 2 de Windows 2000. Este producto es gratuito y se puede obtener en el sitio Web de Microsoft.

- SQL Server 7/2000: va a ser el servidor de base de datos utilizado para los distintos ejemplos del texto, sobre todo en los ejemplos de la parte dedicada a ADO .NET.

- Sistema operativo Windows 2000 Server/Professional. ASP .NET no es soportado por Windows NT ni por Windows 9x.

- El servidor Web Internet Information Server 5.0 (IIS 5.0). Incluido como parte del sistema operativo Windows 2000.

También es recomendable, aunque no obligatorio, disponer de la nueva versión de Visual Studio, que no es otra que Visual Studio .NET. Visual Studio .NET ofrece el primer entorno de desarrollo para la plataforma .NET. Ofrece un entorno integrado desde el que podemos desarrollar nuestra aplicaciones con ASP .NET. En este texto no nos vamos a centrar en esta herramienta, ya que podemos desarrollar nuestras aplicaciones ASP .NET sin ella, aunque dedicaremos un capítulo a realizar una breve descripción de la misma.

Antes de comenzar a instalar ASP .NET debemos instalar un servidor Web para poder albergar nuestras páginas ASP .NET. En realidad no va a ser el servidor Web quién ejecute las páginas ASP .NET, sino que va a ser la plataforma .NET. EN nuestro caso vamos a utilizar el servidor Web que se incluye con el sistema operativo Windows 2000, es decir, Internet Information Server 5.0.

El servidor web

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

Para averiguar si tenemos instalado IIS 5.0 en nuestra máquina podemos realizar una sencilla prueba que consiste en escribir la siguiente URL http://nombreEquipo o bien http://localhost en el navegador Web. Si IIS 5.0 está instalado aparece una ventana similar a la Figura 29.

Otra forma de comprobarlo es acudiendo al menú de inicio de Windows y en el grupo de programas Herramientas administrativas, debemos tener el Administrador de servicios de Internet (Figura 30).

Figura 29

Figura 30

Si observamos que IIS 5.0 no se encuentra instalado en nuestra máquina deberemos ir al Panel de Control y en la opción Agregar/quitar programas seleccionar componentes de Windows. Una vez hecho esto marcaremos el componente Servicios de Internet Information Server y procederemos a su instalación. Esto se puede ver en la Figura 31.

A continuación vamos comentar la instalación de ASP .NET, se debe tener en cuenta que esta instalación es únicamente necesaria en las máquinas que van a realizar las funciones de servidores Web con soporte para aplicaciones ASP .NET, en el cliente no es necesario realizar ningún tipo de instalación o configuración para poder ejecutar las páginas ASP .NET, únicamente debemos tener un navegador Web como cliente, como puede ser Internet Explorer.

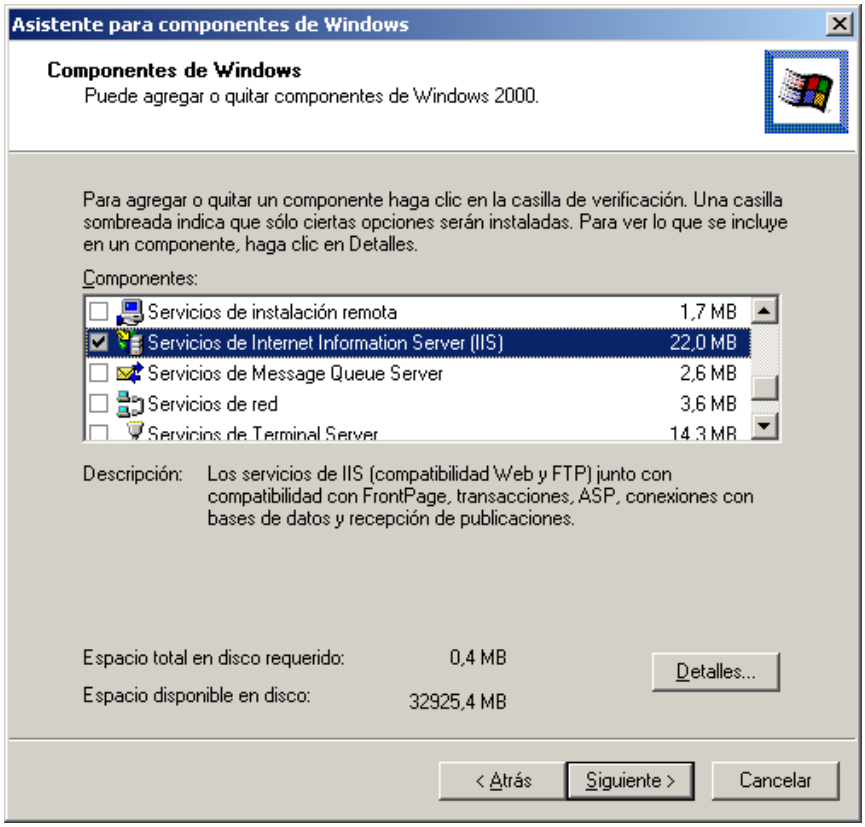


Figura 31

Instalación de ASP .NET

ASP .NET lo podemos obtener de dos formas distintas, a través del .NET Framework SDK y a través de Microsoft ASP .NET Premium Edition. Recomiendo que el lector se instale la versión completa del .NET Framework SDK, ya que contiene abundante documentación como pueden ser tutoriales y una completa referencia de la plataforma .NET, incluyendo por lo tanto una referencia de ASP .NET y de C# que son dos aspectos que nos interesan, y que nos pueden servir como material de apoyo de este texto.

El .NET Framework SDK lo podemos obtener de la dirección de Microsoft http://msdn.microsoft.com/downloads o bien si disponemos de Visual Studio .NET, el .NET Framework SDK se encuentra el disco compacto llamado Windows Component Update. El .NET Framework SDK se instala junto con Visual Studio .NET, de esta forma si tenemos Visual Studio .NET ya tendremos instalado el .NET Framework SDK.

Microsoft ASP .NET Premium Edition también se puede obtener de la misma dirección, pero como ya he comentado es más recomendable instalar el .NET Framework SDK. Si accedemos a este sitio Web de Microsoft, veremos una página similar a la de la Figura 32, en la que podemos seleccionar el producto deseado, además en este sitio Web hay bastante información sobre la plataforma .NET, por lo tanto su visita es más que recomendable.

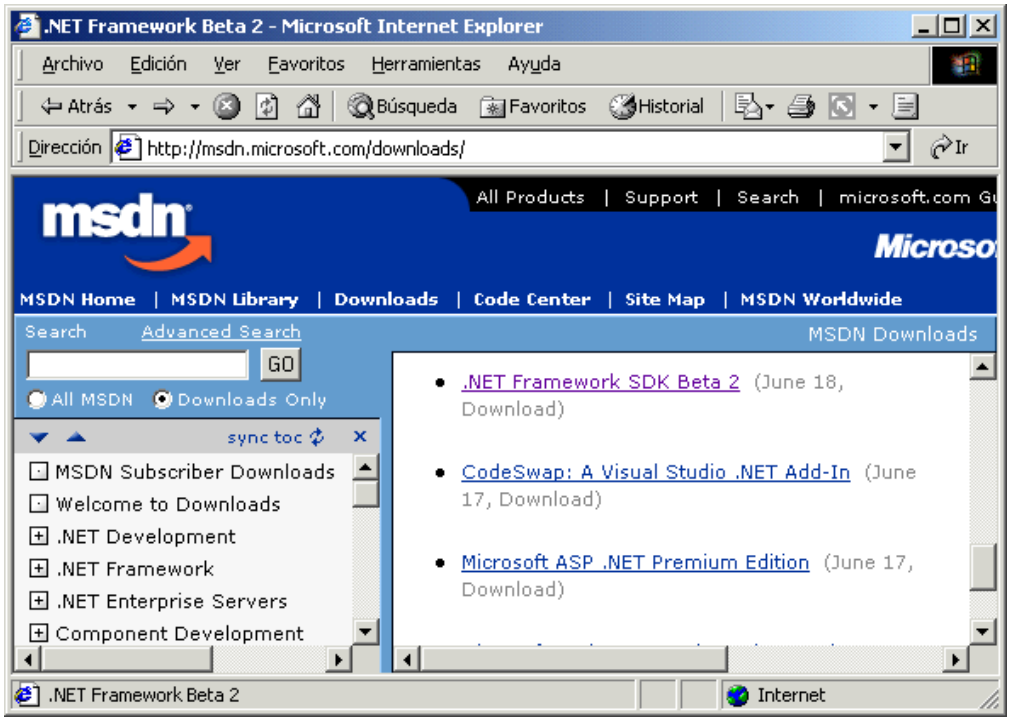


Figura 32

Una vez que tenemos los requisitos necesarios para instalar el .NET Framework SDK, es decir, Internet Explorer 6 y el Service Pack 2 de Windows 2000, podemos proceder con su instalación. Si realizamos la instalación de Visual Studio .NET y utilizamos el disco etiquetado como Windows Component Update, este disco de actualización nos instalará previamente los requisitos necesarios del .NET Framework SDK.

La instalación del .NET Framework SDK es muy sencilla, sólo debemos lanzar el ejecutable de instalación que mostrará un asistente que nos guiará por el proceso de instalación. En la Figura 33 se puede ver la pantalla que aparece en la fase de instalación del .NET Framework.

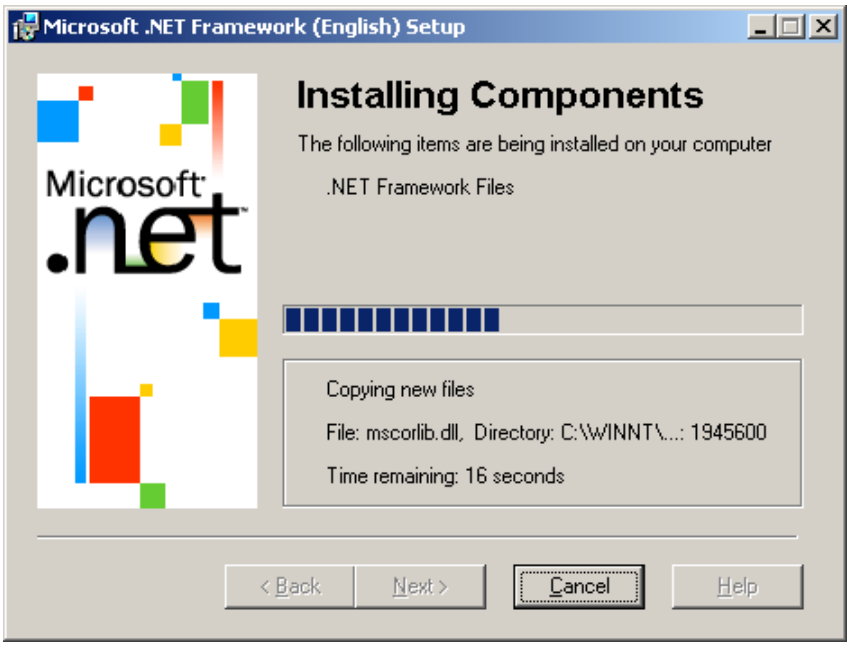


Figura 33

Al realizarse la instalación del .NET Framework SDK se nos indicará que de forma opcional se puede instalar Microsoft Data Access Components 2.7. Y también se nos preguntará si deseamos instalar Microsoft Windows Instaler 2.0, a esta pregunta debemos contestar afirmativamente.

Una vez instalado el .NET Framework SDK, es bastante interesante instalar los tutoriales que vienen con el paquete de desarrollo, estos tutoriales son denominados QuickStart Tutorials (tutoriales rápidos).

La instalación de estos tutoriales es muy sencilla, simplemente debemos acudir a la opción del menú de inicio Microsoft .NET Framework SDK|Samples and QuickStart Tutorials. Al seleccionar esta opción aparece una página Web que nos indica las instrucciones a seguir para instalar los tutoriales y los ejemplos.

La instalación se limita a pulsar dos enlaces. Primero el enlace "Install the .NET Framework Samples Database", y a continuación, una vez finalizada la primera fase de la instalación, se deberá pulsar el enlace "Complete the installation". Con estos dos sencillos pasos ya tendremos instalada la documentación correspondiente a los tutoriales y ejemplos de la plataforma .NET.

Para acceder al tutorial de ASP .NET deberemos escribir la siguiente URL en nuestro equipo http://localhost/quickstart/ASPPlus/, y tendremos acceso a una página como la de la Figura 34.

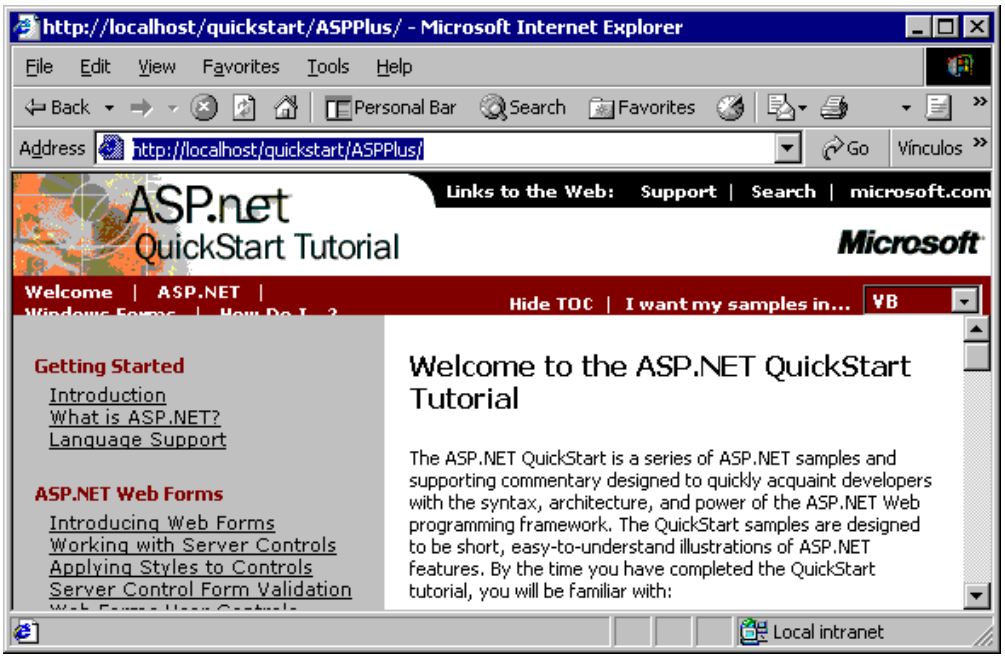


Figura 34

A modo de resumen podemos decir que para poder desarrollar páginas ASP .NET en nuestro equipo tenemos que seguir los siguientes pasos:

1. Instalar el servidor Web Internet Information Server.
2. Instalar el navegador Internet Explorer 5.5 o superior y el Service Pack 2 de Windows 2000.
3. Instalar el .NET Framework SDK.

Visual Studio .NET

Ya hemos comentado que no es necesario disponer de una herramienta de desarrollo determinada para crear nuestras aplicaciones ASP .NET. Pero Microsoft nos ofrece una nueva versión de Visual Studio, en este caso es la versión de Visual Studio para la plataforma .NET, es decir, Visual Studio .NET.

La instalación de Visual Studio .NET incluye la instalación del .NET Framework SDK. La instalación resulta muy sencilla, se introduce el primer disco de Visual Studio .NET, y a continuación se nos pedirá el disco etiquetado como Windows Component Update, el proceso de instalación de VS .NET se ha comentado en detalle en el capítulo anterior.

Desde este nuevo disco se instalarán automáticamente los requisitos necesarios para instalar Visual Studio .NET, que son los siguientes:

- Windows 2000 Service Pack 2.
- Microsoft Windows Instaler 2.0.
- Microsoft FrontPage 2000 Web Extensions Client.
- Setup Run-time Files.
- Microsoft Internet Explorer 6.0 e Internet Tools.
- Microsoft Data Access Components 2.7.
- Microsoft .NET Framework

Una vez instalados los componentes del disco Windows Component Update el proceso de instalación nos pedirá el disco uno de Visual Studio .NET para iniciar realmente la instalación de la herramienta de desarrollo. A partir de aquí la instalación de Visual Studio .NET es muy sencilla, el asistente nos guiará por la misma, aunque resultará lenta, pudiendo llegar a durar de una hora y media a dos horas.

Una vez instalado Visual Studio .NET podemos realizar nuestra primera aplicación ASP .NET. Para ello seleccionamos la opción del menú inicio llamada Microsoft Visual Studio .NET 7.0, como se puede observar en la Figura 35, no existe una opción para cada herramienta, como sucedía en versiones anteriores (Visual Basic, Visual InterDev, Visual C++, etc.), sino que tenemos un único punto de arranque y una única herramienta de desarrollo que muestra un entorno de desarrollo común.

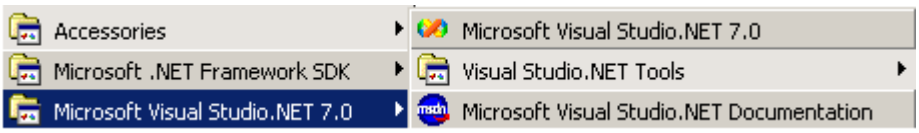


Figura 35

Cuando se inicia la ejecución de Visual Studio .NET debemos seleccionar la creación de un nuevo proyecto, esto se puede hacer con la opción de menú File|New|Project. Aparecerá entonces una ventana similar a la de la Figura 36. En esta ventana que debemos seleccionar el tipo de aplicación que vamos a desarrollar. En nuestro caso seleccionaremos la aplicación de tipo ASP .NET Web Application, utilizando el lenguaje C#.

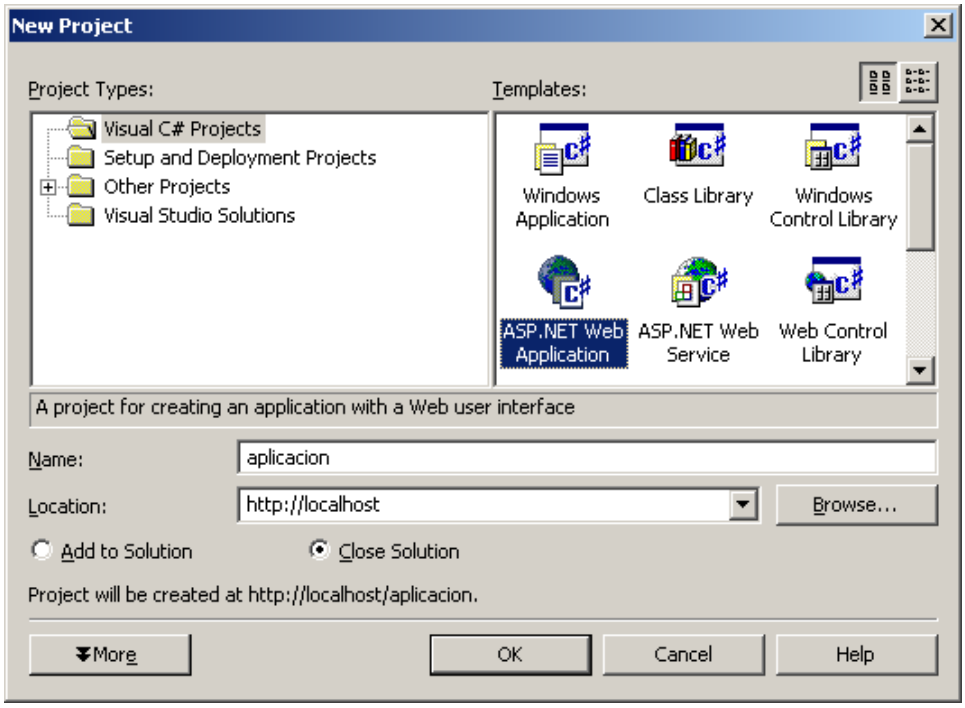


Figura 36

También debemos indicar el nombre del directorio que va a contener la aplicación y el servidor en el que se va a desarrollar. En nuestro caso como se puede ver en la Figura 36, la aplicación se llama aplicacion y se va a crear en la URL http://localhost/aplicacion, localhost va a ser el nombre genérico con el que se hace referencia a la máquina local, es decir, a nuestro propio equipo.

Cuando pulsamos el botón etiquetado como OK Visual Studio .NET procede a crear la aplicación ASP .NET en el lugar y con el nombre indicados. De forma automática se crean una serie de ficheros en la aplicación ASP .NET, que se pueden observar en la Figura 37, esta figura muestra el explorador de la solución (Solution Explorer) de Visual Studio .NET.

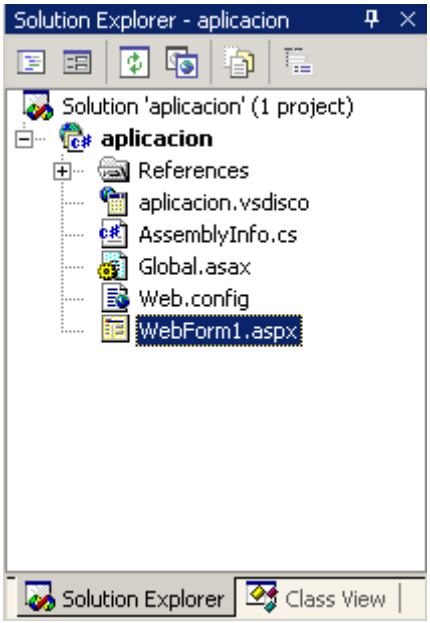


Figura 37

Los ficheros creados son: el fichero especial GLOBAL.ASAX, con funciones similares la fichero GLOBAL.ASA de las versiones anteriores de ASP, es decir, la de definir los eventos propios de la aplicación; el fichero WEB.CONFIG que va a contener la configuración de la aplicación ASP .NET, y una página ASP .NET llamada WEBFORM1.ASPX, que va a ser una página que contiene un Web Form y que podremos utilizar en nuestra aplicación ASP .NET de la forma que deseemos.

Si pulsamos sobre este fichero podemos acceder a la página ASP .NET para añadir un sencillo código fuente que nos permita mostrar la fecha y hora actuales. El código de la página se puede ver en el Código fuente 63.

```
<%@ Page language="c#"%>
<html>
<body>
<%DateTime ahora=DateTime.Now;%>
<%=ahora%>
</body>
</html>
```

Código fuente 63

Para ejecutar nuestra página ASP .NET debemos escribir la URL http://localhost/aplicación/webform1.aspx en nuestro navegador Web, el resultado de la ejecución de esta página ASP .NET se puede ver en la Figura 38.

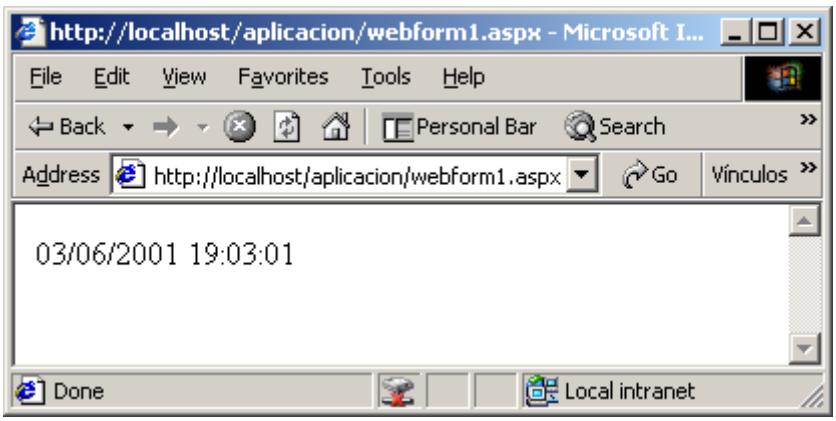


Figura 38

A grandes rasgos se han comentado los distintos elementos que nos van a permitir desarrollar aplicaciones ASP .NET.

En el siguiente capítulo trataremos una de las características o elementos fundamentales que presenta ASP .NET, los formularios Web (Web Forms).

7

Web Forms: introducción y controles HTML

Introducción a los Web Forms

Los Web Forms (formularios Web) es una característica de ASP .NET que ofrece un nuevo modelo de programación que nos va a permitir generar contenidos dinámicos de una forma más sencilla. Esta característica no existía en versiones anteriores de ASP.

Los Web Forms vienen a sustituir e integrar los formularios HTML dentro del entorno de programación de ASP .NET. Un Web Form en última instancia va a generar el código HTML correspondiente al formulario que estamos diseñando.

Una de las grandes ventajas que ofrecen los Web Forms es que ofrecen un modelo de programación muy similar al de los tradicionales formularios de Visual Basic, incluso podemos realizar el tratamiento de eventos del cliente desde código del servidor. Se puede decir que vamos a realizar el tratamiento de los eventos del cliente con código ASP .NET, es decir, con código que será ejecutado en el servidor Web.

Los Web Forms tienen tanta importancia dentro de ASP .NET que incluso en el entorno de desarrollo Visual Studio .NET cuando queremos añadir una nueva página, no parece la opción de añadir nueva página ASP .NET, sino que se muestra la posibilidad de añadir un nuevo Web Form, es decir, en algunos lugares las páginas ASP .NET también se denominan Web Forms.

A continuación se van a describir de forma general las ventajas que nos ofrecen los Web Forms, en los siguientes capítulos se verán en detalle el modelo de programación ofrecidos por los Web Forms junto

con los controles ASP .NET de servidor. Las ventajas que nos ofrecen los Web Forms son las siguientes:

- Nos permiten reutilizar controles de interfaz de usuario (UI) que encapsulan su funcionalidad, reduciendo de esta forma el código que el desarrollador debe escribir.

- Ofrece un código más limpio y claro en la página, ya que no aparece mezclado con código HTML como ocurría en la versiones anteriores de ASP, en las que era bastante usual intercalar código HTML con código ASP, aunque en ASP .NET esto también se permite por compatibilidad, pero si nos ceñimos al modelo de programación que ofrecen los Web Forms no es necesario.

- Podemos construir todo el interfaz del Web Form a través de herramientas o entornos de desarrollo, al igual que construimos los formularios con Visual Basic arrastrando y soltando los controles del interfaz de usuario. Visual Studio .NET nos ofrece una vista de diseño para los Web Forms, que incluso nos permite diseñar los Web Forms situando sus controles según las posiciones x e y de la pantalla.

- Integra el tratamiento de eventos del cliente con el código correspondiente del servidor. El tratamiento de eventos es muy similar al que ya se utilizaba en los formularios de las anteriores versiones de Visual Basic.

- Realizan un mantenimiento del estado del formulario (Web Form) de forma automática entre distintas peticiones, es decir, si seleccionamos una serie de valores en el Web Form, al enviarlo de forma automática se van a mantener estos valores en el caso de que se vuelva a recargar el formulario.

- Ofrece una mayor separación entre la lógica de la aplicación y la presentación de la misma.

- Permite utilizar un gran conjunto de controles de servidor que generarán el código HTML necesario para mostrar el Web Form en la página cargada por el cliente en su navegador. Después veremos que dentro de los Web Forms podemos encontrar varios tipos de controles.

Para crear un formulario Web o Web Form únicamente debemos utilizar una etiqueta <FORM>, al estilo de las etiquetas del lenguaje HTML. Pero presenta la particularidad que debe poseer al propiedad runat con el valor server. En el Código fuente 64 se puede ver un fragmento de código que todo Web Form debe poseer.

```
<form id="WebForm" method="post" runat="server">
</form>
```

Código fuente 64

Como se puede comprobar es muy similar al código de un formulario de HTML. Por lo tanto el código necesario para convertir un formulario HTML en un Web Form es bastante sencillo, eso sí, sin tener en cuenta los controles de servidor que puede contener el Web Form.

Dentro del Web Form se encontrarán los distintos controles de servidor de ASP .NET necesarios para recoger la información correspondiente. Es decir, un Web Form se va a encontrar compuesto por los distintos controles de servidor de ASP .NET, que realizarán las mismas funciones que los típicos controles del lenguaje HTML que constituían los campos de un formulario.

Estos controles de servidor están muy unidos a los Web Form, ya que es dentro de los formularios Web dónde vamos a hacer uso de estos controles. Los controles de servidor, también denominados controles de servidor ASP .NET o simplemente controles ASP .NET, suponen otra nueva y revolucionaria funcionalidad ofrecida por ASP .NET. Los controles ASP .NET combinados con los Web Forms nos van a permitir construir páginas ASP .NET dinámicas de forma muy sencilla.

Podemos decir que los Web Form junto con los controles ASP .NET ofrecen un nuevo modelo de programación dentro de ASP .NET.

Introducción a los controles ASP .NET

Los controles ASP .NET son una serie de objetos de servidor que generarán el correspondiente código HTML para que el usuario pueda utilizarlos en la página cargada en su navegador. Estos controles nos van a permitir manejar eventos generados por el cliente, como puede ser una pulsación de un botón, con código de servidor.

Gracias a la los controles ASP .NET ya no será necesario utilizar los típicos campos de formulario de HTML, es decir, los controles del lenguaje HTML, sino que al ejecutarse la página ASP .NET correspondiente los controles de servidor de ASP .NET generarán todo el código equivalente en el lenguaje HTML, para que así pueda ser interpretado el resultado de la ejecución de la página por cualquier navegador Web.

Los controles ASP .NET nos una muestra más del interesante cambio en la filosofía de desarrollo que aporta ASP .NET, ahora el código de servidor, va a estar separado de forma más clara, que en el pasado, del código del cliente, dando lugar a un código mucho más sencillo, estructurado y por lo tanto legible.

Cada control se corresponde con una clase determinada perteneciente a un espacio con nombre determinado. Existe un gran número de controles de servidor y cada uno de ellos pertenece a una clase del .NET Framework.

Los numerosos controles de servidor incluidos en ASP .NET se pueden agrupar en cuatro categorías o familias:

- Controles intrínsecos, que a su vez se subdividen en dos grupos, y que representan elementos HTML. Los dos grupos que representan los controles intrínsicos se denominan: controles HTML y controles Web, más adelante veremos las diferencias y utilización de estos dos tipos de controles.

- Controles de lista, utilizados para distribuir y mostrar datos en una página.

- Controles ricos, ofrecen funcionalidades avanzadas de interfaz de usuario.

- Controles de validación, ofrecen distintos tipos de validación de entrada de datos.

Las tres últimas categorías se pueden decir que ese encuentran reunidas dentro de los controles Web, es decir, los controles HTML están limitados únicamente a los controles intrínsecos.

A lo largo de los siguientes apartados vamos a ir comentando brevemente los distintos tipos de controles de servidor.

Además de los controles HTML estándar que representan campos de un formulario, los controles de servidor de ASP .NET permiten a los desarrolladores utilizar controles más ricos y complejos, como

puede ser el control de rotación de anuncios (AdRotator), para algunos lectores este control les será familiar de las versiones anteriores de ASP .NET.

Cada control ASP .NET es capaz de ofrecer un completo modelo de objetos que contiene propiedades, métodos y eventos, no debemos olvidar que los controles ASP .NET son instancias de objetos que podemos encontrar en distribuidos en dos espacios con nombre del .NET Framework, se trata de los espacios con nombre: System.Web.UI.HtmlControls y System.Web.UI.WebControls.

En algunos casos nos puede interesar utilizar controles Web, y en otros controles HTML, los controles Web deberán ser usados en las siguientes situaciones:

- Preferimos un modelo de programación similar a Visual Basic.
- Estamos desarrollando Web Forms que deben ser mostrados por varios tipos de navegadores.
- Se necesita una funcionalidad específica, como puede ser un calendario o un rotador de anuncios.

Y los controles HTML es preferible utilizarlos en las siguientes situaciones:

- Preferimos un modelo de objetos similar al lenguaje HTML.
- Estamos trabajando con páginas Web existentes y las queremos migrar a Web Forms.
- El control debe interactuar con script de cliente y de servidor.

Hola Mundo con Web Forms

Antes de seguir comentando más aspectos de los Web Forms y de los controles de servidor de ASP .NET vamos a mostrar y comentar un sencillo ejemplo que utiliza un Web Form con varios controles ASP .NET, se podría considerar que es el típico ejemplo "Hola Mundo" pero con un Web Form.

El ejemplo consiste en crear un Web Form que contenga un botón que al ser pulsado muestre por pantalla el mensaje "Hola Mundo". En el Código fuente 65 se puede ver el código de la página ASP .NET que contiene este formulario Web.

```
<html>
<body>
<script language="C#" runat="server">
void Pulsado(Object sender, EventArgs args){
      etiqueta.InnerText ="Hola Mundo";
}
</script>
<form id="controlesHTML" method="post" runat="server">
<input type="button" id="boton" runat="server"  value="Pulsar"
                                    onserverclick="Pulsado" NAME="boton">
<div id="etiqueta" runat="server"></div>
</form>
</body>
</html>
```

Código fuente 65

En la Figura 39 se puede ver el resultado de la ejecución de esta página ASP .NET.

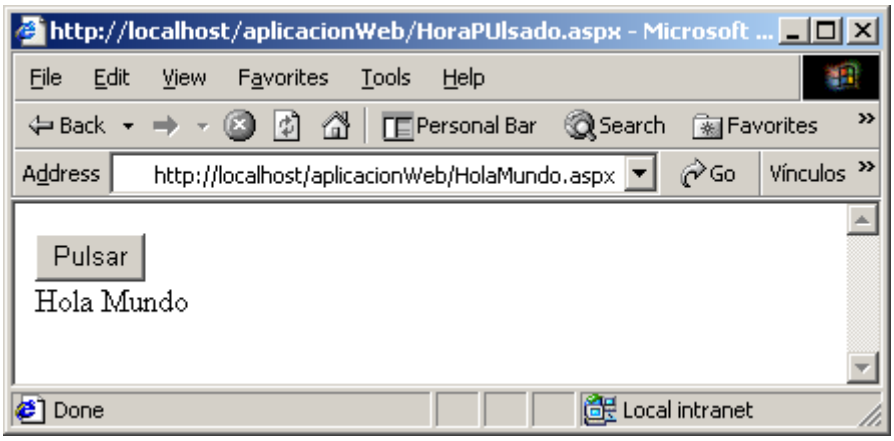


Figura 39

El Web Form contiene dos controles ASP .NET, un botón que será un objeto de la clase HtmlInputButton y una etiqueta que será un objeto de la clase HtmlGenericControl, ambas clases se encuentran en el espacio de nombres System.Web.UI.HtmlControls. En este caso el Web Form para interactuar con el usuario utiliza controles HTML.

Además, a la vista del código del ejemplo se pueden hacer los siguientes comentarios. La sintaxis que ofrecen los controles HTML de servidor es muy similar a la sintaxis de los controles clásicos del lenguaje HTML. Para identificar de forma única un control ASP .NET se debe utilizar la propiedad id, mediante este identificador podemos hacer referencia al objeto correspondiente en la página. Para indicar que en la pulsación del botón se debe ejecutar un método determinado, es decir, el método que realiza el tratamiento del evento, se utiliza la propiedad onserverclick, esto es específico de los controles HTML, luego veremos las diferencias de sintaxis con los controles Web, en el caso de los controles HTML la sintaxis de los mismo le será muy familiar al lector ya que es al estilo del lenguaje HTML, sin embargo, como veremos más adelante, la sintaxis de los controles Web está basada en XML.

Si queremos utilizar controles Web en nuestro ejemplo anterior en lugar de controles HTML podemos utilizar el Código fuente 66.

```
<html>
<body>
<script language="c#" runat="server">
void Pulsado(Object fuente, EventArgs args){
      etiqueta.Text="Hola Mundo";
}
</script>
<form id="formulario" method="post" runat="server">
      <asp:label  id="etiqueta" runat="Server"></asp:label>
      <asp:button id="boton" onclick="Pulsado" runat="server"
text="Pulsa"></asp:button>
</form>
</body>
</html>
```

Código fuente 66

En este nuevo ejemplo se puede comprobar la diferencia existente entre ambos tipos de controles ASP .NET, por lo menos en lo que se refiere a nivel de sintaxis. Como se puede observar el código de servidor, es decir, el código escrito en C# permanece idéntico en ambos ejemplos.

Para terminar con el comentario de este ejemplo de Hola Mundo con un Web Form, podemos ver el código fuente de la página HTML que se ha generado como resultado de la ejecución de nuestra página ASP .NET, este código se ofrece a continuación (Código fuente 67).

```
<html>
<body>
<form name="formulario" method="post" action="Holamundo.aspx" id="formulario">
<input type="hidden" name="__VIEWSTATE"
value="dDwtOTk1MjE0NDA4O3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDE+Oz47bDx0PHA8cDxsPFRleHQ7Pjts
PEhvbGEgTXVuZG87Pj47Pjs7Pjs+Pjs+Pjs+" />

      <span id="etiqueta">Hola Mundo</span>
      <input type="submit" name="boton" value="Pulsa" id="boton" />
</form>
</body>
</html>
```

Código fuente 67

Como se puede comprobar se han generado de forma automática las etiquetas HTML correspondientes, pero también se ha añadido dentro del formulario un campo oculto denominado _VIEWSTATE con un extraño valor. Este campo oculto, que volveremos a comentar en próximos ejemplos, permite al Web Form mantener el estado entre distintas llamadas, es decir, en distintas peticiones se va conservando el valor de los campos del formulario.

Controles HTML

Estos controles son uno de los tipos de controles intrínsecos que podemos encontrar dentro de los controles ASP .NET. Los controles HTML ofrecen a los desarrolladores de entornos Web la potencia de los Web Forms manteniendo la familiaridad y facilidad de uso de las etiquetas HTML que representan los campos de un formulario.

Como ya se comentó estos controles tienen el mismo aspecto que una etiqueta HTML a excepción de que presentan el atributo runat con el valor server. Otra diferencia es que posee un atributo especial para el tratamiento de eventos del cliente con código del servidor. Si queremos realizar el tratamiento de eventos con código del servidor, utilizaremos el atributo onserverevento, a este atributo se le indica como valor el nombre del método que queremos ejecutar. El tratamiento de eventos con los controles ASP .NET lo retomaremos más adelante.

Para permitir hacer referencia a los controles dentro del código fuente de la página, se debe utilizar el atributo id, de esta forma el objeto que se corresponde con el control de servidor se puede utilizar en el código de servidor para utilizar sus métodos o manipular sus propiedades.

En el Código fuente 68 se puede ver la sintaxis general que presentan los controles HTML, y en el Código fuente 69 se muestra un ejemplo de utilización de un Web Form con varios controles HTML de ASP .NET. Se trata de un formulario con una lista desplegable y un botón, al pulsar el botón se muestra el elemento seleccionado de la lista en un texto de un elemento .

```
<input type="elementoHTML" id="identificador" runat="server" >
```

Código fuente 68

```
<%@ Page language="c#"%>
<HTML>
<script language="C#" runat="server">
      void Pulsado(Object sender, EventArgs args){
            etiqueta.InnerText=lista.Value;
      }
</script>
<body>
<form id="WebForm" method="post" runat="server">
<select id="lista" runat="server">
      <option selected>La Naranja Mecánica</option>
      <option>El Resplador</option>
      <option>Lolita</option>
      <option>2001</option>
</select>
<input type="button" id="boton" runat="server" value="seleccionar"
onserverclick="Pulsado">
<div id="etiqueta" runat="server"></div>
</form>
</body>
</HTML>
```

Código fuente 69

En la Figura 40 se puede observar un ejemplo de ejecución de esta página.

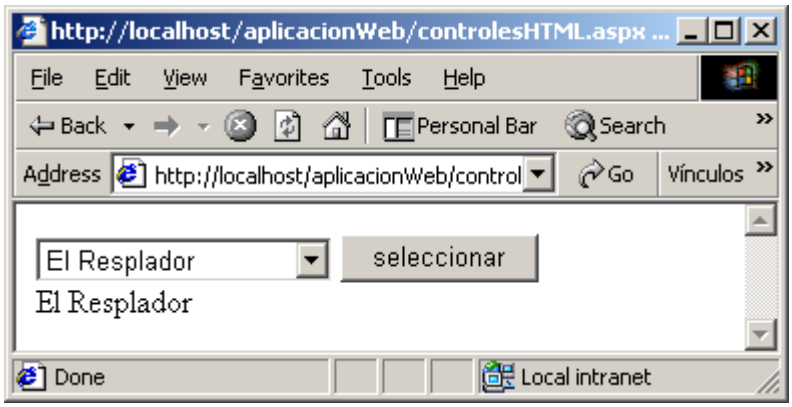


Figura 40

Un aspecto importante es que, como se puede comprobar, no hemos utilizado la colección Form del objeto Request para obtener los valores de los elementos enviados en el formulario, sino que simplemente hemos accedido a las propiedades de los controles a través del nombre con el que los hemos declarado, es decir, al estilo de los formularios de Visual Basic. Esto es posible gracias a que cada control del formulario es un objeto (objetos de la página ASP .NET), es decir una instancia de una clase determinada, y por lo tanto cada uno de ellos poseerá sus propiedades y métodos.

En este mismo ejemplo también se puede observar que en distintas llamadas al formulario se mantiene el estado del mismo, es decir, se mantiene en la lista el elemento seleccionado, sin que tengamos que

escribir ningún código adicional, esto es posible gracias al campo oculto que genera el Web Form y que se denomina _VIEWSTATE. Por lo tanto los Web Forms nos permiten en distintas peticiones conservar los valores de los campos de los formularios.

Estos controles se ofrecen para mantener la compatibilidad con versiones anteriores de ASP, así si queremos que una etiqueta HTML pase a ser un control HTML simplemente debemos indicar la propiedad runat con el valor server, normalmente se suelen utilizar el siguiente tipo de controles intrínsecos que veremos en el próximo apartado, se trata de los controles Web.

El espacio con nombre (NameSpace) en el que podemos encontrar los controles HTML se denomina System.Web.UI.HtmlControls, no debemos olvidar que aunque tengan un gran parecido con la sintaxis HTML, los controles HTML, como todos los controles ASP .NET, tienen sus clases correspondientes dentro del .NET Framework.

El elemento clave para trabajar con estos controles de servidor es el atributo runat=”server”. Cuando se establece este atributo se activan los eventos para su tratamiento en el servir y el mantenimiento automático del estado de los controles dentro del Web Form. Si no se establece este atributo el control funcionará como un elemento del lenguaje HTML. Dentro de un Web Form podemos tener controles ASP .NET y controles pertenecientes a elementos HTML, aunque no es muy común.

A continuación vamos a describir brevemente las distintas clases que se corresponden con los controles HTML y que se encuentran dentro del espacio con nombre System.Web.UI.HtmlControls, suponemos que el lector conoce el lenguaje HTML. Estas clases ofrecen al desarrollador el acceso en el servidor a las distintas etiquetas HTML que finalmente se enviarán al cliente Web como el resultado de la ejecución de la página ASP .NET correspondiente.

A algunas descripciones de las clases de los controles HTML les acompaña un ejemplo de utilización, en estos ejemplos veremos como se utilizan los controles HTML dentro de una página ASP .NET y también veremos algunos ejemplos con tratamiento de eventos.

Las distintas clases que representan a los controles HTML son las que se comienzan a detallar a continuación.

HtmlAnchor

Esta clase se corresponde con la etiqueta <a>, es decir, es el control HTML que nos permite manejar enlaces. El Código fuente 70 muestra la utilización de esta clase, como se puede ver es muy sencilla y es muy similar a su utilización dentro del lenguaje HTML. En este caso al pulsar un botón se cambia la propiedad HRef. del objeto de la clase HtmlAnchor, para que el enlace apunte a otro destino distinto. Aquí nos adelantamos al siguiente control HTML y hacemos uso de un objeto de la clase HtmlButton.

```
<%@ Page language="c#" %>
<script language="c#" runat="server">
      void BotonPulsado(Object fuente, EventArgs argumento){
            enlace.HRef="HolaMundo.aspx";
      }
</script>
<HTML>
<body>
<form id="formulario" method="post" runat="server">
      <a href="fecha.aspx" id="enlace" runat="server">Pulsar enlace</a><br>
      <button runat="server" id="boton" onserverclick="BotonPulsado">
     <b>Pulsa</b></button>
```

```
</form>
</body>
</HTML>
```

Código fuente 70

Como se puede comprobar en el código anterior, se ha utilizado un método para el tratamiento de eventos que hemos denominado BotonPulsado(). Este método se ejecutará cuando se pulsa el botón y recibe como parámetros un objeto de la clase genérica Object que representa al objeto que ha generado el evento, es decir, la fuente u origen del evento, y también un objeto de la clase EventArgs que representa al evento que se ha producido, pulsación, selección, etc.

Para indicar el método que queremos que se ejecute cuando se pulse sobre el objeto de la clase HtmlButton utilizamos la propiedad onserverclick de la etiqueta <button>, a esta propiedad le asignaremos el nombre del método correspondiente.

Si vemos el código fuente que se genera (Código fuente 71) como resultado de la ejecución de esta página ASP .NET, podemos comprobar que se genera código JavaScript para que la etiqueta <button> pueda generar el evento de pulsación del botón y enviar el formulario al servidor para así cambiar el enlace. El enlace se traduce directamente a una etiqueta <a> con sus propiedades correspondientes.

```
<HTML>
<body>
<form name="formulario" method="post" action="enlace.aspx" id="formulario">
<input type="hidden" name="__VIEWSTATE" value="dDwxNDg4NzI4MDg0Ozs+" />

      <a href="/aplicacionWeb/fecha.aspx" id="enlace">Pulsar enlace</a><br>
      <button language="javascript" onclick="__doPostBack('boton','')"
id="boton"><b>Pulsa</b></button>

<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
      function __doPostBack(eventTarget, eventArgument) {
            var theform = document.formulario;
            theform.__EVENTTARGET.value = eventTarget;
            theform.__EVENTARGUMENT.value = eventArgument;
            theform.submit();
      }
// -->
</script>
</form>
</body>
</HTML>
```

Código fuente 71

En la Figura 41 se puede ver un ejemplo de ejecución de esta sencilla página.

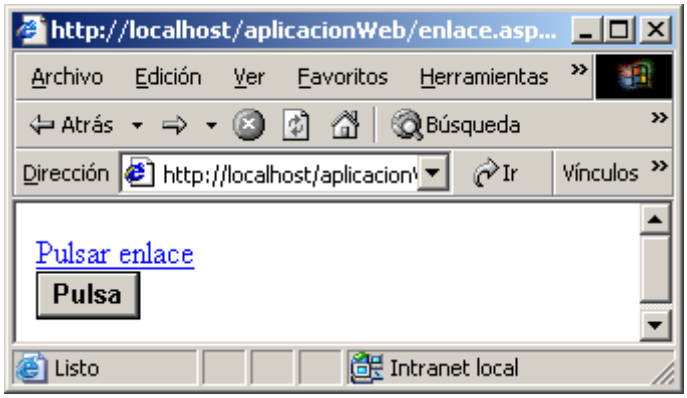


Figura 41

HmlButton

Este control HTML generará la etiqueta <button> de HTML, esta etiqueta pertenece a la especificación 4.0 de HTML, y únicamente es soportada por el navegador Web Internet Explorer 4.0 o superior. Esta control HTML se utilizará cuando sea necesario personalizar al máximo el aspecto de un botón, ya que permite construir botones formados por otros elementos de HTML. Un ejemplo de utilización de esta clase se puede ver en el Código fuente 70.

HtmlForm

Esta clase representa a la etiqueta <form> en el servidor, es la que permite definir un Web Form, que va a realizar la labor de contenedor para una serie de controles de servidor dentro de la página ASP .NET. Todos los controles ASP .NET que deseemos enviar al servidor (post) debemos incluirlos dentro de un control de la clase HtmlForm.

Ejemplos de utilización de este control ya los hemos estado viendo en varios ejemplos a lo largo del texto, y en los siguientes ejemplos también lo veremos, ya que es una pieza fundamental de los Web Forms.

HtmlGenericControl

Esta clase se utilizará para representar las etiquetas HTML que no poseen una correspondencia directa con las clases del .NET Framework, como puede suceder con las etiquetas , o <body>, entre otras.

El ejemplo del Código fuente 72 muestra la utilización de la clase HtmlGenericControl a través de una etiqueta <body>, en este caso al seleccionar un elemento de la lista y pulsar el botón que posee el formulario se mostrará el documento HTML con el color seleccionado de la lista desplegable.

```
<%@ Page language="c#"%>

<HTML>
<script language="C#" runat="server">
      void CambiaColor(Object sender, EventArgs args){
            cuerpo.Style["background-color"]=lista.Value;
      }
```

```
</script>
<body id="cuerpo" runat="server">
<form id="WebForm" method="post" runat="server">
<select id="lista" runat="server" NAME="lista">
      <option value="red" selected>rojo</option>
      <option value="green">verde</option>
      <option value="blue">azul</option>
      <option value="yellow">amarillo</option>
</select>
<input type="button" id="boton" runat="server" onserverclick="CambiaColor"
value="Cambia">
</form>
</body>
</HTML>
```

Código fuente 72

En este caso además de utilizar un control HTML de la clase HtmlGenericControl, también se utilizan dos objetos de las clases HtmlInputButton y HtmlSelect, que se corresponden con el botón y la lista desplegable del formulario.

En la Figura 42 se puede ver un ejemplo de ejecución de la página ASP .NET que nos ocupa.

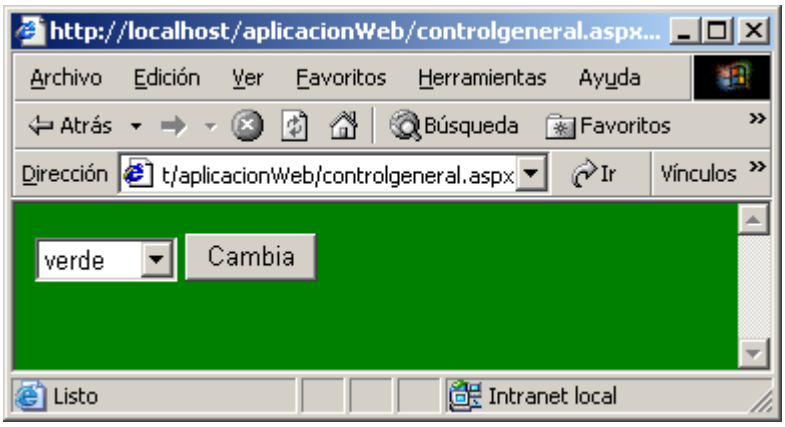


Figura 42

En este caso el botón actúa como una etiqueta <input type=”button”>, es decir, un botón del formulario que ni lo envía ni lo resetea.

En este ejemplo también se muestra como aplicar estilos a los controles ASP .NET, se utiliza la propiedad Style que funciona como una colección permitiendo acceder a las distintas propiedades del control en lo que a hojas de estilo en cascada se refiere (CSS, Cascading Style Sheet), en este caso se manipula la propiedad background-color, para establecer el color de fondo de la página actual.

Si realizamos varias pruebas con esta página, podemos ver en acción una de las características de los Web Forms, el mantenimiento del estado de forma automática de los distintos controles del formulario. El Código fuente 73 es el código HTML que se genera como resultado de la ejecución de la página ASP .NET del ejemplo.

```
<HTML>
<body id="cuerpo" style="background-color:green;">
<form name="WebForm" method="post" action="controlgeneral.aspx" id="WebForm">
```

```
<input type="hidden" name="__VIEWSTATE"
value="dDwxMjg1MTY5MjM4O3Q8O2w8aTwxPjs+O2w8dDxwPGw8c3R5bGU7PjtsPGJhY2tncm91bmQtY29s
b3I6Z3JlZW5cOzs+Pjs7Pjs+Pjs+" />

<select name="lista" id="lista">
	<option value="red">rojo</option>
	<option selected="selected" value="green">verde</option>
	<option value="blue">azul</option>
	<option value="yellow">amarillo</option>
</select>
<input language="javascript" onclick="__doPostBack('ctrl0','')" name="ctrl0"
type="button" value="Cambia" />

<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
	function __doPostBack(eventTarget, eventArgument) {
		var theform = document.WebForm;
		theform.__EVENTTARGET.value = eventTarget;
		theform.__EVENTARGUMENT.value = eventArgument;
		theform.submit();
	}
// -->
</script>
</form>
</body>
</HTML>
```

Código fuente 73

HtmlImage

Esta clase permite utilizar la etiqueta <img> de HTML en el servidor, es decir, el control <img runat=”server”>. Nos permite por lo tanto manejar imágenes.

El Código fuente 74 es un ejemplo de utilización de este control HTML, consiste en mostrar la imagen que se indique en la caja de texto, es decir, en un objeto de la clase HtmlInputText. La imagen se actualiza con la que indiquemos cada vez que se pulse el botón del formulario.

```
<%@ Page language="c#"%>

<HTML>
<script language="C#" runat="server">
	void CambiaImagen(Object sender, EventArgs args){
		imagen.Src=texto.Value;
	}
</script>
<body id="cuerpo" runat="server">
<form id="WebForm" method="post" runat="server">
<input type="text" id="texto" runat="server" value="">
<input type="button" runat="server" onserverclick="CambiaImagen" value="Cambia"
ID="boton">
<img src="figura1.png" runat="server" id="imagen">
</form>
</body>
</HTML>
```

Código fuente 74

En la Figura 43 se puede ver un ejemplo de ejecución de este formulario Web.

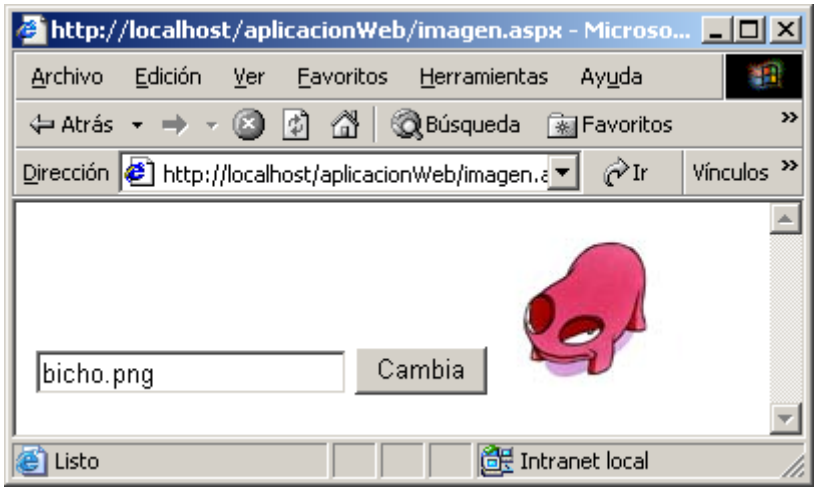


Figura 43

HtmlInputButton

Representa en el servidor las etiquetas <input type="button">, <input type="submit"> y <input type="reset">, es decir, los botones que podemos encontrar dentro de un formulario, botón normal, de envío y de limpieza o restauración del formulario. El tipo de botón que se genera viene dado por el valor de la propiedad type.

Hemos visto ya un par de ejemplos de utilización de este control HTML en combinación con otros controles dentro de un Web Form.

HtmlInputCheckBox

Como el lector habrá sospechado, este control se corresponde con la etiqueta <input type="checkbox">, y gracias a el tenemos acceso en el servidor a este tipo de elemento. Un ejemplo de utilización del control HTML se ofrece en el Código fuente 75.

HtmlInputFile

Esta clase ofrece el acceso a la etiqueta <input type="file">, se debe tener en cuenta que para usar este control el atributo de Enctype de la etiqueta <form> que contenga a este elemento debe tener el valor "multipart/form-data".

HtmlInputHidden

Esta clase se corresponde con la etiqueta <input type="hidden">, y se trata del control de servidor que nos permite utilizar campos ocultos dentro de un Web Form.

HtmlInputImage

Esta clase representa a la etiqueta <input type=”image”>, es decir, ofrece acceso a los campos imagen de un formulario, es decir, los botones gráficos de un formulario.

El Código fuente 75 es un ejemplo de uso de este control HTML, al pulsar la imagen se envía el contenido del formulario Web al servidor y se muestra en un control HtmlGenericControl, que en este caso se corresponde con una etiqueta de HTML.

```
<%@ Page language="c#"%>

<HTML>
<script language="C#" runat="server">
      void Page_Load(Object fuente,EventArgs evento){
            if(Page.IsPostBack)
                  informacion.InnerText=texto.Value+" "+casilla.Checked;
      }
</script>
<body id="cuerpo" runat="server">
<form id="WebForm" method="post" runat="server">
<input type="text" id="texto" runat="server" value="" NAME="texto">
<input type="checkbox" id="casilla" runat="server">
<input type="image" runat="server" src="boton.png" ID="botonImagen">
<div runat="server" id="informacion"></div>
</form>
</body>
</HTML>
```

Código fuente 75

Como se puede comprobar los elementos del formulario que se envían son dos controles de las clases HtmlText y HtmlInputCheckBox.

En este caso el evento que se trata es un evento de la página, representada por la clase Page, esta clase la trataremos en profundidad en un próximo capítulo. El evento tratado se denomina Page_Load, este evento tendrá lugar cada vez que se cargue la página. Para distinguir si es la primera vez que se ha cargado la página, para recuperar o no los valores del formulario, se hace uso de la propiedad IsPostBack de la clase Page, esta propiedad tendrá el valor verdadero si la página ha sido enviada al servidor alguna vez, por ejemplo mediante un botón de submit o con el control HTML que nos ocupa. En la Figura 44 se ofrece el resultado de la ejecución de este nuevo ejemplo.

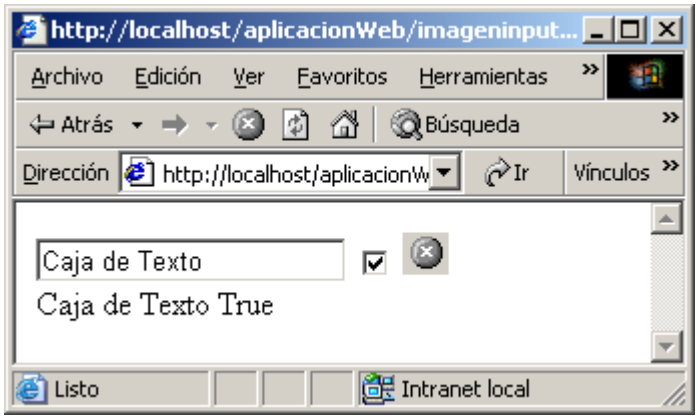


Figura 44

HtmlInputRadioButton

El control HTML que es instancia de esta clase generará en el cliente una etiqueta <input type=”radio”>, permitiendo al código de servidor interactuar con este elemento.

HtmlInputText

Se corresponde con las etiquetas <input type=”text”> y <input type=”password”>. El tipo de caja de texto que se genera viene dado por el valor de la propiedad type. Este control ya lo hemos visto en acción en varios ejemplos.

HtmlSelect

Este control HTML se corresponde con una lista desplegable del lenguaje HTML, es decir, con una etiqueta <select>.

HtmlTable

Este control se corresponde con una tabla del lenguaje HTML, es decir, permite el acceso desde el código de servidor a una etiqueta <table>. Esta clase posee una colección llamada Rows, que contiene objetos de la clase HtmlTableRow.

En el Código fuente 76 se muestra un ejemplo de utilización de este control HTML, se trata de un formulario que permite indicar el número de filas y de columnas que deseamos que tenga una tabla.

```
<html>
<script language="C#" runat="server">
      void Page_Load(Object sender, EventArgs e) {
            if (Page.IsPostBack){
                  int numeroFilas = int.Parse(filas.Value);
                  int numeroColumnas = int.Parse(columnas.Value);
                  for (int j=1; j<=numeroFilas; j++) {
                        HtmlTableRow fila = new HtmlTableRow();
                        for (int i=1; i<=numeroColumnas; i++) {
                              HtmlTableCell columna = new HtmlTableCell();
                              columna.InnerText="["+j+","+i+"]";
                              fila.Cells.Add(columna);
                        }
                        tabla.Rows.Add(fila);
                  }
            }
      }

    </script>
<body>
<form runat="server" ID="formulario">

<table id="tabla" CellPadding=5 CellSpacing="0" Border="1" runat="server">
</table>
<br>
Filas:
<select id="filas" runat="server">
  <option Value="1">1</option>
  <option Value="2">2</option>
```

```
  <option Value="3">3</option>
  <option Value="4">4</option>
  <option Value="5">5</option>
</select>
Columnas:
<select id="columnas" runat="server" NAME="Select2">
  <option Value="1">1</option>
  <option Value="2">2</option>
  <option Value="3">3</option>
  <option Value="4">4</option>
  <option Value="5">5</option>
</select>
<input type="submit" value="Generar tabla" runat="server" ID="boton">

</form>

</body>
</html>
```

Código fuente 76

Como se puede comprobar se utilizan objetos de la clase HtmlTableRow y HtmlTableCell para crear las filas y columnas de la tabla. Para añadir una columna se debe añadir un objeto HtmlTableCell a la colección Cells del objeto HtmlTableRow correspondiente, y para añadir una fila se añade un objeto HtmlTableRow a la colección Rows del objeto de la clase HtmlTable.

En la Figura 45 se puede ver un ejemplo de ejecución de esta página.

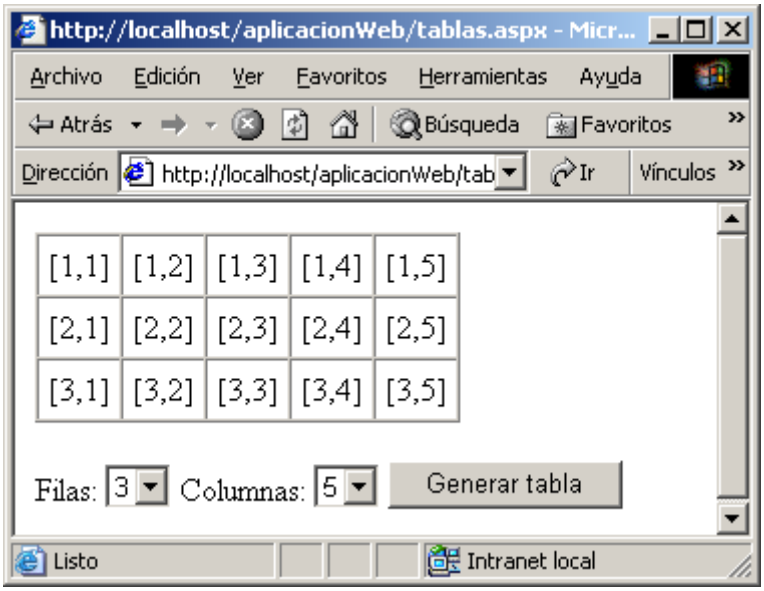


Figura 45

HtmlTableRow

Clase relacionada con la anterior, es el control HTML que permite manipular una fila de una tabla. Esta clase posee una colección llamada Cells, que contiene objetos de la clase HtmlTableCell.

HtmlTableCell

Clase que permite el acceso a las celdas de una tabla, es decir, se corresponde con las etiquetas <td> y <th>.

HtmlTextArea

Este último control HTML permite utilizar y manipular en el servidor la etiqueta <textarea>.

Estos han sido los distintos controles HTML que podemos encontrar en ASP .NET, en el siguiente capítulo veremos otro tipo de controles, los denominados controles Web.

Correspondencia entre controles HTML y etiquetas HTML

En este último apartado se ofrece una tabla (Tabla 6) que pretende ser un resumen de las equivalencias entre los controles HTML y las etiquetas HTML con las que se corresponden.

Control HTML	Etiqueta HTML correspondiente
HtmlAnchor	<a>
HtmlButton	<button>
HtmlSelect	<select>
HtmlTextArea	<textarea>
HtmlInputButton	<input type="button">, <input type="reset">, <input type="submit">
HtmlInputCheckBox	<input type="checkbox">
HtmlInputRadioButton	<input type="radio">
HtmlInputText	<input type="text">, <input type="password">
HtmlInputHidden	<input type="hidden">
HtmlInputImage	<input type="image">
HtmlInputFile	<input type="file">
HtmlForm	<form>
HtmlImage	<img>

HtmlTable	<table>
HtmlTableRow	<tr>
HtmlTableCell	<td>,<th>
HtmlGenericControl	Cualquier etiqueta HTML sin equivalencia, como pueden ser: , , <body>, etc.

Tabla 6

8

Web Forms: Controles Web intrínsecos

Introducción a los Controles Web

En este capítulo, tal como hacíamos en el anterior para los controles HTML, vamos a comentar la utilización de los distintos controles Web que ofrece ASP .NET.

Estos controles ASP .NET cumplen la misma funcionalidad que los anteriores, los controles HTML, en este caso muestran una sintaxis basada en XML, y no existe una correlación directa entre los controles y las etiquetas HTML que se generan, tal como si sucedía con los controles HTML.

Estos controles pertenecen al espacio con nombre System.Web.UI.WebControls, y a diferencia que el espacio con nombre dónde encontramos los controles HTML, el espacio con nombre que contiene los controles Web, contiene al resto de familias de controles ASP .NET, que seguiremos viendo a lo largo del texto, es decir, contiene un mayor número de controles.

En el Código fuente 77 se puede ver la sintaxis general que presentan los controles Web, como ya habíamos adelantado su sintaxis está basada en la sintaxis XML, y en el Código fuente 78 se muestra un ejemplo de utilización de un Web Form con varios controles Web de ASP .NET. Se trata de un formulario con una lista desplegable y un botón, al pulsar el botón se muestra el elemento seleccionado de la lista en una etiqueta, es el mismo ejemplo que se utilizaba en el capítulo anterior para presentar los controles HTML, pero ha sido rescrito con controles Web.

```
<asp:nombreControl id="identificador" runat="server"></asp:nombreControl>
```

Código fuente 77

```
<%@ Page language="c#"%>

<HTML>
<script language="C#" runat="server">
      void Pulsado(Object sender, EventArgs args){
            etiqueta.Text=lista.SelectedItem.Text;
      }
</script>
<body>
<form id="WebForm" method="post" runat="server">
<asp:ListBox id="lista" runat="server">
      <asp:ListItem selected>La Naranja Mecánica</asp:ListItem>
      <asp:ListItem>El Resplador</asp:ListItem>
      <asp:ListItem>Lolita</asp:ListItem>
      <asp:ListItem>2001</asp:ListItem>
</asp:Listbox>
<asp:Button Runat="server" ID="boton" OnClick="Pulsado" Text="seleccionar"/><br>
<asp:Label id="etiqueta" runat="server"/>
</form>
</body>
</HTML>
```

Código fuente 78

En la Figura 46 se puede observar un ejemplo de ejecución de esta página.

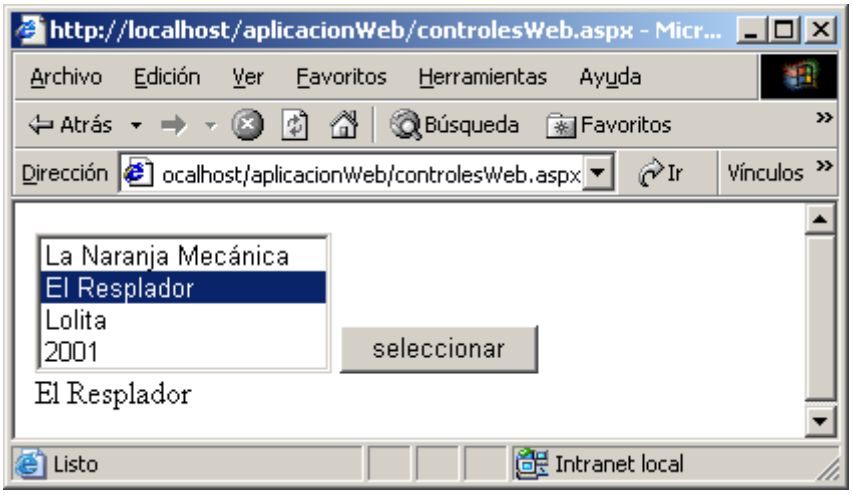


Figura 46

Si comparamos el código fuente de este ejemplo con el correspondiente de los controles HTML, vemos que los mayores cambios se pueden apreciar en la sintaxis de los controles del Web Form, y también en las propiedades de los controles.

Como se puede comprobar existen algunas diferencias a la hora de utilizar controles HTML y controles Web, además de las más evidentes relativas a la distinta sintaxis a la hora de declarar los controles. Algunos nombres de las propiedades cambian, el control HTML button posee la propiedad value, y sin embargo el control Web equivalente posee la propiedad text.

A la hora de indicar los métodos para el tratamiento de eventos de la pulsación de un botón del ratón en los controles HTML utilizamos la propiedad onserverclick, y en los controles Web utilizamos la propiedad onclick. En los controles HTML se debe distinguir si se trata de un evento de cliente o de

servidor, ya que podemos tratar ambos tipos, sin embargo desde los controles Web sólo tenemos la posibilidad de utilizar eventos de servidor, es decir, eventos que son tratados por rutinas en el servidor.

Los controles Web ofrecen un mayor nivel de abstracción que los controles HTML, y su modelo de objetos no refleja la sintaxis HTML necesariamente. Cuando la página se carga en el navegador, el control Web determina el tipo de navegador que ha realizado la petición, y de acuerdo con esta información genera el código HTML apropiado, podemos decir que en este aspecto se trata de controles inteligentes.

Los controles Web además de ofrecer los controles intrínsecos al igual que lo hacían los controles HTML, también incluyen los otros tres tipos de controles de servidor que podemos utilizar desde ASP .NET, es decir, los controles de lista, los controles ricos y los controles de validación. Es decir los controles Web engloban al resto de los controles de ASP .NET, por lo tanto son mucho mas completos que los controles HTML, ya que los controles HTML están limitados únicamente a los controles intrínsecos.

De esta forma tendremos controles Web intrínsecos (que son los equivalentes a los controles HTML), controles Web de lista, controles Web ricos y controles Web de validación. En este capítulo trataremos los controles Web intrínsecos, es decir, los que generarán etiquetas HTML que se corresponden con los elementos del lenguaje HTML.

Los beneficios que se desprenden de utilizar controles Web intrínsecos sobre los controles HTML vistos anteriormente son.

- Se ofrece una convención de nomenclatura para controles similares.
- Existen propiedades comunes para todos los controles , ofreciéndose un modelo de objetos más robusto.
- Se incluyen propiedades fuertemente tipadas específicas de los controles.
- Se genera código HTML específico para cada navegador Web.

En algunos casos nos puede interesar utilizar controles Web, y en otros controles HTML, los controles Web deberán ser usados en las siguientes situaciones:

- Preferimos un modelo de programación similar a Visual Basic.
- Estamos escribiendo Web Forms que deben ser mostrados por varios tipos de navegadores.
- Se necesita una funcionalidad específica, como puede ser un calendario o un rotador de anuncios, que se corresponden con los controles ricos, o bien necesitamos realizar algún tipo de validación o presentación de datos.

Y los controles HTML es preferible utilizarlos en las siguientes situaciones:

- Preferimos un modelo de objetos similar al lenguaje HTML.
- Estamos trabajando con páginas Web existentes y las queremos migrar a Web Forms.
- El control debe interactuar con script de cliente y de servidor, en lo que al tratamiento de eventos se refiere.

En el siguiente apartado nos vamos a ocupar de los controles Web que podemos encontrar dentro del grupo de controles intrínsecos.

Controles Web intrínsecos

Al igual que sucedía con los controles HTML, para permitir hacer referencia a los controles dentro del código fuente de la página, se debe utilizar el atributo id, de esta forma el objeto que se corresponde con el control de servidor se puede utilizar en el código de servidor para utilizar sus métodos o manipular sus propiedades.

La sintaxis de los controles Web ya la hemos comentado anteriormente, pero se debe señalar que podemos encontrar en algunos casos dos tipos de sintaxis, la primera de ellas se puede ver en el Código fuente 79, y se utiliza cuando el control Web no tiene cuerpo, y la segunda (Código fuente 80), cuando el control Web tiene cuerpo, como puede ser una lista (objeto ListBox) con elementos. Cuando el control Web no posee cuerpo podemos utilizar las dos de manera indistinta.

```
<asp:nombreControl id="identificador" runat="server"/>
```

Código fuente 79

```
<asp:nombreControl id="identificador" runat="server"></asp:nombreControl>
```

Código fuente 80

A continuación vamos a ir comentando las principales clases del espacio con nombre System.Web.UI.WebControls que se corresponden con los controles Web intrínsecos. Estas clases ofrecen al desarrollador el acceso en el servidor a las distintas etiquetas HTML que finalmente se enviarán al cliente Web como el resultado de la ejecución de la página ASP .NET correspondiente.

A algunas descripciones de las clases de los controles Web les acompaña un ejemplo de utilización, en estos ejemplos veremos como se utilizan los controles Web dentro de una página ASP .NET que contiene un Web Form y también veremos algunos ejemplos con tratamiento de eventos.

Button

Este control Web representa un botón, se corresponde con la etiqueta <input type=”submit”>, es decir, un botón que envía el contenido de un formulario al servidor.

Esta clase se corresponde con el control Web <asp:Button>, en este caso el nombre de las clases y el nombre de los controles Web coinciden exactamente, lo que ya no existe es una correspondencia directa con los elementos HTML que se generan como resultado de la ejecución de la página ASP .NET.

En el Código fuente 81 se ofrece un ejemplo de utilización de esta clase, en esta página al pulsar el botón, objeto de la clase Button, se muestra la hecha y hora actuales en un objeto de la clase Label. El tratamiento de eventos es prácticamente idéntico al de los controles HTML, la única diferencia es que se utiliza la propiedad OnClick de la clase Button.

```
<%@ Page language="c#"%>

<HTML>
<script language="C#" runat="server">
      void Pulsado(Object sender, EventArgs args){
            DateTime ahora;
            ahora = DateTime.Now;
            etiqueta.Text ="El momento actual es: " + ahora.ToString();
      }
</script>
<body>
<form id="WebForm" method="post" runat="server">
<asp:Button Runat="server" ID="boton" OnClick="Pulsado" Text="Seleccionar"/><br>
<asp:Label id="etiqueta" runat="server"/>
</form>
</body>
</HTML>
```

Código fuente 81

En la Figura 47 se puede ver un ejemplo de ejecución de esta página, y en el Código fuente 82 se puede ver el código HTML que se devuelve al navegador.

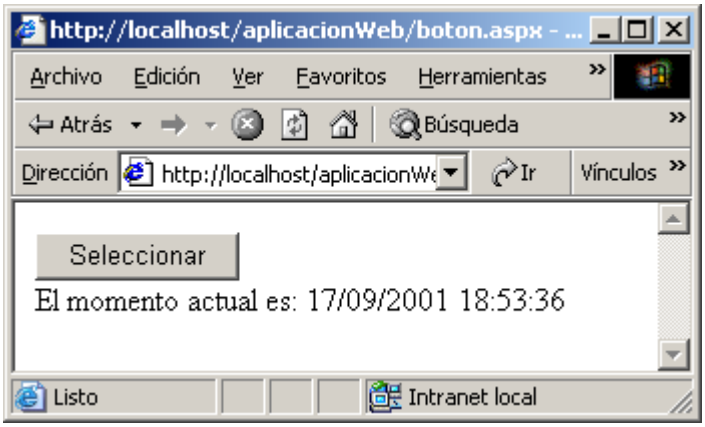


Figura 47

```
<HTML>

<body>
<form name="WebForm" method="post" action="boton.aspx" id="WebForm">
<input type="hidden" name="__VIEWSTATE"
value="dDw1ODM3NzA0MzM7dDw7bDxpPDI+Oz47bDx0PDtsPGk8Mz47PjtsPHQ8cDxwPGw8VGV4dDs+O2w8
RWwgbW9tZW50byBhY3R1YWwgZXM6IDE3LzA5LzIwMDEgMTk6MDA6NTI7Pj47Pjs7Pjs+Pjs+Pjs+" />

<input type="submit" name="boton" value="Seleccionar" id="boton" /><br>
<span id="etiqueta">El momento actual es: 17/09/2001 19:00:52</span>
</form>
</body>
</HTML>
```

Código fuente 82

CheckBox

Control Web que se corresponde con una etiqueta <input type="checkbox"> del lenguaje HTML. Este control Web tiene dos tratamientos distintos dentro de un Web Form, como un control de entrada de datos del usuario (valor verdadero o falso) más o bien como un botón de envío del formulario, este segundo comportamiento lo tendrá si especificamos el valor true en la propiedad AutoPostBack.

En el Código fuente 83 se muestra un ejemplo de utilización del control Web CheckBox, al pulsar el botón se comprueba si el objeto de la clase CheckBox se encuentra activado o no.

```
<html>
<head>
<script language="C#" runat="server">
       void BotonPulsado(Object Fuente, EventArgs e) {
       if (check.Checked == true) {
             etiqueta.Text = "El CheckBox está chequeado";
        }
        else {
           etiqueta.Text = "El CheckBox no está chequeado";
        }
       }

    </script>

</head>
<body>

<form runat="server" ID="formulario">
<asp:CheckBox id="check" Text="Elemento CheckBox" runat="server" /><br>
<asp:button text="Submit" OnClick="BotonPulsado" runat="server" ID="boton"/><br>
<asp:Label id="etiqueta" font-name="arial" font-size="10pt" runat="server"/>
</form>

</body>
</html>
```

Código fuente 83

En la Figura 48 se puede comprobar el resultado de la ejecución de este ejemplo.

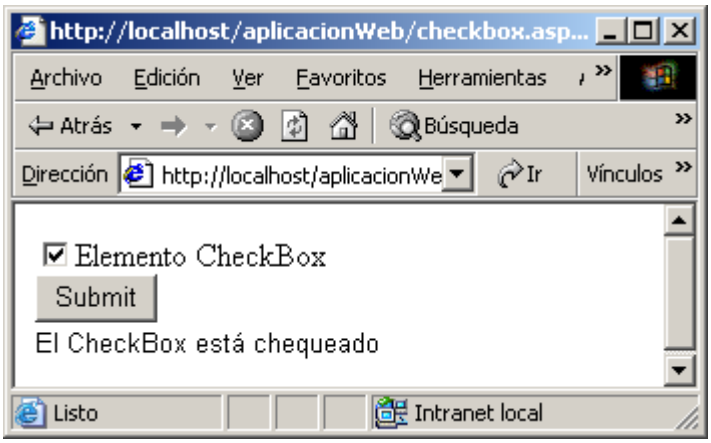


Figura 48

Para utilizar controles checkbox múltiples podemos hacer uso del control CheckBoxList que se comenta a continuación.

CheckBoxList

Este control permite utilizar una lista de selección múltiple de elementos checkbox. Este objeto posee una colección Items que contiene todos los objetos CheckBox. Se puede especificar a través de las propiedades RepeatDirection y RepeatLayout la distribución de los CheckBox en la página.

Por defecto la propiedad RepeatLayout tiene el valor RepeatLayout.Table, en este caso los CheckBox se muestran dentro de una tabla, también se puede asignar el valor RepeatLayout.Flow, este valor de la propiedad provocará que los CheckBox se distribuyan libremente en la pagina sin incluirse en ninguna tabla.

La propiedad RepeatDirection tiene por defecto el valor RepeatDirection.Vertical, es decir, los CheckBox se distribuirán de forma vertical, también le podemos asignar el valor RepeatDirection.Horizontal para conseguir que los controles se muestran de forma horizontal.

En el ejemplo del Código fuente 84 se muestra un control CheckBoxList que contiene una serie de controles CheckBox, y dentro del mismo Web Form se permite indicar la distribución de los elementos del CheckBoxList.

```
<html>
<head>
<script language="C#" runat="server">

void BotonPulsado(object fuente, EventArgs e) {
      String s = "Elementos seleccionados:<br>";
      for (int i=0; i < check.Items.Count; i++) {
            if ( check.Items[ i ].Selected ){
                  s = s + check.Items[i].Text;
           s = s + "<br>";
        }
    }
    etiqueta.Text = s;
}

void PulsadoDistribucion(Object fuente, EventArgs e) {
      if (distribucion.Checked == true) {
            check.RepeatLayout = RepeatLayout.Table;
    }
    else {
          check.RepeatLayout = RepeatLayout.Flow;
    }
}

void PulsadoDireccion(Object sender, EventArgs e) {
      if (direccion.Checked == true) {
            check.RepeatDirection = RepeatDirection.Horizontal;
    }
    else {
          check.RepeatDirection = RepeatDirection.Vertical;
    }
}

</script>
</head>
<body>
```

```
<form runat=server ID="WebForm">
<asp:CheckBoxList id="check" runat="server" font-name="Verdana" font-size="8pt">
      <asp:ListItem>Elemento 1</asp:ListItem>
    <asp:ListItem>Elemento 2</asp:ListItem>
    <asp:ListItem>Elemento 3</asp:ListItem>
    <asp:ListItem>Elemento 4</asp:ListItem>
    <asp:ListItem>Elemento 5</asp:ListItem>
    <asp:ListItem>Elemento 6</asp:ListItem>
</asp:CheckBoxList>
<hr>
<asp:CheckBox id="distribucion" OnCheckedChanged="PulsadoDistribucion"
Text="Distribución en tabla" AutoPostBack="true" runat="server" /><br>

<asp:CheckBox id="direccion" OnCheckedChanged="PulsadoDireccion" Text="Horizontal"
AutoPostBack="true" runat="server" /><br>

<asp:Button id="boton" Text="Enviar" onclick="BotonPulsado" runat="server"/><br>
<hr>
<asp:Label id="etiqueta" font-name="Verdana" font-size="8pt" runat="server"/>

</form>

</body>
</html>
```

Código fuente 84

El evento que se lanza al pulsar sobre un CheckBox es el evento CheckedChanged. En el ejemplo anterior también se puede preciar la forma en la que se recorre la colección Items de la clase CheckBoxList para consultar el valor de cada uno de los CheckBox.

Cada elemento del CheckBoxList se corresponde con un objeto de la clase ListItem, esta clase va a representar a cada uno de los elementos de una lista, en este caso se trata de definir los elementos de un objeto CheckBoxList, pero también se utiliza con otras clases como la clase DropDownList que veremos en el apartado siguiente. Un ejemplo de ejecución de esta página se puede observar en la Figura 49.

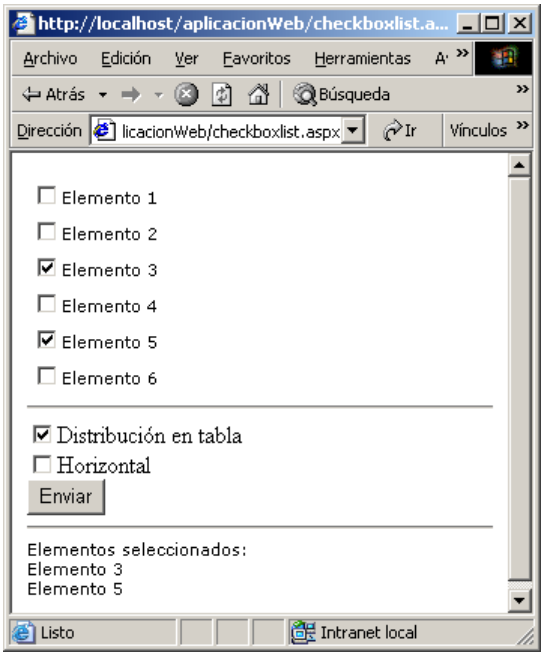


Figura 49

DropDownList

Este control Web representa una lista desplegable, y se corresponde con la etiqueta <select> de HTML. Al igual que el control Web CheclBox también posee la propiedad AutoPostBack, si establecemos a True esta propiedad al seleccionar un elemento de la lista se enviará el formulario al servidor, es decir, es equivalente a la pulsación de un botón de tipo submit.

Al igual que sucedía con el control Web CheckBoxList, cada uno de los elementos del control DropDownList se indica con un objeto de la clase ListItem.

El Código fuente 85 es un sencillo ejemplo de utilización de este control Web, y la Figura 50 es el resultado de su ejecución.

```
<html>
<head>
<script language="C#" runat="server">
void SeleccionLista(Object fuente, EventArgs e) {
     imagen.ImageUrl=lista.SelectedItem.Value;
     imagen.AlternateText=lista.SelectedItem.Text;
}
</script>
</head>
<body>
<form runat="server" ID="WebForm">
<asp:Image ID="imagen" ImageUrl="" AlternateText="Sin imagen" runat="server" />
<asp:DropDownList AutoPostBack="True" OnSelectedIndexChanged="SeleccionLista"
id="lista" runat="server">
    <asp:ListItem Value="bicho.png">Bicho</asp:ListItem>
    <asp:ListItem Value="ImagenDos.gif">Imagen dos</asp:ListItem>
    <asp:ListItem Value="prueba.gif">Imagen de prueba</asp:ListItem>
</asp:DropDownList>
</form>
</body>
</html>
```

Código fuente 85

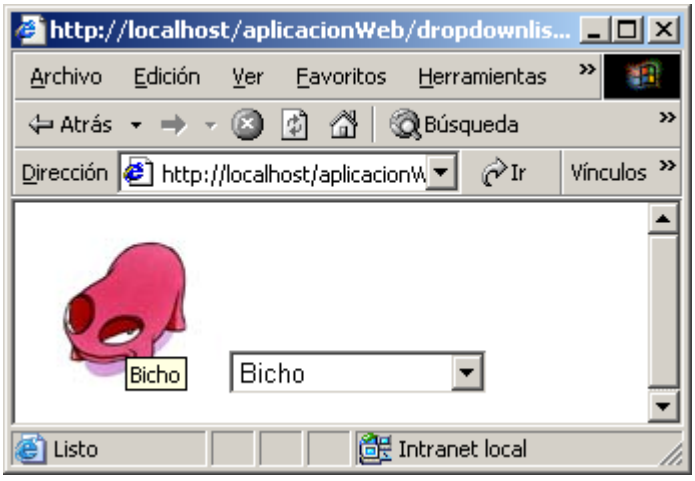


Figura 50

Se trata de un Web Form que posee un objeto de la clase DropDownList con nombres de ficheros de imagen, y un objeto de la clase Image (que veremos mas adelante en su apartado correspondiente) que mostrará la imagen que ha sido seleccionada en la lista desplegable.

El Código fuente 86 se corresponde con el código HTML que se genera al ejecutar la página ASP .NET del ejemplo.

```
<html>
<head>

</head>
<body>
<form name="WebForm" method="post" action="dropdownlist.aspx" id="WebForm">
<input type="hidden" name="__VIEWSTATE"
value="dDwxMTgyNDMwMzIyO3Q8O2w8aTwyPjs+O2w8dDw7bDxpPDE+O2k8Mz47PjtsPHQ8cDxwPGw8SW1h
Z2VVcmw7QWx0ZXJuYXRlVGV4dDs+O2w8YmljaG8ucG5nO0JpY2hvOz4+Oz47Oz47dDx0PDs7bDxpPDE+Oz4
+Ozs+Oz4+Oz4=" />

<img id="imagen" src="/aplicacionWeb/bicho.png" alt="Bicho" border="0" />
<select name="lista" id="lista" onchange="__doPostBack('lista','')"
language="javascript">
      <option value="">Sin imagen</option>
      <option selected="selected" value="bicho.png">Bicho</option>
      <option value="ImagenDos.gif">Imagen dos</option>
      <option value="prueba.gif">Imagen de prueba</option>

</select>

<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
      function __doPostBack(eventTarget, eventArgument) {
            var theform = document.WebForm;
            theform.__EVENTTARGET.value = eventTarget;
            theform.__EVENTARGUMENT.value = eventArgument;
            theform.submit();
      }
// -->
</script>
</form>
</body>
</html>
```

Código fuente 86

El evento SelectedIndexChanged se produce cada vez que se modifica la selección de un elemento de la lista y además la propiedad AutoPostBack tiene el valor True.

HyperLink

Este otro control Web representa un enlace en la página, por lo tanto generará en el cliente una etiqueta <a>.

Image

Control Web que representa una imagen dentro de una página ASP .NET, generará como resultado de su ejecución una etiqueta <img> de HTML. Para indicar la imagen que deseamos mostrar se utilizará la propiedad ImageURL, un ejemplo de utilización de este control ya lo hemos visto en un apartado anterior.

ImageButton

Este control es muy similar al anterior, pero además de mostrar una imagen posee la característica adicional de funcionar como un botón de tipo submit, es decir, al pulsar el botón se envían los contenidos del formulario al servidor. Este control generan en el cliente la etiqueta HTML <input type=”image”>.

El Código fuente 87 muestra la utilización de este control Web, al pulsar sobre la imagen se enviará el formulario al servidor, es decir, se ha ce un submit y se muestra el contenido de las cajas de texto ,que son objetos de la clase TextBox, (que veremos más adelante) en un objeto de la clase Label.

```
<html>
<head>
<script language="C#" runat="server">
void Page_Load(Object fuente,EventArgs evento){
       if(Page.IsPostBack){
              texto.Text=cajaUno.Text+"/"+cajaDos.Text;
       }
}
</script>
</head>
<body>
<form runat="server" ID="WebForm">
<asp:TextBox ID="cajaUno" runat="server"/>
<asp:TextBox ID="cajaDos" runat="server"/>
<asp:ImageButton ID="imagen" ImageUrl="boton.png" AlternateText="pulsa"
runat="server" />
<asp:Label ID="texto" runat="server"/>
</form>
</body>
</html>
```

Código fuente 87

El proceso de mostrar el contenido de las cajas de texto se lleva a cabo en el método Page_Load, es decir, en el método que trata el evento de carga de la página. En este método se consulta la propiedad IsPostBack del objeto Page, que representa a la página actual. Más adelante veremos en profundidad el interesante objeto Page.

El Código fuente 88 es el código HTML que se genera como resultado de la ejecución del ejemplo, y la Figura 51 es un ejemplo de ejecución de la página.

```
<html>
<head>

</head>
```

```
<body>
<form name="WebForm" method="post" action="imagebutton.aspx" id="WebForm">
<input type="hidden" name="__VIEWSTATE"
value="dDw1ODY5NTk1Njg7dDw7bDxpPDI+Oz47bDx0PDtsPGk8Nz47PjtsPHQ8cDxwPGw8VGV4dDs+O2w8
Q2FqYSAxL0NhamEgMjs+Pjs+Ozs+Oz4+Oz4+O2w8aW1hZ2VuOz4+" />

<input name="cajaUno" type="text" value="Caja 1" id="cajaUno" />
<input name="cajaDos" type="text" value="Caja 2" id="cajaDos" />
<input type="image" name="imagen" id="imagen" src="/aplicacionWeb/boton.png"
alt="pulsa" border="0" />
<span id="texto">Caja 1/Caja 2</span>
</form>
</body>
</html>
```

Código fuente 88

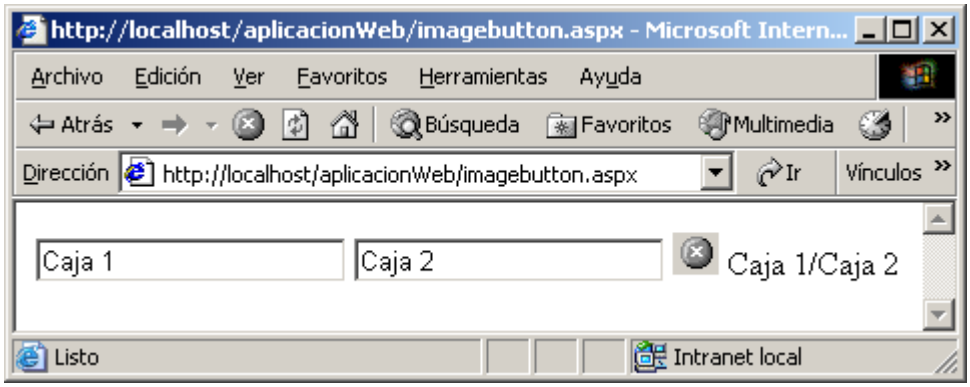


Figura 51

Label

Este control ya lo hemos utilizado en ejemplos anteriores y representa una etiqueta de texto, que generará una etiqueta de HTML.

LinkButton

Este control va a representar un botón que presenta un estilo similar a los enlaces, el control LinkButton presenta una apariencia similar a un control Hyperlink, pero sin embargo ofrece la misma funcionalidad que un control Button, es decir, presenta un texto a modo de enlace que al pulsarlo enviará el formulario en el que se encuentre al servidor.

El Código fuente 89 se trata de un Web Form que posee un control LinkButton, al pulsar sobre el objeto LinkButton se envía el formulario al servidor y se muestra en un control Label los valores de los controles TextBox. Este ejemplo es muy similar al comentado con el control Web ImageButton.

```
<html>
<head>
<script language="C#" runat="server">
void Page_Load(Object fuente,EventArgs evento){
      if(Page.IsPostBack){
            texto.Text=cajaUno.Text+"/"+cajaDos.Text;
      }
```

```
}
</script>
</head>
<body>
<form runat="server" ID="WebForm">
<asp:TextBox ID="cajaUno" runat="server"/>
<asp:TextBox ID="cajaDos" runat="server"/>
<asp:LinkButton ID="imagen" Text="pulsa" runat="server" />
<asp:Label ID="texto" runat="server"/>
</form>
</body>
</html>
```

Código fuente 89

En la Figura 52 se puede ver el resultado de la ejecución del ejemplo del control Web LinkButton.

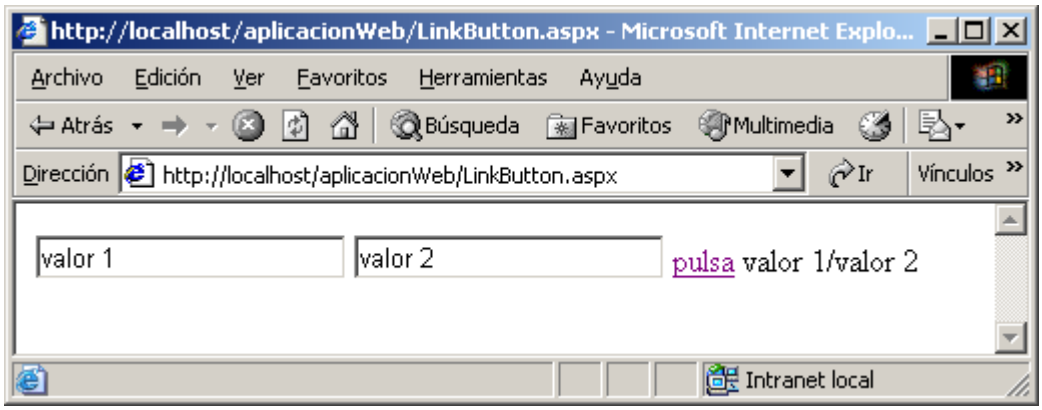


Figura 52

El control LinkButton generará la etiqueta <a> en el código HTML, precisamente el Código fuente 90 es el código HTML que se genera al ejecutar el ejemplo.

```
<html>
<head>

</head>
<body>
<form name="WebForm" method="post" action="LinkButton.aspx" id="WebForm">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTQ1NDA3NzI3NTt0PDtsPGk8Mj47PjtsPHQ8O2w8aTw3Pjs+O2w8dDxwPHA8bDxUZXh0Oz47
bDx2YWxvciAxL3ZhbG9yIDI7Pj47Pjs7Pjs+Pjs+Pjs+" />

<input name="cajaUno" type="text" value="valor 1" id="cajaUno" />
<input name="cajaDos" type="text" value="valor 2" id="cajaDos" />
<a id="imagen" href="javascript:__doPostBack('imagen','')">pulsa</a>
<span id="texto">valor 1/valor 2</span>

<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
       function __doPostBack(eventTarget, eventArgument) {
              var theform = document.WebForm;
              theform.__EVENTTARGET.value = eventTarget;
              theform.__EVENTARGUMENT.value = eventArgument;
              theform.submit();
```

```
      }
// -->
</script>
</form>
</body>
</html>
```

Código fuente 90

ListBox

Este nuevo control Web que pasamos a comentar representa una lista de selección sencilla o múltiple, es similar al control DropDownList, pero en este caso se muestran varios elementos de la lista y se permite la selección múltiple. Este control Web también se corresponde con una etiqueta <select> en el código HTML generado, de la misma forma que sucedía con el control DropDownList, pero en este caso con distintas propiedades.

En la propiedad Rows del control ListBox indicamos el número de filas visibles que va a mostrar el control, y en la propiedad SelectionMode indicamos si se permite la selección múltiple, mediante el valor Multiple, o bien la selección simple mediante el valor Single, este es el valor por defecto.

El Código fuente 91 muestra una lista de selección múltiple y un control Button, al pulsar sobre el botón se muestra en un control Label los valores de los elementos que se han seleccionado de la lista.

```
<html>
<head>
<script language="C#" runat="server">
void Page_Load(Object fuente,EventArgs evento){
 if(Page.IsPostBack){
   texto.Text="Elementos seleccionados: <br>";
   for (int i=0; i<lista.Items.Count; i++) {
     if (lista.Items[i].Selected)
       texto.Text=texto.Text + lista.Items[i].Value+"-"+lista.Items[i].Text+"<br>";
   }
 }
}
</script>
</head>
<body>
<form runat="server" ID="WebForm">
<asp:ListBox id="lista"  width="100" rows="3" runat="server"
SelectionMode="Multiple">
      <asp:ListItem Value="1">Uno</asp:ListItem>
      <asp:ListItem Value="2">Dos</asp:ListItem>
    <asp:ListItem Value="3">Tres</asp:ListItem>
    <asp:ListItem Value="4">Cuatro</asp:ListItem>
    <asp:ListItem Value="5">Cinco</asp:ListItem>
    <asp:ListItem Value="6">Seis</asp:ListItem>
</asp:ListBox>
<asp:Button id="boton" runat="server" text="pulsa"/><br>
<asp:Label id="texto" Runat="server"/>
</form>
</body>
</html>
```

Código fuente 91

Como se puede comprobar en el código anterior el control ListBox también se utiliza en combinación con controles ListItem, que representan cada uno de los elementos de la lista. La propiedad Items contiene todos los elementos del control ListBox, y podemos utilizar la propiedad Selected de cada elemento para comprobar si el elemento de la lista ha sido seleccionado o no, tal como se muestra en el código anterior.

Un ejemplo de ejecución de la página ASP .NET anterior puede ser el mostrado en la Figura 53.

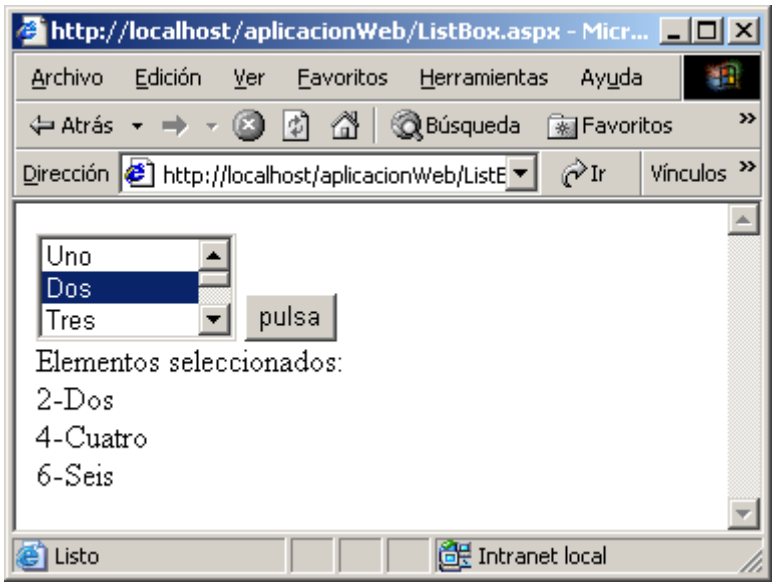


Figura 53

A esta ejecución de la página ASP .NET le corresponde el código HTML del Código fuente 92.

```
<html>
<head>
</head>
<body>
<form name="WebForm" method="post" action="ListBox.aspx" id="WebForm">
<input type="hidden" name="__VIEWSTATE"
value="dDwxNTI4OTM1MjcxO3Q8O2w8aTwyPjs+O2w8dDw7bDxpPDU+Oz47bDx0PHA8cDxsPFRleHQ7Pjts
PEVsZW1lbnRvcyBzZWxlY2Npb25hZG9zOiBcPGJyXD4yLURvc1w8YnJcPjQtQ3VhdHJvXDxiclw+Ni1TZWl
zXDxiclw+Oz4+Oz47Oz47Pj47Pj47bDxsaXN0YTs+Pg==" />
<select name="lista" id="lista" size="3" multiple="multiple" style="width:100px;">
      <option value="1">Uno</option>
      <option selected="selected" value="2">Dos</option>
      <option value="3">Tres</option>
      <option selected="selected" value="4">Cuatro</option>
      <option value="5">Cinco</option>
      <option selected="selected" value="6">Seis</option>
</select>
<input type="submit" name="boton" value="pulsa" id="boton" /><br>
<span id="texto">Elementos seleccionados: <br>2-Dos<br>4-Cuatro<br>6-
Seis<br></span>
</form>
</body>
</html>
```

Código fuente 92

Panel

Este otro control Web intrínseco se utiliza para agrupar controles y realizar la función de contenedor de los mismos. El control Panel posee una propiedad llamada Controls que es una colección que contiene todos los controles incluidos dentro del objeto Panel.

En el Código fuente 93 se muestra un ejemplo sencillo de utilización de este control, en este caso se utiliza en combinación con un control CheckBox, mediante el valor del mismo indicaremos si se va a mostrar el contenido del objeto Panel o no. El objeto Panel contiene tres objetos TextBox.

```
<html>
<head>
<script language="C#" runat="server">
void Pulsado(Object Sender, EventArgs e) {
   if (check.Checked) {
            contenedor.Visible=true;
   }else {
       contenedor.Visible=false;
   }
}
</script>

</head>
<body>

<form runat="server" ID="Formulario">
<asp:Panel ID="contenedor" runat="server" BackColor=#ff9900 Visible="False">
<asp:TextBox ID="cajaTextoUno" runat="server"/>
<asp:TextBox ID="cajaTextoDos" runat="server"/>
</asp:Panel>
<asp:CheckBox id="check" Text="Mostrar contenido panel" runat="server"
OnCheckedChanged="Pulsado" AutoPostBack="True"/><br>

</form>

</body>
</html>
```

Código fuente 93

Como se puede comprobar se ha establecido a True el valor de la propiedad AutoPostBack del control CheckBox, para que al cambiar el valor de este control se envíe al servidor el formulario Web. En la Figura 54 se muestra un ejemplo de ejecución de esta página.

El Código fuente 94 es el código HTML que genera este ejemplo, de este código se desprende que el control Panel se corresponde con una etiqueta de HTML.

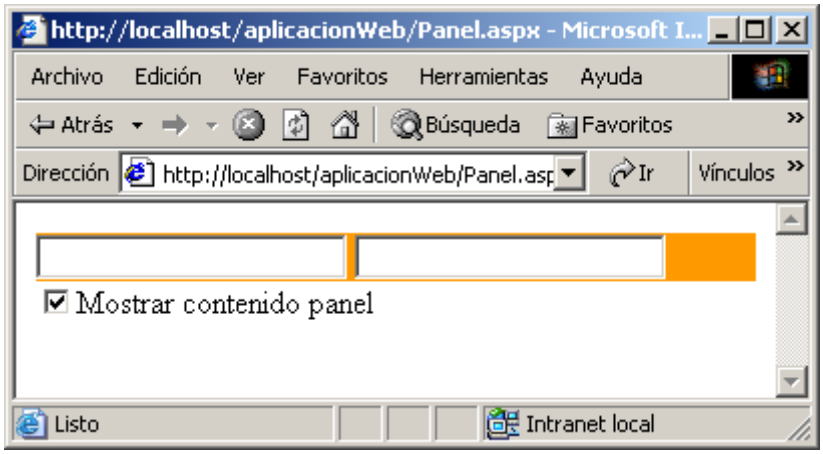


Figura 54

```
<html>
<head>


</head>
<body>

<form name="Formulario" method="post" action="Panel.aspx" id="Formulario">
<input type="hidden" name="__VIEWSTATE"
value="dDwzODMxODUyNDY7dDw7bDxpPDI+Oz47bDx0PDtsPGk8MT47aTwzPjs+O2w8dDxwPHA8bDxWaXNp
YmxlOz47bDxvPHQ+Oz4+Oz47Oz47dDxwPHA8bDxDaGVja2VkOz47bDxvPHQ+Oz4+Oz47Oz47Pj47Pj47bDx
jaGVjazs+Pg==" />

<div id="contenedor" style="background-color:#FF9900;">

<input name="cajaTextoUno" type="text" id="cajaTextoUno" />
<input name="cajaTextoDos" type="text" id="cajaTextoDos" />

</div>
<input id="check" type="checkbox" name="check" checked="checked"
onclick="__doPostBack('check','')" language="javascript" /><label
for="check">Mostrar contenido panel</label><br>

<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
	function __doPostBack(eventTarget, eventArgument) {
		var theform = document.Formulario;
		theform.__EVENTTARGET.value = eventTarget;
		theform.__EVENTARGUMENT.value = eventArgument;
		theform.submit();
	}
// -->
</script>
</form>

</body>
</html>
```

Código fuente 94

PlaceHolder

Control Web que realiza también la función de un contenedor de controles Web, pero en este caso no genera ningún código HTML, se utiliza para añadir controles Web de forma dinámica en la página ASP .NET en un punto determinado, para ello se utiliza su propiedad Controls.

El Código fuente 95 muestra un objeto PlaceHolder al que se le añaden tres objetos TextBox en el evento PageLoad.

```
<html>
<head>
<script language="C#" runat="server">
void Page_Load(Object Sender, EventArgs e) {
      TextBox textoUno =new TextBox();
      contenedor.Controls.Add(textoUno);
      TextBox textoDos =new TextBox();
      contenedor.Controls.Add(textoDos);
      TextBox textoTres =new TextBox();
      contenedor.Controls.Add(textoTres);
}
</script>

</head>
<body>

<asp:PlaceHolder ID="contenedor" runat="server"/>
</body>
</html>
```

Código fuente 95

RadioButton

Este control representa un botón de opción que se corresponde con el elemento <input type=”radio”> de HTML. Muestra la misma funcionalidad, es decir, permite seleccionar una opción dentro de un mismo grupo de opciones. Las opciones se agrupan mediante la propiedad GroupName.

En el ejemplo (Código fuente 96) se agrupan tres controles RadioButton y al cambiar el valor de uno de ellos se muestra en un objeto Label el texto de la opción seleccionada. Como se pude comprobar se hace uso de la propiedad AutoPostBack y del evento CheckedChanged, al igual que se hacia en el control CheckBox para indicar que se ha modificado la selección.

```
<html>
<head>
<script language="C#" runat="server">
void Pulsado(Object Sender, EventArgs e) {
      if (opc1.Checked) {
            seleccion.Text="Se ha elegido la "+opc1.Text;
      }else if (opc2.Checked) {
            seleccion.Text="Se ha elegido la "+opc2.Text;
      }else if (opc3.Checked) {
            seleccion.Text="Se ha elegido la "+opc3.Text;
      }
}
</script>
```

```
</head>
<body>
<form runat="server" ID="Formulario">
<asp:RadioButton id="opc1" Text="Selección uno" runat="server" GroupName="grupo"
OnCheckedChanged="Pulsado" AutoPostBack="True"/><br>
<asp:RadioButton id="opc2" Text="Selección dos" runat="server" GroupName="grupo"
OnCheckedChanged="Pulsado" AutoPostBack="True"/><br>
<asp:RadioButton id="opc3" Text="Selección tres" runat="server" GroupName="grupo"
OnCheckedChanged="Pulsado" AutoPostBack="True"/><br>
<asp:Label ID="seleccion" Runat="server"></asp:Label>
</form>
</body>
</html>
```

Código fuente 96

Un ejemplo de ejecución del código anterior se puede ver en la Figura 55.

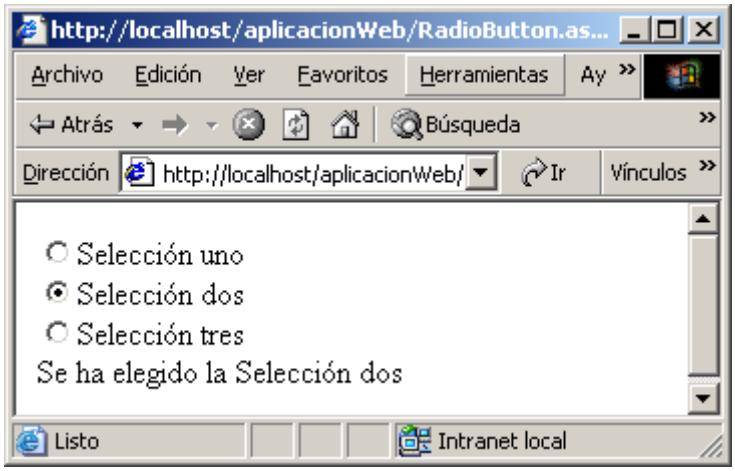


Figura 55

Y el Código fuente 97 es el código HTML que se genera.

```
<html>
<head>

</head>
<body>
<form name="Formulario" method="post" action="RadioButton.aspx" id="Formulario">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTczODk2MDM2ODt0PDtsPGk8Mj47PjtsPHQ8O2w8aTwxPjtpPDM+O2k8NT47aTw3Pjs+O2w8
dDxwPHA8bDxDaGVja2VkOz47bDxvPGY+Oz4+Oz47Oz47dDxwPHA8bDxDaGVja2VkOz47bDxvPHQ+Oz4+Oz4
7Oz47dDxwPHA8bDxDaGVja2VkOz47bDxvPGY+Oz4+Oz47Oz47dDxwPHA8bDxUZXh0Oz47bDxTZSBoYSBlbG
VnaWRvIGxhIFNlbGVjY2nDs24gZG9zOz4+Oz47Oz47Pj47Pj47bDxvcGMxO29wYzE7b3BjMjtvcGMzO29wY
zM7Pj4=" />
<input id="opc1" type="radio" name="grupo" value="opc1"
onclick="__doPostBack('opc1','')" language="javascript" /><label
for="opc1">Selección uno</label><br>
<input id="opc2" type="radio" name="grupo" value="opc2" checked="checked"
onclick="__doPostBack('opc2','')" language="javascript" /><label
for="opc2">Selección dos</label><br>
<input id="opc3" type="radio" name="grupo" value="opc3"
onclick="__doPostBack('opc3','')" language="javascript" /><label
for="opc3">Selección tres</label><br>
<span id="seleccion">Se ha elegido la Selección dos</span>
```

```
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
      function __doPostBack(eventTarget, eventArgument) {
            var theform = document.Formulario;
            theform.__EVENTTARGET.value = eventTarget;
            theform.__EVENTARGUMENT.value = eventArgument;
            theform.submit();
      }
// -->
</script>
</form>
</body>
</html>
```

Código fuente 97

RadioButtonList

Este control permite utilizar una lista de selección múltiple de controles RadioButton. Este objeto posee una colección Items que contiene todos los objetos RadioButton. Se puede especificar a través de las propiedades RepeatDirection y RepeatLayout la distribución de los RadioButton en la página, como se puede apreciar es muy similar al control CheckBoxList, pero en este caso se utilizan controles RadioButton.

Para mostrar un ejemplo de utilización de este control Web podemos rescribir la página ASP .NET que se utilizaba en el ejemplo del control CheckBoxList. El Código fuente 98 sería el nuevo código, en este caso se ha utilizado también dos controles CheckBox.

```
<html>
<head>
<script language="C#" runat="server">
void PulsadoDistribucion(Object fuente, EventArgs e) {
      if (distribucion.Checked == true) {
            check.RepeatLayout = RepeatLayout.Table;
    }
    else {
          check.RepeatLayout = RepeatLayout.Flow;
    }
}

void PulsadoDireccion(Object sender, EventArgs e) {
      if (direccion.Checked == true) {
            check.RepeatDirection = RepeatDirection.Horizontal;
    }
    else {
          check.RepeatDirection = RepeatDirection.Vertical;
    }
}
</script>
</head>
<body>
<form runat=server ID="WebForm">
<asp:RadioButtonList id="check" runat="server" font-name="Verdana" font-size="8pt">
      <asp:ListItem>Elemento 1</asp:ListItem>
    <asp:ListItem>Elemento 2</asp:ListItem>
    <asp:ListItem>Elemento 3</asp:ListItem>
    <asp:ListItem>Elemento 4</asp:ListItem>
    <asp:ListItem>Elemento 5</asp:ListItem>
```

```
    <asp:ListItem>Elemento 6</asp:ListItem>
</asp:RadioButtonList>
<hr>
<asp:CheckBox id="distribucion" OnCheckedChanged="PulsadoDistribucion"
Text="Distribución en tabla" AutoPostBack="true" runat="server" /><br>
<asp:CheckBox id="direccion" OnCheckedChanged="PulsadoDireccion" Text="Horizontal"
AutoPostBack="true" runat="server" /><br>
</form>

</body>
</html>
```

Código fuente 98

En la Figura 56 se puede ver un ejemplo de ejecución del código anterior.

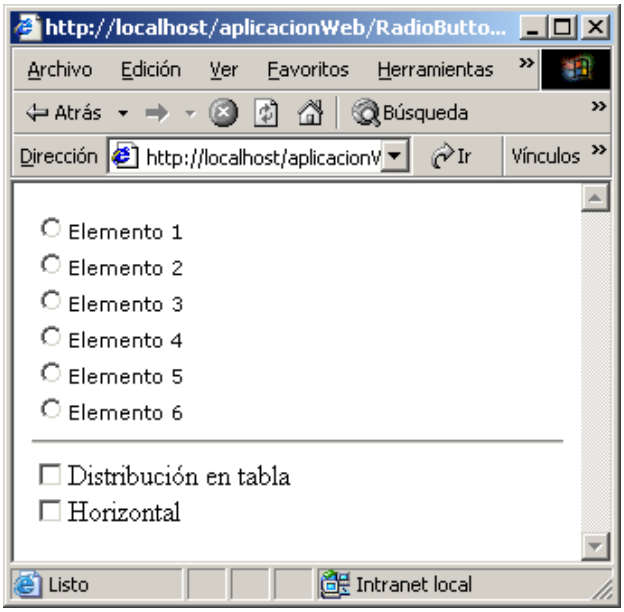


Figura 56

Table, TableRow y TableCell

Estos controles Web se encuentran muy relacionados entre sí, mediante la utilización de todos ellos podremos generar tablas en HTML.

El control Web Table se corresponde con una tabla del lenguaje HTML, es decir, permite generar tablas del lenguaje HTML. Esta clase posee una colección llamada Rows, que contiene objetos de la clase TableRow.

En el Código fuente 99 se muestra un ejemplo de utilización de este control Web, se trata de un formulario Web que permite indicar el número de filas y de columnas que deseamos que tenga una tabla.

```
<html>
<script language="C#" runat="server">
      void Page_Load(Object sender, EventArgs e) {
```

```
            if (Page.IsPostBack){
                  int numeroFilas = int.Parse(filas.SelectedItem.Value);
                  int numeroColumnas = int.Parse(columnas.SelectedItem.Value);
                  for (int j=1; j<=numeroFilas; j++) {
                        TableRow fila = new TableRow();
                        for (int i=1; i<=numeroColumnas; i++) {
                              TableCell columna = new TableCell();
                              columna.Text="["+j+","+i+"]";
                              fila.Cells.Add(columna);
                        }
                        tabla.Rows.Add(fila);
                  }
            }
      }
</script>
<body>
<form runat="server" ID="formulario">

<asp:table GridLines="Both" id="tabla" CellPadding=5 CellSpacing="0"
runat="server"/>
<br>
Filas:
<asp:DropDownList id="filas" runat="server">
  <asp:ListItem Value="1">1</asp:ListItem>
  <asp:ListItem Value="2">2</asp:ListItem>
  <asp:ListItem Value="3">3</asp:ListItem>
  <asp:ListItem Value="4">4</asp:ListItem>
  <asp:ListItem Value="5">5</asp:ListItem>
</asp:DropDownList>
Columnas:
<asp:DropDownList id="columnas" runat="server">
  <asp:ListItem Value="1">1</asp:ListItem>
  <asp:ListItem Value="2">2</asp:ListItem>
  <asp:ListItem Value="3">3</asp:ListItem>
  <asp:ListItem Value="4">4</asp:ListItem>
  <asp:ListItem Value="5">5</asp:ListItem>
</asp:DropDownList>
<asp:Button text="Generar tabla" runat="server" ID="boton"/>
</form>
</body>
</html>
```

Código fuente 99

Como se puede comprobar se utilizan objetos de la clase TableRow y TableCell para crear las filas y columnas de la tabla. Para añadir una columna se debe añadir un objeto TableCell a la colección Cells del objeto TableRow correspondiente, y para añadir una fila se añade un objeto TableRow a la colección Rows del objeto de la clase Table.

En este ejemplo se ha utilizado la propiedad GridLines del control Table para indicar que se desean mostrar todas las líneas de la tabla, tanto las horizontales como las verticales, para ello se le ha asignado el valor Both. En la Figura 57 se puede ver un ejemplo de ejecución de esta página.

Y el Código fuente 100 es el código HTML que se genera al ejecutar la página ASP .NET.

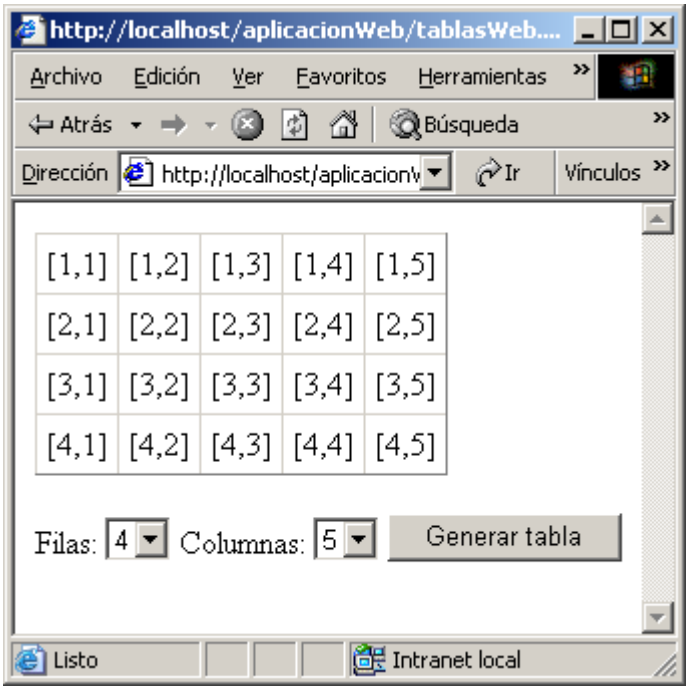


Figura 57

```
<html>
<body>
<form name="formulario" method="post" action="tablasWeb.aspx" id="formulario">
<input type="hidden" name="__VIEWSTATE" value="dDw5ODQ1ODY2OTc7Oz4=" />
<table id="tabla" cellspacing="0" cellpadding="5" rules="all" border="1"
style="border-collapse:collapse;">
	<tr>
		<td>
			[1,1]
		</td><td>
			[1,2]
		</td><td>
			[1,3]
		</td><td>
			[1,4]
		</td><td>
			[1,5]
		</td>
	</tr><tr>
		<td>
			[2,1]
		</td><td>
			[2,2]
		</td><td>
			[2,3]
		</td><td>
			[2,4]
		</td><td>
			[2,5]
		</td>
	</tr><tr>
		<td>
			[3,1]
		</td><td>
			[3,2]
		</td><td>
			[3,3]
```

```
            </td><td>
                  [3,4]
            </td><td>
                  [3,5]
            </td>
      </tr><tr>
            <td>
                  [4,1]
            </td><td>
                  [4,2]
            </td><td>
                  [4,3]
            </td><td>
                  [4,4]
            </td><td>
                  [4,5]
            </td>
      </tr>
</table>
<br>
Filas:
<select name="filas" id="filas">
      <option value="1">1</option>
      <option value="2">2</option>
      <option value="3">3</option>
      <option selected="selected" value="4">4</option>
      <option value="5">5</option>

</select>
Columnas:
<select name="columnas" id="columnas">
      <option value="1">1</option>
      <option value="2">2</option>
      <option value="3">3</option>
      <option value="4">4</option>
      <option selected="selected" value="5">5</option>

</select>
<input type="submit" name="boton" value="Generar tabla" id="boton" />
</form>
</body>
</html>
```

Código fuente 100

TextBox

Este control Web representa una caja de texto, y ya ha sido utilizado en algunos ejemplos dentro de este capítulo.

Este ha sido el último de los controles Web intrínsecos, en el siguiente capítulo veremos otros dos grupos de controles Web, los controles Web ricos (rich controls) y los de validación.

A continuación vamos a comentar una característica que resulta común para todos los controles Web y que puede resultar muy útil a la hora de mejorar y personalizar el aspecto de dichos controles, se trata de aplicar estilos a los controles.

Aplicando estilos a los controles Web

La especificación de hojas de estilo en cascada (Cascading Style Sheets, CSS) nos permiten definir la apariencia y aspecto de los elementos de una página HTML, entendiendo por elementos las distintas etiquetas de HTML.

CSS (Cascade Style Sheets, hojas de estilo en cascada) es una recomendación basada en el estándar de Internet del W3C, que describe el conjunto de símbolos, reglas y asociaciones existentes entre los elementos del lenguaje que forman un documento HTML y su apariencia gráfica en pantalla: color, fondo, estilo de fuente, posición en el espacio...etc.

CSS establece los elementos necesarios para independizar el contenido de los documentos HTML de la forma con la que se presentan en pantalla en función del navegador empleado. CSS extiende las normas de formato del documento, sin afectar a la estructura del mismo. De esta manera, un error en un formato, un navegador que no soporta colores...o cualquier otro error relacionado con el formato del documento no invalida el mismo. CSS además también aporta elementos dinámicos de formato, pues es posible capturar el estilo de los elementos cuando se produce una reacción sobre ellos. De esta manera es posible evitar la escritura de código JavaScript para conseguir efectos tan vistosos como el subrayado automático de hipervínculos, botones gráficos, fondos de color en controles input html... etc.

En este apartado no vamos a entrar a comentar en más profundidad el uso de las hojas de estilo ni su funcionalidad, si algún lector desea más información sobre el tema puede acudir al texto "HTML Dinámico, modelos de Objetos y JavaScript" realizado por mis compañeros Marino Posadas y José Luis Hevia.

Los controles Web ofrecen un completo soporte para las hojas de estilo, es decir, podemos aplicar los estilos de forma similar a como lo hacemos en HTML para personalizar el aspecto de nuestros controles.

Existen varias formas de aplicar estilos a los controles Web. A continuación vamos a comentar cada una de las posibilidades existentes.

Una de las formas de utilizar estilos con los controles Web es a través de una serie de propiedades que presentan los propios controles Web. Estas propiedades se encuentran fuertemente tipadas y representan los estilos más frecuentes que se pueden aplicar al control, color, tamaño de la letra, color de fondo, dimensiones, etc. La ventaja de aplicar los estilos de esta manera es que en tiempo de compilación se comprueba que las propiedades y sus valores sean los correctos.

En el Código fuente 101 se puede ver un control Web Label sobre el que se aplican una serie de estilos utilizando las propiedades que ofrece el control, como se puede ver el grado de personalización del aspecto de los controles Web es muy alto. En la Figura 58 se puede ver el aspecto que se le ha asignado al control Web.

```
<%@ Page language="c#"%>
<html>
<body>
<asp:Label  runat="server" id="etiqueta" BackColor="Yellow" BorderColor="Red"
BorderStyle="Dotted"  ForeColor="Green" Font-Size="12" Font-Bold="True" Font-
Italic="True" Font-Name="Courier">Aplicando estilos sobre controles Web</asp:Label>
</body>
</html>
```

Código fuente 101

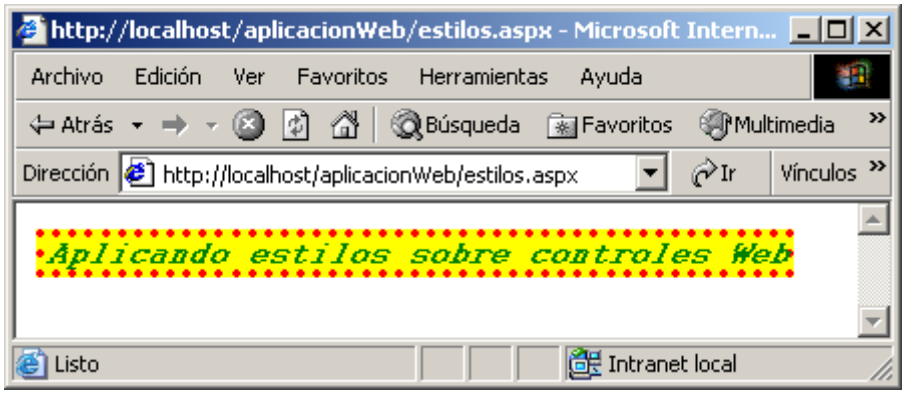


Figura 58

Si observamos el siguiente código (Código fuente 102) podemos ver el código HTML que se genera al ejecutar la página ASP .NET.

```
<html>
<body>
<span id="etiqueta" style="color:Green;background-color:Yellow;border-
color:Red;border-style:Dotted;font-family:Courier;font-size:12pt;font-
weight:bold;font-style:italic;">Aplicando estilos sobre controles Web</span>
</body>
</html>
```

Código fuente 102

Otra forma de aplicar estilos a un control Web es mediante la propiedad CssClass, a este propiedad le asignaremos el nombre de clase que define el estilo que se desea aplicar sobre el control Web correspondiente. En el Código fuente 103 se puede ver un ejemplo de utilización de esta propiedad. En este caso se ha creado una clase en la propia página, en la Figura 59 se puede ver el resultado de este nuevo ejemplo.

```
<%@ Page language="c#"%>
<html>
<head>
<style>
       .clase{font: 14pt verdana;background-color:lightblue;color:purple;}
</style>
</head>
<body>
<asp:Label  runat="server" id="etique" CssClass="clase">Aplicando estilos sobre
controles Web</asp:Label>
</body>
</html>
```

Código fuente 103

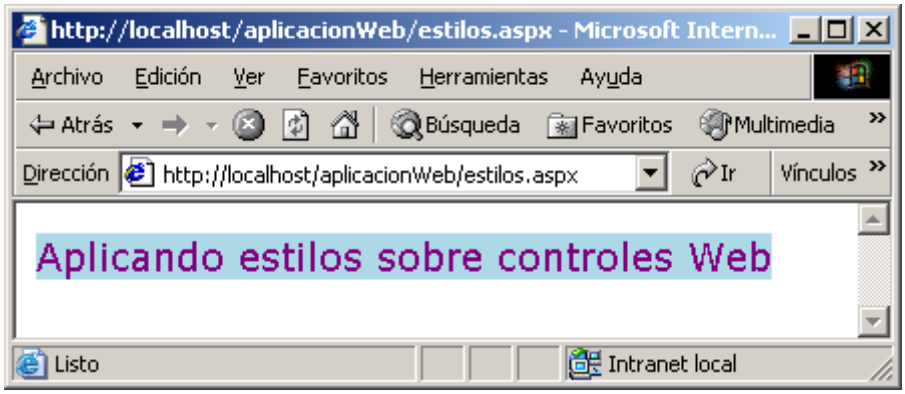


Figura 59

Los estilos también se pueden asignar a través del método ApplyStyle. Este método recibe como parámetro un objeto de la clase Style al que previamente hemos asignado una serie de valores a sus propiedades para definir el estilo que deseamos aplicar al control. En el Código fuente 104 se puede ver como se utiliza este método, y en la Figura 60 se puede ver el resultado del mismo.

```
<%@ Page language="c#"%>
<%@ Import Namespace="System.Drawing" %>

<script language="C#" runat="server">
void Page_Load(Object Sender, EventArgs e){
       Style estilo=new Style();
       estilo.BorderColor=Color.Green;
       estilo.BorderStyle=BorderStyle.Dashed;
       estilo.ForeColor=Color.Blue;
       estilo.BackColor=Color.Pink;
       estilo.Font.Name="Courier";
       estilo.Font.Size=14;
       estilo.Font.Italic=true;
       etiqueta.ApplyStyle(estilo);
}
</script>
<html>
<body>
<asp:Label runat="server" id="etiqueta">Aplicando estilos sobre controles
Web</asp:Label>
</body>
</html>
```

Código fuente 104

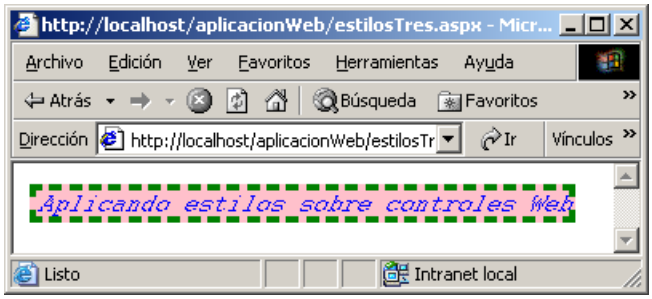


Figura 60

Como se puede comprobar en el código anterior ha sido necesario importan un NameSpace, en concreto el espacio con nombre System.Drawing, para tener acceso a las clases Color y BorderStyle. Para ello se ha utilizado la directiva Import. En un próximo capítulo veremos en detenimiento las distintas directivas que se pueden aplicar en una página ASP .NET.

El Código fuente 105 es el código HTML que se genera para poder aplicar sobre el elemento HTML el estilo indicado en la página.

```
<html>
<body>
<span id="etiqueta" style="color:Blue;background-color:Pink;border-
color:Green;border-style:Dashed;font-family:Courier;font-size:14pt;font-
style:italic;">Aplicando estilos sobre controles Web</span>
</body>
</html>
```

Código fuente 105

Y la última posibilidad de la que disponemos para aplicar estilos a nuestros controles Web es mediante la propiedad Style, que es una colección que nos permite asignar valores a las distintas propiedades del estilo del control. En este caso no se realiza ningún tipo de comprobación para indicar si la propiedad existe o no, únicamente se van añadiendo al control para luego mostrarse en el código HTML como parte de la propiedad style del elemento HTML correspondiente.

En el siguiente ejemplo (Código fuente 106) se muestra como utilizar la propiedad Style, y en el Código fuente 107 se muestra el código HTML que genera la ejecución de la página.

```
<%@ Page language="c#"%>
<script language="C#" runat="server">
void Page_Load(Object Sender, EventArgs e){
      etiqueta.Style["font-size"]="12pt";
      etiqueta.Style["font-family"]="verdana";
      etiqueta.Style["background-color"]="pink";
      etiqueta.Style["color"]="red";
      etiqueta.Style["border-style"]="Dotted";
      etiqueta.Style["border-color"]="Blue";
}
</script>
<html>
<body>
<asp:Label runat="server" id="etiqueta">Aplicando estilos sobre controles
Web</asp:Label>
</body>
</html>
```

Código fuente 106

```
<html>
<body>
<span id="etiqueta" style="font-size:12pt;font-family:verdana;background-
color:pink;color:red;border-style:Dotted;border-color:Blue;">Aplicando estilos
sobre controles Web</span>
</body>
</html>
```

Código fuente 107

Correspondencia entre controles Web intrínsecos y etiquetas HTML

Al igual que hicimos con los controles HTML con los control Web intrínsecos también vamos a mostrar en una tabla (Tabla 7) la correspondencia que existe entre estos controles y las etiquetas HTML que se generan en el navegador que solicita la ejecución de las páginas ASP .NET.

Si comparamos esta correspondencia de etiquetas y controles con la correspondencia que veíamos en el capítulo anterior con los controles HTML, vemos que con los controles Web la correspondencia en algunos casos no están evidente como sucedía con los controles HTML.

Control Web intrínseco	Etiqueta HTML correspondiente
Button	<input type="submit">
CheckBox	<input type="checkbox">
DropDownList	<select>
HyperLink	<a>
Image	<img>
ImageButton	<input type="image">
Label	<span>
LinkButton	<a>
ListBox	<select>
Panel	<div>
PlaceHolder	No genera código HTML
RadioButton	<input type="radio">
Table	<table>
TableCell	<td>,<th>
TableRow	<tr>
TextBox	<input type="text">

Tabla 7

9

Web Forms: controles ricos y de validación

Controles ricos

Este es el segundo grupo de controles Web, los controles ricos (rich controls). En este grupo de controles Web se encuentran una serie de controles avanzados que ofrecen una gran funcionalidad, pero en la versión beta 2 de ASP .NET únicamente nos encontramos con dos controles: el control rotador de anuncios (AdRotator) y el control calendario (Calendar), se espera que se añadan nuevos controles en siguientes versiones de la tecnología.

Al igual que los controles intrínsecos pertenecen también al espacio de nombres System.Web.UI.WebControls.

AdRotator

El control rotador de anuncios (AdRotator), no es ninguna novedad de ASP .NET, ya que disponíamos de este control en versiones anteriores de ASP a través de un componente de servidor. Este control permite mostrar una serie de anuncios, en forma de imágenes, dentro de una página ASP, alternando la aparición de las imágenes.

Para indicar las imágenes y enlaces de los anuncios utilizamos un fichero especial en formato XML, este fichero tiene un formato determinado que vamos a comentar a continuación.

Todo el fichero se encuentra encerrado entre dos elementos <Advertisements>, y cada anuncio viene representado por un elemento <Ad>, y cada elemento que se corresponde con un anuncio tiene una serie de propiedades que definen dicho anuncio, estas propiedades son:

- ImageUrl: es la única de las propiedades que es obligatoria, y es utilizada para indicar la ruta en la que se encuentra el fichero de la imagen. Esta ruta puede ser absoluta o relativa.

- NavigateUrl: esta propiedad indica el enlace que va a tener la imagen, si no se establece esta propiedad al pulsar la imagen del anuncio no ocurrirá nada.

- AlternateText: el texto que se indique en esta propiedad se traducirá en la propiedad Alt de la etiqueta <img> de la imagen del anuncio, es decir, será el ToolTip que aparecerá cuando situemos el cursor del ratón sobre la imagen. Es importante saber que el componente AdRotator da como resultado final una etiqueta <img> de HTML.

- Impressions: el número que presente esta propiedad indicará el peso que se le va a asignar a cada uno de los anuncios incluidos en el fichero XML. El anuncio tendrá más peso según el valor de este número, es decir, si deseamos que el anuncio tenga un peso alto y que por lo tanto aparezca muchas veces, a la propiedad Impressions se le debe dar un valor alto comparado con el resto de valores de los anuncios restantes.

- Keyword: palabra clave que se le puede asignar a la imagen del anuncio.

Una vez definido el fichero con el formato adecuado debemos guardarlo con cualquier nombre pero con extensión XML, y debe encontrarse en el servidor Web en el que vaya utilizarse.

Para utilizar el control Web AdRotator ya sólo nos queda escribir dentro de una página ASP .NET el código correspondiente que instancie un control Web AdRotator. Para el control AdRotator debemos indicar el fichero de anuncios, esto lo realizamos mediante la propiedad AdvertisementFile, que contendrá la ruta al fichero XML.

En el Código fuente 108 se muestra un ejemplo de fichero de anuncios (AdvertisementFile). En este caso son tres anuncios los que se van a ir rotando, dos de ellos tienen el mismo peso y el tercero es el que debe aparecer más veces, ya que tiene el valor más alto en la propiedad Impressions. Cada una de las imágenes tiene su enlace correspondiente.

```
<Advertisements>

<Ad>
    <ImageUrl>imagen1.jpg</ImageUrl>
    <NavigateUrl>http://www.elcampusdigital.com</NavigateUrl>
    <AlternateText>Texto de la imagen</AlternateText>
    <Impressions>50</Impressions>
 </Ad>

 <Ad>
    <ImageUrl>imagen2.jpg</ImageUrl>
    <NavigateUrl>http://www.almagesto.com</NavigateUrl>
    <AlternateText>Segundo Anuncio</AlternateText>
    <Impressions>50</Impressions>
 </Ad>

  <Ad>
    <ImageUrl>imagen3.jpg</ImageUrl>
    <NavigateUrl>http://www.lalibreriadigital.com</NavigateUrl>
    <AlternateText>Libros en formato digital</AlternateText>
    <Impressions>70</Impressions>
```

```
 </Ad>

</Advertisements>
```

Código fuente 108

En el Código fuente 109 se muestra el código necesario para hace uso de este control Web.

```
<%@ Page Language="c#"%>
<html>
<head>
    <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
</head>
<body>
<asp:adrotator runat="Server" AdvertisementFile="anuncios.xml" bordercolor="black"
borderwidth="1" ID="anuncios"></asp:adrotator>
</body>
</html>
```

Código fuente 109

Y en la Figura 61 se puede ver un ejemplo de ejecución.

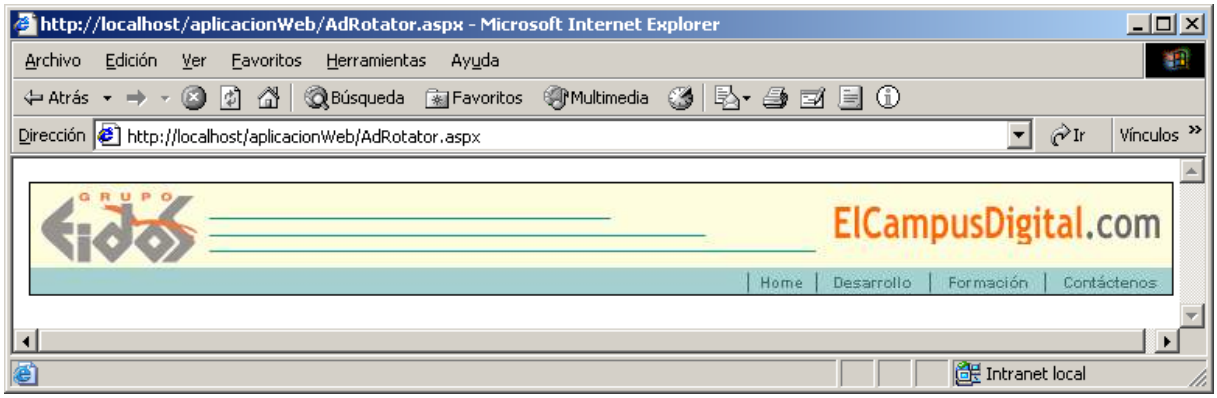


Figura 61

Como ya se ha comentado el control AdRotator generará e código HTML necesario para mostrar la imagen con las propiedades adecuadas que se han indicado a través del fichero XML. En el siguiente código (Código fuente 110) se muestra el código HTML que se generaría para el caso de la figura anterior.

```
<html>
<head>
    <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
</head>
<body>
<a id="anuncios" href="http://www.elcampusdigital.com" target="_top"><img
src="/aplicacionWeb/imagen1.jpg" alt="Texto de la imagen" style="border-
color:Black;border-width:1px;border-style:solid;" /></a>
</body>
</html>
```

Código fuente 110

Calendar

El otro control avanzado o rico que encontramos en la beta 2 de ASP .NET es el control calendario, que ofrece un completo calendario. Es muy sencillo de utilizar aunque ofrece un alto grado de personalización. Este control va a presentar un calendario que permite una navegación por fechas y también selecciones de fecha, el calendario se encuentra basado completamente en código JavaScript.

Para instanciar un control Calendar dentro de una página ASP .NET debemos utilizar la etiqueta <asp:calendar>, y además debemos incluirla dentro de un Web Form.

En el Código fuente 111 se muestra el uso más sencillo del control Calendar, en este caso se toman todas sus propiedades por defecto, el resultado se puede observar en la Figura 62.

```
<%@ Page Language="c#"%>
<HTML>
  <HEAD>
    <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
</HEAD>
<body>
<form id="formulario" method=post runat="server">
<asp:calendar runat="server" id=Calendar1></asp:calendar>
</form>
</body>
</HTML>
```

Código fuente 111

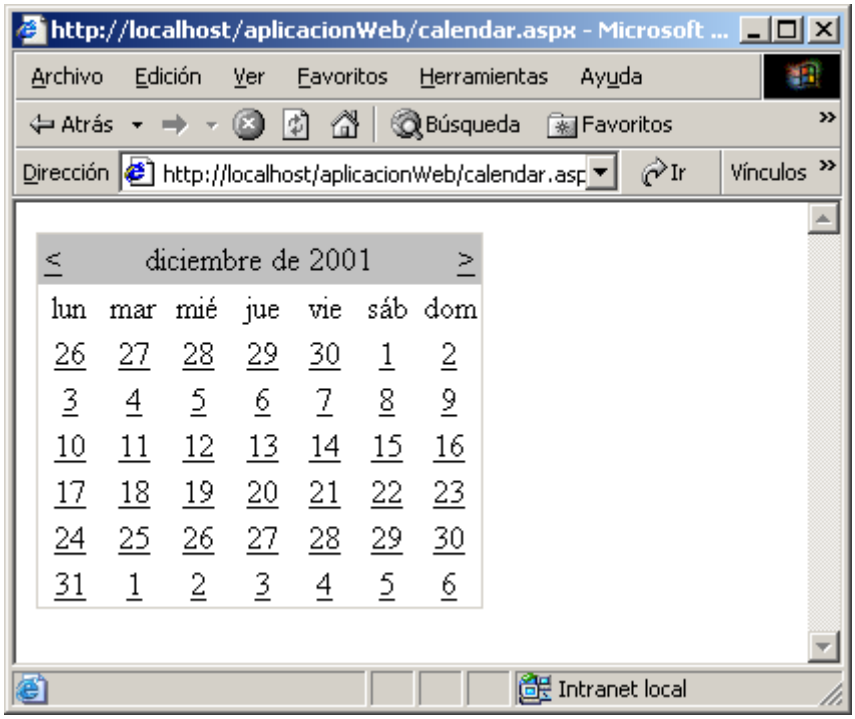


Figura 62

Como ya se ha comentado podemos personalizar el calendario como queremos, ya que se ofrecen un gran número de propiedades que lo permiten. Una de estas propiedades, llamada SelectionMode, es la

que permite indicar el modo de selección de fecha deseado. La propiedad SelectionMode del control Web Calendar puede presentar los siguientes valores:

- Day: únicamente se permite la selección de un solo día del mes.
- DayWeek: se permite la selección de un día del calendario o bien de una semana completa.
- DayWeekMoth: esta es la selección más completa, se permite elegir un día, una semana o el mes completo.
- None: con este último valor deshabilitaremos la selección de fecha en el calendario.

Por defecto esta propiedad tiene el valor Day.

Según sea el modo de selección de fecha el control Calendar presentará un aspecto distinto. Para la selección de un día aparecerán subrayados cada uno de los días del mes, para la selección de semanas mostrará en la columna izquierda una serie de enlaces que permiten seleccionar la semana, y para seleccionar el mes se ofrece en la esquina superior izquierda un enlace. También es posible indicar el aspecto de estos enlaces, pero esto lo veremos más adelante, cuando comentemos otras propiedades del control Calendar.

En el Código fuente 112 se muestra un ejemplo que permite indicar, mediante un control intrínseco DropDownList, el modo de selección de un control avanzado Calendar. Al seleccionar el valor correspondiente se le asignará a la propiedad SelectionMode.

```
<%@ Page Language="c#"%>
<script language="C#" runat="server">
void Page_Load(Object Sender, EventArgs e){
      calendario.SelectionMode = (CalendarSelectionMode)lista.SelectedIndex;
      if (calendario.SelectionMode == CalendarSelectionMode.None)
            calendario.SelectedDates.Clear();
}

void fechaSeleccionada(object s, EventArgs e) {
    switch (calendario.SelectedDates.Count) {
        case (0):
            etiqueta.Text = "No se pueden seleccionar fechas";
         break;
        case (1):
            etiqueta.Text = "La fecha seleccionada es " +
                 calendario.SelectedDate.ToShortDateString();
            break;
       case (7):
            etiqueta.Text = "La selección es la semana que comienza el " +
               calendario.SelectedDate.ToShortDateString();
            break;
       default:
            etiqueta.Text = "La selección es el mes " +
              calendario.SelectedDate.ToShortDateString();
            break;
    }
}
</script>
<HTML>
  <HEAD>
    <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
</HEAD>
<body>
<form id="formulario" method=post runat="server">
```

```
<asp:calendar runat="server" id="calendario"
OnSelectionChanged="fechaSeleccionada">
</asp:calendar>
<asp:DropDownList runat="server" id="lista" AutoPostBack="True">
      <asp:ListItem Value="None" >Sin selección</asp:ListItem>
    <asp:ListItem Selected Value="Day" >Día</asp:ListItem>
    <asp:ListItem Value="DayWeek" >Día y semana</asp:ListItem>
    <asp:ListItem Value="DayWeekMonth" >Día, semana y mes</asp:ListItem>
</asp:DropDownList><br>
<asp:Label runat="server" id="etiqueta"></asp:Label>
</form>
</body>
</HTML>
```

Código fuente 112

Como se puede comprobar se ha asignado a la propiedad AutoPostBack del control DropDownList el valor True, de esta forma cuando seleccionemos el valor de la lista se enviará de forma automática el formulario, y en el método Page_Load se le asignará a la propiedad SelectionMode el valor de la lista. A la hora de asignar el valor de la lista a la propiedad SelectionMode del control Calendar se le hace una conversión de tipos mediante la sintaxis de casting, esto es necesario ya que los valores de la lista son de tipo cadena, y los valores que acepta la propiedad SelectionMode son constantes de la clase CalendarSelectionMode.

En este ejemplo también se puede observar la forma de recuperar el valor de la fecha seleccionada en el calendario en cada uno de los diferentes casos. El evento utilizado es OnSelectionChanged, este evento se lanzará cada vez que cambie la selección de la fecha en el calendario. En el presente ejemplo se han considerado los tres tipos de selección: por un día, una semana o un mes completo. Para distinguir cada caso se ha consultado la propiedad SelectedDates, que es una colección que contiene todas las fechas seleccionadas del calendario y según sea el tipo de selección se indicará un mensaje u otro.

El método ToShortDateString() convierte el objeto DateTime que se encuentra en la propiedad SelectedDate en una representación sencilla de una cadena de caracteres de la fecha seleccionada, del tipo día, mes y año. En la Figura 63 se puede ver un ejemplo de ejecución del código anterior.

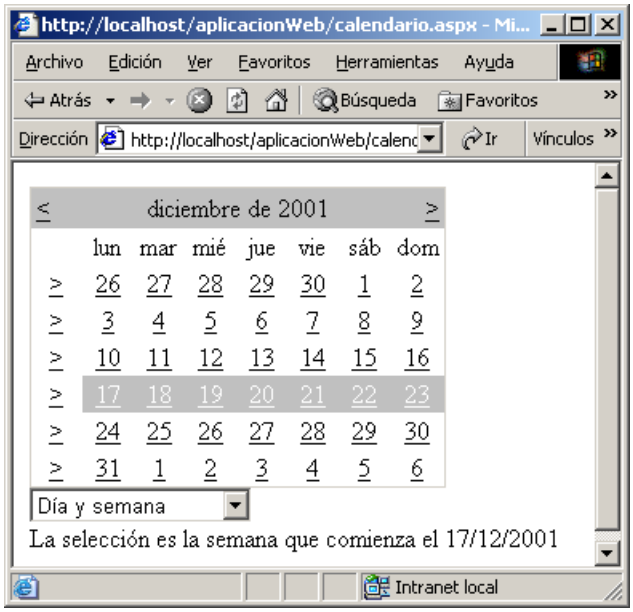


Figura 63

El control avanzado Calendar ofrece un gran número de propiedades para personalizar el aspecto del mismo, estas propiedades se basan en los estilos que se pueden utilizar a la hora de mostrar el calendario. Estas propiedades se puede indicar de dos maneras distintas, bien dentro de la propia etiqueta <asp:calendar>, como se puede observar en el Código fuente 113, o bien dentro de la etiqueta calendar pero como subetiquetas incluidas en la misma, como se muestra en el Código fuente 114. Comparando ambos códigos se pueden ver las diferencias en la sintaxis.

```
<%@ Page Language="c#"%>
<HTML>
  <HEAD>
    <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
</HEAD>
<body>
<form id="formulario" method=post runat="server">
<asp:calendar runat="server" id="calendario"
      DayHeaderStyle-Font-Bold="True" DayHeaderStyle-BackColor="Yellow"
      DayStyle-Font-Name="Arial" DayStyle-Font-Size="8">
</asp:calendar>
</form>
</body>
</HTML>
```

Código fuente 113

```
<%@ Page Language="c#"%>
<HTML>
  <HEAD>
    <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
</HEAD>
<body>
<form id="formulario" method=post runat="server">
<asp:calendar runat="server" id="CalendarioDos">
<DayHeaderStyle Font-Italic="True" Font-Bold="True" BackColor="Blue">
</DayHeaderStyle>
<DayStyle Font-Name="Courier" Font-Size="8"></DayStyle>
</asp:calendar>
</form>
</body>
</HTML>
```

Código fuente 114

A continuación vamos a describir algunas de las propiedades del control Calendar, en esta caso se tratan de propiedades que nos van a permitir especificar el aspecto que van a tener cada uno los distintos elementos o secciones del control Calendar:

- DayHeaderStyle: indica el estilo que se va a utilizar para mostrar el nombre del día de la semana del calendario (lun,mar,mié). En el caso de estas propiedades tienen a su vez otra serie de propiedades que definen el estilo, y que en algunos casos son valores numéricos o boléanos u otras constantes. Así por ejemplo esta propiedad puede tener una serie de valores como los siguientes: Font-Size (tamaño de letra), Font-Italic (para indicar si la letra va a ser cursiva), BackColor (color del fondo), etc. Todas estas propiedades se repetirán en las distintas propiedades del control Calendar, ya que al fin y al cabo nos permiten definir el estilo que se va a aplicar a una propiedad determinada. Se debe tener en cuenta que el control Calendar es

un control Web y como tal se le pueden aplicar estilos, la aplicación de los estilos sobre los controles Web se trataba con detenimiento en el capítulo anterior.

- DayStyle: permite definir el estilo que van a tener los días del mes actual, es decir, nos permite indicar el aspecto que tienen los números del día del mes.

- NextPrevStyle: indica el aspecto que van a tener los enlaces que al pulsarlos nos desplazan al mes anterior y al mes siguiente, es decir, los enlaces de anterior y siguiente que permiten la navegación a través de los meses del calendario.

- OtherMothDayStyle: define el aspecto que van a tener los días del calendario que no pertenecen al mes actual, pero que se muestran en la primera y última semana.

- SelectedDayStyle: permite especificar el estilo que va a mostrar el día o conjunto de fechas seleccionados del calendario.

- SelectorStyle: indica el aspecto que va a tener el selector de fechas, es decir, el selector que permite seleccionar las semanas completas y/o los meses completos.

- TitleStyle: esta propiedad permite establecer el aspecto del título del mes actual.

- TodayDayStyle: define el estilo del día actual, esta propiedad puede ser útil si queremos resaltar de alguna forma el día actual dentro del calendario.

- WeendEndDayStyle: este último estilo hace referencia al aspecto que van a tener los días del fin de semana del calendario. Puede ser útil si queremos diferenciar gráficamente los días festivos.

El siguiente código fuente (Código fuente 115), además de mostrar como se utilizan estas propiedades del control avanzado Calendar, también muestra la distinta sintaxis que podemos emplear a la hora de definir los valores de dichas propiedades.

```
<%@ Page Language="c#"%>
<HTML>
  <HEAD>
    <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
</HEAD>
<body>
<div align="center">
<form id="formulario" method=post runat="server">
<asp:calendar runat="server" id="calendario"
      SelectionMode="DayWeekMonth"
      DayHeaderStyle-Font-Bold="True" DayHeaderStyle-BackColor="Yellow"
      DayStyle-Font-Name="Arial" DayStyle-Font-Size="8"
      NextPrevStyle-BackColor="Red" NextPrevStyle-ForeColor="Yellow"
      OtherMonthDayStyle-BackColor="#cccccc" OtherMonthDayStyle-Font-Italic="True"
             OtherMonthDayStyle-ForeColor="white"
      SelectedDayStyle-BackColor="Red"
      SelectorStyle-Font-Size="8"
      TitleStyle-Font-Name="Arial" TitleStyle-ForeColor="Red"
      TodayDayStyle-BorderStyle="Dashed" TodayDayStyle-BorderWidth="1"
             TodayDayStyle-BorderColor="Green"
      WeekendDayStyle-ForeColor="Red">
</asp:calendar>
<hr>
<asp:calendar runat="server" id="CalendarioDos" SelectionMode="DayWeekMonth">
```

```
<DayHeaderStyle Font-Italic="True" Font-Bold="True" BackColor="Blue">
</DayHeaderStyle>
<DayStyle Font-Name="Courier" Font-Size="8"></DayStyle>
<NextPrevStyle BorderStyle="Dashed" BorderWidth="1"></NextPrevStyle>
<OtherMonthDayStyle Font-Size="7" Font-Name="Arial"></OtherMonthDayStyle>
<SelectedDayStyle Font-Italic="True" Font-Size="10"></SelectedDayStyle>
<SelectorStyle  Font-Size="7" BorderColor="Blue" BorderStyle="Outset"
BorderWidth="1"></SelectorStyle>
<TitleStyle Font-Name="Courier" BorderStyle="Dashed" BorderWidth="3"
BorderColor="Orange"></TitleStyle>
<TodayDayStyle Font-Bold="True" ForeColor="Red"></TodayDayStyle>
<WeekendDayStyle Font-Bold="True" ForeColor="Gray"></WeekendDayStyle>
</asp:calendar>
</form>
</div>
</body>
</HTML>
```

Código fuente 115

En la Figura 64 se puede ver el aspecto que tendrían los dos objetos Calendar que se han creado para el ejemplo.

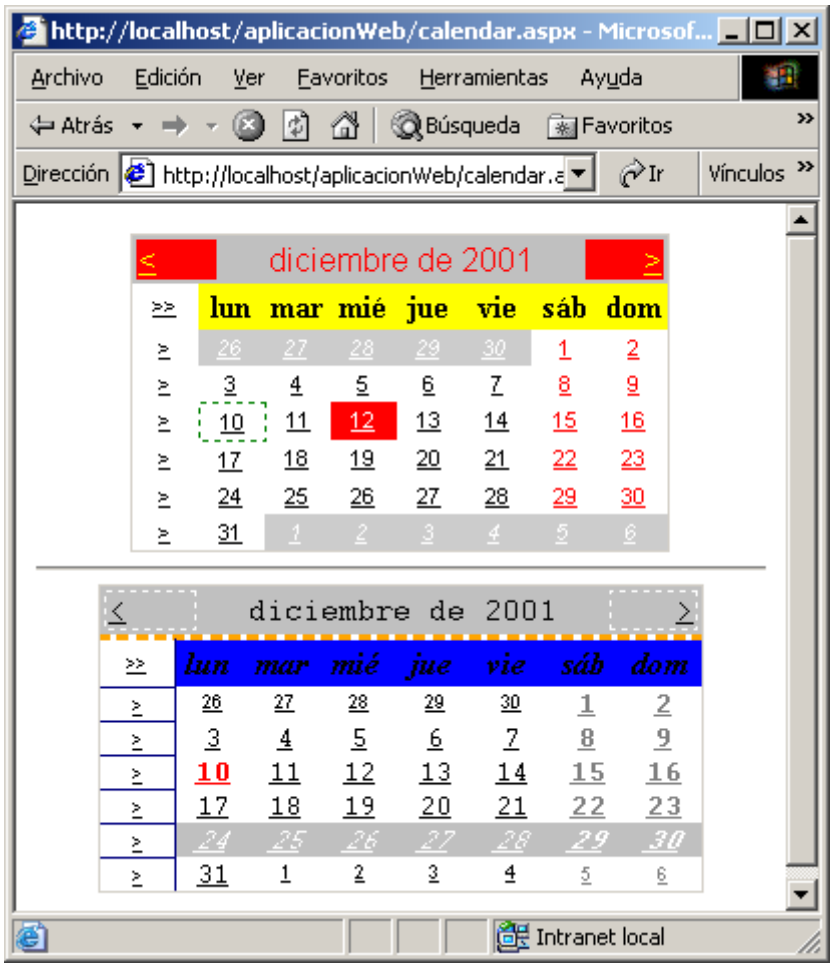


Figura 64

En el siguiente bloque se comentan otra serie de propiedades, también pertenecientes al control Calendar, pero en este caso tienen que ver con el formato en el que se muestran algunos de los elementos del calendario. Estas propiedades no tiene dos tipos de sintaxis, sino que se deben utilizar como propiedades de la etiqueta <asp:calendar>, no es posible utilizarlas como subetiquetas.

- DayNameFormat: este propiedad permite indicar el formato en el que va a aparecer el nombre del día de la semana, los valores que puede tomar esta propiedad son los siguientes: FirstLetter (la primera letra del día), FirstTwoLetters (las dos primeras letras del día), Full (el nombre del día de la semana completo), Short (las tres primeras letras del día), este último valor es el valor por defecto que presenta esta propiedad.

- NextPrevFormat: indica el formato que va a tener los enlaces de mes anterior y siguiente del calendario. Los valores que puede tomar esta propiedad son: FullMonth (el nombre completo del mes), ShortMonth (la abreviatura del mes, se muestran las tres primeras letras), CustomText (texto personalizado, el texto personalizado a mostrar se indicará en la propiedades PrevMonthText y NextMonthText).

- PrevMonthText y NextMonthText: propiedades relacionadas con la anterior, contendrán el texto personalizado que se mostrará en los controles de navegación del mes anterior y mes siguiente respectivamente. Aquí podemos utilizar el texto que deseemos, incluso como se verá más adelante (Código fuente 116) podemos indicar una imagen utilizando el texto HTML correspondiente.

- TitleFormat: especifica el formato del título, puede tener dos valores: Month (se muestra el mes actual) y MonthYear (se muestran el mes y año actuales).

- SelectMonthText y SelectWeekText: al igual que se puede especificar un texto determinado para mostrar la navegación de los meses mediante las propiedades PrevMonthText y NextMonthTex, con estas otras dos propiedades podemos especificar el texto que queremos utilizar en los selectores de mes y de semana, respectivamente. Al igual que seucede en el caso anterior también podemos utilizar etiquetas HTML.

En el siguiente código (Código fuente 116) se muestra la utilización de estas propiedades en dos objetos Calendar distintos, el efecto de los distintos valores en estos calendarios se puede apreciar en la Figura 65.

```
<%@ Page Language="c#"%>
<HTML>
  <HEAD>
    <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
</HEAD>
<body>
<div align="center">
<form id="formulario" method=post runat="server">
<asp:calendar runat="server" id="calendario"
       SelectionMode="DayWeekMonth" DayNameFormat="Full"
       NextPrevFormat= "FullMonth" TitleFormat="Month"
       SelectMonthText="|->" SelectWeekText="--">
</asp:calendar>
<hr>
<asp:calendar runat="server" id="CalendarioDos" SelectionMode="DayWeekMonth"
 NextPrevFormat= "CustomText"
 PrevMonthText="<img src=anterior.gif>"
 NextMonthText="<img src=siguiente.gif>"
 TitleFormat="MonthYear"
 SelectMonthText="<img border=0 src=selecmes.gif>"
```

```
 SelectWeekText="<img border=0 src=selecsemana.gif>">
</asp:calendar>
</form>
</div>
</body>
</HTML>
```

Código fuente 116

Con este ejemplo de algunas de las propiedades del control Calendar damos por finalizado el apartado dedicado a los controles Web avanzados. En el siguiente apartado vamos a comentar el siguiente tipo de controles Web, los controles Web de validación.

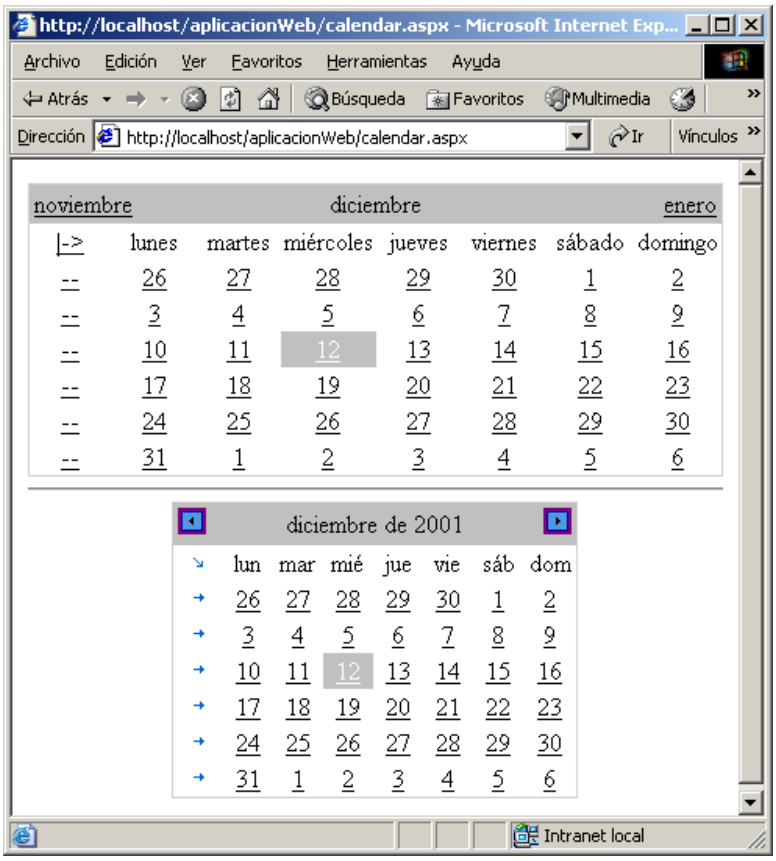


Figura 65

Controles de validación

Este grupo de controles nos permiten validar la entrada dada por el usuario a través de los controles de un Web Form, esta validación se puede realizar en el cliente o en el servidor. Si el navegador lo soporta (navegador de alto nivel) se realizará en el cliente a través de script de cliente, pero si el navegador no lo soporta (navegador de bajo nivel), la validación se llevará a cabo en el servidor, lo que supone una conexión más con el servidor.

Si queremos forzar que la validación se realice en el servidor podemos utilizar la propiedad ClientTarget de la directiva Page, a la que le asignaremos el valor "DownLevel", esto lo veremos más adelante con algunos ejemplos.

ASP .NET ofrece seis controles de validación que pertenecen también al espacio de nombres System.Web.UI.WebControls, y son los siguientes:

- RequiredFieldValidator: este control valida si se ha rellenado un campo con un valor, es decir, valida los campos requeridos.
- CompareValidator: comprueba si un control contiene un valor determinado o coincide con el contenido de otro control, se utiliza con un operador de comparación.
- RangeValidator: valida si el valor indicado por el usuario se encuentra dentro de un rango determinado.
- RegularExpresionValidator: es el más versátil de los controles de validación, valida si el valor indicado por el usuario se corresponde con una expresión.
- CustomValidator: este control permite especificar una función de validación personalizada.
- ValidationSummary: este control contiene todos los mensajes de error que se han producido y muestra una lista de ellos en la página.

La validación se realiza sobre los controles de un Web Form, y un mismo control puede tener distintos tipos de validación, es decir, se pueden utilizar distintos controles de validación sobre un mismo control a validar. Estos controles de validación se utilizan de la misma forma que el resto de los controles Web de ASP .NET.

El código estándar que se suele utilizar con los controles Web de validación se puede observar en el Código fuente 117.

```
<asp:control_validación id="identificador" runat="server"
  controlToValidate="id_control"
  errorMessage="Mensaje de error"
  display="static|dynamic|none">
</asp:control_validación>
```

Código fuente 117

En la propiedad controlToValidate debemos indicar el identificador del control que deseamos validar, es decir, el valor de la propiedad id del control.

La propiedad errorMessage contiene una cadena de texto que será el mensaje de error que se muestra si la entrada ofrecida por el usuario no es válida, y la propiedad display indica como se muestra el control de validación en la página, si indicamos el valor static se reservará espacio en la página para el mensaje de error, con el valor dynamic se desplazarán los controles para mostrar el mensaje de error, y con none no se mostrará el mensaje de error.

Si especificamos algún texto entre las etiquetas de apertura y cierre del control Web de validación se mostrará ese texto si se produce un error, en lugar del texto indicado en la propiedad errorMessage, que será descartado.

Para saber si se han realizado con éxito todas las validaciones de una página se puede consultar la propiedad isValid del objeto Page. Si devuelve verdadero esta propiedad, indica que todas las validaciones se han realizado correctamente. El objeto Page, que veremos con detenimiento en el

capítulo correspondiente, va a representar a la páginas ASP .NET actual, este objeto va a estar disponible en cada página y pertenece al espacio con nombre (NameSpace) System.Web.UI.

Cada control de validación tendrá unas propiedades específicas, así por ejemplo el control de validación RangeValidator posee las propiedades minimumVale y maximumValue para indicar los valores mínimo y máximo del rango a validar, y la propiedad type para indicar el tipo de dato que se desea validar.

Después de esta breve introducción general a los controles Web de validación vamos a tratar en detalle cada uno de estos controles mostrando su utilización mediante el ejemplo correspondiente.

RequiredFieldValidator

Como ya hemos adelantado este control Web de validación es utilizado para verificar que se ha dado algún valor a un campo, es decir, valida que se ha asignado un valor a un campo requerido de un Web Form.

En el Código fuente 118 se puede ver la utilización de un par de controles RequiredFieldValidator, en este caso cada uno de los controles hacen que los campos nombre y apellidos del Web Form sean campos requeridos, respectivamente, es decir, es obligatorio que el usuario rellene ambos campos del formulario. En el caso de que alguno de los campos no tengan valor se mostrará el mensaje de error correspondiente. La validación se realiza cuando se produce el envío del formulario, en este caso, al pulsar el botón correspondiente, si la página no es válida, es decir, no se han realizado las validaciones de forma correcta y el cliente que utilizamos es un cliente de alto nivel, la página no será enviada al servidor.

```
<%@ Page language="c#"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
<head>
<meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" Content="C#">
</head>
<body>
<form id="formulario" method="post" runat="server">
Nombre:<asp:TextBox Id="nombre" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator id="validadorNombre"  runat="server"
ControlToValidate="nombre" ErrorMessage="Debe indicar su nombre" Display="Dynamic"
 Font-Italic="True" Font-Name="Courier" Font-Size="7">
</asp:RequiredFieldValidator>
<br>
Apellidos:<asp:TextBox Id="apellidos" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator id="validadorApellidos"  runat="server"
ControlToValidate="apellidos" ErrorMessage="Debe indicar sus apellidos"
Display="Dynamic"
 ForeColor="Green">
</asp:RequiredFieldValidator>
<br>
Teléfono:<asp:TextBox Id="telefono" runat="server"></asp:TextBox><br>
<asp:Button ID="boton" runat="server" Text="Enviar datos"></asp:Button>
</form>

</body>
</html>
```

Código fuente 118

El mensaje de error aparecerá en rojo de forma predeterminada, a no ser que indiquemos las propiedades correspondientes dentro del control RequiredFieldValidator para especificar el estilo que van a tener los mensajes de error que se muestren en pantalla. Se debe señalar, como algo común para todos los controles Web de validación, que el control Web debe ir ubicado en el código de la página en el lugar en el que queramos que aparezca el mensaje de error. En la Figura 66 se puede ver el aspecto de los dos mensajes de error al no indicar ninguno de los campos requeridos del formulario.

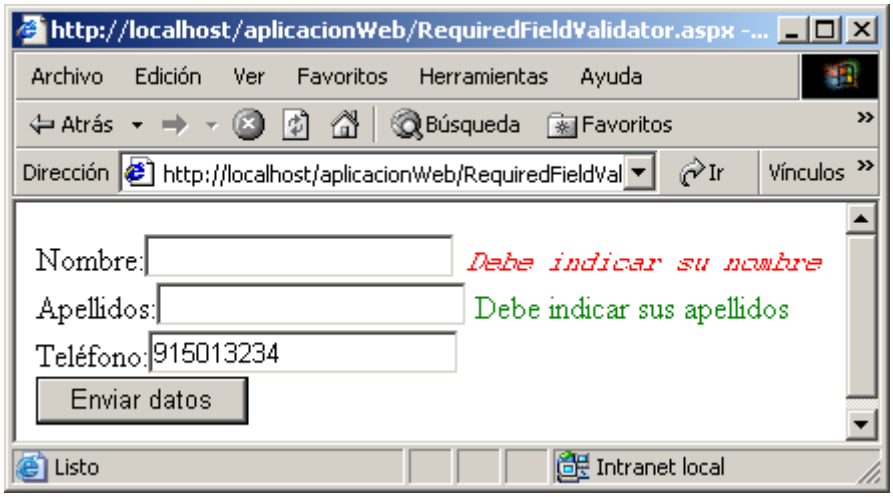


Figura 66

Al no indicar nada en la propiedad ClientTarget de la directiva Page (en un próximo capítulo veremos cada una de las directivas existentes en ASP .NET), se realizará la validación en el cliente o en el servidor según la versión del navegador, es decir, si el navegador soporta DHTML.

Si deseamos que el mensaje de error aparezca de forma automática cuando se pierde el foco del control, deberemos situar el texto del mensaje de error entre las etiquetas de apertura y de cierre de los controles RequiredFieldValidator en lugar de indicar el texto de error en la propiedad ErrorMessage, esta forma de funcionamiento será posible únicamente en clientes de alto nivel, es decir, navegadores en últimas versiones que soporta DHTML de forma completa.

En el Código fuente 119 se puede ver el aspecto que tendría la página ASP .NET si la rescribimos para mostrar de forma dinámica los mensajes de error, sin necesidad de pulsar el botón de envío.

```
<%@ Page language="c#"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
<head>
<meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" Content="C#">
</head>
<body>
<form id="formulario" method="post" runat="server">
Nombre:<asp:TextBox Id="nombre" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator id="validadorNombre"  runat="server"
ControlToValidate="nombre" Display="Dynamic"
 Font-Italic="True" Font-Name="Courier" Font-Size="7">Debe indicar su
nombre</asp:RequiredFieldValidator>
<br>
Apellidos:<asp:TextBox Id="apellidos" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator id="validadorApellidos"  runat="server"
ControlToValidate="apellidos" Display="Dynamic"
```

```
ForeColor="Green">Debe indicar sus apellidos</asp:RequiredFieldValidator>
<br>
Teléfono:<asp:TextBox Id="telefono" runat="server"></asp:TextBox><br>
<asp:Button ID="boton" runat="server" Text="Enviar datos"></asp:Button>
</form>
</body>
</html>
```

Código fuente 119

Aunque si realizamos varias pruebas vemos que en algunos casos no funciona correctamente esta forma de indicar los mensajes de error, esperemos que esto se solucione en la versión final de la plataforma .NET.

El código HTML generado en este ejemplo se puede ver en el Código fuente 120, como se puede apreciar se genera código JavaScript para realizar la validación en el cliente (en el navegador de alto nivel).

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
<head>
<meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" Content="C#">
</head>
<body>
<form name="formulario" method="post" action="requiredfieldvalidator.aspx"
language="javascript" onsubmit="ValidatorOnSubmit();" id="formulario">
<input type="hidden" name="__VIEWSTATE" value="dDwtMjI3OTYzNDMwOzs+" />

<script language="javascript"
src="/aspnet_client/system_web/1_0_2914_16/WebUIValidation.js"></script>


Nombre:<input name="nombre" type="text" id="nombre" />
<span id="validadorNombre" controltovalidate="nombre" display="Dynamic"
evaluationfunction="RequiredFieldValidatorEvaluateIsValid" initialvalue=""
style="color:Red;font-family:Courier;font-size:7pt;font-
style:italic;display:none;">Debe indicar su nombre</span>
<br>
Apellidos:<input name="apellidos" type="text" id="apellidos" />
<span id="validadorApellidos" controltovalidate="apellidos" display="Dynamic"
evaluationfunction="RequiredFieldValidatorEvaluateIsValid" initialvalue=""
style="color:Green;display:none;">Debe indicar sus apellidos</span>
<br>
Teléfono:<input name="telefono" type="text" id="telefono" /><br>
<input type="submit" name="boton" value="Enviar datos" onclick="if
(typeof(Page_ClientValidate) == 'function') Page_ClientValidate(); "
language="javascript" id="boton" />

<script language="javascript">
<!--
      var Page_Validators =  new Array(document.all["validadorNombre"],
document.all["validadorApellidos"]);
            // -->
</script>


<script language="javascript">
<!--
var Page_ValidationActive = false;
```

```
if (typeof(clientInformation) != "undefined" &&
clientInformation.appName.indexOf("Explorer") != -1) {
    if (typeof(Page_ValidationVer) == "undefined")
        alert("No se puede encontrar la biblioteca de secuencias de comandos
/aspnet_client/system_web/1_0_2914_16/WebUIValidation.js. Intente colocar este
archivo manualmente o reinstalarlo ejecutando 'aspnet_regiis -c'.");
    else if (Page_ValidationVer != "121")
        alert("Esta página utiliza una versión incorrecta de WebUIValidation.js. Se
esperaba la versión 121. La biblioteca de secuencias de comandos es " +
Page_ValidationVer + ".");
    else
        ValidatorOnLoad();
}

function ValidatorOnSubmit() {
    if (Page_ValidationActive) {
        ValidatorCommonOnSubmit();
    }
}
// -->
</script>
</form>
</body>
</html>
```

Código fuente 120

CompareValidator

Este control Web de validación lo utilizaremos para comparar el valor que contiene un control Web con un valor específico, que puede ser un valor determinado especificado como una constante o bien un valor de otro control Web. Este control Web se utilizará con un operador de comparación, y presenta las siguientes propiedades adicionales:

- ValueToCompare: esta propiedad se utilizará cuando deseemos comparar el valor de un control Web con un valor constante, esta propiedad es la que contendrá este valor constante.

- ControlToCompare: esta propiedad se utilizará cuando deseemos compara el valor de un control Web con otro, esta propiedad contendrá el identificador del otro control contra el que se quiere comparar el control Web en cuestión.

- Type: indica el tipo de dato de los valores que se van a comparar, esta propiedad puede presentar los siguientes valores, Currency, Date, Double, Integer y String, que se corresponden con los distintos tipos de datos que podemos indicar.

- Operator: esta última propiedad va a contener el tipo de operador que se va a utilizar en la comparación de los dos valores. Esta propiedad puede tener los siguientes valores, Equal (igual), GreaterThan (mayor), GreaterThanEqual (mayor o igual), LessThan (menor), LessThanEqual (menor o igual), NotEqual (distinto) y DataTypeCheck (validación de tipo). Estos valores son bastante descriptivos, ya que se corresponden con los operadores de comparación correspondientes, el único que se debe comentar es el último de estos valores. El operador DataTypeCheck es utilizado para validar el tipo de datos del control Web, en este caso no se compara con otro control, sino que únicamente se comprueba el tipo del valor del control.

Vamos a comentar el funcionamiento y utilización de este control Web de validación con un par de ejemplos ilustrativos.

En el este primer ejemplo (Código fuente 121) se muestra un formulario Web que posee un único campo que nos solicita la edad. Para validar este control Web del formulario vamos a utilizar dos controles CompareValidator y un control RequiredFieldValidator. El primer control de comparación va a verificar que el valor indicado para la edad sea mayor o igual que cero, y el segundo control de comparación va a verificar que el valor de la edad sea de tipo entero. El control RequiredFieldValidator, como no es de extrañar, va a comprobar que el indicamos algún valor en el campo edad.

En la Figura 67 se puede ver un ejemplo de ejecución de esta página ASP .NET.

```
<%@ Page language="c#"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
<head>
<meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" Content="C#">
</head>
<body>
<form id="formulario" method="post" runat="server">
Edad:<asp:TextBox Id="edad" runat="server"></asp:TextBox>
<br>
<asp:CompareValidator id="validadorComparaEdad"  runat="server"
ControlToValidate="edad" ValueToCompare="0" Type="Integer"
Operator="GreaterThanEqual"
ErrorMessage="Debe ser mayor o igual que cero"
Display="Dynamic" Font-Italic="True" Font-Name="Courier" Font-Size="7">
</asp:CompareValidator>
<br>
<asp:CompareValidator id="validadorTipoDatoEdad"  runat="server"
ControlToValidate="edad" Type="Integer" Operator="DataTypeCheck"
ErrorMessage="Debe ser un entero"
Display="Dynamic" Font-Italic="True" Font-Name="Courier" Font-Size="7">
</asp:CompareValidator>
<asp:RequiredFieldValidator id="validadorNombre"  runat="server"
ControlToValidate="edad" ErrorMessage="Debe indicar su edad" Display="Dynamic">
</asp:RequiredFieldValidator>
<br>

<asp:Button ID="boton" runat="server" Text="Enviar datos"></asp:Button>
</form>
</body>
</html>
```

Código fuente 121

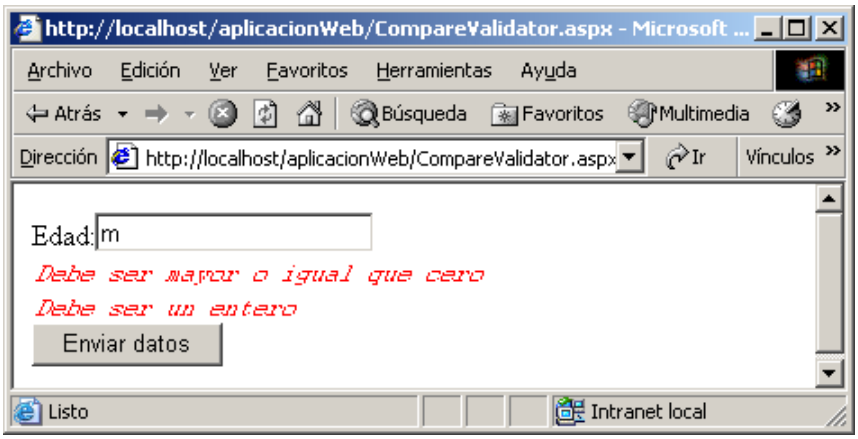


Figura 67

En el siguiente ejemplo(Código fuente 122) vamos a ir comparando de forma dinámica los valores de cadena de dos objetos TextBox de un Web Form. El operador se selecciona en cada caso de una lista desplegable del formulario, representada por un objeto de la clase ListBox. Al pulsar el botón del formulario se indicará si la comparación ha sido válida o no, para ello se consulta la propiedad isValid del objeto Page actual, es decir, el objeto que representa a la página ASP .NET actual. El resultado de la comparación se indicará en un objeto Label

En este caso el valor de la propiedad Operator del control de comparación se asigna de forma dinámica.

Para que esta página ASP .NET funcione de manera correcta debemos indicar que el cliente Web es de bajo nivel mediante la propiedad ClientTarget de la directiva Page, de esta forma la validación se realizará siempre en el servidor Web, en lugar de en el cliente.

```
<%@ Page ClientTarget="downlevel" language="c#"%>
<html>
<head>
<script language="C#" runat="server">
void EnviarFormulario(Object sender, EventArgs e) {
    if (Page.IsValid) {
       resultado.Text = "Comparación correcta";
    }else{
       resultado.Text = "Comparación incorrecta";
    }
}

void SeleccionLista(Object sender, EventArgs e) {
  controlComparacion.Operator = (ValidationCompareOperator) lista.SelectedIndex;
}
</script>

</head>
<body>
<form runat="server" id="formulario">
<table border="1">
<tr valign="top">
      <td>
            Cadena uno:</font><br>
            <asp:TextBox id="campoUno" runat="server"></asp:TextBox>
    </td>
    <td>
     Operador de comparación:<br>
     <asp:ListBox id="lista" OnSelectedIndexChanged="SeleccionLista"
                                                    runat="server">
          <asp:ListItem Selected Value="Equal" >Igual (=)</asp:ListItem>
          <asp:ListItem Value="NotEqual" >Distinto(!=)</asp:ListItem>
          <asp:ListItem Value="GreaterThan" >Mayor(>)</asp:ListItem>
         <asp:ListItem Value="GreaterThanEqual" >Mayor o igual (>=)</asp:ListItem>
          <asp:ListItem Value="LessThan" >menor(<)</asp:ListItem>
          <asp:ListItem Value="LessThanEqual" >menor o igual (<=)</asp:ListItem>
   </asp:ListBox>
   </td>
   <td>
   Cadena dos:<br>
   <asp:TextBox id="campoDos" runat="server"></asp:TextBox><br>
   <asp:Button runat=server Text="Validate" ID="boton" onclick="EnviarFormulario"/>
   </td>
</tr>
</table>

<asp:CompareValidator id="controlComparacion"
ControlToValidate="campoUno" ControlToCompare = "campoDos"
```

```
Type="String" runat="server"/>
<br>

<asp:Label ID="resultado" Font-Name="verdana" Font-Size="10pt" runat="server"/>

</form>
</body>
</html>
```

Código fuente 122

En la Figura 68 se puede ver un ejemplo de ejecución de esta nueva página ASP .NET.

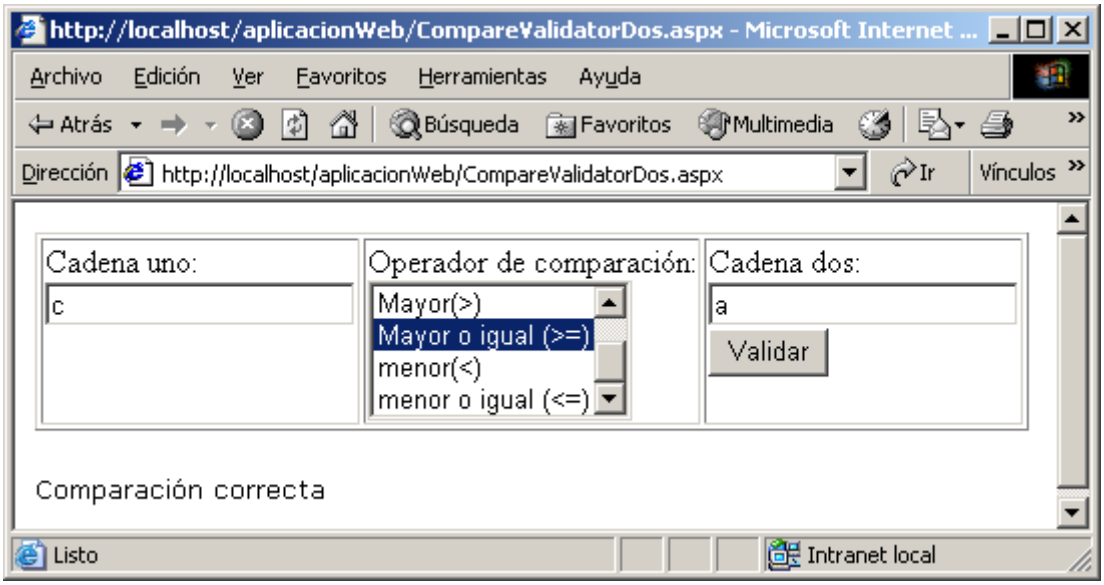


Figura 68

RangeValidator

Este control Web se utilizará cuando sea necesario comprobar si un valor de entrada de un control Web se encuentra comprendido en un rango determinado.

El control RangeValidator ofrece tres propiedades adicionales para llevar a cabo su labor de validación, las propiedades son las siguientes:

- MinimumControl: esta propiedad define el valor mínimo del rango permitido para el control.
- MaximumControl: esta propiedad define el valor máximo del rango permitido para el control.
- Type: indica el tipo de dato que se utiliza para comparar los valores dentro del rango especificado, esta propiedad puede tener los siguientes valores, Currency, Date, Double, Integer y String. Esta propiedad ya se utilizaba con el control CompareValidator.

En el Código fuente 123 se puede ver un ejemplo de utilización del control RangeValidator, en este caso se comprueba que el valor indicado se encuentre comprendido entre 1 y 100.

```
<%@ Page language="c#"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<html>
<body>
<form id="formulario" method="post" runat="server">
Valor:<asp:TextBox Id="valor" runat="server"></asp:TextBox>
<br>
<asp:RangeValidator id="validadorRango"  runat="server"
ControlToValidate="valor" ErrorMessage="Debe indicar un entero en el rango 1-100"
Display="Dynamic"  MinimumValue="1" MaximumValue="100" Type="Integer">
</asp:RangeValidator>
<br>
<asp:Button ID="boton" runat="server" Text="Enviar datos"></asp:Button>
</form>

</body>
</html>
```

Código fuente 123

En la Figura 69 se puede ver un ejemplo de ejecución de la página anterior.

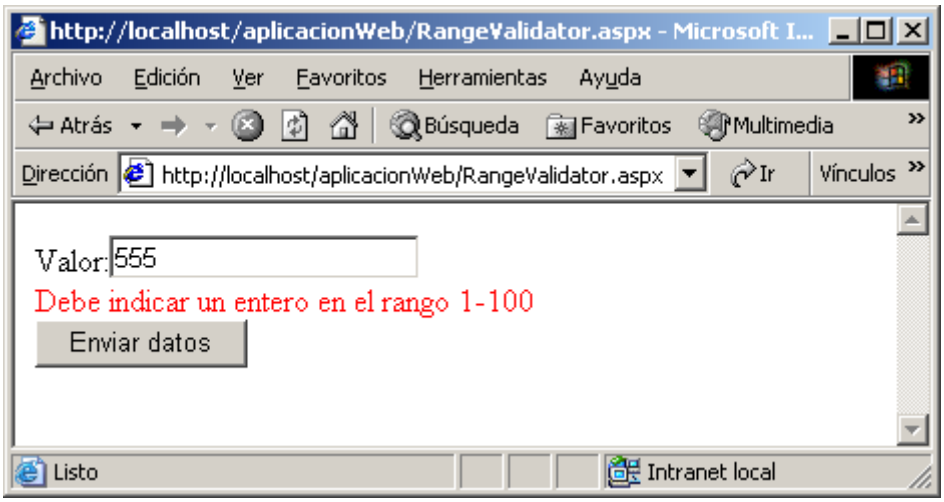


Figura 69

RegularExpressionValidator

Este otro control Web de validación es utilizado para comparar el valor del control correspondiente con una expresión regular.

Este control debe hacer uso de la propiedad RegularExpressionValidator para indicar la expresión regular con la que se va a comparar el valor indicado en el control Web que se desea validar.

En el Código fuente 124 se puede ver un ejemplo de validación de una expresión regular, en este caso el patrón que define la expresión regular es el de un dígito que debe ir seguido de un guión y a continuación debe aparecer otro dígito.

```
<%@ Page language="c#"%>
<html>
<body>
```

```
<form id="formulario" method="post" runat="server">
Valor:<asp:TextBox Id="valor" runat="server"></asp:TextBox>
<br>
<asp:RegularExpressionValidator id="validadorExpresion"  runat="server"
ControlToValidate="valor" ErrorMessage="Debe indicar dos dígitos en el formato 2-3"
Display="Dynamic" ValidationExpression="[0-9]-[0-9]">
</asp:RegularExpressionValidator>
<br>
<asp:Button ID="boton" runat="server" Text="Enviar datos"></asp:Button>
</form>
</body>
</html>
```

Código fuente 124

El resultado de el ejecución de este ejemplo se puede apreciar en la Figura 70.

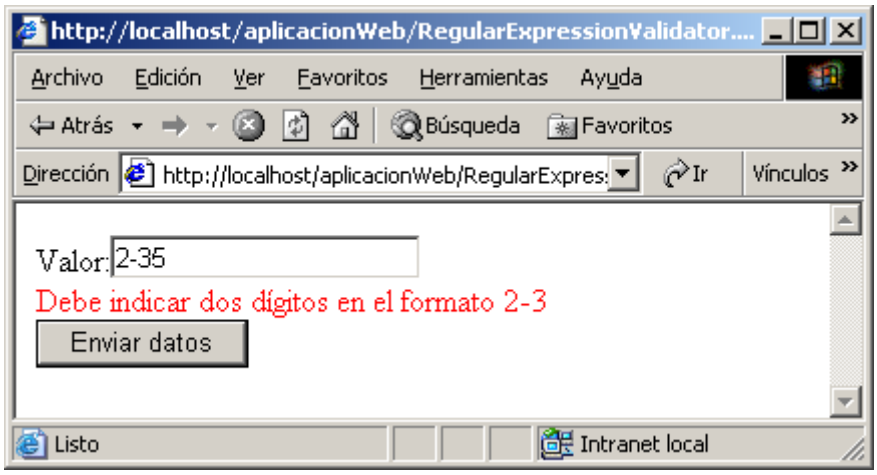


Figura 70

ValidationSummary

Este control difiere del resto, ya que en realidad no va a realizar un tipo de validación determinada, sino que contiene todos los mensajes de error que se han producido y muestra una lista de ellos en la página, en el caso de que la página no sea válida, es decir, algunas de las validaciones han fallado.

Este control mostrará todos los mensajes indicados en la propiedad ErrorMessage de cada uno de los controles de validación cuya validación ha fallado en la página. Así en este caso los controles de validación no mostrarán el mensaje de error de la propiedad ErrorMessage, sino que mostrarán si procede el texto incluido entre sus etiquetas de inicio y fin.

Del control ValidationSummary podemos destacar las siguientes propiedades:

- HeaderText: esta propiedad contendrá el texto que va a mostrarse como cabecera a mode de descripción de los errores.

- ShowSummary: esta propiedad, que puede tener los valores True/False, indicará si desemoas mostrar o no el resumen de validación de la página ASP .NET.

- DisplayMode: en esta propiedad podemos indicar la forma en la que se mostrarán las distintas descripciones de cada uno de los errores de validación encontrados en la página, puede presentar los siguientes valores, BulletList (una lista con puntos), List (una lista) y SingleParagraph (en un único párrafo).

En el siguiente ejemplo (Código fuente 125) tenemos un Web Form que posee dos campos, uno de ellos se valida con un control RequiredFieldValidator y otro con un control CompareValidator, para mostrar un resumen de los errores de validación en la página utilizaremos un objeto ValidationSummary.

```
<%@ Page language="c#"%>
<html>
<body>
<form id="formulario" method="post" runat="server">
Valor uno:<asp:TextBox Id="valorUno" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator id="validadorUno"  runat="server"
ControlToValidate="valorUno" ErrorMessage="Debe indicar el valor uno"
Display="Dynamic">
*
</asp:RequiredFieldValidator>
<br>
Valor dos:<asp:TextBox Id="valorDos" runat="server"></asp:TextBox>
<asp:CompareValidator id="validadorDos"  runat="server"
ControlToValidate="valorDos" Operator="GreaterThan" ValueToCompare="10"
ErrorMessage="El valor dos debe ser mayor que 10" Display="Dynamic" Type="Integer">
*
</asp:CompareValidator>
<br>
<asp:Button ID="boton" runat="server" Text="Enviar datos"></asp:Button>
<br>
<asp:ValidationSummary ID="resumen"  runat="server"
HeaderText="Se han encontrado los siguientes errores en la página"
ShowSummary="True" DisplayMode="BulletList"/>
</form>
</body>
</html>
```

Código fuente 125

En la Figura 71 se puede ver un ejemplo de ejecución del código anterior.

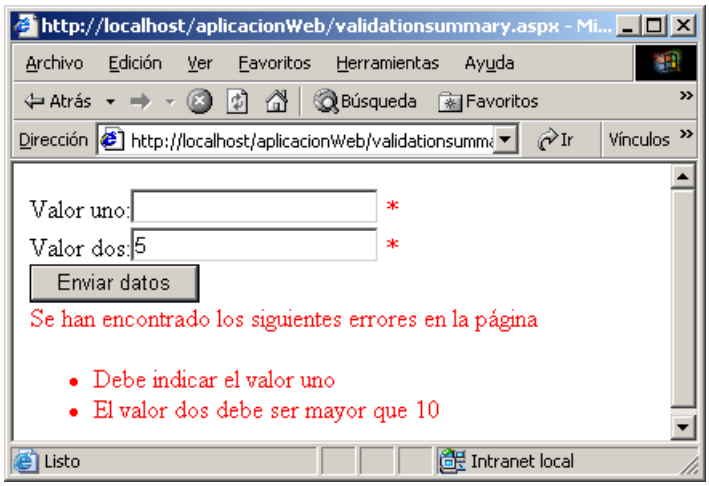


Figura 71

CustomValidator

Este último control Web de validación se utilizará para realizar una validación personalizada, este control llamará a una función definida por el usuario que será la que realice la validación. Esta función puede estar definida en el cliente o bien en el servidor.

Este control de validación ofrece las siguientes propiedades destacadas:

- ClientValidationFunction: en esta propiedad indicaremos el nombre de la función del lado del cliente que realizará la validación deseada, en este caso la función será una función definida en un lenguaje de script de cliente , como puede ser JavaScript, y que sea soportado por el navegador.

- OnServerValidate: en esta propiedad debemos indicar el nombre de la función de servidor que se va a encargar de realizar la validación correspondiente, esta función se ejecutará cuando se produzca el evento ServerValidate, es decir, esta propiedad va a ser el manejador del evento que se produce cuando se realiza la validación en el servidor.

Lo mejor es ofrecer las funciones de validación en el cliente y en el servidor, así nos aseguraremos que la validación siempre se realizará con independencia del tipo de navegador que ha cargado la página ASP .NET correspondiente.

En los dos casos la función de validación va a recibir dos parámetros, uno de la clase Object, que representa al control de la clase CustomValidator, y otro objeto de la clase ServerValidateEventArgs, que se va a utilizar para indicar si la validación ha sido correcta o no. El objeto de clase ServerValidateEventArgs posee la propiedad Value para obtener el valor del control que se desea validar y la propiedad IsValid, en la que se indicará el resultado de la validación.

En el Código fuente 126 se ofrece un objeto CustomValidator que realiza una validación con una función de servidor para comprobar si un el número indicado en el control TextField es par o no, la validación consiste en aceptar únicamente como válidos números pares.

```
<%@ Page language="c#"%>
<script language="C#" runat="server">
void EnviarFormulario(Object sender, EventArgs e) {
       if (Page.IsValid) {
              resultado.Text = "La página es válida";
       }else{
              resultado.Text = "La página NO es válida";
       }
}
void validaParServidor(object source, ServerValidateEventArgs args){
       try {
       int i = int.Parse(args.Value);
       args.IsValid = ((i%2) == 0);
       if (i%2==0){
              args.IsValid = true;
         }else{
              args.IsValid = false;
         }
     }catch{
        args.IsValid = false;
     }
}
</script>

<html>
```

```
<body>
<form id="formulario" method="post" runat="server">
Número par:<asp:TextBox Id="numero" runat="server"></asp:TextBox>

<asp:CustomValidator id="validador"  runat="server"
ControlToValidate="numero" ErrorMessage="Debe indicar un número par"
OnServerValidate="validaParServidor" Display="Dynamic">
</asp:CustomValidator>

<br>
<asp:Button ID="boton" runat="server" Text="Enviar datos"
OnClick="EnviarFormulario">
</asp:Button>
<br>
<asp:Label runat="server" ID="resultado"/>
</form>
</body>
</html>
```

Código fuente 126

Como se puede ver en el código anterior, se utiliza un bloque try{}catch para realizar la conversión a un númeor entero, esto se hace así para evitar errores a la hora de realizar la conversión, y aque el usuario puede introducir en el objeto TextField valores alfanuméricos.

El resultado de la ejecución de la página anterior se puede ver en la Figura 72.

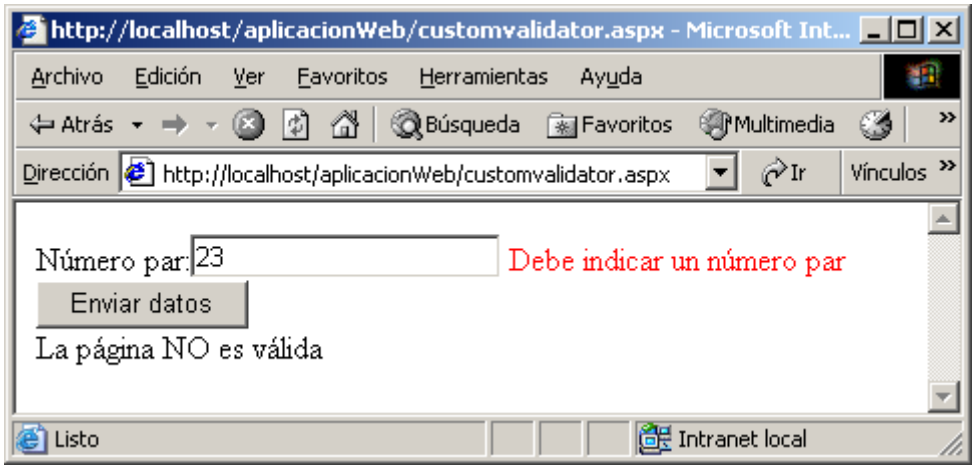


Figura 72

El Código fuente 127 es una versión distinta del ejemplo anterior, en este caso la función que se utiliza para realizar la validación es una función de cliente realizada en JavaScript.

```
<%@ Page language="c#"%>
<script language="javascript">
function validaParCliente(source, arguments){
	if(arguments.Value%2==0){
		arguments.IsValid=true;
	}else{
		arguments.IsValid=false;
	}
}
</script>
```

```
<html>
<body>
<form id="formulario" method="post" runat="server">
Número par:<asp:TextBox Id="numero" runat="server"></asp:TextBox>

<asp:CustomValidator id="validador"  runat="server"
ControlToValidate="numero" ErrorMessage="Debe indicar un número par"
ClientValidationFunction="validaParCliente"
Display="Dynamic">
</asp:CustomValidator>
<br>
<asp:Button ID="boton" runat="server" Text="Enviar datos">
</asp:Button>
</form>
</body>
</html>
```

Código fuente 127

Con este control de validación finaliza el presente capítulo. En el siguiente capítulo trataremos el último grupo de controles Web, los controles Web de lista que hacen uso de la característica de Data Binding o de conexión con datos, que también se comentará en dicho capítulo.

10

Web Forms: plantillas y Data Binding

Introducción

Como su nombre indica en este nuevo capítulo trataremos dos características que nos ofrece ASP .NET y que se encuentran dentro del entorno de los Web Forms, se trata de las plantillas (templates) y del mecanismo de conexión con datos o Data Binding, estas dos características están muy relacionadas con el último tipo de controles Web, que veremos en el siguiente capítulo y que introduciremos en el actual, se trata de los controles de lista. Comúnmente los controles de lista se utilizan en combinación con plantillas (definen su aspecto) y Data Binding (define el origen de los datos).

La funcionalidad de Data Binding no es algo nuevo dentro del mundo de la programación en general, ya la presentaba Visual Basic es versiones anteriores, pero si es nuevo para el mundo de la programación de páginas ASP. Mediante este mecanismo podemos asociar unos datos con un control del interfaz de usuario correspondiente, ASP .NET permite establecer el origen de los datos de los controles ASP .NET ofreciendo compatibilidad con todos los navegadores.

La idea del mecanismo de Data Binding es muy sencilla, se conectan controles Web a una fuente de datos y ASP .NET muestra la información de forma automática, es una forma muy sencilla de organizar la información en listas, cajas de texto, grids, etc, precisamente estos controles son los controles llamados controles Web de lista (list-bound controls).

Por otro lado el mecanismo de plantillas nos va a permitir definir el aspecto que van a mostrar los controles de lista, ya que, como veremos a continuación, estos controles no ofrecen un aspecto determinado, sino que debemos indicar el formato que van a tener los datos que contienen los controles.

Introducción a los controles de lista

Estos controles se encuentran especializados en mostrar listados de datos en las páginas ASP .NET, tarea que suele ser bastante común en la mayoría de las aplicaciones ASP .NET, gracias a estos controles, que como ya habíamos dicho, en capítulos anteriores, pertenecen al espacio de nombres System.Web.UI.WebControls, las tareas de relacionadas con los listados de información son mucho más sencillas, estos controles ASP .NET, en combinación con las plantillas y Data Binding, generan de forma automática el interfaz de usuario que muestra la información permitiendo la paginación, filtrado y otras operaciones sobre los contenidos correspondientes.

Como ya se comentó en la introducción, los controles de tipo lista se suelen utilizar en combinación con otra nueva característica ofrecida por el entorno de los Web Forms, se trata de Data Binding, es decir, la conexión de los datos con los controles en los que se van a mostrar, esta característica la podemos encontrar en versiones anteriores de Visual Basic.

Existen tres controles estándar dentro de esta familia: Repeater, DataList y DataGrid.

Relacionado con los controles de lista, más estrechamente con los controles Repeater y DataList, nos encontramos un interesante mecanismo que nos ofrece ASP .NET denominado plantillas (templates), que nos va a permitir personalizar el aspecto de la información que vamos a mostrar, el control es absoluto, existiendo para ello cinco tipos distintos de plantillas, que los comentaremos en detalle en el apartado correspondiente. Estas plantillas se utilizan en combinación con los controles Web DataList y Repeater.

A continuación se va a describir de forma muy breve cada uno de los controles de lista:

- Repeater: este control muestra elementos de datos en una lista, por sí sólo no genera una salida visible para el usuario, para que sea visible se debe utilizar junto con las plantillas. Es el más sencillo de los tres controles.

- DataList: similar al control anterior pero permite una mayor personalización a la hora de mostrar la información en forma de lista. También permite la selección y edición de elementos. También debe utilizarse junto con las plantillas.

- DataGrid: este control muestra la información en forma de tabla con celdas y columnas, generando el código HTML equivalente a una tabla. Es el más complejo de los tres controles de lista ofreciendo un alto grado de personalización y una serie de características muy interesantes como pueden ser la ordenación, paginación, definición de columnas, edición, etc.

Aunque los controles de lista los vamos a utilizar en algunos ejemplos en los siguientes apartados será en el siguiente capítulo cuando los veamos con detenimiento. En el siguiente apartado se va a comentar el mecanismo de plantillas del que hacen uso estos controles Web.

Plantillas (templates)

La mayoría de los controles Web tienen un aspecto estándar definido, pero los controles Repeater y DataList no ofrecen una salida sino que debemos especificar el aspecto que van a tener a través del mecanismo de templates.

En ASP .NET podemos encontrar una serie de plantillas (templates) que nos van a permitir definir la presentación y aspecto que van a tener los datos. Existen cinco plantillas que se pueden utilizar con los controles Repeater y DataList y son los siguientes:

- HeaderTemplate: representa la primera línea para los datos, es decir, es la cabecera de los datos. Un uso típico de esta plantilla es para indicar el inicio de una elemento contenedor de información, como puede ser una tabla.

- ItemTemplate: representa cada línea de los datos, es decir, representa cada elemento o registro. Dentro de esta plantilla se debe utilizar la sintaxis especial de Data Binding, que veremos más adelante. En el caso de una tabla sería cada una de sus filas.

- AlternatingItemTemplate: esta plantilla se aplica a elementos alternos, y tiene la misma funcionalidad que la anterior.

- SeparatorItemTemplate: esta plantilla permite separar cada uno de los elementos, es decir, cada una de las líneas de datos, pueden ser saltos de línea, líneas o indicadores de fin e inicio de columna de una fila de una tabla.

- FooterTemplate: es la última línea de salida, es decir, el pie de los datos. Normalmente suele contener el elemento que cierra el contenedor, es decir, cierra el elemento que se ha abierto previamente con la plantilla HeaderTemplate.

Las plantillas también se pueden utilizar con el control DataGrid, pero no directamente sino a través de sus columnas.

Para utilizar las plantillas únicamente debemos utilizar la etiqueta con el nombre de plantilla correspondiente, podemos utilizar cualquiera de los cinco anteriores. Entre las dos etiquetas de inicio y finalización del elemento <NombrePlantilla> debemos indicar el código HTML que queremos que se muestre en cada caso, para formar parte del aspecto de los datos.

A continuación vamos a ver un ejemplo de utilización de plantillas. En el Código fuente 128 se puede ver la utilización de cuatro de las cinco plantillas comentadas a través de un control de lista Repeater. En este caso al control Repeater se le ha asignado como origen de datos una tabla de una base de datos, el lector no tiene porque entender todavía este código, ya que se trata de acceso a datos mediante ADO .NET, que se verá en próximos capítulos.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
   SqlConnection conexion = new SqlConnection(
                                "server=angel;database=pubs;uid=sa;pwd=");
   SqlDataAdapter comando = new SqlDataAdapter("select * from Titles", conexion);
   DataSet ds = new DataSet();
   comando.Fill(ds, "Titles");
   repiteDatos.DataSource = ds.Tables["Titles"].DefaultView;
   repiteDatos.DataBind();
}
</script>

<body>

<asp:Repeater id="repiteDatos" runat="server">

      <HeaderTemplate>
        <table width="100%" style="font: 8pt verdana">
          <tr bgcolor="ff00d8">
            <th>Título</th>
```

```
            <th>Tipo</th>
            <th>Precio</th>
          </tr>
      </HeaderTemplate>

      <ItemTemplate>
       <tr bgcolor="FFECD8">
       <td><%# DataBinder.Eval(Container.DataItem, "title") %></td>
       <td><%# DataBinder.Eval(Container.DataItem, "type") %></td>
       <td align="right">
       <%# DataBinder.Eval(Container.DataItem, "price", "{0:f} €") %></td>
       </tr>
      </ItemTemplate>

      </SeparatorTemplate>
      <AlternatingItemTemplate>
       <tr style="background-color:FFEC00">
       <td><%# DataBinder.Eval(Container.DataItem, "title") %></td>
       <td><%# DataBinder.Eval(Container.DataItem, "type") %></td>
       <td align="right">
       <%# DataBinder.Eval(Container.DataItem, "price", "{0:f} €") %></td>
       </tr>
      </AlternatingItemTemplate>

      <FooterTemplate>
        </table>
      </FooterTemplate>

  </asp:Repeater>
</body>
</html>
```

Código fuente 128

Además de adelantar un fragmento de código que muestra el acceso a datos, mediante ADO .NET, también se ha adelantado en este ejemplo la utilización del mecanismo de Data Binding. La forma más sencilla de establecer el origen de los datos de un control que muestra un conjunto de datos (ListBox, Repeater, DataGrid, etc.) es utilizar la propiedad DataSource del control correspondiente para indicar la fuente de los datos y a continuación llamar al método DataBind() del control.

También es posible llamar al método DataBind() de la página ASP .NET, representada mediante el objeto Page, con lo que se establecen todos lo enlaces con los datos existentes en todos los controles de la página.

En la propiedad DataSource del control Web podemos especificar un amplio tipo de fuentes de datos: un array, un objeto DataView de ADO .NET (como vimos en el ejemplo anterior), una expresión, etc.

Una vez establecido el origen de datos, otro aspecto que debemos tener en cuenta es la sintaxis de la característica Data Binding, para indicar en qué lugar se mostrará cada uno de los distintos datos.

El ejemplo anterior no tiene desperdicio, ya que muestra también la sintaxis que se debe utilizar para poder hacer uso del mecanismo de Data Binding, mediante el carácter especial almohadilla (#) y el método Eval() de la clase DataBinder, todo esto lo veremos en mayor detalle en el apartado correspondiente.

En la Figura 73 se puede ver el aspecto que tendría el control Repeater en combinación con las plantillas.

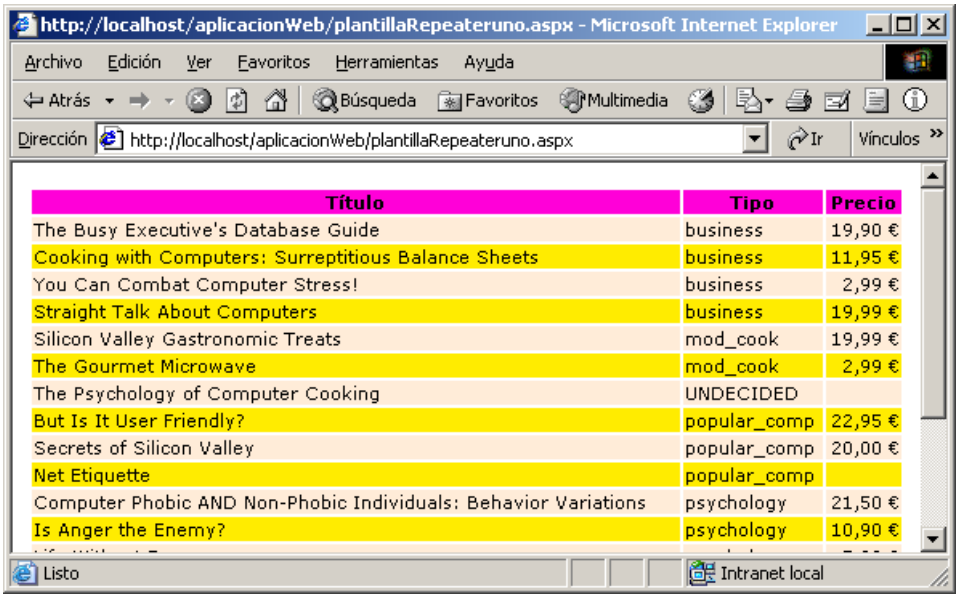


Figura 73

El Código fuente 129 ofrece un fragmento del código HTML generado por esta página ASP .NET.

```
<html>
<body>
 <table width="100%" style="font: 8pt verdana">
   <tr bgcolor="ff00d8">
      <th>Título</th>
      <th>Tipo</th>
      <th>Precio</th>
   </tr>

   <tr bgcolor="FFECD8">
       <td>The Busy Executive's Database Guide</td>
       <td>business    </td>
       <td align="right">19,90 €</td>
   </tr>

    <tr style="background-color:FFEC00">
       <td>Cooking with Computers: Surreptitious Balance Sheets</td>
       <td>business    </td>
       <td align="right">11,95 €</td>
    </tr>

       ..................................

    <tr style="background-color:FFEC00">
       <td>Sushi, Anyone?</td>
       <td>trad_cook   </td>
       <td align="right">14,99 €</td>
    </tr>

 </table>
</body>
</html>
```

Código fuente 129

Una vez realizados estos comentados sobre el código de ejemplo vamos a centrarnos en el aspecto que más nos interesa, que no es otro que la definición de plantillas. En el ejemplo se han utilizado cuatro plantillas (<template>) distintas con el control Repeater.

Se ha utilizado una plantilla de cabecera de la información (HeaderTemplate) para definir la cabecera de una tabla, otra de pie (FooterTemplate) para cerrar la definición de la tabla definida en la plantilla HeaderTemplate,. Otra plantilla más para definir el aspecto de cada elemento (ItemTemplate), en este caso se define como va a ser cada fila de la tabla y una platilla muy similar que define un aspecto para elementos alternativos (AlternatingItemTemplate), es decir, el aspecto de las filas alternativas de la tabla.

En el Código fuente 130 se puede observar la utilización de un objeto Repeater que utiliza el mecanismo de plantillas y de Data Binding de una manera distinta. En este caso el origen de los datos del control Repeater es un array de objetos String, y se ha utilizado una plantilla más, la plantilla que indica el separador de cada elemento (SeparatorTemplate).

```
<%@ Page Language="c#"%>
<script language="c#" runat="server">
void Page_Load(Object objFuente,EventArgs args){
      String[] datos=new String [5];
      datos[0]="Bora-bora";
      datos[1]="Morea";
      datos[2]="Pascua";
      datos[3]="Papeete";
      datos[4]="Santiago de Chile";
      lista.DataSource=datos;
      lista.DataBind();
}
</script>
<html>
<body>
<form id="DataBindingSencillo" method="post" runat="server">
<asp:Repeater id="lista" runat="server">
      <HeaderTemplate>
      <h1>Destinos</h1>
      </HeaderTemplate>
      <ItemTemplate>
            <font face="courier" size="2" color="blue">
                  <%#Container.DataItem%>
            </font>
      </ItemTemplate>
      <AlternatingItemTemplate>
            <font face="arial" size="2" color="green">
                  <%#Container.DataItem%>
            </font>
      </AlternatingItemTemplate>
      <SeparatorTemplate>
            <hr>
      </SeparatorTemplate>
      <FooterTemplate>
            <hr><br>
            <small><i>
            <a href="mailto:reservas@correo.es">reservas@correo.es</a>
            </i></small>
      </FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Código fuente 130

En este nuevo ejemplo se ha utilizado: una plantilla HeaderTemplate para indicar el título de los elementos que se van a mostrar; una plantilla ItemTemplate que define el tipo de letra con el que va a aparecer cada elemento, debido a que el origen de datos es más sencillo (se trata de un array de cadenas) se ha utilizado una sintaxis de Data Binding más sencilla que la anterior para hacer referencia a los elementos; también aparece una platilla AlternatingItemTemplate que cambia el color y el tipo de letra; una plantilla SeparatorTemplate que muestra una línea de separación entre cada uno de los elementos; y por último una plantilla FooterTemplate que muestra una dirección de correo de contacto.

En el Código fuente 131 se puede ver el código HTML que se genera al ejecutar esta página ASP .NET.

```
<html>
<body>
<form name="DataBindingSencillo" method="post" action="plantillarepeaterdos.aspx"
id="DataBindingSencillo">
<input type="hidden" name="__VIEWSTATE"
value="dDwyMDA1MDYwODM7dDw7bDxpPDE+Oz47bDx0PDtsPGk8MT47PjtsPHQ8cDxsPF8hSXRlbUNvdW50
Oz47bDxpPDU+Oz4+O2w8aTwxPjtpPDM+O2k8NT47aTw3PjtpPDk+Oz47bDx0PDtsPGk8MD47PjtsPHQ8QDx
Cb3JhLWJvcmE7Pjs7Pjs+Pjt0PDtsPGk8MD47PjtsPHQ8QDxNb3JlYTs+Ozs+Oz4+O3Q8O2w8aTwwPjs+O2
w8dDxAPFBhc2N1YTs+Ozs+Oz4+O3Q8O2w8aTwwPjs+O2w8dDxAPFBhcGVldGU7Pjs7Pjs+Pjt0PDtsPGk8M
D47PjtsPHQ8QDxTYW50aWFnbyBkZSBDaGlsZTs+Ozs+Oz4+Oz4+Oz4+Oz4=" />
<h1>Destinos</h1>
<font face="courier" size="2" color="blue">
      Bora-bora
</font>
<hr>
<font face="arial" size="2" color="green">
      Morea
</font>
<hr>
<font face="courier" size="2" color="blue">
      Pascua
</font>
<hr>
<font face="arial" size="2" color="green">
      Papeete
</font>
<hr>
<font face="courier" size="2" color="blue">
      Santiago de Chile
</font>
<hr><br>
<small><i>
<a href="mailto:reservas@correo.es">reservas@correo.es</a>
</i></small>
</form>
</body>
</html>
```

Código fuente 131

En la Figura 74 se puede observar el aspecto que ofrece este control Repeater al utilizar las plantillas comentadas.

Una vez que ya hemos visto la utilización de plantillas vamos a tratar una característica que se encuentra muy relacionada con las mismas y que a su vez está muy relacionada con los controles de lista, no es otra que la característica de Data Binding o de conexión con los datos que ofrece la tecnología ASP .NET.

Figura 74

Data Binding

Ya hemos adelantado a lo largo del capítulo algunos de los aspectos del mecanismo de Data Binding que ofrece ASP .NET, pero en este apartado lo vamos a ver con un mayor detenimiento y también se ofrecerán varios ejemplos ilustrativos.

El mecanismo de Data Binding (conexión con los datos o enlace con los datos) es una nueva facilidad que ofrece ASP .NET, pero que debe ser familiar para los lectores que conozcan el entorno de desarrollo de Visual Basic. Mediante este mecanismo podemos asociar unos datos con un control del interfaz de usuario correspondiente definido dentro de un Web Form.

ASP .NET permite establecer el origen de los datos de los controles ASP .NET ofreciendo compatibilidad con todos los navegadores.

La idea del mecanismo de Data Binding es muy sencilla, se conectan controles Web a una fuente de datos y ASP .NET muestra la información de forma automática, es una forma muy sencilla de organizar la información en listas, cajas de texto, grids, párrafos, etc.

La forma más sencilla de establecer el origen de los datos de un control que muestra un conjunto de datos (ListBox, Repeater, DataGrid, etc.) es utilizar la propiedad DataSource del control correspondiente para indicar la fuente de los datos y a continuación llamar al método DataBind() del control. También es posible llamar al método DataBind() de la página ASP.NET, con lo que se establecen todos lo enlaces con los datos existentes en todos los controles de la página.

En la propiedad DataSource del control Web podemos especificar un amplio tipo de fuentes de datos: un array, un objeto DataView de ADO .NET (como vimos en el apartado anterior), una expresión, un array, el resultado de un método, una propiedad, etc.

La sintaxis general para las expresiones de Data Binding se puede observar en el Código fuente 132, dónde se ofrecen dos escenarios, la asignación de una expresión Data Binding a una propiedad de un control Web y también un segundo escenario en el que se muestra la expresión de Data Binding en cualquier lugar de la página ASP .NET que deseemos.

```
<asp:ControlWeb propiedad="<%# expresión Data Bindging %>" runat="server" />
Texto literal en la página: <%# expresión Data Binding %>
```

Código fuente 132

Los lectores familiarizados con versiones anteriores de ASP, habrán comprobado que la sintaxis del mecanismo de Data Binding es muy similar a la sintaxis abreviada <%= %>, que se correspondía en ASP (y sigue correspondiendo a lo mismo en ASP .NET) a la instrucción Response.Write.

Sin embargo el comportamiento de la nueva sintaxis <%# %> es bastante diferente, mientras que la sintaxis abreviado de la instrucción Response.Write se evaluaba mientras la página era procesada, la sintaxis de Data Binding de ASP .NET se evalúa sólo cuando el método DataBing() correspondiente es invocado.

ASP .NET ofrece una sintaxis muy flexible para establecer el mecanismo de Data Binding mediante los delimitadores <%# %>, a continuación se ofrece un esbozo de una serie de ejemplos que demuestran la utilización de esta sintaxis en distintas situaciones:

- Se utiliza una propiedad como origen de datos: Cliente <%# IDCliente %>.
- Una colección: <asp:ListBox id="lista" datasource="<%# miArray %>" runat="server">
- Una expresión: Contacto: <%# cliente.Nombre+" "+cliente.Apellidos %>
- El resultado de un método: Código: <%# getCodigo() %>

Todas las expresiones de Data Binding, con independencia del lugar en el que se ubiquen, deben estar contenidas entre los caracteres delimitadores <%# %>.

El Código fuente 133 nos muestra el mecanismo de conexión con datos (Data Binding) en plena acción, se muestran varios ejemplos de conexión con datos para distintos controles Web dentro de una página ASP .NET.

```
<%@ Page Language="C#"%>
<html>
<script runat="server">
String[] datosDos=new String [5];
ArrayList datos=new ArrayList();
void Page_Load(Object objFuente, EventArgs args){
      datos.Add("Uno");
      datos.Add("Dos");
      datos.Add("Tres");
      datos.Add("Cuatro");
      datos.Add("Cinco");
      datosDos[0]="Bora-bora";
      datosDos[1]="Morea";
      datosDos[2]="Pascua";
      datosDos[3]="Papeete";
      datosDos[4]="Santiago de Chile";
```

```
        listaDos.DataBind();
        lista.DataBind();
        hora.DataBind();
        caja.DataBind();
}
</script>
<body>
<form id="EjemplosDataBinding" method="post" runat="server">
Números:
<asp:DropDownList id="lista" runat="server" DataSource="<%# datos %>">
</asp:DropDownList><br>
Destinos:
<asp:ListBox id="listaDos" runat="server" DataSource="<%# datosDos %>">
</asp:ListBox><br>
<asp:Label ID="hora" runat="server" Text="<%# DateTime.Now %>"/><br>
Navegador:
<asp:TextBox ID="caja" runat="server" Text="<%# Page.Request.Browser.Browser %>">
</asp:TextBox>
</form>
</body></html>
```

Código fuente 133

En este ejemplo se han establecido como orígenes de datos de los controles DropDownList y ListBox un objeto ArrayList y un arrary de objetos String, respectivamente, para el objeto Label se le ha asignado a su propiedad Text el valor devuelto por la propiedad Now, y para el control TextBox se ha seguido el mismo criterio pero utilizando el objeto Request, que es uno de los objetos integrados en ASP .NET y que veremos en el capítulo correspondiente y que será familiar para aquellos lectores que ya conozcan el entorno de ASP.

El resultado de la ejecución de este ejemplo de Data Binding se puede ver en la Figura 75.

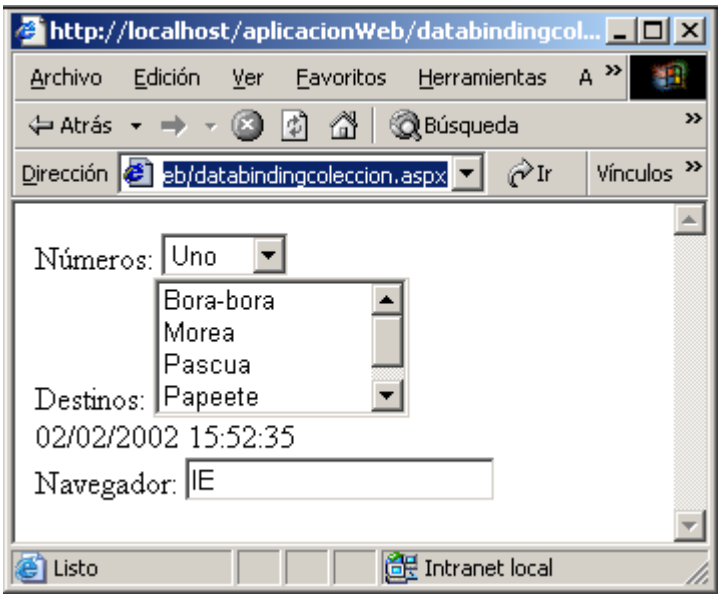


Figura 75

Y se generaría el código HTML que ofrece el Código fuente 134.

```
<html>

<body>
<form name="EjemplosDataBinding" method="post" action="databindingcoleccion.aspx"
id="EjemplosDataBinding">
<input type="hidden" name="__VIEWSTATE"
value="dDwxOTg4MDI4Nzk2O3Q8O2w8aTwyPjs+O2w8dDw7bDxpPDE+O2k8Mz47aTw1Pjs+O2w8dDx0PDt0
PGk8NT47QDxVbm87RG9zO1RyZXM7Q3VhdHJvO0NpbmNvOz47QDxVbm87RG9zO1RyZXM7Q3VhdHJvO0NpbmN
vOz4+Oz47Oz47dDx0PDt0PGk8NT47QDxCb3JhLWJvcmE7TW9yZWE7UGFzY3VhO1BhcGVldGU7U2FudGlhZ2
8gZGUgQ2hpbGU7PjtAPEJvcmEtYm9yYTtNb3JlYTtQYXNjdWE7UGFwZWV0ZTtTYW50aWFnbyBkZSBDaGlsZ
Ts+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDs+O2w8MDIvMDIvMjAwMiAxNTo1MjozNTs+Pjs+Ozs+Oz4+Oz4+Oz4=
" />

Números:
<select name="lista" id="lista">
	<option value="Uno">Uno</option>
	<option value="Dos">Dos</option>
	<option value="Tres">Tres</option>
	<option value="Cuatro">Cuatro</option>
	<option value="Cinco">Cinco</option>

</select><br>
Destinos:
<select name="listaDos" id="listaDos" size="4">
	<option value="Bora-bora">Bora-bora</option>
	<option value="Morea">Morea</option>
	<option value="Pascua">Pascua</option>
	<option value="Papeete">Papeete</option>
	<option value="Santiago de Chile">Santiago de Chile</option>

</select><br>
<span id="hora">02/02/2002 15:52:35</span><br>
Navegador:
<input name="caja" type="text" value="IE" id="caja" />
</form>
</body></html>
```

Código fuente 134

Podemos modificar el ejemplo anterior para establecer la conexión con los datos en el momento en el que pulsemos un control Button de la página, así el mecanismo de Data Binding no está establecido desde el principio al cargar la página, y además en este caso el método DataBind() no se va a lanzar sobre cada control, sino que se va a lanzar sobre el objeto Page que representa a la página ASP .NET actual. El resultado de estas modificaciones da lugar al ejemplo del Código fuente 135.

```
<%@ Page Language="C#"%>
<html>
<script runat="server">
String[] datosDos=new String [5];
ArrayList datos=new ArrayList();
void Page_Load(Object objFuente, EventArgs args){
	datos.Add("Uno");
	datos.Add("Dos");
	datos.Add("Tres");
	datos.Add("Cuatro");
	datos.Add("Cinco");
	datosDos[0]="Bora-bora";
	datosDos[1]="Morea";
	datosDos[2]="Pascua";
	datosDos[3]="Papeete";
	datosDos[4]="Santiago de Chile";
```

```
}

void EstableceDataBinding(Object objFuente, EventArgs args){
      Page.DataBind();
      //o también:
      //DataBind();
}
</script>
<body>
<form id="EjemplosDataBinding" method="post" runat="server">
Números:
<asp:DropDownList id="lista" runat="server" DataSource="<%# datos %>">
</asp:DropDownList><br>
Destinos:
<asp:ListBox id="listaDos" runat="server" DataSource="<%# datosDos %>">
</asp:ListBox><br>
<asp:Label ID="hora" runat="server" Text="<%# DateTime.Now %>"/><br>
Navegador:
<asp:TextBox ID="caja" runat="server" Text="<%# Page.Request.Browser.Browser %>">
</asp:TextBox><br>
<asp:Button ID="boton" Runat="server" Text="Establecer Data Binding"
OnClick="EstableceDataBinding"></asp:Button>
</form>
</body></html>
```

Código fuente 135

De esta forma al cargar la página ASP .NET y no pulsar el botón, tendría el aspecto de la Figura 76, es decir, todos los controles aparecerían sin sus datos correspondientes hasta que no se pulse el botón.

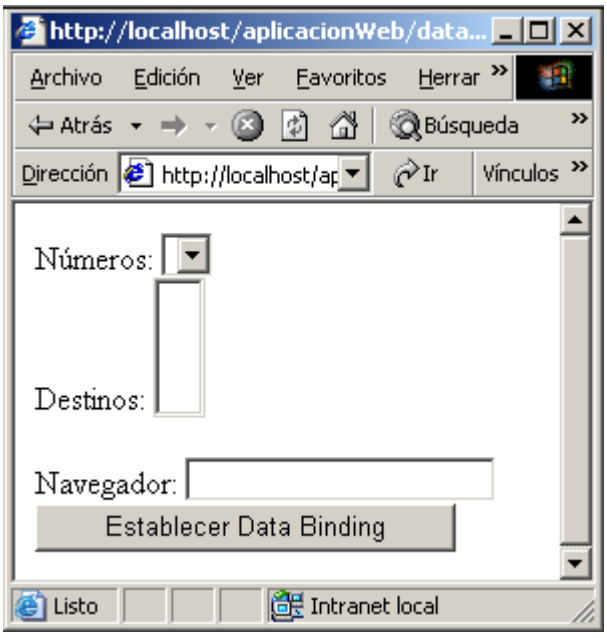


Figura 76

En los siguientes subapartados vamos a mostrar a través de ejemplos algunas de las distintas posibilidades que nos ofrece la conexión con datos en distintos casos y escenarios.

Estableciendo como origen de datos propiedades

Mediante ASP .NET podemos utilizar como origen de los datos en el mecanismo de Data Binding propiedades de la página (objeto Page) o bien propiedades de otros controles Web.

En el Código fuente 136 se muestra como se establece en una conexión de datos, que posee como origen una propiedad de sólo lectura de la página ASP .NET, y que se conecta con la propiedad Text de un control Web de la clase Label. En este caso se han definido dos propiedades para la página que se mostrarán en dos objetos Label.

```
<%@ Page Language="C#"%>
<html>
<head>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
       etiquetaUno.DataBind();
       etiquetaDos.DataBind();
}

string nombrePagina{
  get {
       return "Mi Página";
  }
}

int numeroPagina{
  get {
    return 11;
  }
}
</script>

</head>
<body>
<form runat="server" ID="WebForm">
Nombre página:
   <asp:Label Runat="server" ID="etiquetaUno" Text="<%# nombrePagina %>">
   </asp:Label><br>
Número de página:
   <asp:Label Runat="server" ID="etiquetaDos" Text="<%# numeroPagina %>">
   </asp:Label><br>
</form>
</body>
</html>
```

Código fuente 136

Como se puede comprobar la sintaxis de Data Binding en este caso es muy sencilla, únicamente debemos utilizar los delimitadores <%# %> y acceder a la propiedad deseada.

El resultado de la ejecución de la página anterior se puede observar en la Figura 77.

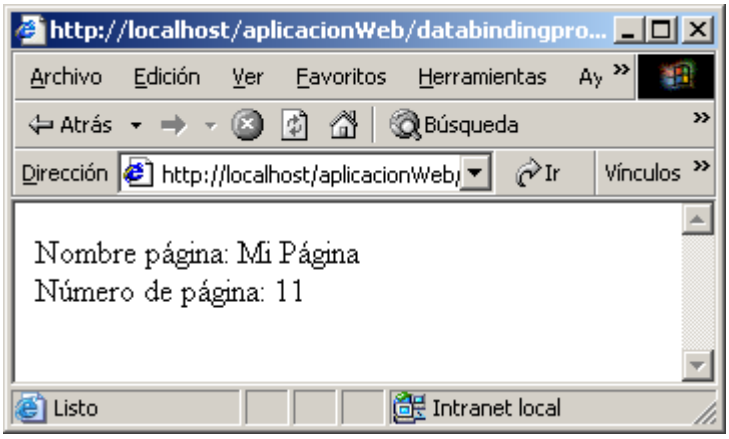


Figura 77

Y se genera el código HTML que se puede ver en el Código fuente 137.

```
<html>
<head>

</head>
<body>
<form name="WebForm" method="post" action="databindingpropiedad.aspx" id="WebForm">
<input type="hidden" name="__VIEWSTATE"
value="dDwxNDI0NDUwMTY1O3Q8O2w8aTwyPjs+O2w8dDw7bDxpPDE+O2k8Mz47PjtsPHQ8cDxwPGw8VGV4
dDs+O2w8TWkgUMOhZ2luYTs+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDs+O2w8MTE7Pj47Pjs7Pjs+Pjs+Pjs+"
/>

Nombre página:
   <span id="etiquetaUno">Mi Página</span><br>
Número de página:
   <span id="etiquetaDos">11</span><br>
</form>
</body>
</html>
```

Código fuente 137

Si queremos utilizar el mecanismo directamente sobre la página ASP .NET, sin necesidad de utilizar controles Label utilizaremos el Código fuente 138.

```
<%@ Page Language="C#"%>
<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
      Page.DataBind();
}

string nombrePagina{
  get {
      return "Mi Página";
  }
}

int numeroPagina{
  get {
```

```
    return 11;
  }
}
</script>
<body>
Nombre página: <%# nombrePagina %><br>
Número de página: <%# numeroPagina %><br>
</body>
</html>
```

Código fuente 138

En el Código fuente 139 se puede ver otro ejemplo de Data Binding con propiedades, en este nuevo ejemplo la propiedad que funciona como origen de datos es una propiedad de otro control Web en la misma página ASP .NET. La propiedad de origen en este caso es la propiedad Text de la propiedad SelectedItem del control DropDownList, de esta forma el objeto Label mostrará el valor del elemento seleccionado de la lista.

```
<html>
<script language="C#" runat="server">

void Page_Load(Object Sender, EventArgs E) {
      if (!Page.IsPostBack) {
            ArrayList valores = new ArrayList();
            valores.Add ("Agapornis");
            valores.Add ("Ninfa");
            valores.Add ("Iguana");
            valores.Add ("Varano");
            valores.Add ("Tortuga");
            lista.DataSource=valores;
            lista.DataBind();
    }
}

void BotonPulsado(Object sender, EventArgs e) {
      etiqueta.DataBind();
}
</script>
<body>
<form runat=server ID="Formulario">
      <asp:DropDownList id="lista" runat="server" />
      <asp:button Text="Enviar" OnClick="BotonPulsado" runat=server
ID="boton"/><br>
    <asp:Label id="etiqueta" font-name="Verdana" font-size="10pt" runat="server"
    Text="<%# lista.SelectedItem.Text %>"/>
</form>
</body>
</html>
```

Código fuente 139

En la Figura 78 se puede ver un ejemplo de ejecución del código anterior.

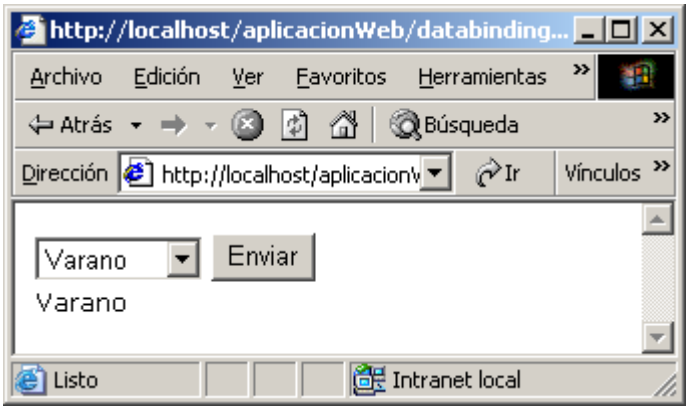


Figura 78

Estableciendo como origen de datos colecciones y listas

Normalmente este tipo de DataBinding se suele aplicar a controles Web como puede ser controles DropDownList, DataList, DataGrid, ListBox, etc.

En el Código fuente 140 se muestra un ejemplo de cómo se establece un objeto ArrayList como origen de datos de un objeto DropDownList. Al seleccionar un elemento de la lista se muestra dentro de un control Web de la clase Label. Como se puede comprobar la sintaxis de Data Binding sigue siendo muy sencilla, indicamos el nombre del objeto ArrayList entre los ya conocidos delimitadores <%# %>.

```
<%@ Page Language="C#"%>
<html>
<script runat="server">
String[] datosDos=new String [5];
ArrayList datos=new ArrayList();
void Page_Load(Object objFuente, EventArgs args){
      datos.Add("Uno");
      datos.Add("Dos");
      datos.Add("Tres");
      datos.Add("Cuatro");
      datos.Add("Cinco");
      DataBind();
}
</script>
<body>
<form id="EjemplosDataBinding" method="post" runat="server">
Números:
<asp:DropDownList id="lista" runat="server" DataSource="<%# datos %>">
</asp:DropDownList>
</form>
</body></html>
```

Código fuente 140

En este caso se utiliza la propiedad DataSource del objeto DropDownList para indicar el origen de los datos.

En la Figura 79 se puede ver un ejemplo de ejecución de la página ASP .NET anterior.

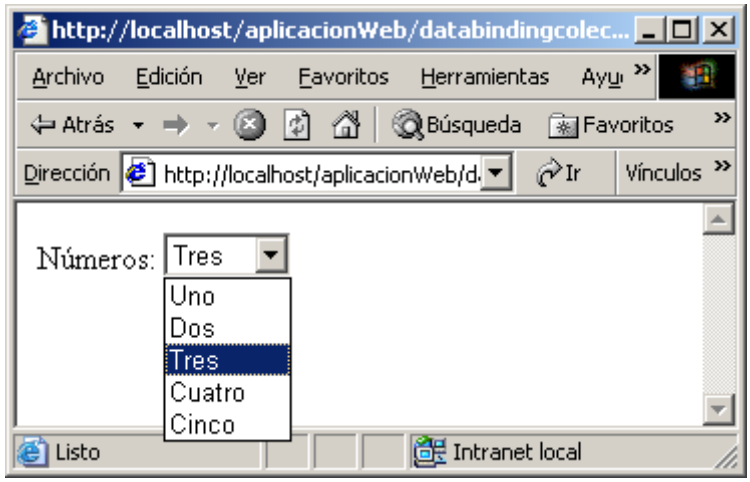


Figura 79

Y el código HTML que se generaría sería similar al del Código fuente 141.

```
<html>

<body>
<form name="EjemplosDataBinding" method="post" action="databindingcoleccion.aspx"
id="EjemplosDataBinding">
<input type="hidden" name="__VIEWSTATE"
value="dDwxNTc0NDE0MzM7dDw7bDxpPDI+Oz47bDx0PDtsPGk8MT47PjtsPHQ8dDw7dDxpPDU+O0A8VW5v
O0RvcztUcmVzO0N1YXRybztDaW5jbzs+O0A8VW5vO0RvcztUcmVzO0N1YXRybztDaW5jbzs+Pjs+Ozs+Oz4
+Oz4+Oz4=" />

Números:
<select name="lista" id="lista">
      <option value="Uno">Uno</option>
      <option value="Dos">Dos</option>
      <option value="Tres">Tres</option>
      <option value="Cuatro">Cuatro</option>
      <option value="Cinco">Cinco</option>

</select>
</form>
</body></html>
```

Código fuente 141

Hay ocasiones en que el origen de los datos no presenta una estructura de datos sencilla, sino que tiene un estructura similar a una tabla de una base de datos, poseyendo el origen de datos distintos campos con información. En estos casos es necesario mapear los campos del origen de los datos a las propiedades del control Web correspondiente sobre el que se va a establecer el Data Binding.

Un caso típico de este origen de datos es un objeto de la clase HashTable, que contiene un conjunto de valores identificados por una clave, es decir, por cada elemento de datos tenemos dos valores, representados mediante los pares clave (Key) y valor (Value).

Estas estructuras de datos complejas de utilizan como orígenes de datos de controles Web de lista, es decir, con controles Repeater, DataList y DataGrid, que se combinarán a su vez con el mecanismo de platillas para dar un aspecto determinado a la información que van a contener.

En el Código fuente 142 se muestra un ejemplo de Data Binding que utiliza como origen de datos un objeto HashTable y este origen de datos se conecta con un control DataList.

```
<%@ Page language="c#"%>
<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
       Hashtable h = new Hashtable();
       h.Add ("clave1", "valor1");
       h.Add ("clave2", "valor2");
       h.Add ("clave3", "valor3");
       lista.DataSource = h;
       lista.DataBind();
}

</script>
<body>

<form runat="server" ID="formulario">
       <asp:DataList id="lista" runat="server"
          BorderColor="black"
          BorderWidth="1"
          GridLines="Both"
          CellPadding="4"
          CellSpacing="0"
          >

            <ItemTemplate>
                <%# ((DictionaryEntry)Container.DataItem).Key %> :
                <%# ((DictionaryEntry)Container.DataItem).Value %>
            </ItemTemplate>

      </asp:DataList>

</form>
</body>
</html>
```

Código fuente 142

Como se puede apreciar entre los delimitadores <%# %> se utiliza la instrucción Container.DataItem. Esta expresión nos permite indicar una columna determinada dentro de el origen de datos, para establecer el mecanismo de Data Binding con el control Web correspondiente, en este caso se trata de indicar que en la primera columna de la tabla representada por la plantilla (template) aplicada sobre el control DataList, se van a mostrar las claves de el objeto HashTable, y en la segunda columna los valores del objeto HashTable.

Container hace referencia al contenedor de los datos (objeto DataList, Repeater o DataGrid), y la propiedad DataItem devuelve un objeto de la clase Object que hace referencia a la fila dentro de la estructura del origen de datos.

Para obtener el valor de la clave se utiliza la propiedad Key, y para obtener el valor del elemento del objeto HashTable se utiliza la propiedad Value. Para poder utilizar estas propiedades, como se puede apreciar en el Código fuente 142, se debe hacer la conversión (casting) de tipos correspondientes, ya que la propiedad DataItem siempre nos devolverá un objeto de la clase genérica Object.

En este ejemplo se ha utilizado por código, en el evento de carga de la página, la propiedad DataSource del objeto DataList para indicar el origen de datos, y la sintaxis de Data Binding se ha utilizado para indicar en que lugar se van a mostrar los distintos campos del origen de datos.

El resultado de la ejecución del código del ejemplo anterior se puede ver en la Figura 80.

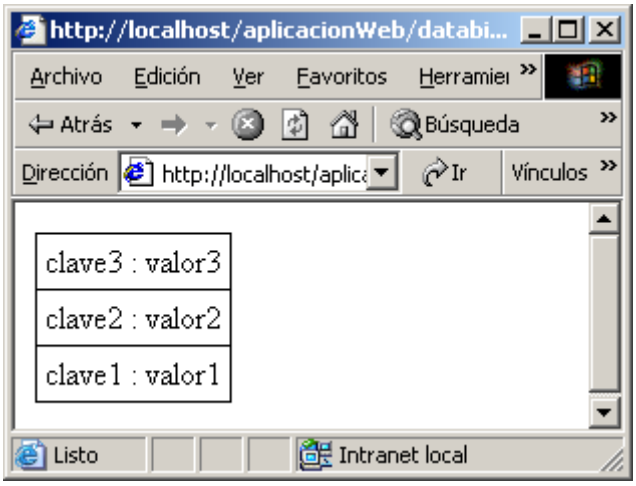


Figura 80

Y el Código fuente 143 es el código HTML que se generaría.

```
<html>
<body>
<form name="formulario" method="post" action="databindingcontaineruno.aspx"
id="formulario">
<input type="hidden" name="__VIEWSTATE"
value="dDwyNDkyNDgzNDI7dDw7bDxpPDI+Oz47bDx0PDtsPGk8MT47PjtsPHQ8QDA8cDxwPGw8RGF0YUtl
eXM7XyFJdGVtQ291bnQ7PjtsPGw8PjtpPDM+Oz4+Oz47Oz57Oz57Oz47bDxpPDA+O2k8MT47aTwyPjs+O2w
8dDw7bDxpPDA+Oz47bDx0PEA8Y2xhdmUzO3ZhbG9yMzs+Ozs+Oz4+O3Q8O2w8aTwwPjs+O2w8dDxAPGNsYX
ZlMjt2YWxvcjI7Pjs7Pjs+Pjt0PDtsPGk8MD47PjtsPHQ8QDxjbGF2ZTE7dmFsb3IxOz47Oz47Pj47Pj47P
j47Pj47Pg==" />

      <table id="lista" cellspacing="0" cellpadding="4" rules="all"
bordercolor="Black" border="1" style="border-color:Black;border-width:1px;border-
style:solid;border-collapse:collapse;">
      <tr>
            <td>

               clave3 :
               valor3

            </td>
      </tr><tr>
            <td>

               clave2 :
               valor2

            </td>
      </tr><tr>
            <td>

               clave1 :
               valor1
```

```
            </td>
      </tr>
</table>

</form>
</body>
</html>
```

Código fuente 143

Para aclarar más las cosas se ofrece el Código fuente 144, en este caso el origen de datos es un objeto de la clase DataView que representa una estructura de datos como la de una tabla de una base se datos con sus filas y columnas (este objeto lo veremos en mayor profundidad cuando en siguientes capítulos se trate el acceso a datos mediante ADO .NET). El control Web encargado de mostrar los datos es el ya cada vez más conocido por todos control Repeater.

```
<%@ Page language="c#"%>
<%@ Import namespace="System.Data" %>
<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e ) {
      DataTable dt = new DataTable();
      DataRow dr;
      dt.Columns.Add(new DataColumn("Entero", typeof(Int32)));
      dt.Columns.Add(new DataColumn("Cadena", typeof(string)));
      dt.Columns.Add(new DataColumn("Booleano", typeof(bool)));
      for (int i = 1; i <= 9; i++) {
            dr = dt.NewRow();
            dr[0] = i;
            dr[1] = "Cadena " + i.ToString();
            dr[2] = (i % 2 != 0) ? true : false;
            dt.Rows.Add(dr);
      }
      datos.DataSource = new DataView(dt);
      datos.DataBind();
}

</script>

<body>

<form runat="server" ID="formulario">
      <asp:Repeater id="datos" runat="server">
      <HeaderTemplate>
        <table width="100%" style="font: 8pt verdana">
          <tr bgcolor="ff00d8">
            <th>Entero</th>
            <th>Cadena</th>
            <th>Booleano</th>
          </tr>
      </HeaderTemplate>

      <ItemTemplate>
       <tr bgcolor="FFECD8">
       <td><%# ((DataRowView)Container.DataItem)[0]%></td>
       <td><%# ((DataRowView)Container.DataItem)[1]%></td>
       <td><%# ((DataRowView)Container.DataItem)[2]%></td>
       </tr>
      </ItemTemplate>

      <FooterTemplate>
```

```
        </table>
      </FooterTemplate>

  </asp:Repeater>
</form>
</body>
</html>
```

Código fuente 144

En este caso también hay que utilizar la expresión Container.DataItem para indicar en que posición deseamos mostrar cada columna del objeto DataView, y también se debe realizar la conversión de tipos correspondiente para poder hacer referencia a las columnas de manera correcta.

En la Figura 81 se puede ver el aspecto que tendría esta página ASP .NET.

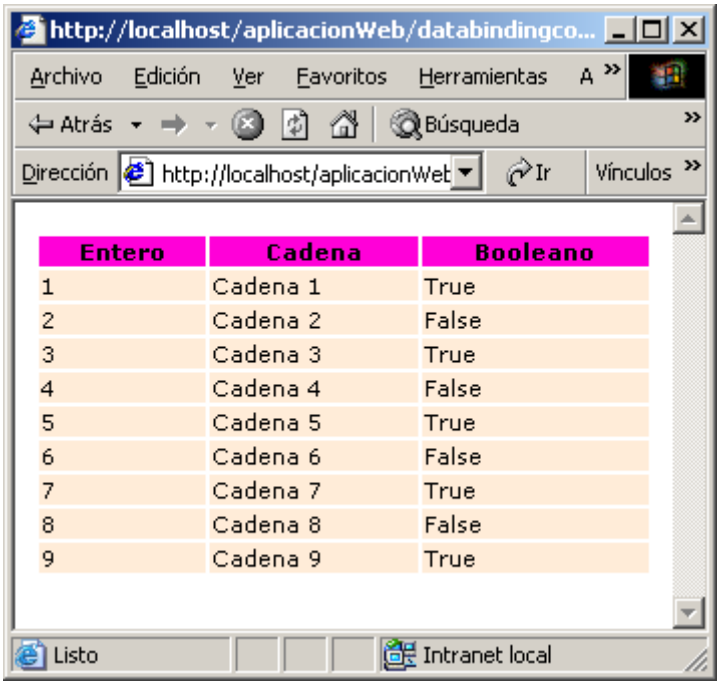


Figura 81

En el apartado dedicado al control Web Repeater veremos más ejemplos en los que el origen de datos es una colección.

Estableciendo como origen de datos expresiones y métodos

En este nuevo ejemplo (Código fuente 145) el origen de los datos va a ser el resultado ofrecido por un método definido dentro de la página ASP .NET y llamado ParOImpar(), que devolverá las cadenas "Par" o "Impar" según el valor del entero que le pasemos por parámetro. Este método se ha conectado con la propiedad Text de un control Label.

Al pulsar sobre el botón (control Web de la clase Button) se pasará al método ParOImpar() el valor indicado en el control TextBox.

```
<%@ Page Language="C#"%>
<html>
<script runat="server">
void BotonPulsado(Object Sender, EventArgs E) {
      DataBind();
}
String ParOImpar(String valor){
      try {
            int i= int.Parse(valor);
            if ((i % 2) == 0)
                  return "Par";
            else
                  return "Impar";
      }catch{
            return "Se ha producido un error";
      }
}
</script>
<body>
<form id="EjemplosDataBinding" method="post" runat="server">
Número:
<asp:TextBox Runat="server" ID="texto"></asp:TextBox>
<asp:Button Runat="server" ID="boton" OnClick="BotonPulsado"
Text="Pulsar"></asp:Button><br>
<asp:Label Runat="server" ID="etiqueta"
 Text="<%# ParOImpar(texto.Text) %>"></asp:Label>
</form>
</body>
</html>
```

Código fuente 145

El resultado de la ejecución de esta nueva página ASP .NET se puede ver en la Figura 82.

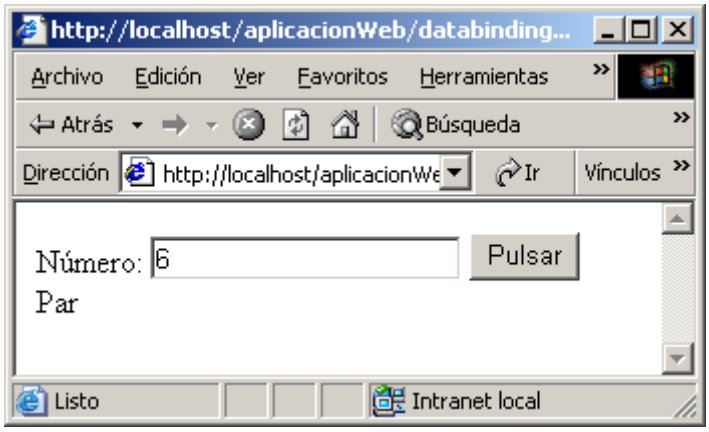


Figura 82

El método DataBinder.Eval()

Aunque este apartado no indica un origen de datos en concreto como sucedía en los anteriores apartados, está estrechamente relacionado con la sintaxis del mecanismo de Data Binding.

El método DataBinder.Eval() es un método estático que nos ofrece ASP .NET para realizar conversiones de tipos a la hora de establecer el mecanismo de Data Binding y también nos permite dar un formato determinado a la información.

Este método es utilizado cuando tenemos que hacer referencia a un valor dentro de un origen de datos y además deseamos darle un formato determinado. Este origen de datos va a contener más de un valor por cada fila, es decir, sería similar a una tabla de una base de datos con sus diferentes campos por cada registro.

Un ejemplo de origen de datos que puede utilizar el método estático DataBinder.Eval() es un objeto de la clase DataView (estos objetos los veremos más adelante en el texto cuando tratemos ADO .NET), que posee una estructura de tabla con filas (registros) y columnas (campos). Este método se utiliza normalmente en combinación con la instrucción Container.DataItem, que ya vimos en un subapartado anterior.

El método DataBinder.Eval() posee tres parámetros: el primer parámetro es un objeto de la clase Object que va a representar el contenedor de los datos, en el caso de los controles Repeater, DataList y DataGrid el contenedor de los datos siempre será la expresión Container.DataItem; el segundo parámetro es un objeto de la clase String que va a hacer referencia al campo de los datos y el tercero es otro objeto String que va a indicar el formato con el que se va a mostrar el campo correspondiente. Para ver en profundidad las cadenas que especifican el formato de la salida de los datos recomiendo al lector que consulte la documentación de referencia ofrecida en Microsoft .NET Framework SDK, aunque más adelante se muestran algunas de las cadenas de formato más comunes.

El método DataBinder.Eval() está sobrecargado y la otra versión del método elimina el tercer parámetro, es decir, no es necesario especificar el formato si no se desea.

Si estamos estableciendo el mecanismo de Data Binding sobre la propia página ASP .NET el primer parámetro del método Eval(), es decir, el que indica el contenedor, será en este caso Page.

La cadena que especifica el formato de salida debe tener los corchetes {} y entre ellos una expresión que define el formato correspondiente. A continuación vamos a pasar a definir esta expresión.

Entre los corchetes en primer lugar siempre debe aparecer el valor cero (0) que hace referencia a modo de variable al valor actual que contiene el campo de datos. A continuación aparecen dos puntos (:) seguidos de los caracteres especiales de formato de cadenas. Fuera de los corchetes podemos indicar cualquier cadena, que será tomada como un literal.

En la Tabla 8 se ofrecen los distintos caracteres especiales que podemos utilizar para formatear valores numéricos.

Carácter de formato	**Descripción**
c	Indica la moneda del sistema (10,20 €)
g	Formato general (depende del formato del dato)
e	Formato científico (3,77e+01)
f	Formato de coma flotante (3,44)
n	Formato numérico

p	Formato porcentual (20%)
x	Formato hexadecimal

Tabla 8

En la Tabla 9 se ofrecen los caracteres para formatear fechas.

Carácter de formato	**Descripción**
d	Fecha corta (d/m/yyyy)
D	Fecha larga (dddd, dd mmmm, yyyy)
f	Fecha larga y hora corta (dddd, dd mmmm yyyy hh:mm)
g	Fecha corta y tiempo corto (d/m/yyyy hh:mm)
m	Mes y día (dd mmmm)
t	Hora corta (hh:mm)
y	Mes y año (mmmm, yyyy)

Tabla 9

También es posible utilizar una serie de caracteres que nos permiten especificar de forma directa el formato que van a tener los valores numéricos. Así por ejemplo, si aplicamos la cadena de formato "00#.##" sobre el número 1,2345 el resultado será "001,23". En la Tabla 10 se comentan estos últimos caracteres de formato.

Carácter de formato	**Descripción**
0	Muestra un cero si no hay un valor significativo, es decir, añade un cero delante de un número o bien al final de un número después de la coma.
#	Se reemplaza únicamente con dígitos significativos, es decir equivale a un dígito en el caso de que éste exista.
.	Muestra la coma decimal.
,	Separa los números en grupos utilizando el carácter correspondiente del sistema actual.

Tabla 10

Una vez comentado en profundidad el método DataBinder.Eval() vamos a verlo en acción en el Código fuente 146. En este ejemplo se ha utilizado como origen de datos una tabla de una base de datos y el control encargado de mostrar estos datos en un control Repeater (que veremos en el próximo capítulo).

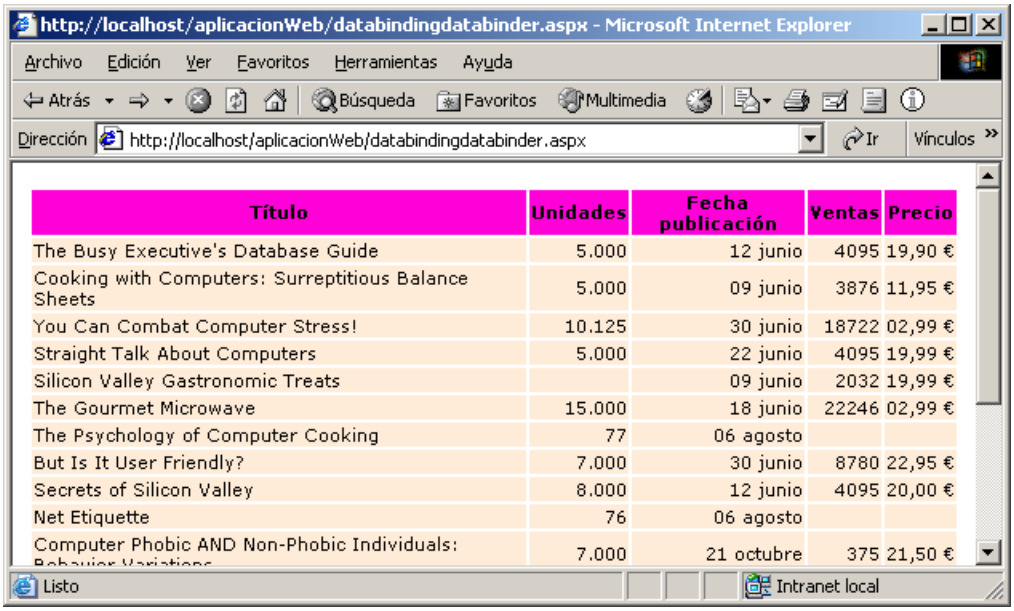
```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
  SqlConnection conexion = new
                       SqlConnection("server=angel;database=pubs;uid=sa;pwd=");
  SqlDataAdapter comando = new SqlDataAdapter("select * from Titles", conexion);
  DataSet ds = new DataSet();
  comando.Fill(ds, "Titles");
  repiteDatos.DataSource = ds.Tables["Titles"].DefaultView;
  repiteDatos.DataBind();
}
</script>

<body>

<asp:Repeater id="repiteDatos" runat="server">

      <HeaderTemplate>
        <table width="100%" style="font: 8pt verdana">
          <tr bgcolor="ff00d8">
            <th>Título</th>
            <th>Unidades</th>
            <th>Fecha publicación</th>
            <th>Ventas</th>
            <th>Precio</th>
          </tr>
      </HeaderTemplate>

      <ItemTemplate>
       <tr bgcolor="FFECD8">
       <td><%# DataBinder.Eval(Container.DataItem, "titulo") %></td>
       <td align="right">
       <%# DataBinder.Eval(Container.DataItem, "unidades","{0:#,###}") %></td>
       <td align="right">
       <%# DataBinder.Eval(Container.DataItem, "fecha","{0:m}") %></td>
       <td align="right">
       <%# DataBinder.Eval(Container.DataItem, "ventas","{0:g}") %></td>
       <td align="right">
       <%# DataBinder.Eval(Container.DataItem, "precio", "{0:0#.00} €") %></td>
       </tr>
      </ItemTemplate>

      <FooterTemplate>
        </table>
      </FooterTemplate>

  </asp:Repeater>
</body>
</html>
```

Código fuente 146

Para la columna título de la tabla de la base de datos no se ha especificado ningún formato, pero si para las siguientes columnas: se ha indicado que en el campo unidades se agrupen los miles; en el

campo fecha de publicación se ha formateado la fecha con el día y el mes; para el campo ventas se ha indicado el formato general; y para terminar en el campo precio se ha dado un formato de precio para euros, indicando también el carácter (€) de la moneda.

En la Figura 83 se puede ver el aspecto que mostraría esta página.



Título	Unidades	Fecha publicación	Ventas	Precio
The Busy Executive's Database Guide	5.000	12 junio	4095	19,90 €
Cooking with Computers: Surreptitious Balance Sheets	5.000	09 junio	3876	11,95 €
You Can Combat Computer Stress!	10.125	30 junio	18722	02,99 €
Straight Talk About Computers	5.000	22 junio	4095	19,99 €
Silicon Valley Gastronomic Treats		09 junio	2032	19,99 €
The Gourmet Microwave	15.000	18 junio	22246	02,99 €
The Psychology of Computer Cooking	77	06 agosto		
But Is It User Friendly?	7.000	30 junio	8780	22,95 €
Secrets of Silicon Valley	8.000	12 junio	4095	20,00 €
Net Etiquette	76	06 agosto		
Computer Phobic AND Non-Phobic Individuals:	7.000	21 octubre	375	21,50 €

Figura 83

Con este ejemplo se da por finalizado el presente capítulo, en el siguiente trataremos los controles de lista.

11

Web Forms: controles de lista

Introducción

Este es el último de los capítulos dedicado al entorno de programación ofrecido por los Web Forms, como su nombre indica trataremos los controles Web de lista.

Estos controles se encuentran especializados en mostrar listados de datos en las páginas ASP .NET, tarea que suele ser bastante común en la mayoría de las aplicaciones ASP .NET. Los controles de lista son tres y ya los comentamos brevemente en el capítulo anterior, e incluso se realizó algún ejemplo con los más sencillos, de todas formas para refrescar la memoria los vamos a enumerar escuetamente:

- Repeater: este control muestra elementos de datos en una lista, por sí sólo no genera una salida visible para el usuario, para que sea visible se debe utilizar junto con las plantillas. Es el más sencillo de los tres controles.

- DataList: similar al control anterior pero permite una mayor personalización a la hora de mostrar la información en forma de lista. También permite la selección y edición de elementos. También debe utilizarse junto con las plantillas.

- DataGrid: este control muestra la información en forma de tabla con celdas y columnas, generando el código HTML equivalente a una tabla. Es el más complejo de los tres controles de lista ofreciendo un alto grado de personalización y una serie de características muy interesantes como pueden ser la ordenación, paginación, definición de columnas, edición, etc.

En los siguientes apartados vamos a comentar en detalle cada uno de estos controles, que serán acompañados de los ejemplos necesarios.

El control Repeater

Este control de lista va a repetir una serie de elementos en forma de lista según las plantillas que tenga definidas. Como mínimo el control Repeater debe definir una platilla para cada elemento, es decir, una plantilla ItemTemplate, y puede utilizar el resto de plantillas ya comentadas en el capítulo anterior.

Este control es el más sencillo de todos los controles de lista, no ofrece ningún tipo de salida, se debe indicar de forma explícita todo el código HTML que deseemos que presente los datos, debido a que no genera ninguna salida, es necesario utilizar como mínimo una plantilla ItemTemplate.

Ya hemos visto la sencilla utilización del control Repeater en distintos ejemplos del capítulo anterior, pero vamos a ver un ejemplo más en el Código fuente 147.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
       if(!Page.IsPostBack){
              SqlConnection conexion =
                new SqlConnection("server=angel;database=pubs;uid=sa;pwd=");
              SqlDataAdapter comando =
                new SqlDataAdapter("select title,type,price from Titles", conexion);
              DataSet ds = new DataSet();
              comando.Fill(ds, "Titles");
              repiteDatos.DataSource = ds.Tables["Titles"].DefaultView;
              repiteDatos.DataBind();
       }
}
</script>

<body>

<asp:Repeater id="repiteDatos" runat="server">

      <HeaderTemplate>
        <table width="100%" style="font: 8pt verdana">
          <tr bgcolor="ff00d8">
            <th>Título</th>
            <th>Tipo</th>
            <th>Precio</th>
          </tr>
      </HeaderTemplate>

      <ItemTemplate>
       <tr bgcolor="FFECD8">
       <td><%# ((DataRowView)Container.DataItem)[0]%></td>
       <td><%# ((DataRowView)Container.DataItem)[1]%></td>
       <td align="right"><%# ((DataRowView)Container.DataItem)[2]%></td>
       </tr>
      </ItemTemplate>

      <AlternatingItemTemplate>
       <tr style="background-color:FFEC00">
       <td><%# ((DataRowView)Container.DataItem)[0]%></td>
       <td><%# ((DataRowView)Container.DataItem)[1]%></td>
       <td align="right"><%# ((DataRowView)Container.DataItem)[2]%></td>
       </tr>
      </AlternatingItemTemplate>
```

```
      <FooterTemplate>
        </table>
      </FooterTemplate>

  </asp:Repeater>
</body>
</html>
```

Código fuente 147

En este caso el origen de datos del control Repeater es una tabla de una base de datos, y para indicar en que columna de la tabla, que va a generar el control Repeater a través de la plantilla, se muestra cada campo del registro actual, se utiliza la expresión Container.DataItem aplicando el casting (conversión de tipos) necesario para poder tratar a cada elemento de datos como un objeto de la clase DataRowView.

En la Figura 84 se puede ver el resultado de la ejecución de la página.

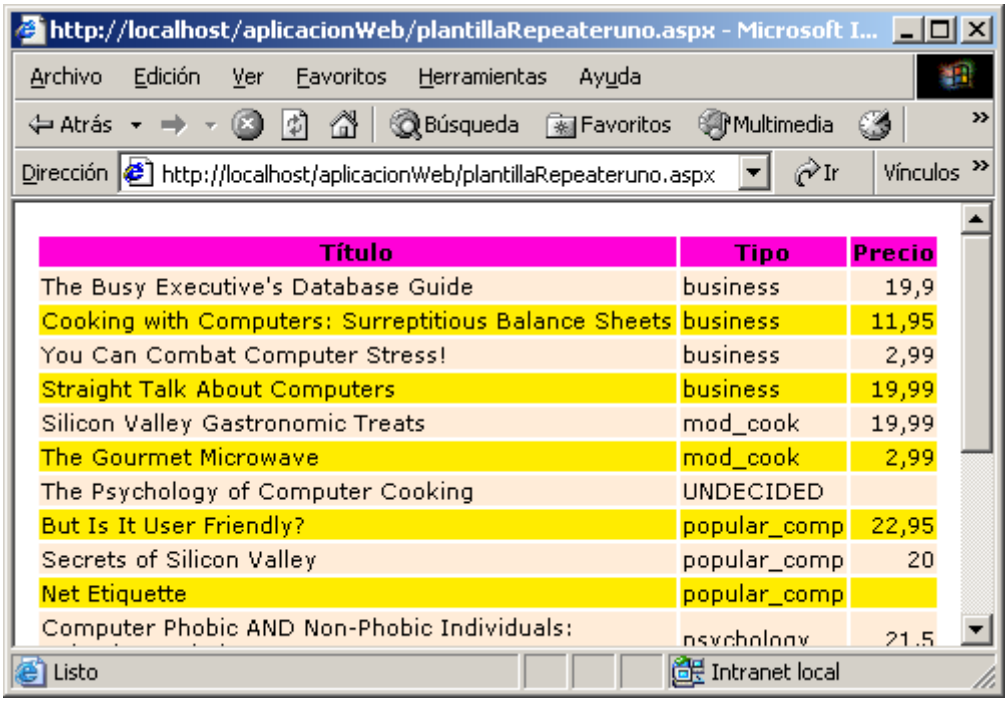


Figura 84

A continuación vamos a tratar el siguiente control de lista, el control DataList que ofrece alguna funcionalidad más.

El control DataList

Al igual que sucedía con el control Repeater, este control de lista debe utilizar al menos la plantilla ItemTemplate. Este control se puede personalizar más ya que incorpora tres propiedades llamadas RepeatDirection, RepeatLayout y RepeatColumns, que permiten indicar de que forma se va a ir mostrando y distribuyendo la información, sin tener que especificarlo a través de las plantillas. Vamos a pasar a comentar cada una de estas propiedades:

- RepeatLayout: esta propiedad indica que distribución se le va a dar a los datos, es decir, a cada elemento de datos incluido en la plantilla DataItem. Esta propiedad puede tomar dos valores, que son Table, para indicar un formato de distribución de tabla típico de HTML con sus filas y columnas correspondientes, y el valor Flow, para indicar que los elementos se van a distribuir sin ningún formato de tabla sino que lo hacen de forma lineal. El valor por defecto de esta propiedad es Table.

- RepeatDirection: indica si los elementos se van a mostrar de forma horizontal o vertical, es decir, la dirección en la que se van a ir repitiendo los distintos elementos contenidos en el control DataList. Los valores que acepta esta propiedad son Vertical y Horizontal. Por defecto esta propiedad tiene el valor Vertical.

- RepeatColumns: esta otra propiedad va a recibir un valor entero que indicará el número de columnas utilizado para mostrar los datos. Por defecto el valor de esta propiedad es 1.

En el Código fuente 148 se muestra un control DataList que distribuye la información de la siguiente forma: en una tabla de la que se muestran todos los bordes, de tres en tres columnas y con un distribución vertical de los elementos. Para indicar que se deben mostrar las líneas de separación de la tabla se le ha asignado ala propiedad GridLines el valor Both.

El origen de los datos del control DataList es una tabla de una base de datos de la que se vana recuperar los campos de nombre y apellidos.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
  if(!Page.IsPostBack){
     SqlConnection conexion =
        new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
     SqlDataAdapter comando =
        new SqlDataAdapter("select firstname, lastname from Employees", conexion);
     DataSet ds = new DataSet();
     comando.Fill(ds, "Employees");
     lista.DataSource = ds.Tables["Employees"].DefaultView;
     lista.DataBind();
  }
}
</script>

<body>

<asp:DataList id="lista" runat="server"
       RepeatDirection="Vertical" RepeatLayout="Table"
       RepeatColumns="3" GridLines="Both">

      <ItemTemplate>
       <%# ((DataRowView)Container.DataItem)["firstname"]%> 
       <%# ((DataRowView)Container.DataItem)["lastname"]%>
      </ItemTemplate>


  </asp:DataList>
</body>
</html>
```

Código fuente 148

En la Figura 85 se puede ver la forma en la se distribuyen los elementos dentro del control DataList.

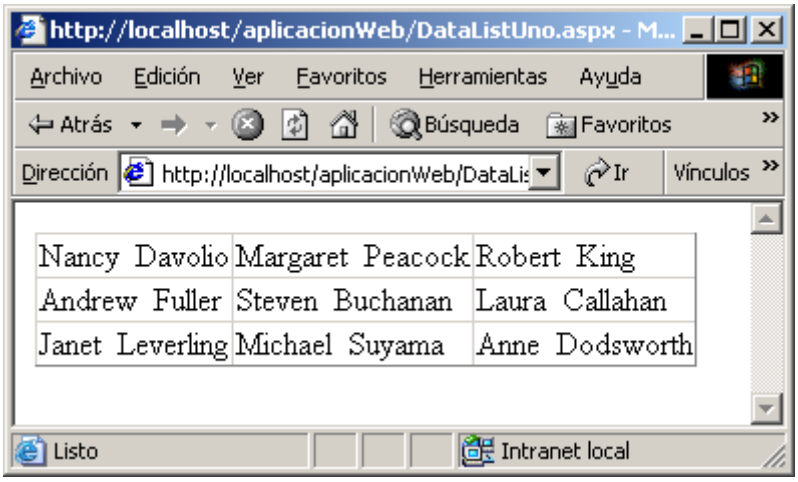


Figura 85

Y el Código fuente 149 es el código HTML que se genera.

```
<html>
<body>
<table id="lista" cellspacing="0" rules="all" border="1" style="border-
collapse:collapse;">
       <tr><td>
       Nancy 
       Davolio
              </td><td>
       Margaret 
       Peacock
              </td><td>
       Robert 
       King
              </td>
       </tr><tr>
              <td>
       Andrew 
       Fuller
              </td><td>
       Steven 
       Buchanan
              </td><td>
       Laura 
       Callahan
              </td>
       </tr><tr>
              <td>
       Janet 
       Leverling
              </td><td>
       Michael 
       Suyama
              </td><td>
       Anne 
       Dodsworth
              </td></tr>
</table>
</body>
```

```
</html>
```

Código fuente 149

Si modificamos el ejemplo anterior para que podamos especificar de forma dinámica en un formulario Web las propiedades de distribución de los elementos del control DataList, obtendremos el Código fuente 150.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
 if(!Page.IsPostBack){
   SqlConnection conexion =
      new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
   SqlDataAdapter comando =
      new SqlDataAdapter("select firstname, lastname from Employees", conexion);
   DataSet ds = new DataSet();
   comando.Fill(ds, "Employees");
   lista.DataSource = ds.Tables["Employees"].DefaultView;
   lista.DataBind();
 }
}

void pulsado(Object Sender, EventArgs E) {
    if (direccion.SelectedIndex == 0)
        lista.RepeatDirection = RepeatDirection.Horizontal;
    else
        lista.RepeatDirection = RepeatDirection.Vertical;
    if (distribucion.SelectedIndex == 0)
        lista.RepeatLayout = RepeatLayout.Table;
    else
        lista.RepeatLayout = RepeatLayout.Flow;
    lista.RepeatColumns=columnas.SelectedIndex+1;
    if ((bordes.Checked) && (lista.RepeatLayout == RepeatLayout.Table)) {
        lista.BorderWidth = 1;
        lista.GridLines = GridLines.Both;
    }else {
        lista.BorderWidth = 0;
        lista.GridLines = GridLines.None;
    }
}

</script>

<body>

<asp:DataList id="lista" runat="server">

      <ItemTemplate>
       <%# ((DataRowView)Container.DataItem)["firstname"]%> 
       <%# ((DataRowView)Container.DataItem)["lastname"]%>
      </ItemTemplate>

  </asp:DataList>

  <form method="post" runat="server" id="formulario">
       Dirección:
        <asp:DropDownList id="direccion" runat="server">
            <asp:ListItem>Horizontal</asp:ListItem>
```

```
            <asp:ListItem>Vertical</asp:ListItem>
        </asp:DropDownList><br>

        Distribución:
        <asp:DropDownList id="distribucion" runat="server">
            <asp:ListItem>Table</asp:ListItem>
            <asp:ListItem>Flow</asp:ListItem>
        </asp:DropDownList><br>

        N° Columnas:
        <asp:DropDownList id="columnas" runat="server">
            <asp:ListItem>1</asp:ListItem>
            <asp:ListItem>2</asp:ListItem>
            <asp:ListItem>3</asp:ListItem>
            <asp:ListItem>4</asp:ListItem>
            <asp:ListItem>5</asp:ListItem>
        </asp:DropDownList><br>

        Mostrar bordes:
        <asp:CheckBox id="bordes" runat="server" /><p>

        <asp:Button id="boton" Text="Enviar" OnClick="pulsado" runat="server"/>
</form>
</body>
</html>
```

Código fuente 150

En la Figura 86 se puede ver un ejemplo de ejecución de este nuevo código, en este enlace se encuentra la página ASP .NET.

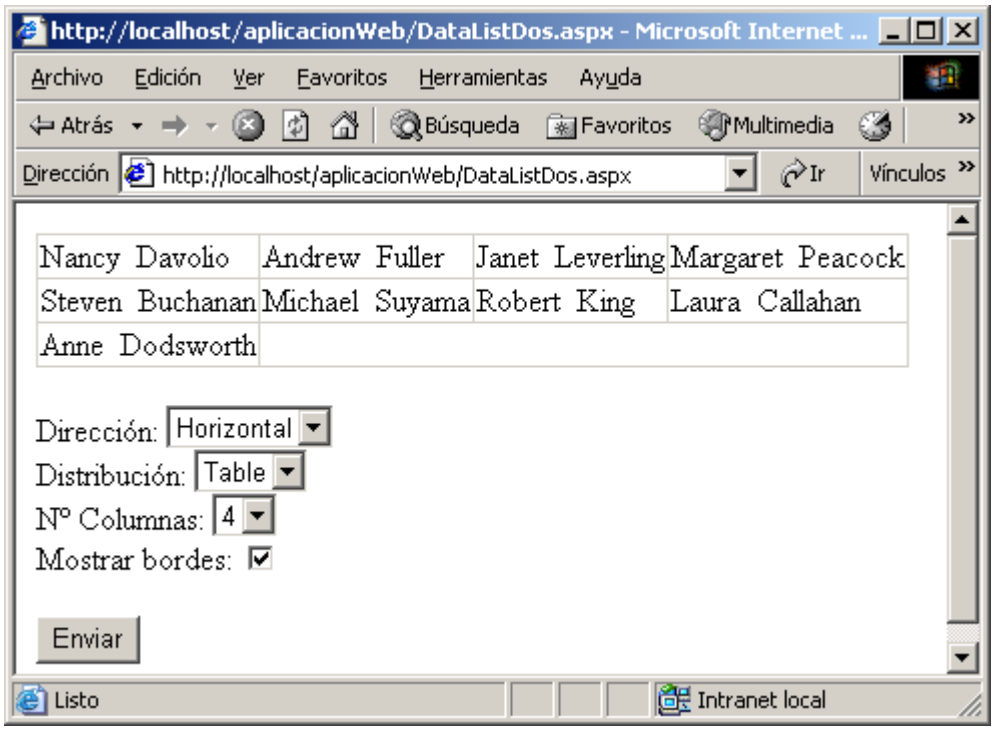


Figura 86

El control DataList permite especificar el estilo que van a tener cada uno de los distintos tipos de plantillas utilizados para mostrar la información, esto se consigue mediante una serie de propiedades

ofrecidas por el control, así por ejemplo tenemos una propiedad denominada ItemStyle-Font-Bold para indicar si la fuente del texto que va a aparecer dentro de una plantilla ItemTemplate va es estar en negrita o no; y existirá una propiedad similar para cada una de las distintas plantillas.

En el Código fuente 151 se puede ver un control DataList sobre el que se aplican estas propiedades para definir el aspecto de las distintas plantillas que utiliza.

```
<%@ Page Language="c#"%>
<script language="c#" runat="server">
void Page_Load(Object objFuente,EventArgs args){
      String[] datos=new String [5];
      datos[0]="Bora-bora";
      datos[1]="Morea";
      datos[2]="Pascua";
      datos[3]="Papeete";
      datos[4]="Santiago de Chile";
      lista.DataSource=datos;
      lista.DataBind();
}
</script>
<html>
<body>
<form id="formulario" method="post" runat="server">
<asp:DataList id="lista" runat="server"
 RepeatColumns="3" RepeatDirection="Horizontal" RepeatLayout="Flow"
HeaderStyle-Font-Underline="True" HeaderStyle-BackColor="yellow"
HeaderStyle-Font-Bold="True" HeaderStyle-Font-Size="16"
HeaderStyle-ForeColor="blue" AlternatingItemStyle-BackColor="green"
ItemStyle-BackColor="red" AlternatingItemStyle-Font-Italic="True"
AlternatingItemStyle-Font-Name="Courier" ItemStyle-Font-Name="Arial"
FooterStyle-Font-Italic="True" FooterStyle-Font-Size="10"
FooterStyle-HorizontalAlign="Left">
      <HeaderTemplate>
      Destinos
      </HeaderTemplate>
      <ItemTemplate>
            <%#Container.DataItem%>
      </ItemTemplate>
      <AlternatingItemTemplate>
            <%#Container.DataItem%>
      </AlternatingItemTemplate>
      <FooterTemplate>
            <a href="mailto:reservas@correo.es">reservas@correo.es</a>
      </FooterTemplate>
</asp:DataList>
</form>
</body>
</html>
```

Código fuente 151

En la Figura 87 se puede ver el aspecto que ofrece el control DataList.

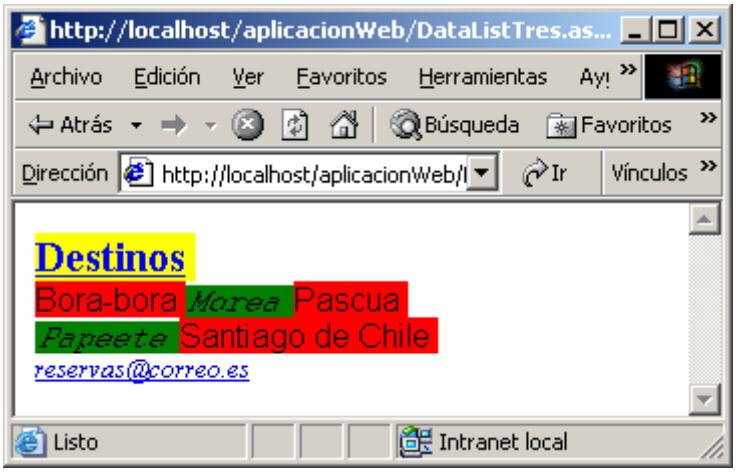


Figura 87

El control DataList permite utilizar dos plantillas (templates) nuevas que no hemos visto con el control Repeater ya que no las soporta, estas dos nuevas plantillas se denominan SelectedItemTemplate y EditItemTemplate. Vamos a comentar a continuación estas plantillas:

- SelectedItemTemplate: esta plantilla define el aspecto que va a tener un elemento cuando es seleccionado dentro de un control DataList o DataGrid, esta plantilla, al igual que la siguiente no se encuentra disponible en el control Repeater. Más adelante veremos como se indica que un elemento determinado ha sido seleccionado, para ello utilizaremos el manejador de eventos OnItemCommand y la propiedad SelectedIndex del control DataList.

- EditItemTemplate: esta segunda plantilla permite definir el aspecto de un elemento cuando pasa a modo de edición, es decir esta plantilla nos va a permitir los datos de un elemento determinado. Más adelante veremos que debemos utilizar el manejador de eventos OnEditCommand y la propiedad EditItemIndex para mostrar esta plantilla y así iniciar el proceso de edición.

Vamos a mostrar con un sencillo ejemplo (Código fuente 152) la utilización de la plantilla SelectedItemTemplate sobre un control DataList, en este caso al seleccionar un elemento determinado se va a mostrar una información detallada del mismo, esta información detallada se define en la plantilla SelectedItemTemplate. Así al seleccionar un empleado se muestra una serie de datos adicionales además de su nombre y apellidos, la sintaxis de Data Binding para mostrar estos datos se puede observar en la plantilla SelectedItemTemplate.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
      if(!Page.IsPostBack) estableceDataBinding();
}

void estableceDataBinding(){
      SqlConnection conexion =
            new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      SqlDataAdapter comando =
            new SqlDataAdapter("select * from Employees", conexion);
      DataSet ds = new DataSet();
      comando.Fill(ds, "Employees");
```

```
       lista.DataSource = ds.Tables["Employees"].DefaultView;
       lista.DataBind();
}
void pulsadoSeleccion(object Sender, DataListCommandEventArgs e) {
       lista.SelectedIndex = e.Item.ItemIndex;
       estableceDataBinding();
}


</script>

<body>
<form id="formulario" runat="server">
<asp:DataList id="lista" runat="server"
        RepeatDirection="Vertical" RepeatLayout="Table"
        RepeatColumns="2" GridLines="Both" OnItemCommand="pulsadoSeleccion"
        >

      <ItemTemplate>
       <asp:Button Text="..." Runat="server"/>
       <%# ((DataRowView)Container.DataItem)["firstname"]%> 
       <%# ((DataRowView)Container.DataItem)["lastname"]%>
      </ItemTemplate>
         <SelectedItemStyle BackColor="yellow" BorderWidth="4">
         </SelectedItemStyle>
         <SelectedItemTemplate>
          Nombre:<b><%# ((DataRowView)Container.DataItem)["firstname"]%></b><br>
       Apellidos:<b><%# ((DataRowView)Container.DataItem)["lastname"]%></b><br>
       Puesto:<b><%# ((DataRowView)Container.DataItem)["title"]%></b><br>
       Dirección:<b><%# ((DataRowView)Container.DataItem)["address"]%></b><br>
          Ciudad:<b><%# ((DataRowView)Container.DataItem)["city"]%></b>

         </SelectedItemTemplate>

  </asp:DataList>
 </form>
</body>
</html>
```

Código fuente 152

Cuando se muestra la plantilla SelectedItemTemplate para un elemento determinado de un control DataList, se dice que estamos en modo de selección. En el código anterior se puede comprobar que se ha utilizado la etiqueta SelectedItemStyle para indicar el estilo de la plantilla SelectedItemTemplate, en este caso se ha indicado el color de fondo y el tipo de borde.

Para poder seleccionar un elemento dentro del control DataList debemos incluir en cada elemento, es decir, en cada plantilla ItemTemplate, un botón de cualquier tipo, en este botón debemos asignar a la propiedad CommandName el valor Select, así indicaremos que se trata del botón que nos permitirá seleccionar un elemento de la lista.

Al pulsar el botón del elemento correspondiente se lanzará el evento ItemCommand para indicar que un elemento determinado ha sido seleccionado por el usuario, y en ese momento se ejecutará el método definido para el manejador de eventos OnItemCommand.

En la Figura 88 se puede ver el aspecto que tendría un control DataList al seleccionar un elemento dentro del mismo y al aplicarle la plantilla SelectedItemTemplate del código anterior.

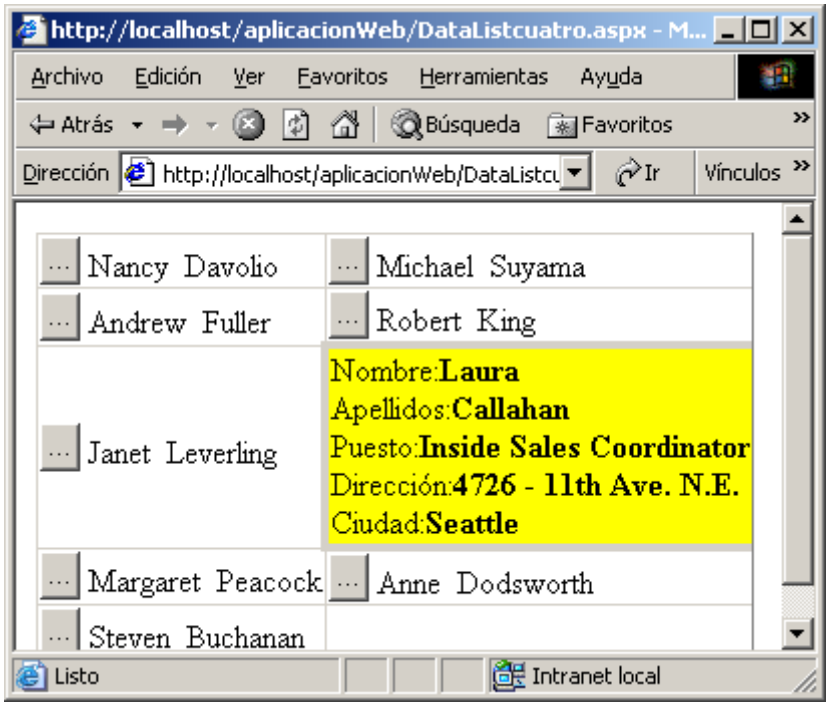


Figura 88

En el método que se ejecuta en el manejador de eventos OnItemCommand indicamos el elemento que se ha seleccionado de la lista actual, para ello a la propiedad SelectedIndex del objeto DataList se le asigna la propiedad ItemIndex, que a su vez es propiedad de la propiedad Item del evento de la clase DataListCommandEventArgs, es decir, la clase que va a representar el evento de la pulsación de un elemento de un control DataList.

Por defecto la propiedad SelectedIndex tiene el valor de –1, es decir, no hay ningún elemento de la lista seleccionado, cuando se le asigna un valor a esta propiedad aplicará la plantilla SelectedItemTemplate sobre el elemento seleccionado.

Además de seleccionar un elemento de la lista también podemos pasar al modo de edición para modificarlo, para ello se hace uso de la plantilla EditItemTemplate. Para mostrar el uso de la plantilla EditItemTemplate vamos a ampliar el ejemplo anterior, además de poder ver la información ampliada de cada elemento de la lista, vamos a poder modificar los valores de los distintos campos. Al editar un elemento vamos a mostrar los valores de sus campos en cajas de texto (controles Web TextBox) que nos van a permitir la edición.

En la plantilla ItemTemplate vamos a añadir un nuevo control Button que nos va a permitir indicar que deseamos pasar al modo de edición de un elemento determinado del objeto DataList, para ello debemos asignar a la propiedad CommandName del botón el valor Edit.

Al pulsar el botón de edición de un elemento determinado se lanzará el evento EditCommand para indicar que un elemento determinado ha sido seleccionado por el usuario para editarlo, y en ese momento se ejecutará el método definido para el manejador de eventos OnEditCommand.

En el método que se ejecuta en el manejador de eventos OnEditCommand indicamos el elemento que se ha quiere edirae de la lista actual, para ello a la propiedad EditItemIndex del objeto DataList se le asigna la propiedad ItemIndex, que a su vez es propiedad de la propiedad Item del evento de la clase DataListCommandEventArgs, es decir, la clase que va a representar el evento de la pulsación de un elemento de un control DataList. También se le asigna a la propiedad SelectedIndex el valor –1, de esta forma se cancela la selección anterior de cualquier otro elemento de la lista.

En la plantilla EditItemTemplate además de mostrar los valores del elemento actual en cajas de texto, también se muestran dos controles LinkButton, que nos van a permitir llevar a cabo la actualización de los datos o bien cancelar la acción de modificación. Estos botones deberán indicar en sus propiedades ComandName los valores Update y Cancel respectivamente, ya que el control DataList también ofrece las propiedades OnUpdateCommand y OnCancelCommand para tratar los eventos de actualización y cancelación de edición de un elemento determinado.

Como se puede ver en el Código fuente 153, debemos tratar ahora distintos eventos: el de selección de un elemento, el de edición de un elemento, el de actualización de un elemento y el de cancelación de edición, como se puede ver el código de la página ASP .NET, que se puede obtener en el siguiente enlace, es más complejo que el ejemplo anterior e incorpora el código necesario para realizar una modificación en la base de datos en el método pulsadoActualizar().

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
      if(!Page.IsPostBack) estableceDataBinding();
}
void estableceDataBinding(){
      SqlConnection conexion =
            new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      SqlDataAdapter comando =
            new SqlDataAdapter("select * from Employees", conexion);
      DataSet ds = new DataSet();
      comando.Fill(ds, "Employees");
      lista.DataSource = ds.Tables["Employees"].DefaultView;
      lista.DataBind();
}
void pulsadoSeleccion(object Sender, DataListCommandEventArgs e) {
      lista.EditItemIndex=-1;
      lista.SelectedIndex = e.Item.ItemIndex;
      estableceDataBinding();
}
void pulsadoEdicion(object Sender, DataListCommandEventArgs e) {
      lista.SelectedIndex=-1;
      lista.EditItemIndex = e.Item.ItemIndex;
      estableceDataBinding();
}
void pulsadoCancelacion(object Sender, DataListCommandEventArgs e) {
      lista.EditItemIndex=-1;
      lista.SelectedIndex=-1;
      estableceDataBinding();
}

void pulsadoActualizar(object Sender, DataListCommandEventArgs e) {
      SqlConnection conexion =
          new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      String strComando;
      String txtNombre=((TextBox)e.Item.FindControl("nombre")).Text;
      String txtApellidos=((TextBox)e.Item.FindControl("apellidos")).Text;
      String txtPuesto=((TextBox)e.Item.FindControl("puesto")).Text;
      String txtDireccion=((TextBox)e.Item.FindControl("direccion")).Text;
      String txtCiudad=((TextBox)e.Item.FindControl("ciudad")).Text;

      strComando="UPDATE Employees SET firstname='" + txtNombre + "', lastname='" +
      txtApellidos + "', address='" +txtDireccion + "', title='" +txtPuesto +
      "', city='" + txtCiudad +
      "'WHERE EmployeeID=" +lista.DataKeys[(int)e.Item.ItemIndex];

      SqlCommand comando = new SqlCommand(strComando, conexion);
```

```
       comando.Connection.Open();
       comando.ExecuteNonQuery();
       lista.EditItemIndex=-1;
       lista.SelectedIndex=-1;
       estableceDataBinding();
}

</script>

<body>
<form id="formulario" runat="server">
<asp:DataList id="lista" runat="server"
        RepeatDirection="Vertical" RepeatLayout="Table"
        RepeatColumns="2" GridLines="Both" OnItemCommand="pulsadoSeleccion"
         OnEditCommand="pulsadoEdicion" OnCancelCommand="pulsadoCancelacion"
          OnUpdateCommand="pulsadoActualizar"
         DataKeyField="employeeid"
        >

      <ItemTemplate>
       <asp:Button Text="..." Runat="server" CommandName="Select"
ID="BotonSeleccionar"/>
       <asp:Button Text="Editar" Runat="server" CommandName="Edit"
ID="BotonEditar"/>
       <%# ((DataRowView)Container.DataItem)["firstname"]%> 
       <%# ((DataRowView)Container.DataItem)["lastname"]%>
      </ItemTemplate>

       <SelectedItemStyle BackColor="yellow" BorderWidth="4">
      </SelectedItemStyle>

    <SelectedItemTemplate>
          Nombre:<b><%# ((DataRowView)Container.DataItem)["firstname"]%></b><br>
       Apellidos:<b><%# ((DataRowView)Container.DataItem)["lastname"]%></b><br>
       Puesto:<b><%# ((DataRowView)Container.DataItem)["title"]%></b><br>
       Dirección:<b><%# ((DataRowView)Container.DataItem)["address"]%></b><br>
          Ciudad:<b><%# ((DataRowView)Container.DataItem)["city"]%></b>
    </SelectedItemTemplate>
    <EditItemStyle HorizontalAlign="Right"  Font-Italic="True"
         BorderStyle="Dotted" BorderWidth="4" BackColor=#ffcc33>
    </EditItemStyle>
   <EditItemTemplate>
          Nombre:
          <asp:TextBox id="nombre"  Runat="server"
          Text='<%# ((DataRowView)Container.DataItem)["firstname"]%>'/><br>
       Apellidos:<asp:TextBox id="apellidos"  Runat="server"
          Text='<%# ((DataRowView)Container.DataItem)["lastname"]%>'/><br>
       Puesto:<asp:TextBox id="puesto"  Runat="server"
          Text='<%# ((DataRowView)Container.DataItem)["title"]%>'/><br>
       Dirección:<asp:TextBox id="direccion"  Runat="server"
          Text='<%# ((DataRowView)Container.DataItem)["address"]%>'/><br>
          Ciudad:<asp:TextBox id="ciudad"  Runat="server"
          Text='<%# ((DataRowView)Container.DataItem)["city"]%>'/><br>
          <asp:LinkButton text="Actualizar" CommandName="Update"
              id="botonActualizar" Runat="server"/>
          <asp:LinkButton text="Cancelar" CommandName="Cancel"
              id="botonCancelar" Runat="server"/>
    </EditItemTemplate>

  </asp:DataList>
 </form>
</body>
</html>
```

Código fuente 153

El método pulsadoActualizar() merece un comentario destacado, sobretodo en lo que a la recuperación de los valores de las cajas de texto se refiere, ya que de momento no nos vamos a ocupar del acceso a datos con ADO .NET. Como se puede observar en el código anterior, se ha utilizado el método FindControl() para poder acceder a el texto de un objeto TextBox determinado, este método recibe como parámetro una cadena de texto que indica el nombre del control Web que se quiere recuperar, una vez que se tiene la referencia al control deseado se transforma al tipo de objeto deseado, en este caso un objeto de la clase TextBox, y ya se puede acceder a su propiedad Text.

Además de recuperar los valores de los campos que se van a modificar en un elemento determinado, también se debe obtener la clave de ese elemento para poder llevar a cabo la actualización en la base de datos. Para obtener el identificador del elemento editado del control DataList debemos acceder a la colección DataKeys, a la que pasaremos como índice el índice del elemento seleccionado actualmente. En la propiedad DataKeyField del control DataList debemos indicar el campo que funciona como clave en la base de datos, en este caso es el campo identificador de empleado.

En la Figura 89 se puede ver el aspecto que presenta la plantilla EditItemTemplate.

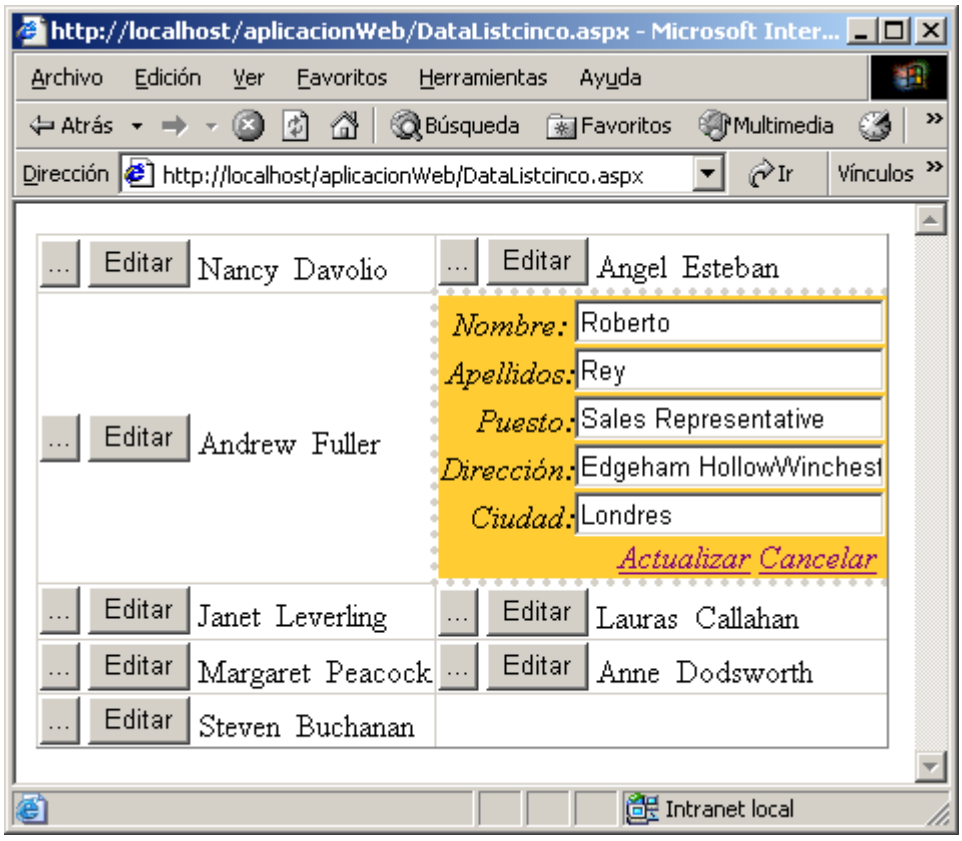


Figura 89

Como se ha podido comprobar el control DataList es mucho más completo y complejo que el control Repeater, en el siguiente apartado vamos a tratar el último control Web de lista, que es mucho más completo que los vistos hasta ahora, ya que además de permitir la selección y edición de elementos con las plantillas correspondientes, permite una mayor personalización a la hora de mostrar y distribuir la información.

El control DataGrid

Este es el último y más completo de los controles Web de lista este control muestra la información en forma de tabla con celdas y columnas, generando el código HTML equivalente a una tabla. Se puede utilizar de forma muy sencilla, o podemos complicar su uso para llegar a altos niveles de personalización a la hora de mostrar la información.

El control DataGrid permite la selección, edición (al igual que sucedía con el control DataList), ordenación y paginación de los datos.

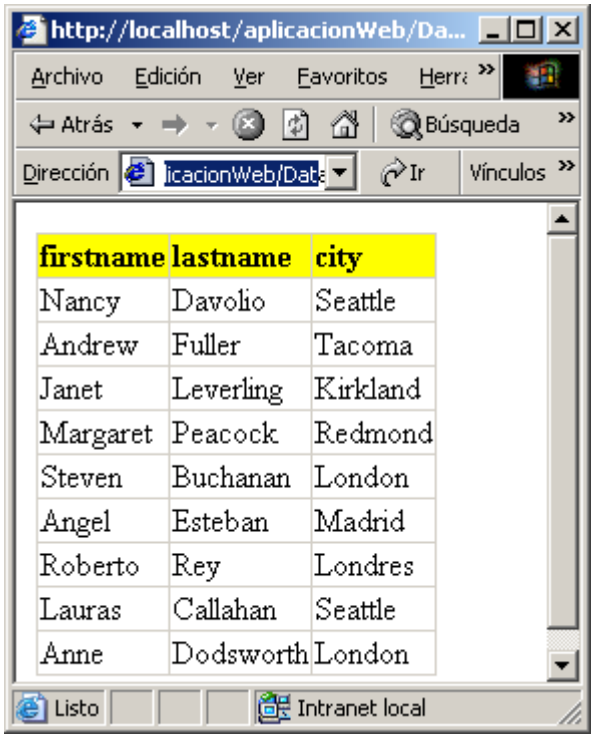
Vamos a mostrar en primer lugar la forma más sencilla de utilizar el control DataGrid, que consiste en establecer la conexión con los datos y luego establecer la conexión de los datos con el control a través del mecanismo de Data Binding.

Por defecto el control DataGrid genera una columna (BoundColumn) por cada campo presente en la fuente de datos, el nombre de los campos aparecerá en la cabecera de las columnas y los valores de los campos se mostrarán en cada columna como etiquetas de texto. En el Código fuente 154 se puede ver un ejemplo sencillo de utilización de este control.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
  SqlConnection conexion =
    new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
  SqlDataAdapter comando =
    new SqlDataAdapter("select firstname, lastname,city from Employees", conexion);
  DataSet ds = new DataSet();
  comando.Fill(ds, "Employees");
  tabla.DataSource = ds.Tables["Employees"].DefaultView;
  tabla.DataBind();
}
</script>

<body>
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow">
</asp:DataGrid>
</body>
</html>
```

Código fuente 154

Y en la Figura 90 se puede ver el resultado de la ejecución de esta página ASP .NET.



firstname	lastname	city
Nancy	Davolio	Seattle
Andrew	Fuller	Tacoma
Janet	Leverling	Kirkland
Margaret	Peacock	Redmond
Steven	Buchanan	London
Angel	Esteban	Madrid
Roberto	Rey	Londres
Lauras	Callahan	Seattle
Anne	Dodsworth	London

Figura 90

El código HTML que genera este ejemplo es el Código fuente 155.

```
<html>
<body>
<table cellspacing="0" rules="all" border="1" id="tabla" style="border-
width:1px;border-style:solid;border-collapse:collapse;">
	<tr style="background-color:Yellow;font-weight:bold;">
		<td>
			firstname
		</td><td>
			lastname
		</td><td>
			city
		</td>
	</tr><tr>
		<td>
			Nancy
		</td><td>
			Davolio
		</td><td>
			Seattle
		</td>
	</tr><tr>
		<td>
			Andrew
		</td><td>
			Fuller
		</td><td>
			Tacoma
		</td>
	</tr>

	[… … …]
```

```
	<tr>
		<td>
			Anne
		</td><td>
			Dodsworth
		</td><td>
			London
		</td>
	</tr>
</table>
</body>
</html>
```

Código fuente 155

En los siguientes subapartados vamos a ir tratando los distintos aspectos de configuración y utilización que nos ofrece el control de lista DataGrid, primero vamos a comenzar por la personalización de columnas.

Definición de columnas dentro de un control DataGrid

Podemos indicar de forma explícita las columnas que deseamos que aparezcan en el control DataGrid, si deseamos hacer esto asignaremos a la propiedad AutoGenerateColums el valor False, por defecto tiene el valor True, también se debe hace uso de la etiqueta <Columns>, que contendrá en su cuerpo la definición de las distintas columnas del control DataGrid.

A la hora de definir las columnas del control DataGrid podemos utilizar distintos tipos de columnas, que se corresponden con las distintas etiquetas que se comentan a continuación:

- BoundColumn: en este caso se muestra el contenido de la columna como una etiqueta, y es el tipo de columna por defecto. Para indicar el campo de datos que se va a mostrar se hace uso de la propiedad DataField, más adelante veremos mediante ejemplos que esta propiedad es utilizada para la misma finalidad en diferentes tipos de columnas.

- HyperLinkColumn: el contenido de la columna se muestra como un hipervínculo, es decir, para mostrar los datos en la columna se utiliza un control Web HyperLink. Este tipo de columna se puede utilizar, por ejemplo, cuando deseamos mostrar información adicional del registro actual en una página distinta.

- ButtonColumn: este tipo de columna permite enviar un evento de pulsación de un elemento determinado del control DataGrid. El aspecto que va a mostrar esta columna es muy similar al de la columna anterior, pero permite lanzar eventos.

- TemplateColumn: en este otro tipo de columna podemos especificar los controles Web que deseemos que aparezcan en la columna correspondiente. Este tipo de columna también permite especificar la correspondencia en lo que a Data Binding se refiere de los campos de datos y las propiedades de los controles.

- EditCommandColumn: este último tipo de columna nos va a permitir editar, modificar y cancelar la edición de una fila determinada dentro del control DataGrid. Esta es una de las columnas más complejas, pero que nos ofrece una gran funcionalidad. Mediante las propiedades EditText, UpdateText y CancelText podremos indicar los texto que aparecenrán en los enlaces que van a permitir la edición, modificación y cancelación de edición, respectivamente, de una fila determinada del control de lista DataGrid.

A continuación vamos a ver mediante ejemplos la utilización de los distintos tipos de columnas que podemos utilizar en un control DataGrid.

En el Código fuente 156 se utilizan dos tipos de columnas, una serie de columnas BoundColumn y una columna HyperLinkColumn. Las columnas BoundColumn son utilizadas para definir las columnas nombre y apellidos del control DataGrid, y la columna HyperLinkColumn se utiliza para mostrar en una nueva ventana los detalles de la fila seleccionada, esta columna va a mostrar el identificador de cada uno de los registros que contiene el control de lista DataGrid.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {

      SqlConnection conexion =
            new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      SqlDataAdapter comando =
            new SqlDataAdapter("select * from Employees", conexion);
      DataSet ds = new DataSet();
      comando.Fill(ds, "Employees");
      tabla.DataSource = ds.Tables["Employees"].DefaultView;
      tabla.DataBind();
}
</script>

<body>
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow"
AutoGenerateColumns="False">

<Columns>
      <asp:HyperLinkColumn HeaderText="Detalles"
        DataNavigateUrlField="EmployeeID"
        DataNavigateUrlFormatString="detalles.aspx?id={0}"
        DataTextField="EmployeeID"
        Target="_blank"/>
      <asp:BoundColumn DataField="firstName" HeaderText="Nombre"/>
      <asp:BoundColumn DataField="lastName" HeaderText="Apellidos"/>
</Columns>

</asp:DataGrid>
</body>
</html>
```

Código fuente 156

Para definir cada una de las columnas BoundColumn se utilizan las propiedades DataField y HeaderText, mediante la primera indicamos el campo de datos que vamos a mostrar en la columna y mediante la segunda indicamos el texto de la cabecera de la columna.

La columna HyperLinkColumn también posee la propiedad HeaderText para indicar el texto del encabezado de la columna, pero además posee estas otras propiedades destacadas:

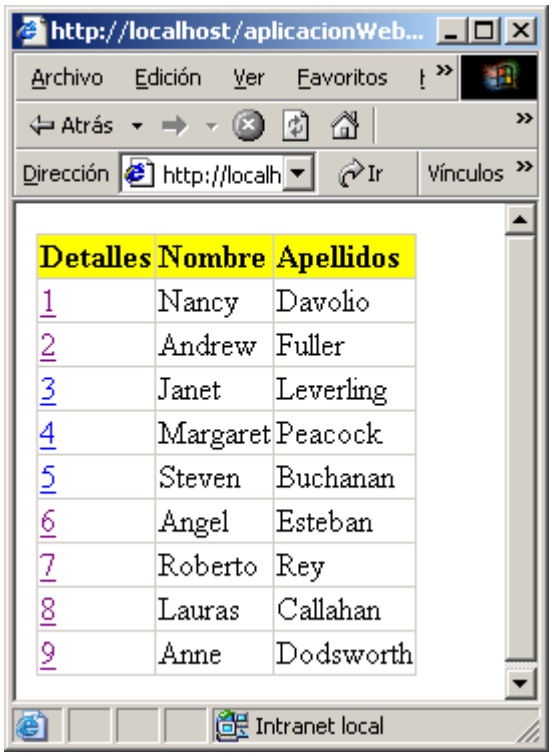
- DataNavigateUrlField: en esta propiedad indicaremos el campo que se va a utilizar dentro de la URL que apuntará a la nueva página en la que se mostrarán los detalles del campo seleccionado.

- DataNavigateUrlFormatString: aquí se indicará el formato de la cadena del enlace que nos llevará a la nueva página.

- DataTextField: esta propiedad contendrá el nombre del campo cuyo texto deseamos que aparezca en la columna.

- Target: en esta última propiedad indicaremos la forma en la que se abrirá la nueva ventana, los valores que puede tomar esta propiedad son los mismos que la propiedad del mismo nombre de una etiqueta <a> de HTML, es decir, _top, _blank, _parent, _search y _self.

El aspecto de esta página ASP .NET se puede ver en la Figura 91.



Detalles	Nombre	Apellidos
1	Nancy	Davolio
2	Andrew	Fuller
3	Janet	Leverling
4	Margaret	Peacock
5	Steven	Buchanan
6	Angel	Esteban
7	Roberto	Rey
8	Lauras	Callahan
9	Anne	Dodsworth

Figura 91

Y el Código fuente 157 es el código HTML que se generaría.

```
<html>
<body>
<table cellspacing="0" rules="all" border="1" id="tabla" style="border-
width:1px;border-style:solid;border-collapse:collapse;">
	<tr style="background-color:Yellow;font-weight:bold;">
		<td>
			Detalles
		</td><td>
			Nombre
		</td><td>
			Apellidos
		</td>
	</tr><tr>
		<td>
```

```
                    <a href="/aplicacionWeb/detalles.aspx?id=1"
target="_blank">1</a>
              </td><td>
                    Nancy
              </td><td>
                    Davolio
              </td>
       </tr><tr>
              <td>
                    <a href="/aplicacionWeb/detalles.aspx?id=2"
target="_blank">2</a>
              </td><td>
                    Andrew
              </td><td>
                    Fuller
              </td>
       </tr>
              […  …  …]
       <tr>
              <td>
                    <a href="/aplicacionWeb/detalles.aspx?id=8"
target="_blank">8</a>
              </td><td>
                    Lauras
              </td><td>
                    Callahan
              </td>
       </tr><tr>
              <td>
                    <a href="/aplicacionWeb/detalles.aspx?id=9"
target="_blank">9</a>
              </td><td>
                    Anne
              </td><td>
                    Dodsworth
              </td>
       </tr>
</table>
</body>
</html>
```

Código fuente 157

Pero antes de pasar a otro ejemplo de utilización de columnas dentro de un DataGrid nos faltaría ver y comentar la página ASP .NET que muestra los detalles del registro seleccionado mediante su columna HyperLinkColumn.

En la página de detalles (Código fuente 158) se recupera el identificador de empleado mediante la colección QueryString del objeto Request, este objeto y su colección serán muy familiares para aquellos lectores que conozcan ASP en alguna de sus versiones anteriores, pero para aquellos que no conozcan ASP adelantaremos que el objeto Request va a representar la petición realizada de una página ASP .NET y su colección QueryString nos va a permitir acceder a cada uno de los elementos presentes en la cadena de consulta de la página ASP .NET actual. Como pueden apreciar los lectores conocedores de versiones previas de la tecnología ASP, el objeto Request de ASP .NET, que va a ser una propiedad del objeto Page que representa al página ASP .NET actual, sigue teniendo la misma funcionalidad y significado que en versiones anteriores.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
```

```
<head>

<script language="C#" runat="server">
String id;
DataView datos;

void Page_Load(Object sender, EventArgs e) {
    id=Request.QueryString["id"];
    SqlConnection conexion =
              new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
       SqlDataAdapter comando =
              new SqlDataAdapter("select * from Employees Where EmployeeID=" + id,
conexion);
       DataSet ds = new DataSet();
       comando.Fill(ds, "Employees");
       datos= ds.Tables["Employees"].DefaultView;
}
</script>

</head>
<body>
<h4><font face="Verdana">Detalles del empleado <%= id %></u></font></h4>
       Nombre:<b><%= ((DataRowView)datos[0])["firstname"]%></b><br>
       Apellidos:<b><%= ((DataRowView)datos[0])["lastname"]%></b><br>
       Ciudad:<b><%= ((DataRowView)datos[0])["city"]%></b><br>
       País:<b><%= ((DataRowView)datos[0])["country"]%></b><br>
</body>
</html>
```

Código fuente 158

El objeto Request lo veremos más adelante cuando veamos la clase Page con sus distintas propiedades y métodos.

Una vez que tenemos el identificador del empleado seleccionado, se hace la consulta correspondiente mediante el código de acceso a datos necesario. No vamos a entrar en detalle en este código de ADO .NET, ya que lo veremos en futuros capítulos, lo que se hace con este código es realizar la sentencia SELECT de SQL que nos permite obtener el registro que se corresponde con el identificador de empleado seleccionado. En la Figura 92 se puede ver un ejemplo de ejecución de esta página de detalles de un empleado.

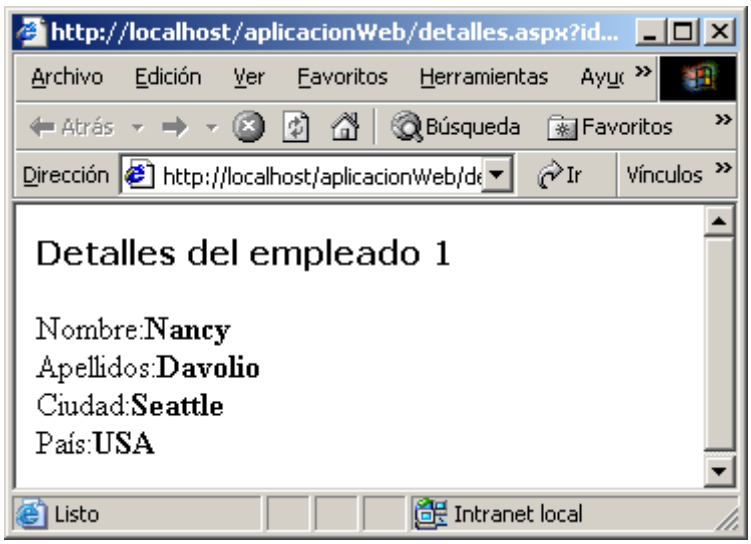


Figura 92

El código fuente de estas dos páginas se puede obtener en el siguiente enlace.

En el siguiente ejemplo vamos a ver la utilización de otros dos tipos de columnas, se trata de las columnas ButtonColumn y TemplateColumn, también se utilizarán columnas del tipo BoundColumn.

En la columna ButtonColumn vamos a mostrar un botón de tipo enlace, es decir, un control LinkButton, al pulsar sobre el botón se indicará en un objeto Label el nombre del empleado que ha sido seleccionado del DataGrid. En la propiedad DataTextField de la columna BoundColumn indicaremos el campo que deseamos mostrar en dicha columna y en la propiedad ButtonType el tipo de botón que va a ser empleado, los valores que puede tomar esta propiedad son dos: LinkButton (valor por defecto) y PushButton.

Al pulsar el botón se ejecutará el método indicado en la propiedad OnItemCommand del control DataGrid, en este caso este método es muy sencillo, únicamente asigna a la propiedad Text de un control Label el nombre y apellidos de un empleado. En el Código fuente 159, que es el código de este nuevo ejemplo, se puede apreciar en el método botonPulsado() la forma en la que se puede obtener el texto de una celda del registro seleccionado, a través de la propiedad Item del objeto de la clase DataGridCommandEventArgs tenemos acceso a la colección Cells que contiene las celdas del registro actual, de cada celda que nos interese podemos acceder a su propiedad Text, así obtendremos los nombres y los apellidos de los empleados.

```
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Data" %>

<HTML>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {

       SqlConnection conexion =
              new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
       SqlDataAdapter comando =
              new SqlDataAdapter("select * from Employees", conexion);
       DataSet ds = new DataSet();
       comando.Fill(ds, "Employees");
       tabla.DataSource = ds.Tables["Employees"].DefaultView;
       tabla.DataBind();
}

void botonPulsado(Object sender, DataGridCommandEventArgs e) {
              String nombre=e.Item.Cells[1].Text;
              String apellidos=e.Item.Cells[2].Text;
        seleccionado.Text="Se ha seleccionado el empleado: "+nombre+" "+apellidos;
}
</script>

<body>
<form runat="server" id="formulario">

<asp:Label Runat="server" ID="seleccionado" Font-Size="Medium"
Font-Bold="True" Font-Name="Verdada" Font-Names="Verdada"></asp:Label>

<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow"
AutoGenerateColumns="False" OnItemCommand="botonPulsado">

<Columns>

       <asp:ButtonColumn DataTextField="EmployeeID"
       HeaderText="Seleccionar" ButtonType="LinkButton"/>
```

```
        <asp:BoundColumn DataField="firstname" HeaderText="nombre"/>
        <asp:BoundColumn DataField="lastname" HeaderText="apellidos"/>

        <asp:TemplateColumn HeaderText="Dirección">
                <ItemTemplate>
                        <%# ((DataRowView)Container.DataItem)["address"]%><br>
                        <%# ((DataRowView)Container.DataItem)["city"]%><br>
                        <%# ((DataRowView)Container.DataItem)["region"]%><br>
                        <%# ((DataRowView)Container.DataItem)["country"]%>
                </ItemTemplate>
        </asp:TemplateColumn>

</Columns>

</asp:DataGrid>
</form>
</body>
</HTML>
```

Código fuente 159

La columna TemplateColumn la vamos a utilizar para definir la columna dentro del control DataGrid que va a mostrar la dirección completa de cada empleado de la tabla. Como se puede ver en el se utiliza una etiqueta <ItemTemplate> para definir el elemento de la columna, es decir, las columnas del tipo TemplateColumn nos van a permitir utilizar el mecanismo de plantillas de la misma forma que lo hacíamos con el control DataList dentro de este mismo capítulo.

En la Figura 93 se puede ver un ejemplo de ejecución de este nuevo DataGrid, y en el siguiente enlace se puede obtener su código fuente completo.

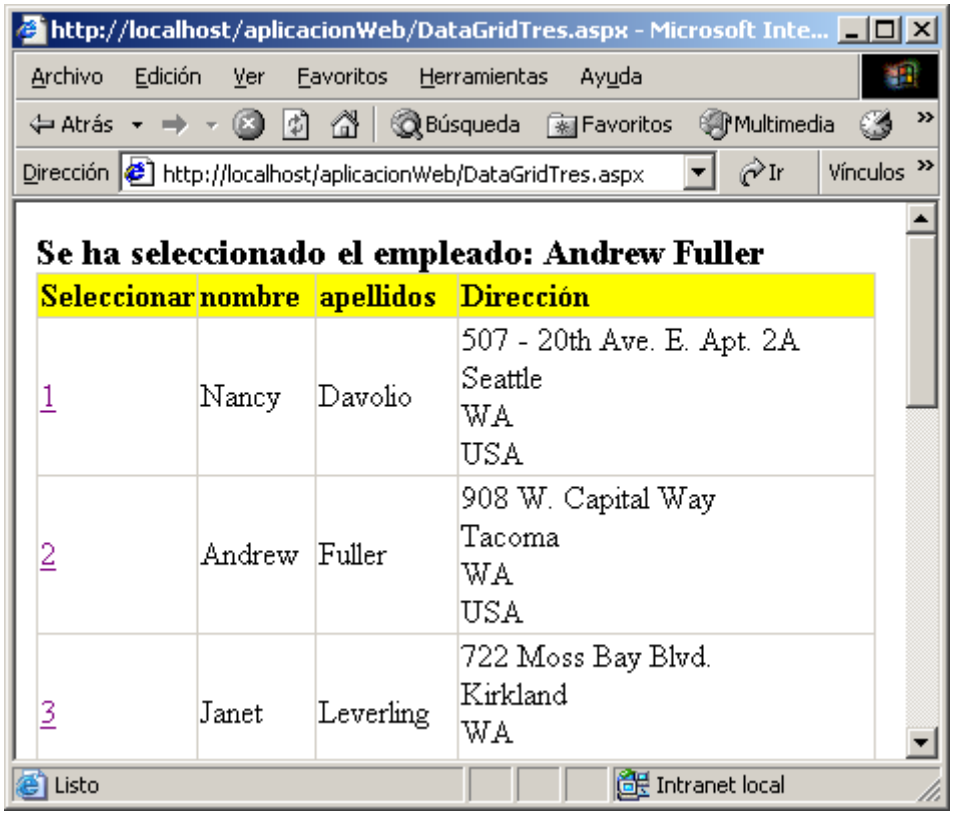


Figura 93

El último ejemplo de utilización de distintos tipos de columnas lo vamos a mostrar a continuación, utilizando el tipo de columna EditCommandColumn y una serie de columnas BoundColumn. La columna EditCommandColumn nos va a permitir editar los campos que deseemos dentro de una fila determinada del control DataGrid.

La edición dentro del control DataGrid con el tipo de columna EditCommandColumn es muy similar a la edición con la plantilla EditItemTemplate dentro del control DataList, es decir, vamos a indicar el elemento que se está editando asignando a la propiedad EditItemIndex del control DataGrid el valor de la propiedad ItemIndex del objeto Item, que representará al elemento que se ha seleccionado para la edición.

La columna EditCommandColumn nos ofrece las propiedades EditText, CancelText y UpdateText para indicar los textos que van a parecer en los distintos botones de enlace, que nos permitirán realizar las operaciones pertinentes, es decir, edición, cancelación de edición y actualización de un elemento del control DataGrid.

Al igual que sucedía con el control DataList, en el control DataGrid vamos a dispones de los manejadores de eventos OnEditCommand, OnCancelCommand y OnUpdateCommand. Como se puede ver en el Código fuente 160, la edición de un elemento de un DataGrid mediante una columna EditCommandColumn es muy similar a la edición de un control DataList mediante una plantilla EditItemTemplate.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
      estableceDataBinding();
}

void estableceDataBinding(){
      SqlConnection conexion =
            new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      SqlDataAdapter comando =
            new SqlDataAdapter("select * from Employees", conexion);
      DataSet ds = new DataSet();
      comando.Fill(ds, "Employees");
      tabla.DataSource = ds.Tables["Employees"].DefaultView;
      tabla.DataBind();

}

void pulsadoEdicion(object Sender, DataGridCommandEventArgs e) {
      tabla.SelectedIndex=-1;
      tabla.EditItemIndex = e.Item.ItemIndex;
      estableceDataBinding();
}

void pulsadoCancelacion(object Sender, DataGridCommandEventArgs e) {
      tabla.EditItemIndex=-1;
      tabla.SelectedIndex=-1;
      estableceDataBinding();
}

void pulsadoActualizar(object Sender, DataGridCommandEventArgs e) {
      SqlConnection conexion =
            new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      String strComando;
      String txtNombre=((TextBox)e.Item.Cells[1].Controls[0]).Text;
```

```
        String txtApellidos=((TextBox)e.Item.Cells[2].Controls[0]).Text;
        strComando="UPDATE Employees SET firstname='" + txtNombre + "', lastname='" +
         txtApellidos + "'WHERE EmployeeID=" + tabla.DataKeys[(int)e.Item.ItemIndex];
         SqlCommand comando = new SqlCommand(strComando, conexion);
        comando.Connection.Open();
        comando.ExecuteNonQuery();
        tabla.EditItemIndex=-1;
        tabla.SelectedIndex=-1;
        estableceDataBinding();
}

</script>

<body>
<form runat="server">
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow"
AutoGenerateColumns="False" OnEditCommand="pulsadoEdicion"
OnCancelCommand="pulsadoCancelacion"  OnUpdateCommand="pulsadoActualizar"
DataKeyField="employeeid">

<Columns>
        <asp:EditCommandColumn EditText="Editar" CancelText="Cancelar"
     UpdateText="Actualizar"  HeaderText="Edición"/>
        <asp:BoundColumn DataField="firstName" HeaderText="Nombre"/>
        <asp:BoundColumn DataField="lastName" HeaderText="Apellidos"/>
        <asp:BoundColumn DataField="city" HeaderText="Ciudad" ReadOnly="True"/>
</Columns>

</asp:DataGrid>
</form>
</body>
</html>
```

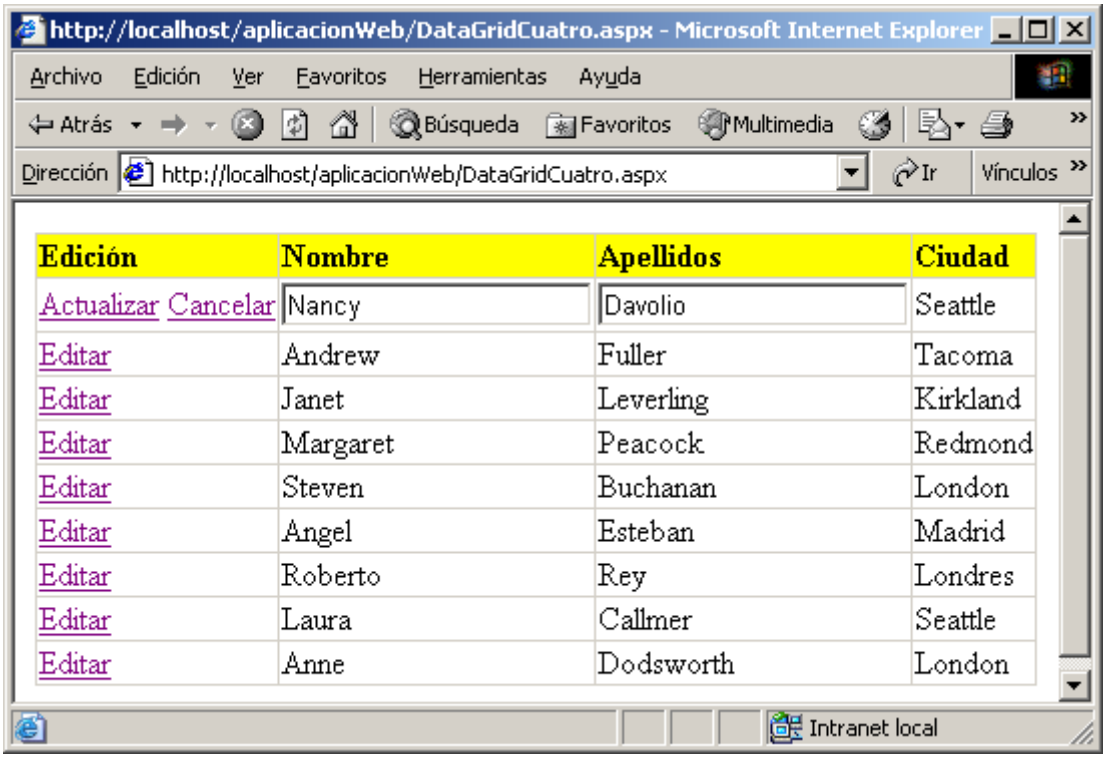
Código fuente 160

En el código anterior se puede ver que hay una columna BoundColumn que no hemos deseado editar, para ello se le ha asignado a su propiedad ReadOnly el valor False. También cabe destacar la forma que tenemos de recuperar los valores de los campos de texto para llevar a cabo la actualización del registro en la base de datos. Utilizamos la colección Cells, que a su vez tiene la colección Controls, la colección Controls contiene todos los controles Web existentes dentro de una celda determinada, debido a que únicamente existe una caja de texto (control TextBox), accedemos a la misma mediante el índice cero.

Cuando se pulsa el botón de enlace de edición aparecen en la columna EditCommandColumn otros dos nuevos botones de enlace que permitir cancelar o realizar la actualización de los campos del registro editado. El aspecto de este ejemplo se puede ver en la Figura 94 y el código fuente completo se encuentra disponible en el siguiente enlace.

Se debe observar que el control DataGrid se encuentra dentro de un Web Form, esto es necesario para que las pulsaciones sobre los distintos botones de enlace se puedan llevar a cabo sin problemas.

Con este ejemplo se da por terminado el apartado dedicado a la definición de columnas en un control DataGrid, a continuación vamos a tratar otras operaciones que se pueden realizar con las columnas.



Edición	Nombre	Apellidos	Ciudad
Actualizar Cancelar	Nancy	Davolio	Seattle
Editar	Andrew	Fuller	Tacoma
Editar	Janet	Leverling	Kirkland
Editar	Margaret	Peacock	Redmond
Editar	Steven	Buchanan	London
Editar	Angel	Esteban	Madrid
Editar	Roberto	Rey	Londres
Editar	Laura	Callmer	Seattle
Editar	Anne	Dodsworth	London

Figura 94

Otras operaciones con columnas

Es posible ocultar y mostrar una columna o columnas determinadas según nos interese, para ello únicamente debemos acceder a la propiedad Visible de la columna correspondiente. Esta propiedad puede tener los valores True/False, para mostrar u ocultar la columna respectivamente, por defecto tiene el valor True.

En el Código fuente 161 se muestra un ejemplo de utilización de la propiedad Visible de las columnas de un DataGrid, se trata de tres controles Web de la clase LinkButton que al ser pulsados ocultarán o mostrarán la columna correspondiente. Se debe prestar atención a la forma que tenemos de diferenciar el botón que ha sido pulsado, la propiedad utilizada para discernir el botón de enlace que ha sido pulsado es la propiedad CommandName, también merece la pena apreciar el mecanismo de casting que ha sido utilizado para acceder a esta propiedad.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {

      SqlConnection conexion =
            new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      SqlDataAdapter comando =
            new SqlDataAdapter("select * from Employees", conexion);
      DataSet ds = new DataSet();
      comando.Fill(ds, "Employees");
      tabla.DataSource = ds.Tables["Employees"].DefaultView;
      tabla.DataBind();
}
```

```
void pulsadoBoton(Object obj, EventArgs e){
	if (((LinkButton)obj).CommandName=="C1"){
		tabla.Columns[0].Visible = !tabla.Columns[0].Visible;
		if(tabla.Columns[0].Visible){
			botonC1.Text="Oculta C1";
		}else{
			botonC1.Text="Muestra C1";
		}
	}else{
		if (((LinkButton)obj).CommandName=="C2"){
			tabla.Columns[1].Visible = !tabla.Columns[1].Visible;
			if(tabla.Columns[1].Visible){
				botonC2.Text="Oculta C2";
			}else{
				botonC2.Text="Muestra C2";
			}
		}else{
			if (((LinkButton)obj).CommandName=="C3"){
				tabla.Columns[2].Visible = !tabla.Columns[2].Visible;
				if(tabla.Columns[2].Visible){
					botonC3.Text="Oculta C3";
				}else{
					botonC3.Text="Muestra C3";
				}
			}
		}
	}
}
</script>

<body>
<form runat="server">
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow"
AutoGenerateColumns="False">

<Columns>
	<asp:BoundColumn DataField="firstName" HeaderText="Nombre"/>
	<asp:BoundColumn DataField="lastName" HeaderText="Apellidos"/>
	<asp:BoundColumn DataField="city" HeaderText="Ciudad"/>
</Columns>

</asp:DataGrid>
<asp:LinkButton runat="server" id="botonC1" Text="Oculta C1"
                               CommandName="C1" OnClick="pulsadoBoton"/>
<asp:LinkButton runat="server" id="botonC2" Text="Oculta C2"
                               CommandName="C2" OnClick="pulsadoBoton"/>
<asp:LinkButton runat="server" id="botonC3" Text="Oculta C3"
                               CommandName="C3" OnClick="pulsadoBoton"/>
</form>
</body>
</html>
```

Código fuente 161

En la Figura 95 se puede ver un ejemplo de ejecución de este ejemplo, y en este enlace se puede obtener el código de la página ASP .NET.

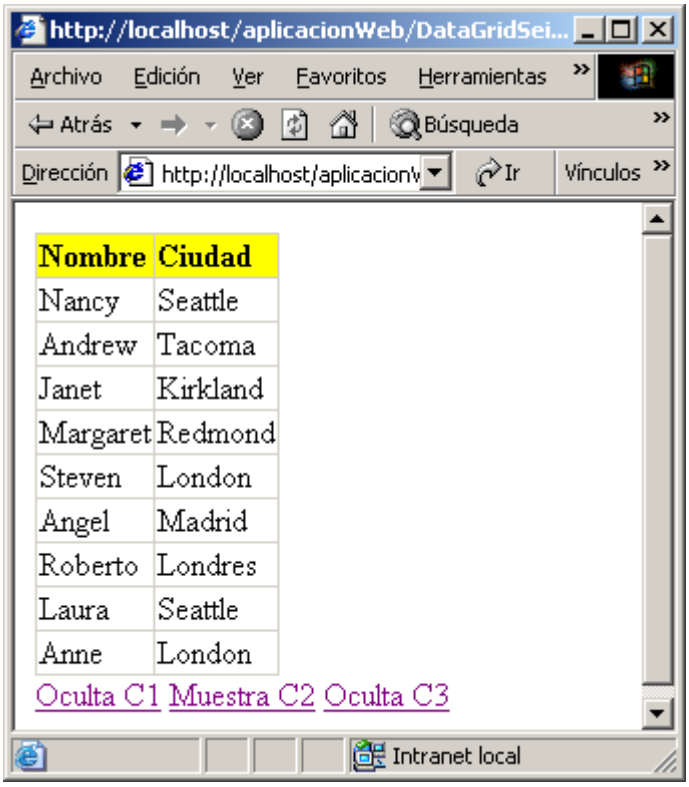


Figura 95

Otra operación que podemos realizar con una columna es ordenar el control DataGrid atendiendo a una columna determinada. Para poder ordenar un control DataGrid mediante una columna determinada, se debe asignar a la propiedad AllowSorting el valor True. Al asignar este valor a la propiedad AllowSorting las cabeceras de las columnas del control DataGrid aparecerán con enlaces, al pulsar cada uno de los enlaces se ejecutará el método indicado en la propiedad OnSortCommand del control DataGrid.

Una vez que se ha pulsado la columna, al manejar nosotros el evento podemos indicar que se ordene la fuente de datos atendiendo a la columna seleccionada del DataGrid, esto se puede observar en el Código fuente 162.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
String columnaOrdenacion="";
void Page_Load(Object sender, EventArgs e) {
       if (!IsPostBack) {
              if (columnaOrdenacion == "") {
                     columnaOrdenacion = "firstname";
         }
         estableceDataBinding();
     }
}

void estableceDataBinding(){
       SqlConnection conexion =
              new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
       SqlDataAdapter comando =
```

```
              new SqlDataAdapter(
              "select firstName,lastName,city,country from Employees", conexion);
       DataSet ds = new DataSet();
       comando.Fill(ds, "Employees");
       ds.Tables["Employees"].DefaultView.Sort=columnaOrdenacion;
       tabla.DataSource = ds.Tables["Employees"].DefaultView;
       DataBind();
}

void pulsadoOrdenar(Object sender, DataGridSortCommandEventArgs e) {
   columnaOrdenacion= (string)e.SortExpression;
   estableceDataBinding();
}

</script>

<body>
<form runat="server">
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow"  AllowSorting="True"
 OnSortCommand="pulsadoOrdenar"/>
 Ordenado por la columna
 <asp:Label runat="server" id="orden" Text="<%# columnaOrdenacion %>"></asp:Label>
</form>
</body>
</html>
```

Código fuente 162

Para ordenar la fuente de datos asignamos a la propiedad Sort de la vista de datos DeafaultView el nombre de la columna que ha sido pulsada del DataGrid, sobre las vistas de datos veremos más ejemplos en el capítulo dedicado al acceso a datos mediante ADO .NET. La columna que se utiliza para ordenar se obtiene a partir de la propiedad SortExpression de la clase DataGridSortCommandEventArgs.

En la Figura 96 se puede observar el aspecto de este ejemplo.

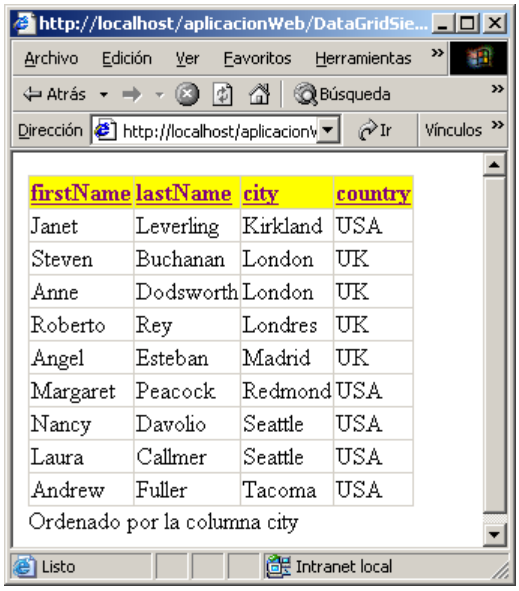


Figura 96

Hasta aquí se han comentado las distintas operaciones que se pueden hacer con las columnas de un control DataGrid, en el siguiente subapartado veremos otra funcionalidad que ofrece este interesante control, se trata de la paginación de registros.

Paginación de un control DataGrid

El mecanismo de paginación es muy común a la hora de mostrar datos en forma de listados, se trata de ir mostrando y agrupando los elementos a mostrar en un número determinado, cada grupo de elementos se va a denominar página. Normalmente en la paginación se ofrecen una serie de controles que nos vana permitir desplazarnos por las distintas páginas de datos.

El control DataGrid nos permite la posibilidad de paginar los datos que contiene, para activar la paginación dentro de un control DataGrid debemos asignar a su propiedad AllowPaging el valor True. El control DataGrid posee un control de navegación para la paginación denominado Pager, por defecto se ofrece un par de botones de enlaces que nos permiten movernos a la siguiente y anterior páginas de registros.

Aunque el control DataGrid ofrece muchas facilidades a la hora de establece la paginación sobre los datos, el evento de cambio de página lo tenemos que gestionar nosotros a través del manejador de eventos OnPageIndexChanged, para cambiar de página debemos asignar a la propiedad CurrentPageIndex de control DataGrid el valor de la propiedad NewPageIndex del objeto de la clase DataGridPageChangedEventArgs, esta clase va a permitir acceder a las propiedades del evento de cambio de página actual dentro de un control DataGrid.

A continuación se va a pasar a comentar una serie de propiedades del control DataGrid que se encuentran relacionadas con el mecanismo de paginación:

- OnPageIndexChanged: en esta propiedad deberemos indicar el nombre del método que deseamos que se ejecute cuando se produce el evento de cambio de página (PagIndexChanged), es decir, cuando se pulsa sobre alguno de los botones de navegación del control DataGrid.

- PagerStyle-NextPageText: existen una serie de propiedades relativas al control de paginación que se utiliza por defecto en el control DataGrid que permiten configuración el aspecto de éste. Esta propiedad indica el texto que se va a mostrar en el botón de enlace que nos va a permitir desplazarnos a la página siguiente de un control DataGrid.

- PagerStyle-PrevPageText: esta propiedad va a contener el texto que se va a mostrar en el botón de enlace que nos va a permitir desplazarnos a la página siguiente dentro de un control DataGrid. Estas dos propiedades tienen sentido cuando estamos utilizando el control de paginación en el modo Siguente/Anterior (NextPrev).

- PagerStyle-Mode: esta propiedad permite indicar el modo utilizado dentro del control de paginación. Podemos asignarle dos valores a esta propiedad, NextPrev, que es el valor por defecto y que ofrece una sencilla navegación que permite desplazarnos a la página siguiente y a la anterior, el segundo valor es NumericPages, este otro modo del control Pager va a mostrar una serie de botones de enlace con los números de página existentes y al pulsar cada uno de ellos nos desplazaremos a la página correspondiente.

- PagerStyle-PageButtonCount: cuando nos encontramos en el modo NumericPages del control de navegación para la paginación, podemos utilizar esta propiedad para indicar el número de botones de enlace que deseamos mostrar de una vez en el control Pager.

En el Código fuente 163 se muestra un ejemplo que consiste en aplicar la paginación a un control DataGrid. En este ejemplo se ofrece un control CheckBox que nos va a permitir cambiar dinámicamente el modo del control Pager. Se debe mostrar atención al método cambiaPagina() ya que en el mismo se muestra la forma que tenemos de cambiar de página dentro del control DataGrid.

En este ejemplo se muestra dentro de un control Web de la clase Label, a modo informativo, el número de página actual sobre el de páginas totales.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
      if (check.Checked) {
            tabla.PagerStyle.Mode=PagerMode.NumericPages;
            tabla.PagerStyle.PageButtonCount=4;
    }
    else{
        tabla.PagerStyle.Mode=PagerMode.NextPrev;
    }
      if (!IsPostBack){
            estableceDataBinding();
      }
}

void estableceDataBinding(){
      SqlConnection conexion =
            new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      SqlDataAdapter comando =
            new SqlDataAdapter("select productname,quantityperunit,unitsinstock "
+

            " from Products", conexion);
      DataSet ds = new DataSet();
      comando.Fill(ds, "Products");
      tabla.DataSource = ds.Tables["Products"].DefaultView;
      tabla.DataBind();
      estadoActual.Text=(tabla.CurrentPageIndex+1) + " de " + tabla.PageCount;

}

void cambiaPagina(Object sender, DataGridPageChangedEventArgs e) {
      tabla.CurrentPageIndex = e.NewPageIndex;
    estableceDataBinding();
}

</script>

<body>
<form runat="server">

<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
 HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow"
AutoGenerateColumns="False"
 AllowPaging="True" PageSize="10"  OnPageIndexChanged="cambiaPagina"
 PagerStyle-Font-Name="arial" PagerStyle-Font-Size="8"
 PagerStyle-HorizontalAlign="Center" PagerStyle-Mode="NextPrev"
 PagerStyle-NextPageText="Siguiente->" PagerStyle-PrevPageText="<-Anterior">

<Columns>
      <asp:BoundColumn DataField="productName" HeaderText="Producto"/>
      <asp:BoundColumn DataField="quantityperunit" HeaderText="Presentación"
```

```
      ItemStyle-HorizontalAlign="Right"/>
      <asp:BoundColumn DataField="unitsinstock" HeaderText="unidades"
       ItemStyle-HorizontalAlign="Right"/>
</Columns>

</asp:DataGrid>

<asp:Label runat="server" id="estadoActual"/>
<asp:Checkbox id="check" runat="server"
Text="Muestra botones numéricos de paginación" Font-Name="Verdana"
Font-Size="8pt" AutoPostBack="true"/>

</form>
</body>
</html>
```

Código fuente 163

En la Figura 97 se puede ver el resultado de la paginación del control DataGrid del ejemplo. En el siguiente enlace se puede obtener el código completo de este ejemplo.

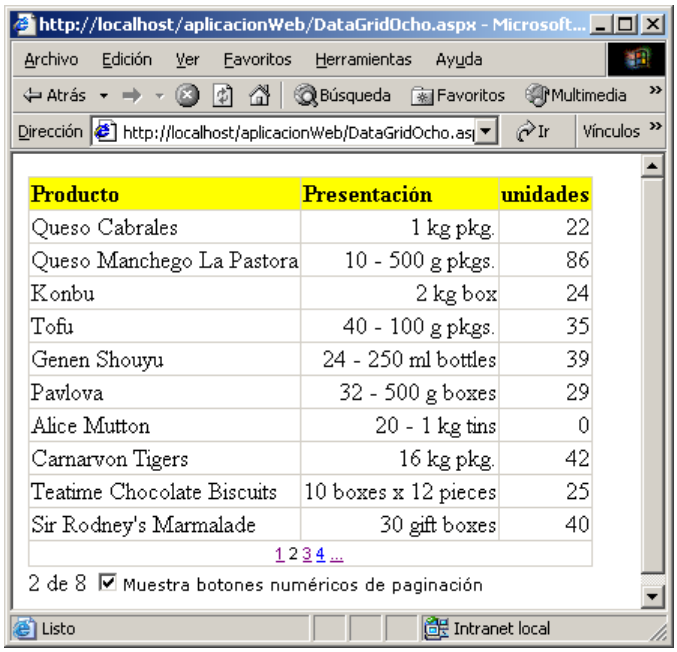


Figura 97

Con el control DataGrid se da por terminado el conjunto de capítulos dedicados al entorno ofrecido por el modelo de programación de los Web Forms. A lo largo de cinco capítulos hemos ido comentando con distintos ejemplos los distintos elementos presentes en el entorno ofrecido por los Web Forms.

Recomiendo al lector que consulte los apartados llamados “Web Forms Controls Reference” y “Web Forms Syntax Reference”, presentes en el tutorial “ASP .NET QuickStart Tutorial” ofrecido en la instalación de la plataforma .NET, ya que son una referencia muy interesante de los controles Web y de los Web Forms en general.

En los siguientes capítulos vamos a tratar detenidamente la clase Page, es decir, la clase que representa a la página ASP .NET actual. En estos capítulos veremos las distintas propiedades y métodos que ofrece la clase Page, así como las directivas que se pueden utilizar dentro de una página.

12

La clase Page

Introducción

Cada vez que una página ASP .NET es demandada desde el servidor Web, es decir, un cliente a través de un navegador Web solicita una URL que indica un fichero .ASPX en el servidor, se crean una serie de componentes que constituyen la página solicitada y que se compilan en una unidad. Estos componentes pueden ser: el fichero .ASPX que ha sido solicitado, la clase .NET que contiene el código de la página, controles de usuario de la página, etc.

La unidad formada por los distintos componentes compilados da lugar a un objeto de la clase Object.Contro.TemplateControl.Page, esta clase que se crea de forma dinámica se instanciará cada vez que se realice una petición del fichero .ASPX. Una vez que se ha instanciado un objeto de la clase esta clase, este objeto será utilizado para procesar las sucesivas peticiones y devolver los datos correspondientes al cliente que haya realizado la solicitud.

El proceso de compilación de la página únicamente se realiza cuando se modifica el código fuente de la página ASP .NET, de esta forma el proceso resultante es bastante eficiente.

Dentro del código de una página ASP .NET siempre vamos a tener acceso a la clase Page, esta clase va a servir como contenedor de todos los componentes que constituyen la página, nos permite el acceso a distintos aspectos de la página, como pueden ser sus propiedades, métodos y eventos.

En resumen y de manera sencilla podemos decir que el objeto instancia de la clase Page, va a representar a la página ASP .NET actual.

A lo largo del presente capítulo vamos a comentar los distintos eventos, propiedades y métodos que nos ofrece la clase Page, también veremos otros aspectos relacionados con la clase Page, como pueden ser la directivas de la página y la separación de código fuente.

Eventos de la página

ASP .NET ofrecen una serie de eventos que nos permiten estructurar el código en las páginas, estos eventos no existían en versiones anteriores de ASP. Los eventos se producen en un orden determinado o en un momento determinado.

A continuación se van a enumerar los distintos eventos que posee la clase Page:

- Init: este evento es lanzado cuando la página es inicializada, cuando es lanzado este evento todavía no se han creado por completo los distintos controles de la página. Este evento es tratado en el método Page_Init.

- Load: este evento se lanzaría a continuación del método Init y es lanzado cuando la página se ha cargado, en este caso todos los controles de la página ya han sido creados. Este evento se lanzará cada vez que la página ASP .NET es ejecutada. Este evento es tratado en el método Page_Load.

- PreRender: el evento se lanzará justo antes de enviar la información al cliente. Este evento es tratado en el método Page_PreRender, y siempre es lanzado después del evento Load.

- UnLoad: este otro evento se lanzará en último lugar, y tiene lugar cuando la página ha finalizado de procesarse, es decir, cuando se ha terminado la ejecución de la página y toda la información ha sido enviada al cliente. Este evento es tratado en el método Page_UnLoad.

- AbortTransaction: los cuatro eventos anteriores se ejecutarán siempre, sin embargo los tres eventos que quedan por comentar se producirán en un momento determinado cuando un cierta situación tenga lugar. Este nuevo evento serán lanzado cuando la transacción en la que está participando la página actual es abortado.

- CommitTransaction: este evento se lanzará cuando la transacción en la que participe la página se lleve a cabo sin errores.

- Error: este evento se lanzará cuando se produzca una excepción no tratada dentro de la página. El método Page_Error se utilizará cuando deseemos realizar nuestro propio tratamiento de errores, esto lo veremos más en detalle en el capítulo dedicado al tratamiento de errores y depuración.

En el Código fuente 164 se muestra una sencilla utilización de algunos de estos métodos. El ejemplo muestra una serie de mensajes en el cliente para ilustrar el orden de ejecución de los distintos eventos.

```
<html>
<script language="C#" runat="server">
void Page_Init(Object sender, EventArgs e) {
      Page.Response.Write("Evento Init<br>");
}

void Page_Load(Object sender, EventArgs e) {
      Page.Response.Write("Evento Load<br>");
}
```

```
void Page_UnLoad(Object sender, EventArgs e) {

}

void Page_PreRender(Object sender, EventArgs e) {
      Page.Response.Write("Evento PreRender<br>");
}
</script>
<body>
<asp:Label runat="server" id="texto">Eventos de la página</asp:Label>
</body>
</html>
```

Código fuente 164

En la Figura 98 se puede ver el resultado de la ejecución de esta página ASP. NET.

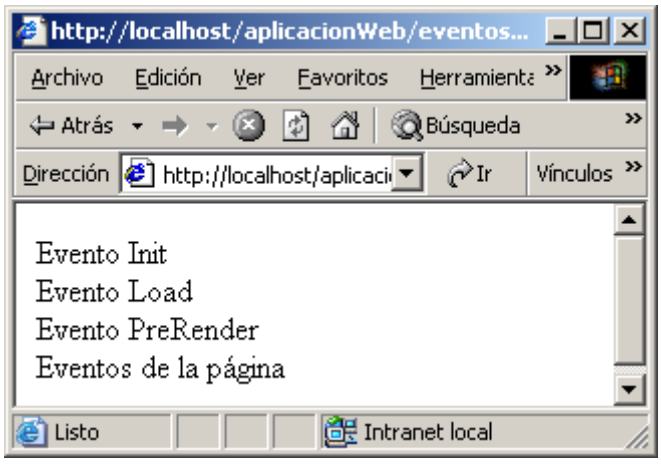


Figura 98

En este código de ejemplo accedemos a una propiedad de la clase Page llamada Response, que es una referencia a un objeto de la clase System.Web.HttpResponse, este objeto permite acceder a la respuesta del protocolo HTTP que se envía al cliente, en este caso el cliente va a ser un navegador Web, es decir, permite manipular la salida que se va a enviar al navegador, en este caso utilizamos el método Write() para escribir una cadena que identifique el evento que se ha producido.

A los lectores conocedores de versiones anteriores de ASP les resultará familiar la propiedad Response de la clase Page, ya que se corresponde con el objeto integrado Response de las otras versiones de la tecnología ASP, más adelante veremos en detalle esta propiedad de la clase Page.

Pero volvamos al tema principal de este apartado que son eventos de la clase Page. A la hora de utilizar los distintos métodos que manejan los ya comentados eventos de la página hay que tener en cuenta una serie de puntos:

- El evento Load se lanza cada vez que la página ASP .NET se ejecuta.
- Una vez cargada la página se pueden ir lanzado los distintos eventos que pueden ser emitidos por los distintos controles Web presentes en la página actual.
- Durante el evento Init se puede acceder a los controles Web de la página ASP .NET, pero estos controles tendrán sus valores por defecto no los valores anteriores que hayan podido ser

establecidos antes de que se produjera el envío de la página, es decir, no se ha conservado el estado de los controles del formulario ya que todavía no se encuentra el campo VIEWSTATE del formulario Web, recuerdo a los que este campo se trata de un campo oculto del formulario denominado _VIEWSTATE que permite mantener el estado entre distintas llamadas, es decir, en distintas peticiones se va conservando el valor de los campos del formulario.

- En el evento UnLoad no se tiene acceso al objeto Page, ya que este evento se produce cuando se destruye dicho objeto.

Con estas consideraciones terminamos el apartado dedicado a los eventos de la página ASP .NET, a continuación vamos a comentar las distintas propiedades de la clase Page.

Propiedades de la clase Page

Para acceder a una propiedad de la clase Page lo podemos hacer de dos formas: indicando delante de la propiedad que se trata de la clase Page o bien utilizando la propiedad directamente sin anteponer nada delante de la misma, en el se muestran estas dos posibilidades, se debe indicar que no existe ningún tipo de mejora el rendimiento si se utiliza una u otra de las dos posibilidades.

```
Page.Response.Write("Es equivalente");
Response.Write("Es equivalente");
```

Código fuente 165

A continuación se va a comentar brevemente las distintas propiedades que nos ofrece la clase Page, no vamos a comentar en detalle cada una de ellas, sino que más adelante, a mediada que se siga avanzando en el desarrollo del texto, se comentarán las más relevantes, aquí únicamente las vamos a enumerar a modo de referencia rápida y para tener una visión completa y general de la clase que nos ocupa.

Siempre que sea aplicable y necesario se comentará la correspondencia de la propiedad con el antiguo modelo de objetos integrado que presentaban las versiones anteriores de ASP. Esto se hace así para que los lectores que hayan utilizado alguna versión previa de ASP se sientan más cómodos con la versión ASP .NET.

Teniendo en cuenta todo lo anterior, podemos decir que las propiedades principales de la clase Page son las siguientes:

- Application: esta propiedad nos va a ofrecer una referencia a un objeto de la clase System.Web.HttpApplicationState, este objeto nos va a permitir almacenar y acceder a información que va a ser común a toda la aplicación Web, es decir, es una información compartida por todos los clientes de una aplicación Web determinada. Esta propiedad es equivalente al objeto integrado Application de anteriores versiones de ASP.

- Cache: esta propiedad permite acceder a un objeto de la clase System.Web.Caching.Cache. La propiedad Cache no tiene ninguna equivalencia con los objetos integrados de las anteriores versiones de ASP, ya que ofrece una nueva funcionalidad que permite almacenar información para que pueda ser mantenida durante distintas peticiones sobre distintas páginas ASP .NET, esta información será global para toda la aplicación. Pero también existe un segundo tipo de caché denominado caché de salida, que no utiliza el objeto Cache, sino que almacena la salida

generada por las páginas ASP, y utiliza esta copia almacenada para las siguientes peticiones de la página.

- ClientTarget: esta propiedad de la clase String nos permite sobrescribir la detección automática de navegador Web ofrecida por ASP .NET, de esta forma podemos indicar un cliente Web específico con unas propiedad y capacidades determinadas.

- Context: propiedad de la clase System.Web.HttpContext, esta propiedad ofrece información sobre la solicitud actual de una página, esta clase va a encapsular toda la información referente a una petición HTTP individual. A través de esta propiedad también podemos tener acceso a distintos objetos de ASP .NET que también son propiedades de ASP .NET, como ocurre con los objetos Application o Cache. Pero lo más común, en el entorno de las páginas ASP .NET, es obtener referencia a estos objetos a través de las propiedades que nos ofrece la clase Page que aquí estamos tratando.

- EnableViewState: propiedad de tipo Boolean que nos permite desactivar el mantenimiento del estado entre distintas llamadas de los controles Web de la página. Su valor por defecto es true.

- ErrorPage: propiedad de la clase String que permite indicar la página de error que se va a utilizar en el caso de que se produzca una excepción no tratada en la página ASP .NET actual. Por defecto esta propiedad no tiene ningún valor, por lo que se utiliza una página ASP .NET genérica para el tratamiento de errores.

- IsPostBack: esta propiedad de la clase Bolean devolverá true si la página ASP .NET actual ya ha sido enviada al servidor en alguna ocasión. Si tiene el valor false indicará que la página es la primera vez que se carga y nos servirá de indicador para poder inicializar los controles Web de la página o bien realizar otras labores de inicialización. Esta propiedad ya la hemos utilizado en diversos ejemplos dentro del entorno de los Web Forms.

- IsValid: esta propiedad también de tipo Bolean, presentará el valor verdadero si todos los controles de validación presentes en la página actual han realizado sus operaciones de validación con éxito. Si alguna de las validaciones de los controles Web de validación falla esta propiedad tendrá el valor falso.

- Request: al igual que sucedía con otras propiedades ya comentadas de la clase Page, esta propiedad ofrece una referencia a un antiguo objeto integrado del modelo de objetos de versiones anteriores de ASP, en este caso nos devuelve una instancia de un objeto de la clase System.Web.HttpRequest. Este objeto va a permitir acceder a todo la información necesaria de la petición del protocolo HTTP que ha sido utilizada para demandar la página de el servidor Web.

- Response: nos seguimos encontrando con referencias a los objetos integrados de versiones anteriores de ASP. El objeto de la clase System.Web.HttpResponse nos va a permitir manipular la respuesta devuelta al cliente que ha realizado la petición sobre la página ASP .NET actual, este objeto representa la respuesta HTTP que se envía al cliente.

- Server: ofrece una referencia a un objeto de la clase System.Web.HttpServerUtility, este objeto tiene la misma funcionalidad que el objeto Server de anteriores versiones de ASP, es decir, es un compendio de utilidades que permiten realizar una serie de operaciones sobre las peticiones recibidas. Una de las funcionalidades que se ofrecía como principal en las versiones anteriores de ASP, ahora en ASP .NET no tiene dicha importancia se trata de la posibilidad de crear componentes existentes en el servidor, ya hemos visto que en ASP .NET los objetos se crean de manera distinta, ahora los instanciamos mediante el operador new e importando el espacio con nombre de sea necesario.

- Session: al igual que sucedía con las últimas propiedades descritas, esta propiedad permite acceder a un objeto incluido dentro del antiguo modelo de objetos de ASP, en este caso se obtiene una instancia de la clase System.Web.SessionState.HttpSessionState, este objeto nos va a permitir almacenar información entre diferentes páginas ASP incluidas en una misma aplicación ASP .NET. La diferencia con el objeto Application se encuentra en el ámbito de las variables, cada variable del objeto Session es particular a una sesión de un usuario determinado, no a toda la aplicación. De esta forma, cada usuario tendrá sus variables y sus valores, sin dar lugar a problemas de concurrencia, tampoco se podrá acceder a distintas variables de sesión, cada usuario tiene su espacio de almacenamiento. Las variables de aplicación son valores globales y comunes a toda la aplicación, y las variables de sesión son particulares para cada usuario de la aplicación.

- Trace: esta propiedad ofrece una referencia a un objeto de clase System.Web.TraceContext, que nos va a permitir acceder a la información detallada de la ejecución de la página ASP .NET actual. Este objeto lo comentaremos en detalle cuando nos ocupemos del tratamiento de errores y al utilización de trazas dentro de ASP .NET. El objeto Trace es un nuevo objeto de ASP .NET y no tiene ninguna correspondencia con los anteriores objetos del modelo de objetos de ASP.

- TraceEnabled: propiedad de tipo booleano que contendrá el valor true si está activado el mecanismo de trazas dentro de la página, cuando tratemos las directivas de la página veremos como activar las trazas dentro de una página ASP .NET.

- User: propiedad que devuelve información sobre el usuario que ha realizado la petición de la página ASP .NET.

- Validators: esta propiedad es una referencia a una colección de todos los controles Web de validación presentes en la página ASP .NET actual.

Una vez comentadas las propiedades y eventos principales de la clase Page nos queda comentar los métodos de dicha clase, para ello no hay nada más que pasar al siguiente apartado.

Métodos de la clase Page

A continuación vamos a pasar a describir brevemente algunos de los métodos que nos ofrece la clase Page:

- DataBind: este método, que ya lo hemos utilizado en algunos ejemplos en capítulos anteriores, establece el enlace de datos de todos los controles Web incluidos en la página ASP .NET actual. Este método no recibe ningún tipo de parámetro y no devuelve ningún tipo de valor.

- FindControl: este método realiza una búsqueda en la página del control Web cuyo nombre se pasa como parámetro, y si lo encuentra devuelve dicho control. Este método recibe como parámetro un objeto de la clase String que va a representar el nombre del control que deseamos localizar, y si el control se encuentra en la página devuelve el objeto correspondiente de la clase Control, sobre el que deberemos aplicar el mecanismo de casting adecuado para poder utilizar el control Web de la forma conveniente.

- LoadControl: carga de forma dinámica un control de usuario (en un futuro capítulo se comentará como crear controles definidos por el usuario). Como parámetro recibe un objeto de la clase String que representa la ruta del fichero .ASCX que va a ser el tipo de fichero

dónde se definen los controles de usuario. Este método devuelve una instancia de dicho control de usuario mediante un objeto de la clase UserControl.

- MapPath: este método devuelve un objeto de la clase String que representa una ruta virtual construida a partir de la ruta física que se indica por parámetro, también como un objeto de la clase String. Este método tiene la misma función que el método MapPath del objeto Server de versiones anteriores de ASP.

- ResolveUrl: este método convierte una URL virtual, que le indicamos por parámetro, en una URL absoluta.

- Validate: método que no posee ningún parámetro y que tampoco devuelve ningún valor, cuando es lanzado se realizan las validaciones correspondientes de todos los controles Web de validación incluidos en la página ASP .NET.

En el siguiente apartado vamos a comentar otro aspecto importante de ASP .NET y que se encuentra estrechamente relacionado con la clase Page, se trata de las directivas que podemos utilizar en una página ASP .NET.

Directivas de la página

En versiones anteriores de ASP únicamente podíamos utilizar una única directiva en la página, por ejemplo para indicar el lenguaje de programación utilizado en la página, sin embargo en ASP .NET podemos utilizar varias sentencias con directivas en la página. Además ASP introduce nuevas directivas.

Mediante estas directivas podremos declarar una serie de atributos de la página ASP .NET actual, estos atributos tendrán repercusión a la hora de crear la página.

Las directivas que presenta ASP .NET son: Page, Control, Import, Register, Assembly, OutputCache y Refence,a continuación vamos a pasar a comentar cada una de estas directivas, mostrando las propiedades que nos ofrecen cada una de ellas.

Estas directivas se pueden situar en cualquier lugar de la página, pero por convención se suelen situar en la zona superior de la misma.

La directiva @Page

La directiva Page es mas compleja de todas, en lo que a número de atributos se refiere, y también la más utilizada. La directiva Page define una serie de atributos específicos para cada página y que serán utilizados por el compilador a la hora de generar el código intermedio al que se compila cada página.

Esta directiva soporta los atributos existentes para la única directiva que se podía utilizar en versiones anteriores de ASP, es decir, ofrece los atributos Language, Transaction, etc. La directiva Page es la directiva 'por defecto', así por ejemplo la directiva <@Language="VBScript"%> de ASP 3.0, tendría el equivalente <@Page Language="VB"> en ASP .NET. La sintaxis general de la directiva Page es la que se muestra en el Código fuente 166.

```
<%@ Page atributo=valor [atributo=valor...] %>
```

Código fuente 166

En una página únicamente puede existir una directiva Page, si deseamos indicar varios atributos para la página deberemos separar los mediante espacios los pares atributo/valor.

A continuación vamos a pasar a describir los distintos atributos de esta directiva:

- AspCompat: a este atributo se le puede asignar los valores true/false, y establece que la página se ejecute en un proceso del tipo Single-thread apartment. Por defecto el valor de esta propiedad es false, si establecemos este atributo a true podemos perder rendimiento en la ejecución de la página ASP .NET correspondiente.

- AutoEventWireup: este atributo también puede presentar los valores true/false, por defecto tiene el valor true, e indica si los eventos de la página (UnLoad, Load, Init, etc.) se van a lanzar de forma automática.

- Buffer: este atributo al igual que los anteriores posee los valores true/false y se utilizará para activa o desactivar el búfer de la página ASP .NET actual. El mecanismo de búfer ya existía en versiones anteriores de ASP, y en ASP .NET está estrechamente relacionado con la propiedad Response de la clase Page, de hecho se puede acceder también a la propiedad Buffer del objeto Response de la clase HttpResponse. Cuando tratemos la clase System.Web.HttpResponse veremos en detalle el mecanismo de búfer. Por defecto este atributo tiene el valor true.

- ClassName: esta propiedad va a presentar como valor una cadena que indica el nombre de la clase de la página actual.

- ClientTarget: contiene una cadena que indica el tipo de navegador Web para el que tienen que generar el código HTML los controles Web.

- CodePage: indica el código de la página.

- CompilerOptions: este atributo es una cadena que indica una lista de opciones para el compilador de la página.

- ContentType: este atributo contiene una cadena que describe un tipo MIME. En este atributo indicaremos el tipo de contenido devuelto por la página ASP .NET.

- Culture: identificador de cultura del sistema, este identificador establece el lenguaje, calendario y sistema de escritura.

- Debug: propiedad que puede presentar los valores true/false y que permite activar la depuración de la página ASP .NET actual. El mecanismo de depuración lo veremos en detalle más adelante en el texto. Por defecto este atributo presenta el valor false.

- Description: cadena que muestra una descripción de la página, se utiliza a modo de documentación.

- EnableSessionState: este atributo permite activar o desactivar el estado de sesión, es decir, permite o no la utilización de la propiedad Session de la clase Page para almacenar información común a la sesión actual del usuario con la aplicación Web. Este atributo puede tener los valores true/false, para indicar si está disponible el estado de sesión a través de la propiedad Session (instancia de la clase System.Web.SessionState.HttpSessionState), y también puede tener el valor ReadOnly, cuando se le asigna este valor la página podrá leer variables de sesión pero no modificarlas ni crear variables nuevas. Por defecto este atributo tiene el valor true.

- EnableViewState: permite activar o desactivar el mantenimiento automático de los valores de los controles Web dentro de un formulario, por defecto tiene el valor true.

- EnableViewStateMac: indica si a la hora de mantener el estado de los controles Web se debe realizar mediante un mecanismo de autenticación de la máquina cliente. El valor por defecto de este atributo es false.

- ErrorPage: este atributo contendrá una URL que indicará la página de error que se va utilizar en el caso de que se produzca una excepción no tratada en el página ASP .NET actual.

- Explicit: la utilización de este atributo de la directiva Page tiene sentido cuando estamos desarrollado nuestras páginas ASP .NET con el lenguaje Visual Basic .NET. Mediante este atributo indicaremos si la declaración de variables es obligatoria o no, es equivalente al modo Option Explicit de Visual Basic. Su valor por defecto es true.

- Inherits: este atributo indica el nombre de una clase de la que la página ASP .NET hereda, este atributo es utilizado para realizar la separación del código de la presentación de la información. Esto lo veremos más en detalle cuando tratemos el mecanismo de Code-Behind (código oculto).

- Language: el lenguaje de la plataforma .NET utilizado en la página, es el lenguaje al que se compilará todo el código fuente de la página.

- LCID: atributo que indica un identificar de localización válido para la página.

- ResponseEncoding: formato de codificación utilizado por el texto enviado como respuesta al cliente.

- Src: este atributo contiene el nombre de un fichero que contiene código que podrá se utilizado en la página, es otra forma de separación de código que nos ofrece ASP .NET, en este caso no se trata de una clase, sino en un bloque de código que se encuentra en un fichero separado de la página ASP .NET, el mecanismo de Code-Behind se retomará en el siguiente y veremos varios ejemplos de su utilización.

- SmartNavigation: este atributo puede tener los valores true/false e indica si se va a utilizar la característica de ASP .NET denominada SmartNavigation (navegación inteligente), esta característica mejora la navegación y transición entre los distintos estados de una página, se eliminan parpadeo, se mantienen las posiciones de scroll, se mantiene el foco e los elementos correspondientes, etc. El valor por defecto de esta propiedad es false.

- Strict: indica si se va utilizar el modo Option Strict de Visual Basic a la hora de compilar la página ASP .NET. Su valor por defecto es false. El modo Option Strict de Visual Basic indica que las conversiones de tipos deben ser estrictas, y por lo tanto no se realizar las conversiones automáticas.

- Trace: indica si se encuentra activado el mecanismo de trazas de la página, para poder utilizar la propiedad Trace de la clase Page. Si se encuentran activadas las trazas, cuando se ejecute la página, a la salida de la página se añadirá toda la información referente al procesamiento de la página. Por defecto presenta el valor false.

- TraceMode: este atributo es un complemento al anterior, y es utilizado para indicar el orden el que se mostrarán las trazas en la página, los valores que puede tomar son SortByTime (ordenadas por tiempo) y SortByCategory (ordenadas por categoría).

- Transaction: indica la configuración relativa a transacciones de la página, tiene el mismo significado que el atributo Transaction que se utilizaba en versiones anteriores de ASP. Los valores que puede presentar este atributo son los siguientes, Required,:se requiere una transacción; RequiresNew, se requiere una transacción nueva; Supported, soporta transacciones; NotSupported, No soporta transacciones. El valor por defecto de esta tributo es NotSupported.

- WarningLevel: este atributo indica el nivel en el que la compilación será interrumpida, los valores que puede presentar son de 0 a 4.

En el Código fuente 167 se muestra un ejemplo de utilización de la directiva Page, en este caso se ha indicado que el tipo de contenido que va a devolver la página ASP .NET es HTML (text/html), el estado de sesión se va a utilizar únicamente en modo de lectura, el lenguaje utilizado en el código fuente de la página es C# y se encuentra activado el mecanismo de trazas de la página.

```
<%@ Page ContentType="text/html" EnableSessionState=ReadOnly
  Language="C#" Trace=true%>
<html>
<body>
<script runat="server">
void Pulsado(Object fuente, EventArgs args){
      etiqueta.Text="Hola Mundo";
}
</script>
<form id="formulario" method="post" runat="server">
      <asp:label  id="etiqueta" runat="Server"></asp:label>
      <asp:button id="boton" onclick="Pulsado" runat="server" text="Pulsa">
     </asp:button>
</form>
</body>
</html>
```

Código fuente 167

La directiva @Import

Esta directiva es utilizada para importar de manera explícita un espacio con nombre (NameSpace) determinado. Al importar un NameSpace determinado tendremos acceso a todas las clases e interfaces definidos en el mismo. Se puede indicar un espacio con nombre de la plataforma .NET Framework o bien un espacio con nombre definido por nosotros mismos.

Esta directiva posee únicamente un atributo llamado NameSpace, su valor será el de un espacio con nombre válido. Únicamente se puede indicar un espacio con nombre por directiva Import, si deseamos importar o utilizar más de un espacio con nombre deberemos utilizar una directiva Import por cada NameSpace. En el Código fuente 168 se puede ver la sintaxis general de esta directiva.

```
<%@ Import NameSpace=”valor” %>
```

Código fuente 168

El .NET Framework de forma automática importa una serie de espacios con nombre para las páginas ASP .NET, por lo que no se tiene que importan de forma explícita, estos NameSpaces son los siguientes:

- System
- System.Collections
- System.Collections.Specialized
- System.Configuration
- System.IO
- System.Text
- System.Text.RegularExpressions
- System.Web
- System.Web.Caching
- System.Web.Security
- System.Web.SessionState
- System.Web.UI
- System.Web.UI.HtmlControls
- System.Web.UI.WebControls

En el Código fuente 169 se muestra un ejemplo de utilización de esta directiva, en este caso se importan dos espacios con nombre que forman parte del acceso a datos con ADO .NET, estos NameSpaces han sido importados para poder escribir el código necesario para acceder a unos datos en la página ASP .NET actual.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
    if(!Page.IsPostBack){
       SqlConnection conexion =
          new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
       SqlDataAdapter comando =
          new SqlDataAdapter("select firstname, lastname from Employees", conexion);
       DataSet ds = new DataSet();
       comando.Fill(ds, "Employees");
       lista.DataSource = ds.Tables["Employees"].DefaultView;
       lista.DataBind();
    }
}
</script>

<body>

<asp:DataList id="lista" runat="server"
        RepeatDirection="Vertical" RepeatLayout="Table"
        RepeatColumns="3" GridLines="Both" ItemStyle-Font-Bold="true">
      <ItemTemplate>
       <%# ((DataRowView)Container.DataItem)["firstname"]%> 
       <%# ((DataRowView)Container.DataItem)["lastname"]%>
      </ItemTemplate>
  </asp:DataList>
</body>
</html>
```

Código fuente 169

La directiva @Implements

Esta directiva nos permite implementar un interfaz determinado dentro de una página ASP .NET, implementar un interfaz implica dar un contenido a sus métodos, propiedades y eventos. Esta directiva presenta únicamente un atributo llamado Interface. Este atributo indicará el nombre del interfaz que se va a implementar en la página ASP .NET.

Al igual que sucedía en la directiva anterior, si deseamos implementar varios interfaces en la página debemos utilizar varias directivas Implements, una por cada interfaz a implementar. En el Código fuente 170 se puede ver la sintaxis general de esta directiva.

```
<%@ Implements Interface="nombreInterfaz" %>
```

Código fuente 170

La directiva @Register

Esta directiva se utilizará para indicar que vamos a utilizar en la página ASP .NET actual un control definido por el usuario. Existen dos tipos de controles definidos por el usuario, los que se denominan control Web Form de usuario o simplemente control de usuario, y los que se denominan controles ASP .NET de servidor o controles de servidor personalizados, más adelante veremos las distintas formas que tenemos de crear nuestros propios controles ASP .NET.

Esta directiva ofrece una serie de atributos que nos van a permitir indicar al compilador los controles personalizados que vamos a utilizar, ya que si al compilador no se le indica la información necesaria acerca de estos controles de usuario se generará un error a la hora de compilar la página ASP .NET. Estos atributos son:

- TagPrefix: este atributo va a contener una cadena que definirá un alias que se utilizará como espacio con nombre de la etiqueta dentro de la página ASP .NET para hace referencia al control Web de usuario.

- TagName: este atributo está muy relacionado con el anterior, y va a contener un alias que se utilizará como clase de la etiqueta dentro de la página ASP .NET que hace referencia al control Web de usuario. La etiqueta que va a hacer referencia al control de usuario se construirá de la forma tagprefix:tagname. Para los controles se servidor no se hace uso de este atributo.

- NameSpace: el espacio con nombre asociado al atributo TagPrefix, es decir, el NameSpace que se corresponde con el alias definido en el atributo TagPrefix. Este atributo se utilizará cuando el control utilizado es un control ASP .NET de servidor, que se va a encontrar definido en un assembly, más adelante veremos lo que es un assembly o ensamblado. Este atributo contiene el espacio con nombre en el que reside el control de servidor, este atributo no se utilizará para los controles de usuario.

- Src: indica la localización absoluta o relativa del fichero que define el control Web de usuario, estos ficheros poseen la extensión ASCX. Este atributo es utilizado cuando deseamos utilizar un control de usuario, que se encontrará definido dentro de un fichero .ASCX, y no en ensamblados, como sucedía con los controles de servidor.

- Assembly: en este atributo indicaremos el nombre del ensamblado que contiene el control ASP .NET de servidor.

Como ya se ha comentado, esta directiva presentará distintos atributos según el tipo de control al que deseamos hacer referencia, en el caso de querer utilizar un control de usuario en la página utilizaremos la sintaxis general del Código fuente 171.

```
<%@ Register TagPrefix="prefijoEtiqueta" TagName="nombreEtiqueta"
                                                    Src="rutaFicheroASCX" %>
```

Código fuente 171

Pero si por el contrario, deseamos hacer uso de un control ASP .NET de servidor utilizaremos la sintaxis que aparece en el Código fuente 172.

```
<%@ Register TagPrefix="prefijoEtiqueta" NameSpace="espacioConNombre"
                                                  Assembly="nombreAssembly" %>
```

Código fuente 172

La directiva Register la volveremos a comentar cuando veamos la creación de controles de usuario y controles ASP .NET de servidor.

La directiva @Assembly

Antes de nada debemos comentar el concepto de assembly dentro de la plataforma .NET. Un ensamblado o assembly, consiste en un conjunto de tipos y recursos, reunidos para formar la unidad más elemental de código que puede ejecutar el entorno de .NET Framework.

De igual forma que los edificios se crean a base de la unión de un conjunto de materiales, dentro de la tecnología .NET, los ensamblados se presentan como los bloques de construcción software, que se unen o ensamblan para crear aplicaciones. Una aplicación desarrollada para .NET Framework puede estar compuesta por uno o varios ensamblados, ver Figura 99.

Podemos establecer una analogía entre un ensamblado y una DLL, ya que ambos contienen clases, que se exponen a otras aplicaciones. Por dicho motivo, a un ensamblado también se le da el nombre de DLL lógica; el término DLL se emplea porque tiene un comportamiento similar al de las DLL's tradicionales, y el término lógica porque un ensamblado es un concepto abstracto, ya que se trata de una lista de ficheros que se referencian en tiempo de ejecución, pero que no se compilan para producir un fichero físico, a diferencia de lo que ocurre con las DLL's tradicionales.

Sin embargo, un ensamblado extiende sus funcionalidades a un horizonte mucho más amplio, ya que puede contener otros elementos aparte de clases, como son recursos, imágenes, controles se servidor, etc.

Por otro lado, simplifican los tradicionales problemas de instalación y control de versiones sobre los programas, uno de los objetivos de la tecnología .NET, en la que en teoría, para instalar una aplicación, sólo sería necesario copiar los ficheros que la componen en un directorio de la máquina que la vaya a ejecutar.

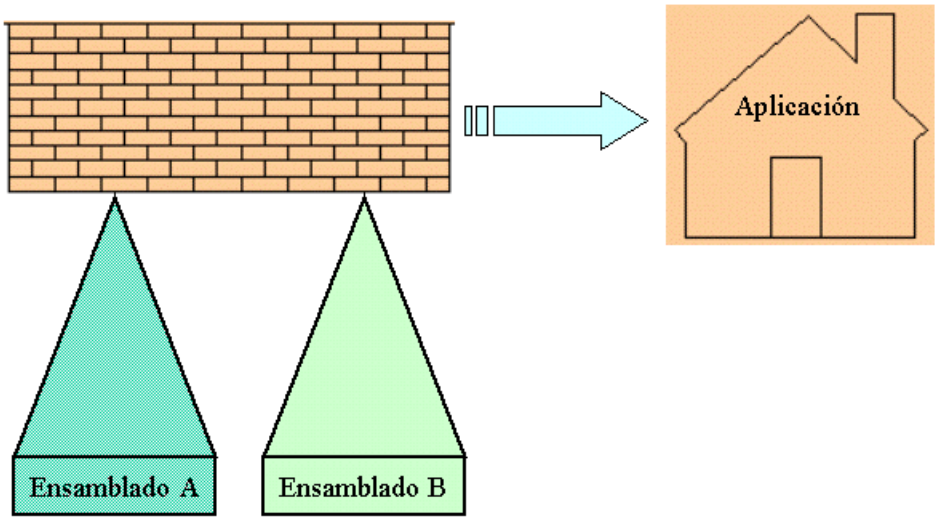


Figura 99

Una vez comentado lo que es un assembly vamos a pasar a describir la directiva que nos va a permitir utilizar un assembly directamente en una página ASP .NET. Para indicar el assembly que deseamos utilizar dentro de una página ASP .NET, esto implica instanciar objetos de las clases contenidas en el assembly y utilizar sus propiedades y métodos como los de cualquier clase del .NET Framework, disponemos de los siguientes atributos:

- Name: en este atributo se indicará el nombre del assembly compilado, este nombre no va a contener ninguna extensión de fichero.

- Src: en este caso se debe indicar el camino al fichero de código fuente que define el assembly, en este caso si se debe indicar la extensión correspondiente.

Normalmente esta directiva no se utiliza, ya que, cuando un assembly se encuentra el directorio bin de la aplicación Web, no es necesario indicar dicho assembly mediante la directiva Assembly, sino que se encuentra disponible para las páginas ASP .NET de la aplicación por encontrarse en el directorio especial bin.

Por lo tanto tendrá sentido utilizar la directiva Assembly cuando deseamos utilizar ensamblados que no se encuentran en la aplicación ASP .NET actual.

Al igual que sucedía con otras directivas anteriores, si deseamos hacer referencia varios ensamblados deberemos utilizar una directiva por cada assembly.

En el Código fuente 173 se muestra la utilización de la directiva Assembly en dos casos distintos, utilizando cada uno de los atributos de los que dispone.

```
<%@ Assembly Name="MiAssembly" %>
<%@ Assembly Src="MiAssembly.cs" %>
```

Código fuente 173

La directiva @OutputCache

Esta directiva es utilizada para controlar la forma en la que se utilizará la caché para la página ASP .NET en el servidor Web. ASP .NET implementa una serie de servicios de caché a dos niveles, uno se denomina caché de salida y otro caché ASP .NET.

El primero de estos servicios de caché es el que se puede configurar mediante la directiva OutputCache, por lo tanto esta directiva nos permite configurar la forma en la que generan la salida las páginas ASP dinámicas que son las mismas para distintos usuarios. Este mecanismo de caché almacena la salida generada por las páginas ASP, y utiliza esta copia almacenada para las siguientes peticiones de la página.

Este tipo caché únicamente será útil si la salida de la página para distintos usuarios es la misma, si se debe generar una salida diferente por cada usuario distinto que acceda a la página no se debe utilizar este mecanismo de caché de salida. ASP .NET puede detectar la cadena de consulta (QueryString) que se añade a la URL de la página que se demanda, y utilizará una copia almacenada en la caché únicamente si coinciden exactamente los valores del QueryString. Más adelante en este texto veremos en más detalle el mecanismo de caché que ofrece ASP .NET en sus dos vertientes.

A continuación vamos a pasar a describir los distintos atributos que nos ofrece la directiva OutputCache:

- Duration: indica el segundos el tiempo que una página va a permanecer en caché. Este atributo es obligatorio utilizarlo en la directiva, sino se utiliza se generará un error.

- Location: indica el lugar en el que es almacenado los datos de la caché, este atributo presenta cinco valores válidos; Any, este es el valor por defecto la caché se puede almacenar en el cliente, en un servidor intermedio, como puede ser un servidor proxy o en el propio servidor de origen de la página; Client, la caché se localiza en el navegador del cliente que realizó la petición de la página ASP .NET; Downstream, la caché se encuentra en un servidor intermedio entre el cliente y el servidor de origen de la página ASP .NET; None, en este caso no se encuentra habilitada la caché para la página actual; y en último lugar tenemos el valor Server, que indicará que la caché se encuentra en el servidor Web en el que se procesa la página.

- VaryByCustom: este atributo es utilizado para identificar requerimientos personalizados para los servicios de caché. Si indicamos al cadena “browser” la caché de la página variará en función del nombre y versión del navegador que realice la petición. EL significado de variar la caché es el de vaciar la caché actual y almacenar una nueva copia de la página ASP .NET en el estado actual.

- VaryByHeader: este atributo contiene una lista de encabezados del protocolo HTTP separados por punto y coma, estos encabezados son los que van a hacer que cambie la caché de la página.

- VaryByParam: en este caso la caché de la página variará atendiendo a los parámetros presentes en el QueryString de la página. Este atributo puede contener una lista de nombre de parámetros separados por punto y coma, en este caso la caché variará atendiendo a los valores de dichos parámetros. Si utilizamos el carácter asterisco (*) la caché variará con cualquier parámetro, pero si indicamos el valor none la caché no variará aunque se modifiquen los valores de los parámetros del QueryString.

En el Código fuente 174 se ofrece un ejemplo de utilización de esta directiva, en este caso la hora y fecha del momento actual cambiarán cada minuto o bien cuando se modifique el valor del parámetro llamado param dentro del QueryString.

```
<%@ Page Language="C#"%>
<%@ OutputCache Duration="60" VaryByParam="param"%>
<html>
<head>
<script runat="server">
void Page_Load(Object objFuente,EventArgs args){
  String mensaje;
  DateTime ahora=DateTime.Now;
  mensaje = "El código de esta página se ejecutó:"
             + "<br><b>" + ahora.ToString() + "</b></p>";
  tiempoEjecucion.Text = mensaje;
}

</script>
</head>
<body>
<form runat="server" id="formulario">
<asp:label id="tiempoEjecucion" runat="server"/>
</form>
</body>
</html>
```

Código fuente 174

La directiva @Reference

Esta última directiva de la página es utilizada para establecer una referencia a un control de usuario o a otra página ASP .NET, de esta forma se pueden cargar dinámicamente en la página actual. Esta directiva posee dos atributos, que serán utilizados según corresponda:

- Page: nombre del fichero ASPX que contiene la página a la que se desea hacer referencia.
- Control: nombre del fichero ASCX que contiene al control al que se desea hacer referencia en la página actual.

Con esta última directiva finalizamos el capítulo actual que ha pretendido dar una visión general de la clase Page.

En el siguiente capítulo trataremos los distintos mecanismos que nos ofrece ASP .NET para separar el código fuente de la presentación, alguno de estos mecanismos ya los hemos mencionado en el capítulo actual, se trata de los controles de usuario y el mecanismo de Code-Behind.

13

Code-Behind y controles de usuario

Introducción

En este capítulo y en el siguiente vamos a comentar las distintas posibilidades que nos ofrece ASP .NET para poder realizar una separación de código fuente frente al contenido o presentación, es decir, se pretende separar la lógica de la aplicación de la lógica de la presentación.

ASP .NET nos ofrece cuatro mecanismos que nos van a permitir realizar esta separación de código y contenido:

- Mediante el uso de ficheros Code-Behind, que son módulos precompilados escritos en cualquier lenguaje de la plataforma .NET Framework.

- Utilizando controles de usuario definidos en ficheros .ASCX.

- Creando componentes para el .NET Framework definidos en unidades denominas assembly, los ensamblados ya se comentaron en el capítulo anterior.

- Construyendo nuestros propios controles de servidor de ASP .NET, al igual que en el caso anterior estos controles de servidor irán compilados en assemblys. Aunque la construcción de controles Web tiene que ver también con la reutilización de código y con la creación de controles de servidor visuales, que generan contenido, se ha creído conveniente incluir este mecanismo que nos ofrece ASP .NET en esta enumeración.

A lo largo del presente capítulo se comentarán los dos primeros mecanismos de separación de código, también se mostrarán diversos ejemplos de utilización. Y en el próximo capítulo veremos como realizar la construcción de componentes .NET y de controles de servidor de ASP .NET.

Code-Behind

Coma ya hemos adelantado ASP .NET ofrece a las páginas un mecanismo denominado Code-Behind (código oculto). Este nuevo mecanismo nos permite separar el interfaz de usuario de la lógica de la aplicación, de esta forma el código fuente puede residir en un fichero separado, que puede ser una clase de Visual Basic .NET, C#, etc. Este código es el que se denomina Code-Behind, ya que permanece oculto. Las ventajas que obtenemos al utilizar este mecanismo son:

- La posibilidad de realizar una clara división del trabajo entre los programadores y los diseñadores.
- Permite a los desarrolladores o programadores utilizar sus entornos preferidos.
- Los autores de código HTML pueden utilizar sus herramientas de diseño de páginas Web.

Se debe señalar que un fichero Code-Behind es al fin y al cabo una clase definida en el lenguaje .NET correspondiente.

Para indicar que una página ASP .NET va a utilizar una clase determinada (como Code-Behind), se utilizan los atributos Inherits y Src de la directiva Page, esta directiva se veía con detenimiento en el capítulo anterior.

En el atributo Inherits debemos indicar el nombre de la clase Code-Behind y en el atributo Src la ruta del fichero de la clase. No es necesario hacer uso del atributo Src, si no indicamos la ruta al fichero que define la clase, en el caso del lenguaje C# se trata de un fichero con la extensión .CS, ASP .NET buscará la clase en el directorio bin de la aplicación.

Una vez indicado en la página ASP .NET la clase que se va a utilizar, mediante la directiva Page, únicamente nos queda definir e implementar la clase Code-Behind.

Vamos a realizar una clase muy sencilla en C# que simplemente muestra un saludo mediante un método. Para realizar la llamada de un método perteneciente a una clase Code-Behind desde una página ASP .NET únicamente debemos escribir el nombre del método como si estuviera definido en la propia página.

El aspecto que presenta una clase que va a ser utilizada desde una página ASP .NET es muy similar al resto de las clases tradicionales, lo que la hace particular es que debe importar una serie de NameSpaces correspondientes al entorno de ASP .NET. Estos espacios de nombres son System y System.Web, que nos van a permitir utilizar los objetos del entorno ASP .NET. Para importar los espacios de nombres se va a utilizar la instrucción using.

Y si también deseamos acceder a los controles Web definidos en la página ASP .NET desde la clase Code-Behind tenemos que importar los espacios de nombres necesarios, como puede ser System.Web.UI.WebControls.

La otra diferencia con las clases convencionales es que nuestra clase Code-Behind debe heredar de la clase System.Web.UI.Page de ASP .NET. Para indicar la clase de la que hereda nuestra clase Code-Behind, debemos situar a continuación del al nombre de la clase actual el nombre de la clase padre

separado mediante dos puntos (:). De esta forma el código de nuestra clase Code-Behind sería el mostrado en el Código fuente 175.

```
using System;
using System.Web;

namespace aplicacionWeb.Ejemplos
{
	public class ClaseSencilla:System.Web.UI.Page
	{
		public void muestraSaludo()
		{
			Response.Write("<b>¡Hola amigo, que tal...!</b><br>");
		}
	}
}
```

Código fuente 175

En este ejemplo hemos creado una clase en C# que hereda de la clase Page, se llama ClaseSencilla y se encuentra en el espacio con nombre aplicacionWeb.Ejemplos. Se ha definido un método público llamado muestraSaludo() que únicamente utiliza el método Write() de la propiedad Response de la clase. No se debe olvidar que tenemos acceso completo a la clase Page, ya que la clase que estamos definiendo es clase hija de la clase Page.

En el Código fuente 176 se muestra el código que contendría la página ASP .NET que desea utilizar la clase Code-Behind anterior, como ya hemos comentado, si copiamos el fichero .CS que contiene el código fuente en C# de la clase Code-Behind en el directorio bin especial de la aplicación ASP .NET actual, no tenemos que utilizar el atributo Src de la directiva Page. Más adelante en este texto veremos la estructura y configuración de una aplicación ASP .NET, de momento cabe destacar la importancia del directo bin de la aplicación, que también será utilizado para definir los componentes de servidor.

```
<%@ Page Language="C#" Src="ClaseSencilla.cs"
Inherits="aplicacionWeb.Ejemplos.ClaseSencilla"%>
<html>
<head>
<script runat="server">
void Page_Load(Object objFuente,EventArgs args){
	muestraSaludo();
}
</script>
<title>Code-Behind</title>
</head>

<body>
Página que utiliza una clase Code-Behind
</body>

</html>
```

Código fuente 176

En el código de la página ASP .NET se ha asignado el valor ClaseSencilla.cs al atributo Src porque en este caso la clase Code-Behind se encuentra en el mismo directorio que la página. Si no deseamos utilizar este atributo podemos copiar la clase al directorio bin de la aplicación ASP .NET, pero para

que todo funcione correctamente debemos seleccionar la opción de generar la aplicación ASP .NET (proyecto) dentro de Visual Studio .NET, en este caso si modificamos la clase Code-Behind, se debe volver a generar el proyecto, ya que en el directorio bin se debe encontrar la clase precompilada.

Como se puede comprobar en el código de la página ASP .NET a la hora de lanzar el método definido en la clase Code-Behind no debemos hacer nada especial, no es necesario compilar previamente la clase Code-Behind, al ejecutar la página ASP .NET que hace uso de la clase, se realiza la compilación de forma automática.

En la Figura 100 se puede ver el resultado de la ejecución de este ejemplo de utilización Code-Behind.

Utilizando los controles Web de la página

Este ejemplo de Code-Behind es muy sencillo y únicamente muestra un mensaje en el navegador, pero en otros casos nos puede interesar operaciones más complejas, como puede ser acceder a controles Web, desde la clase Code-Behind, definidos en una página ASP .NET.

Figura 100

Vamos a realizar otro ejemplo de clase Code-Behind, en este caso se trata de ampliar la clase anterior añadiendo un método que muestre la hora y fecha actuales en un control Web de la clase Label de la propia página ASP .NET.

En este nuevo ejemplo, en nuestra clase Code-Behind debemos importar el NameSpace System.Web.UI.WebControls, para así poder acceder a los controles Web que se encuentran en la página ASP .NET. Otra diferencia que resulta notable es que definimos una variable dentro de la clase que va a ser de la misma clase que el control Web que deseamos manipular, y además debe tener el mismo nombre que se le ha dado al control Web en la página ASP .NET.

Estas variables que van a corresponderse con controles Web de la página ASP .NET no es necesario instanciarlas, ya que esto lo va a hacer la propia página, no debemos olvidar que coincida el nombre de la variable con el del control Web.

El Código fuente 177 es el código de la clase Code-Behind que incorpora esta otra funcionalidad que nos permiten este tipo de clases.

```
using System;
```

```
using System.Web;
using System.Web.UI.WebControls;

namespace aplicacionWeb.Ejemplos
{
	public class ClaseSencillaDos:System.Web.UI.Page
	{
		public Label etiqueta;
		public void muestraSaludo()
		{
			Response.Write("<b>¡Hola amigo, que tal...!</b><br>");
		}

		public void muestraHora()
		{
			DateTime ahora=DateTime.Now;
			etiqueta.Text=ahora.ToString();
		}
	}
}
```

Código fuente 177

Y el Código fuente 178 es el código de la página ASP .NET que utiliza a la clase Code-Behind anterior, en este caso la única diferencia con el ejemplo anterior es el nombre de la clase de la que hereda la página, el nombre del fichero de código fuente y la llamada al nuevo método muestraHora().

```
<%@ Page Language="C#" Src="ClaseSencillaDos.cs"
Inherits="aplicacionWeb.Ejemplos.ClaseSencillaDos"%>
<html>
<head>
<script runat="server">
void Page_Load(Object objFuente,EventArgs args){
	muestraSaludo();
	muestraHora();
}
</script>
<title>Code-Behind</title>
</head>

<body>
Página que utiliza una clase Code-Behind
<br>
<form runat="server">
<asp:Label ID="etiqueta" runat="server"/>
</form>
</body>

</html>
```

Código fuente 178

Tratamiento de eventos

Podemos complicar algo más la clase Code-Behind, en esta nueva ampliación vamos a incluir el tratamiento de eventos de una pulsación (evento OnClick) de un botón que se encuentra en la página ASP .NET, al pulsar el botón se mostrará en el objeto Label la fecha y hora actuales.

En el Código fuente 179 se puede ver el nuevo aspecto que presentará la clase, se ha añadido el método pulsado() que recibe los parámetros adecuados para realizar el tratamiento de eventos del botón que se ha pulsado en la página ASP .NET.

```
using System;
using System.Web;
using System.Web.UI.WebControls;

namespace aplicacionWeb.Ejemplos
{
	public class ClaseSencillaTres:System.Web.UI.Page
	{
		public Label etiqueta;
		public void muestraSaludo()
		{
			Response.Write("<b>¡Hola amigo, que tal...!</b><br>");
		}

		private void muestraHora()
		{
			DateTime ahora=DateTime.Now;
			etiqueta.Text=ahora.ToString();
		}

		public void pulsado(Object sender, EventArgs args)
		{
			muestraHora();
		}
	}
}
```

Código fuente 179

También se ha optado por cambiar el acceso que la página va a tener al método muestraHora(), en este ejemplo se ha indicado que este método es privado, por lo tanto únicamente se va a poder lanzar a través del método pulsado().

La página ASP .NET presenta el Código fuente 180, como se puede observar para indicar el método que va a tratar el evento del control Web Button, se hace de la misma forma que cuando se encuentra el código para el tratamiento del evento en la misma página.

```
<%@ Page Language="C#" Src="ClaseSencillaTres.cs"
Inherits="aplicacionWeb.Ejemplos.ClaseSencillaTres"%>
<html>
<head>
<script runat="server">
void Page_Load(Object objFuente,EventArgs args){
	muestraSaludo();
}
</script>
<title>Code-Behind</title>
</head>

<body>
Página que utiliza una clase Code-Behind
<br>
<form runat="server" ID="Form1">
<asp:Label ID="etiqueta" runat="server"/>
<asp:Button ID="boton"  Runat="server" OnClick="pulsado" Text="pulsa"/>
```

```
</form>
</body>

</html>
```

Código fuente 180

En la Figura 101 se puede ver el nuevo aspecto de la página ASP .NET del ejemplo.

Figura 101

Clases Code-Behind y Visual Studio .NET

Como se ha podido comprobar a lo largo de este apartado, podemos crear las clases Code-Behind tan complicadas como sea necesario, y de esta forma vamos a conseguir que el código fuente de la página, es decir, el código que suele estar encerrado entre las etiquetas <script ruant=”server”></script>, se encuentre completamente separado de nuestra página ASP .NET que contiene un Web Form que muestra un interfaz de usuario.

El propio entorno de desarrollo Visual Studio .NET hace uso de forma automática de clases Code-Behind para los Web Forms que se crean dentro de un proyecto determinado. Para comprobarlo de primera mano vamos a realzar el siguiente proceso dentro de un proyecto de Visual Studio .NET:

- Se añade un nuevo Web Form al proyecto, pulsado con el botón derecho del ratón sobre el nombre del proyecto seleccionamos la opción Agregar|Agregar formulario Web.

- Añadimos desde el cuadro de herramientas un control Web Label y un control Button en la vista de diseño.

- Se hace doble clic sobre el control Button, de forma automática VS .NET nos muestra el fichero .CS que contiene la clase Code-Behind asociada a la página ASP .NET actual. En este momento podemos escribir una línea de código dentro del método en el que nos encontramos posicionados (este método debería ser Button1_Clic), en esta línea de código lo único que hacemos es asignar a la propiedad Text del objeto Label un valor.

- Para poder probar la página ASP .NET con su clase Code-Behind asociada debemos generar el proyecto, para ello acudimos a la opción de menú Generar. En Visual Studio .NET las

clases Code-Behind se compilan dentro de un assembly que se localizará en el directorio bin de la aplicación ASP .NET actual.

Por lo tanto si estamos utilizando VS .NET de la forma antes indicada, para utilizar las clases Code-Behind debemos generar el proyecto. Esto es así debido a que el código que genera de forma automática VS .NET en la directiva Page de la página no incluye el atributo Src, para indicar el lugar en el que se encuentra el fichero de la clase Code-Behind, por lo tanto se buscará en el directorio bin compilado ya como un assembly.

En el Código fuente 181 se ofrece el código que genera de forma automática VS .NET para la clase Code-Behind. En el código aparece destacado en negrita lo más interesante del mismo.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace aplicacionWeb
{
	/// <summary>
	/// Summary description for WebForm3.
	/// </summary>
	public class WebForm3 : System.Web.UI.Page
	{
		protected System.Web.UI.WebControls.Label Label1;
		protected System.Web.UI.WebControls.Button Button1;

		public WebForm3()
		{
			Page.Init += new System.EventHandler(Page_Init);
		}

		private void Page_Load(object sender, System.EventArgs e)
		{
			// Put user code to initialize the page here
		}

		private void Page_Init(object sender, EventArgs e)
		{
		//
		// CODEGEN: This call is required by the ASP.NET Web Form Designer.
		//
			InitializeComponent();
		}

		#region Web Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
			this.Button1.Click +=
				new System.EventHandler(this.Button1_Click);
			this.Load += new System.EventHandler(this.Page_Load);
```

```
            }
            #endregion

            private void Button1_Click(object sender, System.EventArgs e)
            {
                  Label1.Text="Prueba";
            }
      }
}
```

Código fuente 181

Y en el Código fuente 182 se ofrece el código que se genera para la página ASP .NET, al igual que en el código anterior se destaca lo más relevante.

```
<%@ Page language="c#" Codebehind="WebForm3.aspx.cs" AutoEventWireup="false"
Inherits="aplicacionWeb.WebForm3" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
<meta content="Microsoft Visual Studio 7.0" name=GENERATOR>
<meta content=C# name=CODE_LANGUAGE>
<meta content="JavaScript (ECMAScript)" name=vs_defaultClientScript>
<meta content=http://schemas.microsoft.com/intellisense/ie5 name=vs_targetSchema>
  </HEAD>
<body MS_POSITIONING="GridLayout">
<form id=WebForm3 method=post runat="server">
<asp:Label id=Label1 style="Z-INDEX: 101; LEFT: 363px; POSITION: absolute; TOP:
95px" runat="server">Label</asp:Label>
<asp:Button id=Button1 style="Z-INDEX: 102; LEFT: 561px; POSITION: absolute; TOP:
55px" runat="server" Text="Button"></asp:Button></FORM>

  </body>
</HTML>
```

Código fuente 182

A continuación vanos a pasar a comentar la segunda alternativa de la que disponemos si deseamos realizar una separación y estructuración de nuestro código en las páginas ASP .NET.

Controles de usuario

Los controles de usuario (user controls) van a ser páginas ASP .NET que serán utilizadas por otras páginas como controles de servidor. Los ficheros en los que se definan los controles de usuario poseen una extensión .ASCX, gracias a esta extensión se evita que el control de usuario pueda ser ejecutado de forma aislada como si se tratara de una página ASP .NET.

Los controles de usuario nos puede servir para ampliar los controles ofrecidos por ASP .NET. Con una serie de cambios mínimos cualquier página ASP .NET puede convertirse en un control de usuario y así poder ser reutilizada por otra página.

Podríamos decir que los controles de usuario reemplazan de largo a los ficheros include de versiones anteriores de ASP, los controles de usuario van a intentar solventar los mismos problemas, como puede ser la creación de cabeceras, barras de navegación, pies, bloques de código repetitivos, etc., aunque ofrecen una funcionalidad mucho más avanzada que los ficheros de include.

En las primeras versiones que aparecieron de ASP .NET antes de llegar a la versión final, los controles de usuario recibían la denominación de "Pagelet".

Básicamente un control de usuario se compone de HTML y código, pero debido a que un control de usuario siempre va a estar incluido dentro de una página ASP .NET, no se pueden utilizar en los ficheros ASCX las etiquetas <html>,<body> y <form>.

Un control de usuario va a participar en el ciclo completo de la ejecución de cada petición de la página en la que se encuentran incluidos, y además pueden manejar sus propios eventos, encapsulando de esta forma la lógica frente a la página ASP .NET en la que se utilizan, así por ejemplo un control de usuario va a poder implementar su método Page_Load.

Vamos a comenzar por un ejemplo muy básico de control de usuario, se va a tratar de un simple texto para que sea incluido en las páginas que sean necesarias, en este caso se trata de un copyright que puede ser incluido en el pie de una página. El Código fuente 183 muestra el sencillo código que presenta el fichero ASCX que contiene al control, como se puede comprobar se trata de código HTML sencillo.

```
<hr>
<font size=1>© 2002 Grupo EIDOS. Todos los derechos reservados.</font>
```

Código fuente 183

Una vez que tenemos definido nuestro control de usuario en el fichero ASCX correspondiente, para incluirlo en una página ASP .NET debemos hacer uso de la directiva Register, que ya comentamos en el capítulo anterior.

En la directiva Register debemos indicar un NameSpace para el control de usuario, es decir, el prefijo que se incluirá delante del nombre del control dentro de la página, esto se hace mediante el atributo TagPrefix de la directiva Register. Varios controles de usuario pueden pertenecer al mismo espacio de nombres, es decir, pueden tener el mismo valor para el atributo TagPrefix.

También debemos indicar un valor para el atributo TagName y Src. En el atributo TagName indicaremos el nombre del control, es decir, el nombre de la etiqueta que va a identificar al control de forma única, y en el atributo Src indicaremos la ruta virtual al fichero ASCX que contiene al control de usuario.

Una vez registrado el control vamos a poder utilizarlo como si se tratara de un control Web de ASP .NET, se debe indicar que la clase que representa al control de usuario es System.Web.UI.UserControl que hereda directamente de la clase System.Web.UI.Control. Cuando ejecutamos la página ASP .NET que contiene al control, el entorno de ejecución del .NET Framework compila el control de usuario y lo hace disponible para la página.

El Código fuente 184 es el que presentaría la página ASP .NET que hace uso del control de usuario definido anteriormente, en este caso el NameSpace del control es controles y su etiqueta es pie.

```
<%@ Page language="c#"%>
<%@ Register TagPrefix="controles" TagName="pie" Src="ControlPie.ascx"%>
<HTML>
<head><title>Controles de usuario</title></head>
<body>
<h1>Página ASP .NET que utiliza un control de usuario</h1>
```

```
<controles:pie runat="server" id="pieUno"/>
</body>
</HTML>
```

Código fuente 184

Y en la Figura 102 se puede observar el resultado de la ejecución de esta página ASP .NET.

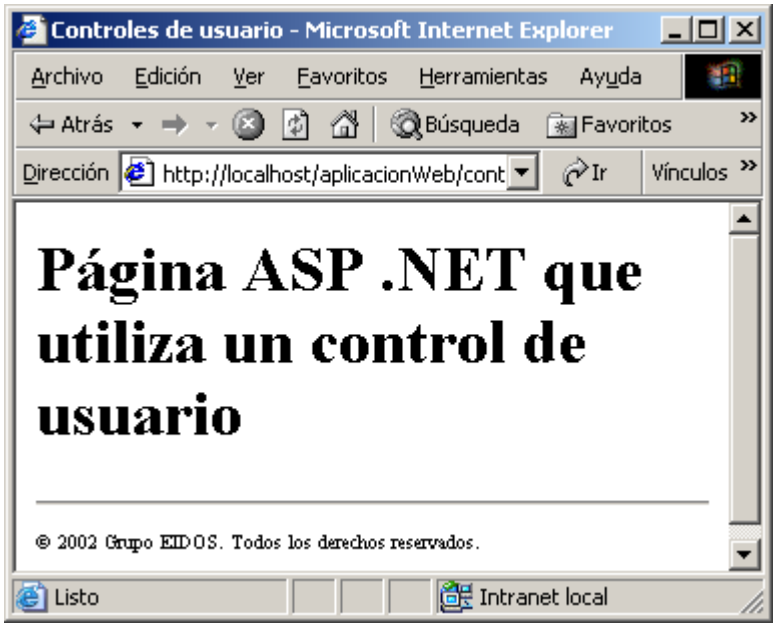


Figura 102

Si deseamos registrar más de un control de usuario en una misma página ASP .NET se deberá utilizar una directiva Register para cada uno de ellos, así si queremos incluir un nuevo control de usuario (Código fuente 185) que funcione como una cabecera deberemos escribir el Código fuente 186 para la página ASP .NET correspondiente.

```
<font size=1>El momento actual es: <%=DateTime.Now.ToString()%></font>
<hr>
```

Código fuente 185

```
<%@ Page language="c#"%>
<%@ Register TagPrefix="controles" TagName="pie" Src="ControlPie.ascx"%>
<%@ Register TagPrefix="controles" TagName="cabecera" Src="ControlCabecera.ascx"%>
<HTML>
<head><title>Controles de usuario</title></head>
<body>
<controles:cabecera runat="server" id="cabeceraUno"/>
<h1>Página ASP .NET que utiliza controles de usuario</h1>
<controles:pie runat="server" id="pieUno"/>
</body>
</HTML>
```

Código fuente 186

Como se puede apreciar en este nuevo ejemplo, el nuevo control de cabecera pertenece al mismo espacio de nombres que el control anterior, y en este caso se ha incluido código en C# para obtener la fecha y horas actuales.

El resultado se puede observar en la Figura 103.

Figura 103

Vamos a realizar un ejemplo un poco más complejo de control de usuario, en este caso el control va a contener controles Web, como si de una página ASP .NET se tratara. El control de usuario va a contener un objeto Text con dos controles de validación, uno del tipo RequiredFieldValidator y otro del tipo RangeValidator. El control de usuario nos a requerir introducir un número entre 1 y 100. En el Código fuente 187 se puede ver como se ha construido este control de usuario.

```
<asp:TextBox Id="valor" runat="server"/><br>
<asp:RequiredFieldValidator id="validadorRequerido"  runat="server"
      ControlToValidate="valor" Display="Dynamic"
      Font-Italic="True" Font-Name="Courier" Font-Size="7"
      ErrorMessage="Debe indicar un número">
</asp:RequiredFieldValidator>
<asp:RangeValidator id="validadorRango"  runat="server"
      ControlToValidate="valor"
       ErrorMessage="Debe indicar un entero en el rango 1-100"
      Display="Dynamic"  MinimumValue="1" MaximumValue="100" Type="Integer"
      Font-Italic="True" Font-Name="Courier" Font-Size="7">
</asp:RangeValidator>
```

Código fuente 187

Como se puede comprobar en el control de usuario no se ha utilizado ningún Web Form que contenga a los controles Web, será responsabilidad de la página ASP .NET que utilice el control de usuario

incluirlo si es necesario dentro de un formulario. No debemos olvidar que no es posible realizar la anidación de formularios.

La página ASP .NET que utiliza este control (Código fuente 188), va a ser muy sencilla, únicamente contiene un formulario Web con un control Web de la clase Button y una instancia de nuestro control, al pulsar el botón se realizará la correspondiente validación, como se puede comprobar el tipo de validación que se realiza es transparente a la página ASP .NET.

```
<%@ Page language="c#"%>
<%@ Register TagPrefix="controles" TagName="validador"
Src="ControlValidacion.ascx"%>
<html>
<head><title>Controles de usuario</title></head>
<body>

<form id="formulario" method="post" runat="server">
Valor:<controles:validador Id="control" runat="server"/>
<br>
<asp:Button ID="boton" runat="server" Text="Enviar datos"></asp:Button>
</form>

</body>
</html>
```

Código fuente 188

En la Figura 104 se puede ver el resultado de la ejecución de esta página ASP .NET

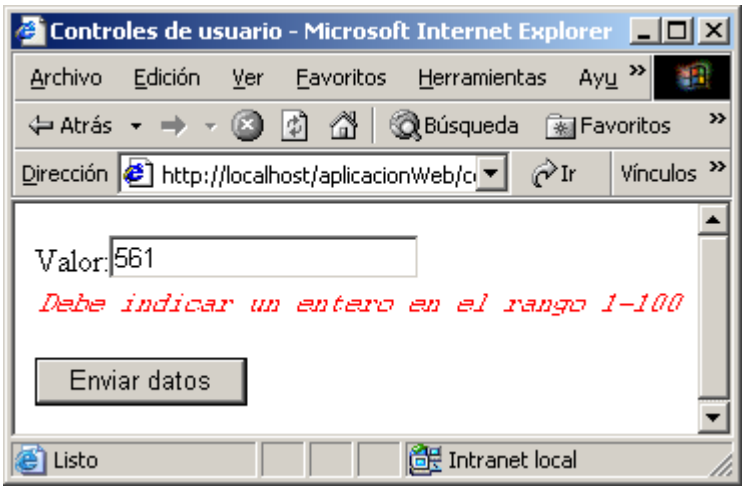


Figura 104

Creando propiedades en los controles de usuario

Estos ejemplos no dejan de ser ejemplos son muy sencillos, ya que se tratan de simples textos y algún control Web, pero los controles de usuario nos ofrecen mucho más que los antiguos ficheros de include, en el siguiente ejemplo vamos a ver como podemos crear propiedades para nuestros controles de usuario, ya que las propiedades públicas y métodos públicos que presente el control de usuario se van a poder utilizar desde la página ASP .NET, incluso las propiedades se van a poder incluir como atributos de la etiqueta del control dentro de la página.

Para definir una propiedad de un control de usuario lo podemos hacer de dos formas, mediante campos públicos (variables públicas) o bien mediante la sintaxis de propiedades, a continuación veremos ambas posibilidades mediante los correspondientes ejemplos.

En este primer ejemplo se van a definir un serie de variables públicas dentro de un control de usuario, estas variables nos van a indicar el texto que va a mostrar el control, el color de dicho texto y el tamaño.

En el Código fuente 189 se ofrece el contenido del fichero .ASCX que define este nuevo control, como se puede comprobar asignamos unos valores por defecto a las variables del control, así si utilizamos el control de usuario dentro de la página ASP .NET sin indicar ninguna de estas propiedades a través de los atributos correspondientes de la etiqueta, se mostrará el control con sus valores por defecto.

```
<script language="C#" runat="server">

  public String Color = "red";
  public String Texto = "Mensaje por defecto";
  public int tamaño=8;

</script>

<span id="Mensaje" style="font-
size:<%=tamaño%>pt;color:<%=Color%>"><%=Texto%></span>
```

Código fuente 189

En el Código fuente 190 se ofrece el código que aparece en la página ASP .NET que utiliza el control de usuario, como se puede observar se han creado dos instancias del control de usuario, una utiliza los valores por defecto y a la otra instancia se le han asignado estos valores a través de los atributos correspondientes en la etiqueta, tal como se hacía con los controles Web de ASP .NET.

```
<%@ Page language="c#"%>
<%@ Register TagPrefix="controles" TagName="texto" Src="ControlTexto.ascx"%>
<HTML>
<head><title>Controles de usuario</title></head>
<body>
<h3>Página ASP .NET que utiliza controles de usuario</h3>
<controles:texto runat="server" id="textoUno"/><br>
<controles:texto runat="server" Texto="Texto nuevo desde la página ASP .NET"
Color="blue" id="TextoDos" tamaño="12"/>
</body>
</HTML>
```

Código fuente 190

Un ejemplo de ejecución de la página anterior se puede ver en la Figura 105.

Vamos ahora a crear un nuevo control de usuario, en este caso utilizando la sintaxis de propiedades que nos ofrece el lenguaje C#. El control de usuario va a ofrecer la misma funcionalidad que el control anterior, es decir, permite indicar un texto con su tamaño y color, pero en este caso en lugar de utilizar variables para cada uno de estos valores vamos a utilizar la sintaxis de propiedades que nos ofrece el

lenguaje C#, y en lugar de utilizar una etiqueta de HTML se va a utilizar un control Web de la clase Label.

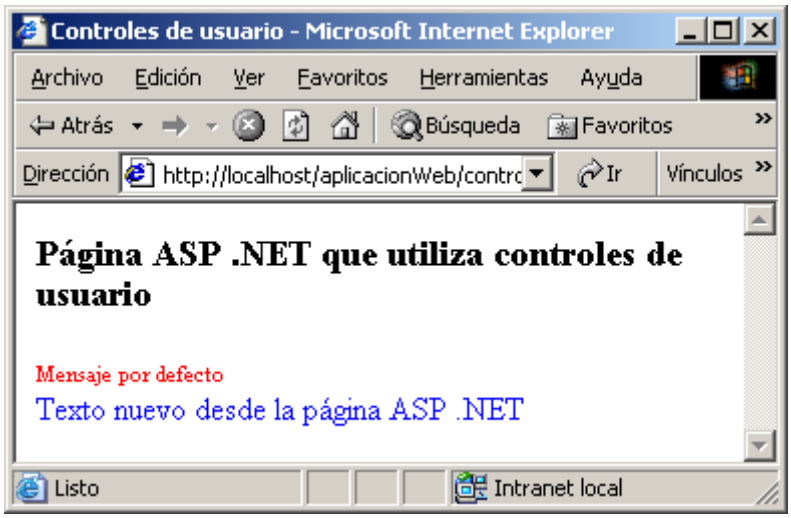


Figura 105

En este caso las propiedades de nuestro control de usuario se van a utilizar para acceder y manipular propiedades del objeto Label.

En el Código fuente 191 se puede ver el nuevo aspecto que ofrece el fichero .ASCX del control. El código de la página ASP .NET es exactamente igual que en el ejemplo anterior, ya que seguimos accediendo a las propiedades a través de atributos de la etiqueta del control.

```
<script runat="server" language="C#">
public int tamaño{
	get{
		return int.Parse(etiqueta.Font.Size.ToString());
	}
	set{
		etiqueta.Font.Size=value;
	}
}

public String texto{
	get{
		return etiqueta.Text;
	}
	set{
		etiqueta.Text=value;
	}
}

public String color{
	get{
		return etiqueta.Style["color"];
	}
	set{
		etiqueta.Style["color"]=value;
	}
}

</script>
```

```
<asp:Label id="etiqueta" runat="server" Text="Por defecto" Color="red" Font-
Size="8"/>
```

Código fuente 191

Como una curiosidad se ofrece la Figura 106, en la que se puede observar el aspecto que le da Visual Studio .NET a los controles de usuario en la vista de diseño de la página ASP .NET que hace uso de ellos.

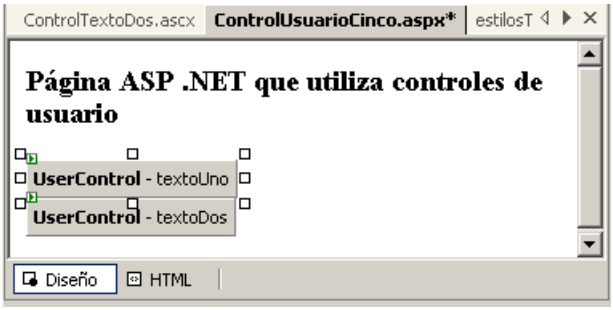


Figura 106

Eventos en los controles de usuario

En los controles de usuario también podemos definir eventos y tratarlos en el propio control, sin que la página ASP .NET tenga que ser consciente de ello, se debe recordar que los controles de usuario se van a poder comportar prácticamente como páginas ASP .NET.

En el siguiente control de usuario (Código fuente 192) se utiliza una lista desplegable, que es un control Web de la clase DropDownList, que al ser seleccionado un elemento de la lista se muestra dichero elemento en un control Label.

```
<script language="C#" runat="server">
private String[] datos=new String [5];
void Page_Load(Object objFuente, EventArgs args){
      if(!IsPostBack){
            datos[0]="Bora-bora";
            datos[1]="Morea";
            datos[2]="Pascua";
            datos[3]="Papeete";
            datos[4]="Santiago de Chile";
            lista.DataBind();
      }
}

void seleccionado(Object sender, EventArgs args){
      etiqueta.Text=lista.SelectedItem.Text;
}

</script>
<asp:DropDownList id="lista" runat="server" DataSource="<%# datos %>"
AutoPostBack="True" OnSelectedIndexChanged="seleccionado"/>
<asp:Label id="etiqueta" runat="server"/>
```

Código fuente 192

Para inicializar el contenido de la lista se ha utilizado el método Page_Load y dentro de este método se pregunta por la propiedad IsPostBack para saber si la página se ha enviado al servidor. También se utiliza el mecanismo de Data Binding de la misma forma que lo hacíamos con las páginas ASP .NET.

En el método seleccionado() se trata el evento de selección de la lista.

El código de la página ASP .NET (Código fuente 193) que utiliza este control es extremadamente sencillo, ya que es el control de usuario el encargado de realizar todo el trabajo.

```
<%@ Register TagPrefix="controles" TagName="lista" Src="ControlLista.ascx"%>
<%@ Page language="c#"%>

<HTML>
  <HEAD><title>Controles de usuario</title>
</HEAD>
<body>
<form id="formulario" method="post" runat="server">
<controles:lista runat="server" id="lista"/>
</form>
</body>
</HTML>
```

Código fuente 193

En la Figura 107 se puede ver un ejemplo de ejecución de esta página.

La directiva @Control

Los controles de usuario presentan una directiva, similar a la directiva Page, llamada Control, esta directiva únicamente se puede utilizar en los fichero ASCX y sólo puede existir una directiva Control por cada control de usuario.

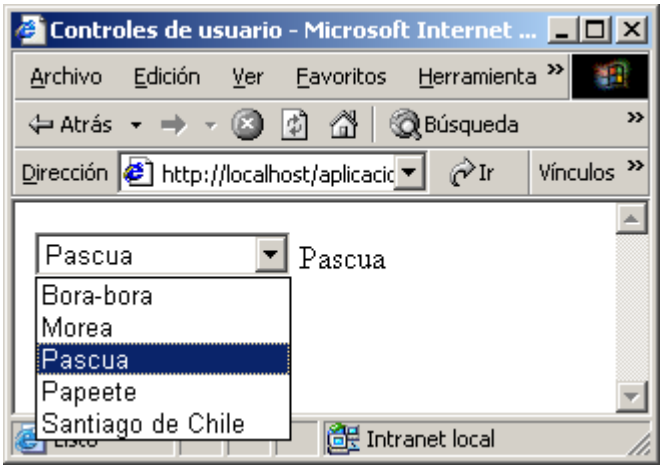


Figura 107

La directiva Control soporta algunos de los atributos que soportaba la directiva Page, algunos de estos atributos son AspCompat y Trace, para establecer las trazas para un control de usuario, se deben activar en la página ASP .NET que contiene a dicho control.

A continuación vamos a comentar los distintos atributos que presenta la directiva Control, vamos a ser muy breves ya que tienen la misma funcionalidad que en la directiva Page pero aplicados al control de usuario actual:

- AutoEventWireup: indica si los eventos (UnLoad, Load, Init, etc.) se van a lanzar de forma automática.
- ClassName: esta propiedad va a presentar como valor una cadena que indica el nombre de la clase del control actual.
- CompilerOptions: este atributo es una cadena que indica una lista de opciones para el compilador.
- Debug: permite activar la depuración para el control de usuario actual
- Description: cadena que muestra una descripción del control, se utiliza a modo de documentación.
- EnableSessionState: este atributo permite activar o desactivar el estado de sesión, es decir, permite o no la utilización de la propiedad Session de la clase Page para almacenar información común a la sesión actual del usuario con la aplicación Web.
- EnableViewState: permite activar o desactivar el mantenimiento automático de los valores de los controles Web dentro de un formulario, por defecto tiene el valor true.
- Explicit: mediante este atributo indicaremos si la declaración de variables es obligatoria o no, es equivalente al modo Option Explicit de Visual Basic.
- Inherits: este atributo indica el nombre de una clase de la que el control hereda, este atributo es utilizado para realizar la separación del código de la presentación de la información.
- Language: el lenguaje de la plataforma .NET utilizado en el control de usuario.
- Src: la ruta del fichero de la clase Code-Behind utilizada por el control.
- Strict: indica si se va utilizar el modo Option Strict de Visual Basic a la hora de compilar el control.
- WarningLevel: este atributo indica el nivel en el que la compilación será interrumpida, los valores que puede presentar son de 0 a 4.

Controles de usuario y clases Code-Behind

Gracias a la directiva Control podemos utilizar también en nuestros controles de usuario el mecanismo de Code-Behind, es decir, en el propio control de usuario podemos separar el código fuente de la presentación, para utilizar una clase Code-Behind desde un control únicamente debemos hacer uso de los atributos Inherits y Src de la directiva Control, de la misma forma que lo hacíamos para las páginas ASP .NET.

A continuación vamos a crear un control de usuario que va a mostrar el momento actual, es decir, la fecha y horas actuales, pero vamos a hacerlo utilizando una clase Code-Behind asociada al control mediante la directiva Control, tal como hemos comentado anteriormente.

Para tener una visión general del ejemplo paso a enumerar los ficheros de código fuente que van a formar parte del mismo:

- Un fichero .ASCX que va a contener la definición del control de usuario (ControlFechaHora.ascx).
- Un fichero .CS que va a contener la clase Code-Behind utilizada por el control de usuario (ClaseSencillaControl.cs).
- Un fichero .ASPX que va a ser la página ASP .NET que va a hacer uso del control de usuario que vamos a desarrollar (ControlUsuarioSiete.aspx).

En primer lugar vamos a crear el fichero .ASCX cuyo código se puede ver en el Código fuente 194, este control va a tener su directiva Control para indicar de la clase Code-Behind que hereda.

```
<%@ Control Language="C#" Src="ClaseSencillaControl.cs"
Inherits="aplicacionWeb.Ejemplos.ClaseSencillaControl"%>
<script runat="server">
void Page_Load(Object objFuente,EventArgs args){
      muestraHora();
}
</script>
<asp:Label ID="etiqueta" runat="server"/>
```

Código fuente 194

Como se puede observar el código del control es muy sencillo, posee un control Web de la clase Label, que será dónde se muestra la fecha y hora actuales, y también se encuentra implementado el método Page_Load, dentro del método que trata el evento de carga de la página se hace una llamada al método muestraHora(), este método pertenece a la clase Code-Behind de la que hereda nuestro control, a continuación vamos a pasar a comentar dicha clase.

La clase Code-Behind utilizada por el control de usuario es muy similar a las clases Code-Behind que veíamos en los ejemplos de utilización de estas clases con páginas ASP .NET, lo que distingue a las clases Code-Behind utilizadas con controles de usuario de las utilizadas con páginas ASP .NET es la clase de la que heredan. En el Código fuente 195 se puede observar que nuestra clase Code-Behind no hereda de la clase System.Web.UI.Page, sino que hereda de la clase System.Web.UI.UserControl.

```
using System;
using System.Web;
using System.Web.UI.WebControls;

namespace aplicacionWeb.Ejemplos
{
      public class ClaseSencillaControl:System.Web.UI.UserControl
      {
            public Label etiqueta;

            public void muestraHora()
            {
                  DateTime ahora=DateTime.Now;
                  etiqueta.Text=ahora.ToString();
            }

            public int tamaño
```

```
            {
                  get
                  {
                        return int.Parse(etiqueta.Font.Size.ToString());
                  }
                  set
                  {
                        etiqueta.Font.Size=value;
                  }
            }

            public String color
            {
                  get
                  {
                        return etiqueta.Style["color"];
                  }
                  set
                  {
                        etiqueta.Style["color"]=value;
                  }
            }
      }
}
```

Código fuente 195

En la clase utilizada por el control de usuario, además de aparecer el método muestraHora() que muestra dentro del objeto Label del control la fecha y hora actuales, también aparecen dos propiedades, se trata de las propiedades que nos van a permitir indicar el tamaño y el color del control Web de la clase Label, y que podrán ser utilizadas como atributos de la etiqueta del control de usuario cuando se utilice en la página ASP .NET.

Como se puede ver en la clase Code-Behind hemos incluido todo el código fuente, incluidas las propiedades del control de usuario, ya que el control de usuario al heredar de la clase Code-Behind heredará las propiedades definidas en la misma. También se debe importar el NameSpace System.Web.UI.WebControls ya que debemos utilizar el control Web de la clase Label definido en el control de usuario.

Y para finalizar nuestro ejemplo nos queda lo más sencillo, utilizar el control de usuario desde la página ASP .NET, para ello utilizaremos el Código fuente 196, que es un código muy similar al de otros ejemplos de utilización de controles de usuario con propiedades, ya que el forma en la que se encuentre implementado el control de usuario es completamente transparente para la página ASP .NET que lo utiliza.

```
<%@ Register TagPrefix="controles" TagName="fecha" Src="ControlFechaHora.ascx"%>
<%@ Page language="c#"%>
<HTML>
  <HEAD><title>Controles de usuario</title>
</HEAD>
<body>
<h4>El control de usuario indica que el momento actual es:</h4>
<form id="formulario" runat="server">
<controles:fecha runat="server" id="fechaUno" color="green" tamaño="8"/>
</form>
</body>
</HTML>
```

Código fuente 196

En la Figura 108 se puede ver un ejemplo de ejecución de esta página ASP .NET.

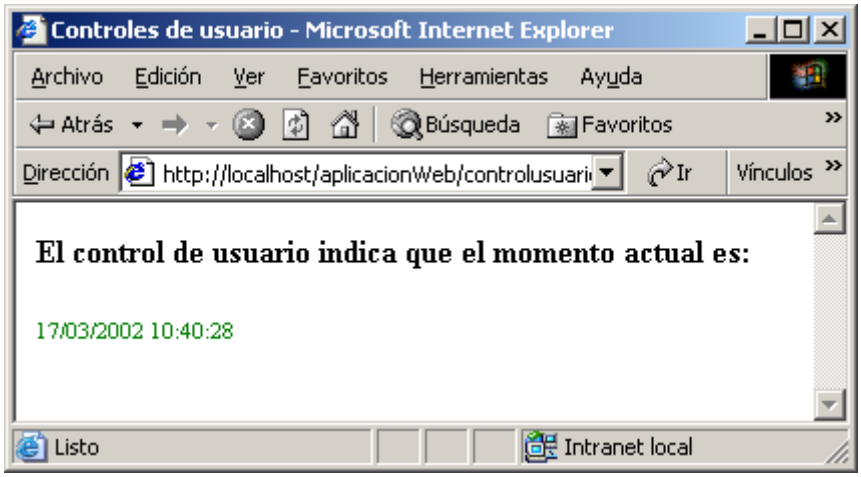


Figura 108

En el siguiente enlace se puede encontrar los tres ficheros implicados en este ejemplo.

Transformación de una página ASP .NET en control de usuario

Como ya hemos dicho en diversas ocasiones a lo largo de este apartado, un control de usuario realiza prácticamente las mismas funciones que una página ASP .NET y presenta un aspecto muy similar, para reforzar esta idea vamos a ver mediante un ejemplo práctico como podemos transformar una página ASP .NET en un control de usuario.

Supongamos que tenemos la página ASP .NET que aparece en el Código fuente 197, esta página ASP .NET establece una conexión con una base de datos y muestra una serie de campos de una tabla dentro de un control Web de la clase DataGrid, esta página ya se utilizó en el capítulo dedicado a los controles Web de lista. Esta página ASP .NET la deseamos transformar en un control de usuario. Este proceso va a resultar muy sencillo.

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script runat="server">
void Page_Load(Object sender, EventArgs e) {

      SqlConnection conexion =
            new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      SqlDataAdapter comando =
            new SqlDataAdapter("select firstname, lastname,city from Employees",
conexion);
      DataSet ds = new DataSet();
      comando.Fill(ds, "Employees");
      tabla.DataSource = ds.Tables["Employees"].DefaultView;
      tabla.DataBind();
}
</script>
```

```
<body>
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow">
</asp:DataGrid>
</body>
</html>
```

Código fuente 197

Lógicamente en primer lugar debemos crear el fichero ASCX que va a contener al control de usuario, a continuación pegamos el código fuente de la página al control de usuario y cambiamos la directiva Page por la directiva Control. Ahora debemos eliminar todas las etiquetas HTML como puede ser <HTML><HEAD><BODY>, y las correspondientes de cierre. Si la página tuviera un Web Form, que no es el caso, también deberíamos eliminarlo.

Una vez hecho lo anterior ya tendríamos disponible nuestro control de usuario para utilizarlo en una página ASP .NET. El código resultante sería el del control de usuario, que se puede ver en el Código fuente 198.

```
<%@ Control language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<script runat="server">
void Page_Load(Object sender, EventArgs e) {

  SqlConnection conexion =
    new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
  SqlDataAdapter comando =
    new SqlDataAdapter("select firstname, lastname,city from Employees", conexion);
  DataSet ds = new DataSet();
  comando.Fill(ds, "Employees");
  tabla.DataSource = ds.Tables["Employees"].DefaultView;
  tabla.DataBind();
}
</script>
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow">
</asp:DataGrid>
```

Código fuente 198

Ya nos queda únicamente mostrar el código de la página ASP .NET que hace uso del control, éste se puede ver en el Código fuente 199.

```
<%@ Register TagPrefix="controles" TagName="datos" Src="ControlPagina.ascx"%>
<%@ Page language="c#"%>
<HTML>
  <HEAD><title>Controles de usuario</title>
</HEAD>
<body>
<h4>DataGrid mostrado por el control de usuario:</h4>
<controles:datos runat="server" id="datosUno"/>
</body>
</HTML>
```

Código fuente 199

En la Figura 109 se puede observar el resultado dela ejecución de esta página ASP .NET.

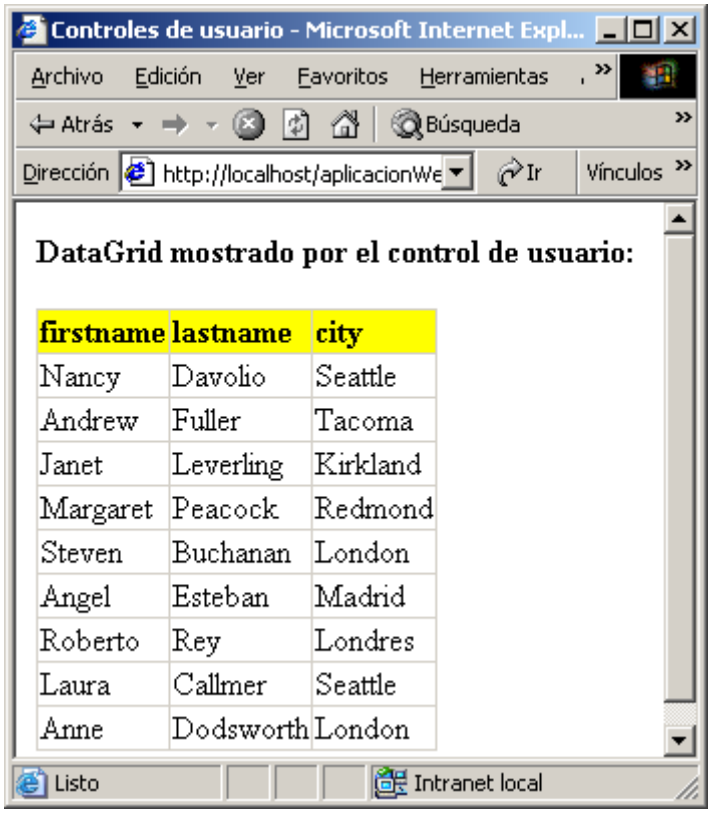


Figura 109

Con este nuevo ejemplo de utilización de controles de usuario se da por finalizado este apartado y el capítulo actual, en el siguiente capítulo vamos a comentar el tercer y último mecanismo que nos ofrece ASP .NET para separar el código de la lógica de la aplicación de la presentación o interfaz de usuario de la misma, se trata de la creación de componentes para ASP .NET.

En el próximo capítulo también veremos otro tema muy relacionado con la separación de código que es la creación de nuestros propios controles Web de servidor de ASP .NET, que a primera vista pueden parecer muy similares a los controles de usuario que acabamos de ver, pero en el siguiente capítulo se comprobará que no es así.

14

Creación de componentes y controles de servidor

Introducción

Este capítulo es continuación del anterior y aquí se enlazan los mecanismos de separación de código con los de creación de controles que pueden ser reutilizados.

En primer lugar nos vamos a ocupar de la creación de componentes para la plataforma .NET Framework, veremos las diferencias y mejores que existen respecto al modelo de componentes de COM+, veremos también como construir componentes y como distribuirlos y utilizarlos dentro de páginas ASP .NET.

Y en segundo lugar vamos a tratar la creación de controles de servidor de ASP .NET. Gracias a la creación de nuestros propios controles ASP .NET de servidor podemos encapsular interfaces personalizados y más funcionalidades en controles que pueden reutilizados en las páginas ASP .NET. No se debe confundir la creación de controles ASP .NET de servidor con los controles de usuario que veíamos en el capítulo anterior. Veremos lo sencillo, en comparación con otros lenguajes, que resulta en ASP .NET escribir nuestros propios controles de servidor.

Introducción a los componentes .NET

Los componentes dentro de ASP .NET los vamos a utilizar para que contengan la lógica de negocio, es decir, para que sean objetos con lógica de negocio (business objects). Estos componentes van a contener bloques de código con una funcionalidad concreta.

Los componentes de software son unidades de código ejecutable que proporcionan una funcionalidad específica para una aplicación. A la aplicación que utiliza código de un componente, creando objetos y llamando a sus propiedades y métodos, se denomina cliente o contenedora.

Las ventajas derivadas de la utilización de los componentes en el desarrollo de software pueden compararse a las de la producción industrial en cadena: en vez de ser un único fabricante el que se encargue, por ejemplo, de la construcción de todas y cada una de las piezas de un coche, existen varios fabricantes, cada uno de ellos especializado en una parte concreta (motor, amortiguadores, motores eléctricos, chapa, caja de cambios, ...), que tienen funcionalidades diferentes. Cada uno de estos fabricantes trabaja de forma independiente a los demás, pero siguiendo unos estándares industriales para que luego las piezas encajen entre sí para formar una única unidad funcional: el coche.

Del mismo modo, la utilización de los componentes en una aplicación informática permite que varias personas distintas, de la misma o distinta empresa, trabajando con distintos lenguajes de programación, puedan crear pequeños bloques de código con una funcionalidad específica. Cada uno de estos componentes deben ajustarse a unos estándares o protocolos, para que luego puedan encajar entre sí y formar una aplicación con funcionalidad completa.

Pero en este capítulo no vamos a entrar en más detalles en lo que a teoría de componentes se refiere, sino que vamos a ponernos manos a la obra y vamos a ir viendo la creación de componentes .NET a través de ejemplos, aunque antes de hacer esto debemos tener en cuenta una serie de puntos acerca de los componentes .NET.

Un problema que tenía el antiguo modelo de componentes (Component Object Model, COM) para los componentes de una aplicación Web es que estos componentes debían ser registrados en el servidor antes de poder ser utilizados desde una aplicación ASP. A menudo la administración remota de estos componentes no resultaba posible, debido a que la aplicación que permitía el registro de los componentes debía ser local al servidor.

Otras de las dificultades que plantea el uso de componentes es que permanecen bloqueados en el disco después de ser utilizados por la aplicación Web correspondiente, y en muchos casos era necesario parar el servidor Web para poder liberar el componente y así poder ser reemplazado por otro o eliminado.

El siguiente paso de COM es COM+. COM+ es un conjunto de servicios que combina COM con Microsoft Transaction Server (MTS) en sistemas Windows 2000. De esta forma el servidor transaccional MTS ha dejado de existir como una entidad separada en Windows 2000, toda su funcionalidad es ahora una parte básica del sistema operativo.

En el sistema operativo Windows 2000 Server la administración de componentes se ha visto mejorada a través de la herramienta administrativa de Servicios de componentes, que permite configurar las aplicaciones COM+.

Los Servicios de componentes ofrecen las siguientes características que están relacionadas con el desarrollo de componentes en la versión 3.0 de ASP:

- Plena integración con Internet Information Server 5.0. Lo cual redunda en una serie de características relacionadas con la creación de aplicaciones ASP, como son:
    - Ejecución de secuencias de comandos de páginas ASP dentro de una transacción gestionada por Servicios de componentes.

  - Protección contra errores de las aplicaciones Web de IIS. Al poder ejecutarse cada aplicación dentro de su propia aplicación COM+ de Servicios de componentes, se permite un aislamiento de los procesos implicados.

  - Inclusión de eventos transaccionales en las páginas ASP.

- Características relacionadas con la administración:

  - La utilidad explorador de Servicios de componentes, que permitirá administrar las aplicaciones COM+.

  - Cierre de aplicaciones COM+ de manera individual, sin necesidad de cerrar el proceso del servidor.

  - Configuración de la activación de aplicaciones COM+.

  - Desplazamiento de componentes entre aplicaciones COM+.

- Características relacionadas con la programación:

  - Se puede llevar a cabo, de manera automática, procedimientos administrativos como la instalación de aplicaciones COM+, mediante objetos de administración programables a través de secuencias de comandos.

Ahora en la plataforma .NET Framework y en ASP .NET se ha tratado de solventar todos los problemas y dificultades que planteaba el desarrollo de componentes en anteriores versiones de las tecnologías implicadas.

Para registrar un componente en ASP :NET debemos situarlo en un directorio determinado de la aplicación ASP .NET, los componentes se van a distribuir de una manera muy sencilla, simplemente se debe copiar al directorio BIN de la aplicación, este directorio debe encontrarse inmediatamente debajo del directorio raíz de la aplicación ASP .NET.

A continuación se comentan las ventajas de almacenar los componentes en el directorio BIN:

- No es necesario registrar el componente en el servidor Web. No es necesario el registro de componentes para hacerlos disponibles para una aplicación ASP .NET. Como ya hemos comentado los componentes se pueden distribuir copiando la DLL del componente (assembly) al directorio BIN de la aplicación correspondiente, esta operación puede ser realizada a través del protocolo FTP. En este punto cabe hacer una aclaración sobre el término DLL y assembly o ensamblado. Podemos establecer una analogía entre un ensamblado y una DLL, ya que ambos contienen clases, que se exponen a otras aplicaciones. Por dicho motivo, a un ensamblado también se le da el nombre de *DLL lógica*; el término *DLL* se emplea porque tiene un comportamiento similar al de las DLL's tradicionales, y el término *lógica* porque un ensamblado es un concepto abstracto, ya que se trata de una lista de ficheros que se referencian en tiempo de ejecución, pero que no se compilan para producir un fichero físico, a diferencia de lo que ocurre con las DLL's tradicionales.

- No es necesario reiniciar el servidor Web. Cuando cambiamos un componente reemplazando la antigua DLL del mismo por una nueva versión únicamente tenemos que sobrescribir el fichero y de forma automática la aplicación o aplicaciones ASP .NET que estén utilizando el componente comenzará a hacer uso de la nueva versión modificada del componente .NET. No es necesario reiniciar el servidor Web, ni el servicio Web ni descargar de memoria las aplicaciones ASP .NET que hacen uso del componente.

- No existen conflictos de espacies de nombre. Cada componente cargado desde el directorio BIN se encuentra limitado al ámbito de la aplicación que se está ejecutando. Esto quiere decir que muchas aplicaciones pueden potencialmente utilizar diferentes componentes con el mismo nombre de NameSpace o de clase, sin ocasionarse por ello ningún tipo de conflicto.

Al nivel más básico un componente va a ser simplemente una clase que podrá ser instanciada desde una página ASP .NET que importa el componente.

Podemos comparar el mecanismo de Code-Behind, que veíamos en el capítulo anterior, con al creación de componentes, de esta comparación es extraen las siguientes conclusiones:

- Las clases Code-Behind por defecto son ficheros sin compilar, aunque vimos en el capítulo anterior que podíamos compilarlas para que se incluyeran en un assembly que puede utilizar la aplicación ASP .NET.

- Las páginas ASP .NET utilizan a las clases Code-Behind a través del mecanismo de herencia, sin embargo los componentes se utilizan importándolos en las páginas correspondientes.

- Para utilizar una clase Code-Behind no hay que instanciarla, únicamente hay que heredar de ella, ya que gracias al mecanismo de herencia tendremos acceso a las propiedades y métodos de la clase padre. Sin embargo para hacer uso de un componente si es necesario realizar la instanciación el mismo, será sobre el objeto que es instancia de la clase del componente sobre el que lanzaremos los métodos correspondientes.

- Las clases Code-Behind se incluyen en una página ASP .NET mediante la directiva @Page, y los componentes se incluyen utilizando la directiva @Import, como si estuviéramos importando una clase o un NameSpace del .NET Framework.

Una vez realizada esta introducción más o menos exhaustiva sobre los componentes .NET vamos a pasar al siguiente apartado, en el que vamos a crear un sencillo componente para ASP .NET.

Creación de componentes .NET

En este apartado vamos a crear una clase que va a ser utilizada dentro de una página ASP .NET como un componente.

La estructura básica de una clase ya la vimos en el capítulo anterior cuando definíamos una clase Code-Behind, la clase de un componente .NET va a ser muy similar pero más genérica. En este caso la clase no tiene porqué heredar de la clase System.Web.UI.Page, ni tampoco tiene que importar unos espacios con nombre determinados, sino heredará de la clase que sea necesaria e importará los NameSpaces que sean necesarios. Lo normal es que se importe por lo menos el NameSpace System.

En un primer ejemplo vamos a mostrar una clase muy sencilla que devuelve mediante un método en una cadena de texto la fecha y horas actuales, este ejemplo ya lo hemos abordado utilizando otros mecanismos de separación de código.

El Código fuente 200 es el código que contiene esta clase en C#, que una vez compilada pasará a ser un componente para ser utilizado dentro de una página ASP .NET.

```
using System;

namespace Componentes.Ejemplos
{
```

```
	public class ComponenteSencillo
	{
		public ComponenteSencillo()
		{
		}

		public String muestraHora()
		{
			DateTime ahora=DateTime.Now;
			return ahora.ToString();
		}
	}
}
```

Código fuente 200

Como se puede observar la clase únicamente importa el NameSpace del sistema, es decir, el espacio de nombre System y no hereda de ninguna clase, así que por defecto heredará de la clase base de la jerarquía de clases de la plataforma .NET, la clase Object.

La clase consta de un constructor, cuya implementación se encuentra vacía, y la clase pertenece al espacio de nombres Componentes.Ejemplos. Posee un único método que devuelve un objeto de la clase String que va a contener la cadena con la fecha y hora actuales.

Para compilar esta clase lo podemos hacer de dos formas, mediante entorno de desarrollo de la plataforma .NET, es decir, Visual Studio .NET o mediante el compilador del lenguaje C#. A continuación vamos a comentar las dos posibilidades.

Si utilizamos el entorno de desarrollo Visual Studio .NET, y nuestra clase del componente se encuentra en el mismo proyecto que la aplicación ASP .NET actual, podemos seleccionar la opción de generar el proyecto, de esta forma se generará el componente también dentro de un assembly para todo el proyecto. Este assembly contendrá todas las clases de la aplicación ASP .NET que representa al proyecto actual, organizadas según los espacios de nombre correspondientes. El assembly se localizará en el directorio BIN y presentará el nombre del proyecto.

Para tener una visión más clara del assembly del proyecto de Visual Studio .NET, y de las clases y espacios de nombre que contiene, podemos seleccionar la opción de mostrar la utilidad Vista de clases, para ello seleccionamos la opción Ver|Vista de clases, y aparecerá un árbol que muestra la jerarquía de espacios de nombres y clases contenidas en los mismos. En la Figura 110 se puede ver la estructura de espacios de nombres que presenta al clase del componente que acabamos de crear.

Desde Visual Studio .NET también tenemos la posibilidad de añadir a la solución actual un nuevo proyecto, en este caso sería un proyecto de biblioteca de clases que va a contener la clase de nuestro componente. De esta forma podemos generar de forma independiente el assembly que va a contener nuestro componente .NET. El fichero DLL generado en este caso es el assembly que deberemos copiar al directorio BIN de la aplicación ASP .NET en la que deseamos hacer uso del mismo.

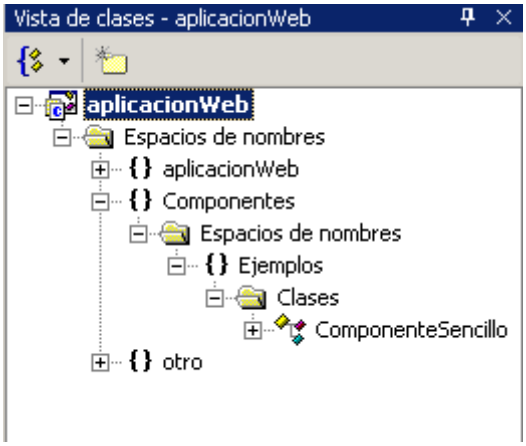


Figura 110

En la Figura 111 se puede observar el aspecto que mostraría en este segundo escenario la Vista de clases.

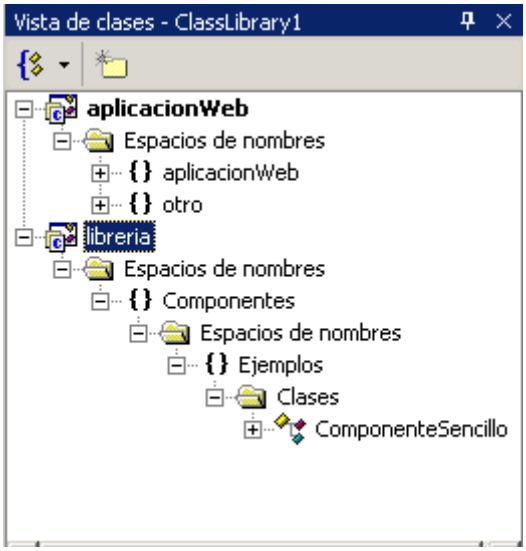


Figura 111

Las páginas ASP .NET utilizarán el componente de la misma forma, sin tener en cuenta si el componente .NET se encuentra en el assembly general del proyecto que representa a la aplicación ASP .NET actual o si se encuentra en un assembly separado, en cualquier caso el assembly resultante debe encontrase en el directorio BIN de la aplicación ASP .NET .

Copiar el assembly que contiene a nuestro componente .NET al directorio BIN de la aplicación ASP .NET es el mecanismo común para registrar un componente en una aplicación ASP .NET, esto es una de las mejoras que presentan los componentes .NET respecto a los componentes COM+, además si queremos modificar el fichero DLL y reemplazarlo por otro no tenemos nada más que sobrescribir, sin necesidad de cerrar ningún tipo de proceso o descargar el sitio Web de memoria.

La otra posibilidad es utilizar el compilador del lenguaje C#, esta utilidad se utiliza se invoca desde la línea de comandos y es llamada CSC. Este compilador es el que se ofrece con la instalación del Microsoft .NET Framework SDK (Software Development Kit).

El compilador posee varios parámetros para indicar de que forma se va a construir el assembly que va a contener al componente, vamos a enumerar los más interesantes:

- out: en este parámetros indicamos la ruta al fichero de salida que se va a generar. En nuestro caso será el fichero DLL que contiene al assembly.

- target: indica el formato del fichero de salida, puede tener los siguientes valores; exe, para indicar que se genere una aplicación de consola ejecutable; library, indica al compilador que debe crear una biblioteca (dinamic-link library, DLL), este será el valor que utilizaremos para compilar nuestra clase del componente en un assembly; module, en este caso se creará un módulo que deberá formar parte de un assembly; winexe, con este parámetro indicamos que deseamos generar una aplicación Windows Forms.

- Nombre de la clase a compilar, después de los parámetros anteriores aparece el nombre de la clase que deseamos compilar.

- reference: en este último parámetro se indican otros assemblies que son utilizados en nuestro componente, en nuestro caso se debe indicar únicamente el assembly System.dll.

El compilador de C# se encuentra en el directorio del .NET Framework dentro del directorio de instalación de Windows, en el caso de mi máquina se encuentra en el directorio C:\WINNT\Microsoft.NET\Framework\v1.0.2914.

En nuestro ejemplo si deseamos generar el assembly para nuestro componente deberíamos ejecutar el Código fuente 201 en la línea de comandos.

```
csc /out:bin\compo.dll /target:library ComponenteSencillo.cs /reference:System.dll
```

Código fuente 201

Una vez que hemos compilado nuestra clase, mediante cualquiera de los procesos indicados anteriormente, y se ha generado el assembly que va a contener al componente, únicamente nos queda copiarlo al directorio BIN de la aplicación ASP .NET para poder hacer uso del componente en una página ASP .NET.

Para utilizar el componente en una página ASP .NET únicamente debemos importar el componente, mediante la directiva @Import, indicando su espacio de nombre. Una vez hecho esto utilizaremos el componente como cualquier componente del .NET Framework, por lo que lo podremos instanciar y utilizar como un objeto dentro de la página ASP .NET, en el Código fuente 202 se muestra una página ASP .NET que hace uso de este sencillo componente de ejemplo.

```
<%@ Import Namespace="Componentes.Ejemplos" %>
<html>
<script language="C#" runat="server">
ComponenteSencillo componente=new ComponenteSencillo();
void Pulsado(Object origen, EventArgs argumentos){
      etiqueta.Text =componente.muestraHora();
}
</script>
<head><title>Componentes .NET</title></head>
<body>

<form id="formulario" method="post" runat="server">
```

```
<asp:button id="boton" text="Pulsar" onclick="Pulsado" runat="Server"/>
<asp:label id="etiqueta" runat="server"/>
</form>

</body>
</html>
```

Código fuente 202

La página ASP .NET crea una instancia del componente y en el evento de pulsación del botón lanza sobre el componente el único método que presenta, y el valor que devuelve se lo asigna a un objeto Label.

En la Figura 112 se puede ver un ejemplo de ejecución de la página ASP .NET.

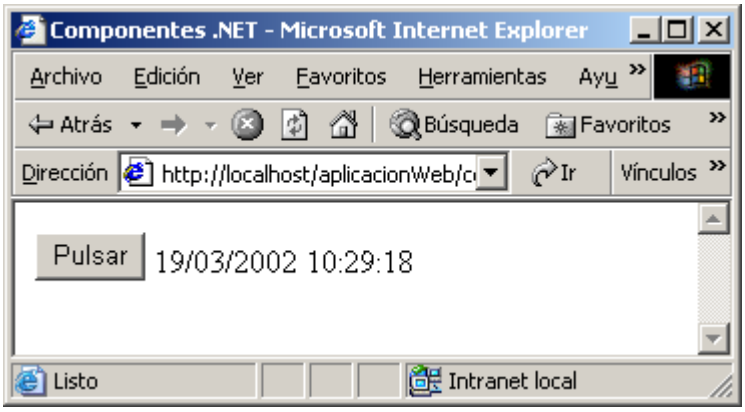


Figura 112

Este ejemplo muestra muy claramente la creación de componentes .NET, pero se trata de un ejemplo muy sencillo, a continuación vamos a mostrar la realización de un nuevo componente .NET con un poco más de complejidad.

En este caso nuestro componente va a poseer un método que nos va a devolver un objeto DataView de ADO .NET que va a representar una vista de una tabla de una base de datos. Nuestro componente va a poseer una propiedad que es un objeto de la clase String y que va a contener la cadena de conexión a la base de datos. Debido a que la clase realiza un acceso a datos es necesario importar los espacios de nombres que permiten hacer estas operaciones.

La propiedad de la clase se llama Conexión y es de lectura y escritura, esta propiedad se inicializa a vacío en el constructor de la clase.

Nuestra nueva clase pertenece al mismo espacio de nombres que la anterior, y la vamos a compilar las dos clases en el mismo assembly, ahora la vista de clases que va a presentar nuestro proyecto librería se puede observar en la Figura 113.

En el Código fuente 203 se puede ver el código completo de la clase de este nuevo componente.

Figura 113

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace Componentes.Ejemplos
{
	public class Datos
	{
		private String dsn;

		public Datos()
		{
			dsn="";
		}

		public Datos(String cadenaConex)
		{
			dsn=cadenaConex;
		}

		public String Conexión
		{
			set
			{
				dsn=value;
			}
			get
			{
				return dsn;
			}
		}

		public DataView devEmpleados()
		{
		  SqlConnection conexion = new SqlConnection(dsn);
		  SqlDataAdapter comando =
		      new SqlDataAdapter(
		          "select firstname, lastname,city from Employees", conexion);
		  DataSet ds = new DataSet();
		  comando.Fill(ds, "Employees");
		  return ds.Tables["Employees"].DefaultView;
		}
	}
}
```

Código fuente 203

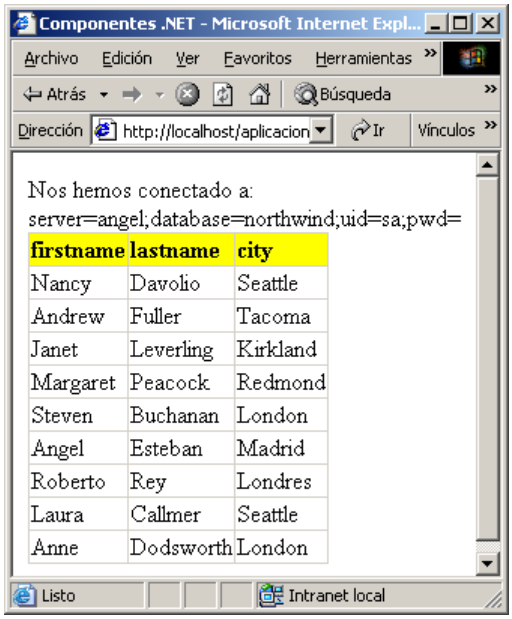
En el código de la clase podemos ver que el constructor se encuentra sobrecargado, en una de sus versiones no recibe ningún parámetro, pero en la otra recibe un objeto de la clase String que va a contener la cadena de conexión. Si utilizamos esta segunda versión no es necesario asignarle un valor a la propiedad Conexión, ya que el constructor lo ha hecho por nosotros.

El Código fuente 204 es el código que se corresponde con la página ASP .NET que hace uso de esta nueva clase, la página debe importar el espacio de nombres ya conocido, asigna a la propiedad de nuestra clase la cadena de conexión, y el resultado devuelto por el método devEmpleados() se lo asigna a la propiedad DataSource del objeto DataGrid presente en la página.

```
<%@ Page language="C#"%>
<%@ Import Namespace="Componentes.Ejemplos" %>
<html>
<head><title>Componentes .NET</title></head>
<script runat="server">
void Page_Load(Object sender, EventArgs e) {
      //Datos componente=new Datos("server=angel;database=northwind;uid=sa;pwd=");
      Datos componente=new Datos();
      componente.Conexión="server=angel;database=northwind;uid=sa;pwd=";
      tabla.DataSource=componente.devEmpleados();
      tabla.DataBind();
      Response.Write("Nos hemos conectado a: " + componente.Conexión);
}
</script>

<body>
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow">
</asp:DataGrid>
</body>
</html>
```

Código fuente 204

En la Figura 114 se puede ver el resultado de la ejecución esta página.



firstname	lastname	city
Nancy	Davolio	Seattle
Andrew	Fuller	Tacoma
Janet	Leverling	Kirkland
Margaret	Peacock	Redmond
Steven	Buchanan	London
Angel	Esteban	Madrid
Roberto	Rey	Londres
Laura	Callmer	Seattle
Anne	Dodsworth	London

Figura 114

En el siguiente enlace se puede obtener el código de la página ASP .NET y de la clase del componente.

Con este ejemplo damos por terminado el presente apartado, en lo que queda de capítulo vamos a comentar como podemos crear controles visuales reutilizables en páginas ASP .NET, se trata de la construcción de controles ASP .NET de servidor. ASP .NET nos va a permitir construir controles visuales que van a mostrarse como código HTML o bien en otro lenguaje como puede ser WML.

Creación de controles ASP .NET de servidor

Mediante al construcción de nuestros propios controles ASP .NET de servidor podemos encapsular la lógica y el interfaz de usuario para que puedan ser reutilizados en distintas páginas ASP .NET, como ya hemos comentado no se deben confundir los controles ASP .NET de servidor con los controles de usuario, ofrecen unos mecanismos distintos y se construyen de maneras muy diferentes, un control de usuario se va a construir normalmente a partir de una página ASP .NET, y un control de servidor se va construir a partir de una clase.

A diferencia de otros entornos como puede ser C/C++, la creación de controles visuales personalizados en ASP .NET resulta muy sencilla. Y como el movimiento se demuestra andando, vamos a crear nuestro primer control ASP .NET de servidor, este control de servidor únicamente va a mostrar un texto.

Para crear un control se servidor sencillo debemos crear un clase que herede de la clase System.Web.UI.Control y que sobrescriba su método Render(). El método Render() recibe como parámetro un objeto de la clase System.Web.UI.HtmlTextWriter, sobre este objeto lanzaremos el método Write() pasándole por parámetro una cadena (String) el código HTML que deseamos que genere nuestro control ASP .NET de servidor.

El método Render() se ejecutará cuando se instancie el control en la página ASP .NET, y generará el código que será mostrado al cliente en la página. El proceso de traducción o rendering del control es una de las fases del ciclo de vida de un control de servidor.

Nuestra clase que va a dar lugar al control de servidor también debe importar los espacios de nombre que resulten necesarios, normalmente se deben importar los siguientes:

- System: este NameSpace contiene las clases básicas del sistema, como puede ser la clase String. El código del control de servidor que vamos a realizar en el ejemplo podrá compilar sin este espacio de nombres, pero es recomendable incluirlo siempre.

- System.Web: este es el espacio de nombres padre para todas las clases de ASP .NET. Contiene clases como HttpRequest, que se corresponde con el objeto Request de ASP .NET, y HttpResponse, que se corresponde con el objeto Response de ASP .NET. De nuevo nuestro código podría compilar sin este NameSpace, pero es una buena práctica incluirlo, ya que en otros casos si que será necesario.

- System.Web.UI: este NameSpace contiene las clases de los controles ASP .NET, las cuales se encuentran divididas en espacios de nombres atendiendo a su familia (System.Web.UI.WebControls, System.Web.UI.HTMLControls, etc.). Esta referencia es necesaria, ya que estamos utilizando las clases Control y HtmlTextWriter.

En el Código fuente 205 se ofrece el código de la clase que nos va a permitir crear un control ASP .NET de servidor.

```
using System;
using System.Web;
using System.Web.UI;

namespace Controles.Ejemplos
{
	public class Texto:Control
	{
		protected override void Render(HtmlTextWriter salida)
		{
			salida.Write("<h1>Control ASP .NET de servidor</h1>");
		}
	}
}
```

Código fuente 205

Como se puede observar se ha utilizado la palabra reservada override para indicar que deseamos sobrescribir el método Render() de la clase Control de la que heredamos.

Una vez tenemos la clase de nuestro control debemos compilarla para que se genere en el assembly correspondiente, al igual que sucedía en el apartado anterior, a la hora de compilar la clase de un componente .NET, también tenemos las mismas opciones para compilar la clase de nuestro control ASP .NET de servidor. Cuando se ha generado el assembly de nuestro control de servidor lo debemos copiar al ya conocido directorio BIN de la aplicación ASP .NET en la que deseamos utilizar el control ASP .NET de servidor.

Para hacer uso del control de servidor en una página debemos utilizar la directiva @Register, esta misma directiva la utilizábamos en el capítulo anterior para registrar los controles de usuario en una página ASP .NET, pero ahora la vamos a utilizar para registrar controles ASP .NET de servidor.

En el atributo TagPrefix indicaremos la etiqueta que precederá al nombre dela clase del control de servidor cuando queramos utilizarlo en la página ASP .NET, en el atributo NameSpace indicamos el espacio de nombres del control y en la propiedad Assembly el nombre del fichero .DLL que contiene assembly, pero sin extensión. Como se puede ver en el Código fuente 206 para hacer referencial al control ASP .NET de servidor utilizamos la sintaxis <TagPrefix:NombreClase>.

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles ASP .NET de servidor</title></head>
<body>
<controles:Texto runat="server"/>
</body>
</html>
```

Código fuente 206

En la Figura 115 se puede ver el resultado de la ejecución de la página ASP .NET que hace uso del control que hemos construido.

Figura 115

Y el Código fuente 207 es el código HTML que se generaría.

```
<html>
<head><title>Controles de servidor</title></head>
<body>
<h1>Control ASP .NET de servidor</h1>
</body>
</html>
```

Código fuente 207

Ya hemos visto con un sencillo ejemplo los distintos pasos que hay que seguir para construir un control ASP .NET de servidor, en los siguientes subapartados vamos a ir viendo aspectos más avanzados relacionados con la construcción de controles de servidor.

Propiedades de los controles de servidor

Para que un control de servidor sea útil debe permitir que el desarrollador de las páginas ASP .NET pueda tener algún tipo de influencia sobre la forma en la que el control se presenta en la página. En la forma más común para realizar esta tarea es mediante atributos.

La creación de atributos a través de propiedades de la clase del control ya lo vimos para el caso de los controles de usuario, para los controles ASP .NET de servidor va a ser muy similar.

Cuando el .NET Framework ejecuta una página ASP .NET y procesa los atributos de un control los hace corresponder con propiedades públicas de la clase en la que se encuentra definido el control de servidor. Los valores de las propiedades se establecen en la fase de inicialización de la página.

Durante la fase de inicialización de la página, una vez que las propiedades del control se han establecido con los valores de los atributos indicados en las etiquetas, se llama al método OnInit() de cada control. En el método OnInit() del control podemos validar las propiedades o realizar otras operaciones como las de establecer los valores por defecto para las propiedades.

Vamos a crear un control muy similar al anterior, pero en este caso va a presentar una propiedad de la clase String llamada Texto, que nos va a permitir indicar al control el texto que deseamos mostrar.

Para implementar la propiedad debemos utilizar la sintaxis de propiedades del lenguaje C# tal como hacíamos en otros ejemplos.

El Código fuente 208 es el código del nuevo control ASP .NET de servidor.

```
using System;
using System.Web;
using System.Web.UI;

namespace Controles.Ejemplos
{
	public class Etiqueta:Control
	{
		private String _texto;

		public String Texto
		{
			get
			{
				return _texto;
			}

			set
			{
				_texto=value;
			}

		}

		protected override void OnInit(EventArgs evento)
		{
			base.OnInit(evento);
			if(_texto==null)
			{
				_texto="Texto por defecto";
			}
		}

		protected override void Render(HtmlTextWriter salida)
		{
			salida.Write("<h3>" + _texto + "</h3>");
		}
	}
}
```

Código fuente 208

Como se puede comprobar este control se encuentra en el mismo espacio de nombres que el del ejemplo anterior. Otro aspecto a destacar es la sobrescritura del método OnInit(), como ya hemos comentado utilizaremos este método para realizar tareas de inicialización de propiedades y de comprobación y validación de los valores de las mismas.

En el método OnInit() se comprueba si la propiedad Texto tiene algún valor, si no posee ningún valor se le asigna un texto por defecto. En este método también se hace referencia a la clase padre, la clase Control, a través de la palabra reservada base, y sobre la clase padre se lanza el método OnInit(), ya que lo estamos sobrescribiendo en la clase actual.

En el método Render() de nuestra nueva clase se escribe en el objeto HtmlTextWriter la propiedad Texto entre las etiquetas HTML correspondientes. Como se puede observar la clase internamente utiliza una variable privada para manejar la propiedad Texto.

Realizamos las labores ya conocidas sobre la clase del componente para hacerla disponible en nuestra aplicación ASP .NET, es decir, se compila y se copia el assembly al directorio BIN de al aplicación. En este caso vamos a crear en la página ASP .NET (Código fuente 209) tres controles de servidor, uno de ellos va a mostrar el valor por defecto de su propiedad Texto, y los otros dos mostrarán el valor especificado en la misma.

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<body>
<controles:Etiqueta runat="server" ID="Etiqueta1"/>
<controles:Etiqueta runat="server" ID="Etiqueta2" Texto="Hola"/>
<controles:Etiqueta runat="server" ID="Etiqueta3" Texto="Mundo"/>
</body>
</html>
```

Código fuente 209

En la Figura 116 se puede ver el resultado de la ejecución de esta página ASP .NET.

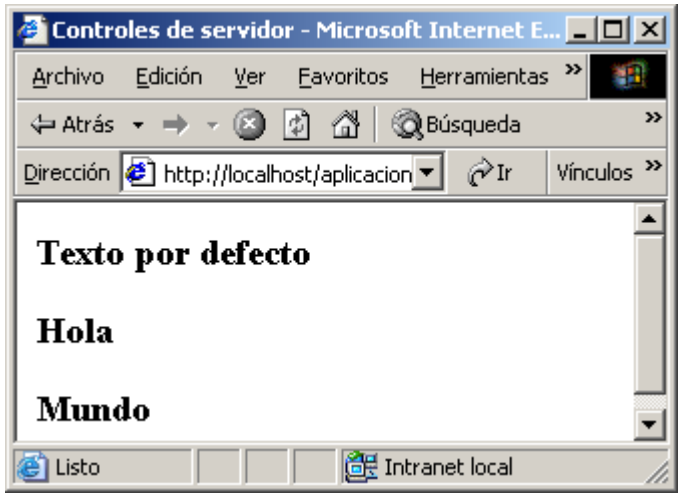


Figura 116

Conversión de los valores de atributos

ASP .NET realiza una conversión inteligente entre los valores de los atributos indicados en la página ASP .NET y los tipos definidos en las propiedades de la clase del control de servidor. Si en la clase del control de servidor una propiedad se encuentra definida de tipo String, ASP .NET realizará un mapeo sencillo y asignará a la propiedad el valor del atributo. Si la propiedad es de tipo int o long, el valor del atributo se convertirá a un valor numérico, y si el valor es una enumeración, ASP .NET encontrará la correspondencia entre el valor de tipo cadena indicado en el atributo con el valor de tipo enumerado de la propiedad de la clase. El mismo tipo de conversión se realizará con valores de tipo booleano.

Para demostrar algunas de estas conversiones vamos a ampliar el ejemplo anterior añadiendo nuevas propiedades. Una de estas propiedades va a ser la propiedad Repetición, que va a ser de tipo entero y va a indicar el número de veces que se va a mostrar el texto que le indiquemos al control, y la otra propiedad es la propiedad ColorLetra, que va a indicar el color en el que va a aparecer el texto, esta propiedad es de la clase System.Drawing.Color.

Para establecer los valores de las propiedades no tenemos que hace nada en especial, únicamente debemos utilizar la sintaxis de propiedades de manera correcta. Por defecto a las nuevas propiedades se les ha asignado los valores de 1 para la propiedad Repetición, y Color.Red para la propiedad ColorLetra.

En el Código fuente 210 se ofrece el código que presentaría nuestra clase con estas nuevas propiedades. Se puede observar que se ha importado el espacio de nombres System.Drawing para poder manejar el objeto que representa el color de las letras de la etiqueta de texto.

```
using System;
using System.Web;
using System.Web.UI;
using System.Drawing;

namespace Controles.Ejemplos
{
	public class EtiquetaDos:Control
	{
		private String _texto;
		private int _repetición=1;
		private Color _colorLetra=Color.Red;

		public String Texto
		{
			get
			{
				return _texto;
			}

			set
			{
				_texto=value;
			}

		}

		public Color ColorLetra
		{
			get
			{
				return _colorLetra;
			}

			set
			{
				_colorLetra=value;
			}

		}

		public int Repetición
		{
			get
			{
				return _repetición;
			}
```

```
                set
                {
                        _repetición=value;
                }

        }

        protected override void OnInit(EventArgs evento)
        {
                base.OnInit(evento);
                if(_texto==null)
                {
                        _texto="Texto por defecto";
                }
        }

        protected override void Render(HtmlTextWriter salida)
        {
                int i;
                for(i=0;i<_repetición;i++)
                {
                  salida.Write("<h3 style='color:" +
                  ColorTranslator.ToHtml(_colorLetra)+"'>"+ _texto + "</h3>");
                }
        }
    }
}
```

Código fuente 210

En el método Render() se ha creado un bucle for para que realice las repeticiones indicadas en la nueva propiedad. En este método también se debe destacar la utilización de una clase auxiliar llamada System.Drawing.ColorTranslator, esta clase se va a emplear para convertir el valor enumerado de la variable _colorLetra en el código HTML correspondiente que nos permite indicar el color de la letra, para ello se ha utilizado el método ToHtml().

El Código fuente 211 es un código de ejemplo de utilización de este control de servidor en una página ASP .NET. Se han utilizado dos controles en la página, el primero toma los valores por defecto, y en el segundo se han indicado todos los valores de sus atributos.

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<body>
<controles:EtiquetaDos runat="server" ID="Etiqueta1"/>
<controles:EtiquetaDos runat="server" ID="Etiqueta2" Repetición="3"
ColorLetra="Green" Texto="Hola"/>
</body>
</html>
```

Código fuente 211

El aspecto de esta página ASP .NET se puede ver en la Figura 117.

Figura 117

En el siguiente enlace se puede obtener la clase para el control de servidor del ejemplo y el código de la página ASP .NET que lo utiliza.

Utilizando los servicios de HtmlTextWriter

A medida que vamos ampliando la funcionalidad de nuestro control de servidor aumenta también la complejidad del código HTML que se va a generar. Para evitar mezclar código HTML con nuestra clase y para evitar errores podemos utilizar los servicios de presentación (rendering) que nos ofrece la clase HtmlTextWriter, mediante estos servicios podremos crear atributos y etiquetas HTML de forma más clara y generando un código más sencillo.

La clase HtmlTextWriter, que utilizamos en el método Render() de nuestro control ASP .NET de servidor, ofrece los siguientes servicios:

- Escritura de HTML o de otros lenguajes de presentación en la salida del usuario.
- Gestión de la creación de elementos bien formados en el lenguaje.
- Creación de atributos.
- Creación de atributos de estilos.

Para ver con claridad como nos puede ayudar la clase HtmlTextWriter a la hora de generar el código HTML que generaría nuestro control de servidor, vamos a modificar el ejemplo del subapartado anterior para que haga uso de estos servicios. El nuevo código de esta clase se puede observar en el Código fuente 212.

```
using System;
using System.Web;
using System.Web.UI;
using System.Drawing;
```

```
namespace Controles.Ejemplos
{
	public class EtiquetaTres:Control
	{
		private String _texto;
		private int _repetición=1;
		private Color _colorLetra=Color.Red;

		public String Texto
		{
			get{return _texto;}
			set{_texto=value;}
		}

		public Color ColorLetra
		{
			get{return _colorLetra;}
			set{_colorLetra=value;}
		}

		public int Repetición
		{
			get{return _repetición;}
			set{_repetición=value;}
		}

		protected override void OnInit(EventArgs evento)
		{
			base.OnInit(evento);
			if(_texto==null)
			{
				_texto="Texto por defecto";
			}
		}

		protected override void Render(HtmlTextWriter salida)
		{
			int i;
			for(i=0;i<_repetición;i++)
			{
				salida.AddStyleAttribute("color",
					ColorTranslator.ToHtml(_colorLetra));
				salida.RenderBeginTag("h3");
				salida.Write(_texto);
				salida.RenderEndTag();
			}
		}
	}
}
```

Código fuente 212

Como se puede comprobar el código es prácticamente idéntico al del apartado anterior, se diferencia en el contenido del método Render(), ya que en este caso no se ha escrito el código HTML directamente en la salida, sino que se han utilizado los métodos ofrecidos por el objeto de la clase HtmlTextWriter.

En este caso se ha utilizado el método AddStyleAttribute() para indicar que se va a aplicar un estilo con el atributo color al siguiente elemento HTML que se indique en la salida.

La llamada al método RenderBeginTag() indica al objeto HtmlTextWriter que se va a mostrar una etiqueta de apertura para el elemento h3, a esta etiqueta de inicio se le añadirá los atributos de estilos indicados previamente con el método AddStyleAttribute(), y también se le añadirán directamente los

atributos indicados previamente con la llamada al método AddAttribute(), este método añadirá un atributo a la etiqueta HTML correspondiente.

A continuación llamados al ya conocido método Write() para escribir el texto que va a ir dentro de las etiquetas HTML. Y por último para cerrar la etiqueta llamamos al método RenderEndTag().

La salida generada por el control de servidor dentro de la página ASP .NET que lo utiliza va a ser igual que en el ejemplo anterior, en el Código fuente 213 se puede ver el código HTML que se genera.

```
<html>
<head><title>Controles de servidor</title></head>
<body>
<h3 style="color:Red;">
      Texto por defecto
</h3>
<h3 style="color:Green;">
      Hola
</h3><h3 style="color:Green;">
      Hola
</h3><h3 style="color:Green;">
      Hola
</h3>
</body>
</html>
```

Código fuente 213

El Código fuente 214 es el código de la página ASP .NET que utiliza esta variación del control ASP .NET de servidor EtiquetaDos.

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<body>
<controles:EtiquetaTres runat="server" ID="Etiqueta1"/>
<controles:EtiquetaTres runat="server" ID="Etiqueta2" Repetición="3"
ColorLetra="Green" Texto="Hola"/>
</body>
</html>
```

Código fuente 214

Propiedades de clase

Las propiedades de clase son aquellas propiedades definidas en al clase de un control ASP .NET de servidor que son objetos de otra clase, y a su vez estos objetos tienen sus propiedades, que son llamadas subpropiedades.

Para acceder a estas propiedades desde las etiquetas de los controles de servidor de una página ASP .NET se debe seguir una sintaxis determinada, esta sintaxis consiste en separar las subpropiedades de las propiedades mediante un guión (-), y también tenemos la posibilidad de utilizar elementos

anidados dentro de las etiquetas principales del control ASP .NET servidor, a continuación veremos estas dos posibilidades mediante un ejemplo.

Vamos a partir de la ya conocida clase que define una etiqueta como control de servidor, en este caso nuestra clase va a tener dos propiedades simples como son Repetición y Texto (con el mismo significado que en ejemplos anteriores), pero además va a tener una propiedad llamada Formato de la clase Formateador, que se encuentra definida en el mismo fichero .CS.

La propiedad Formato, objeto de la clase Formateador, posee dos subpropiedades denominadas Color y Tamaño, que respectivamente se van a utilizar para definir el color y tamaño de la etiqueta de texto del control.

En el Código fuente 215 se ofrece el código completo del fichero .CS que va a contener a la clase Formateador y a la clase EtiquetaCuatro. Como se puede observar en este código la definición de la propiedad Formato de la clase EtiquetaCuatro no posee ninguna dificultad ni ninguna diferencia respecto al resto de las propiedades definidas en dicha clase. La única diferencia es que la propiedad no tiene un método set asociado. Para indicar los valores por defecto de la propiedad Formato se ha utilizado el constructor de la clase a la que pertenece.

```
using System;
using System.Web;
using System.Web.UI;
using System.Drawing;

namespace Controles.Ejemplos
{
	public class Formateador
	{
		private int _tamaño;
		private String _color;

		public Formateador(int _tamaño, System.String _color)
		{
			this._tamaño=_tamaño;
			this._color=_color;
		}
		public String Color
		{
			get {return _color;}
			set {_color = value;}
		}
		public int Tamaño
		{
			get {return _tamaño;}
			set { _tamaño= value;}
		}
	}

	[ParseChildren(true)]
	public class EtiquetaCuatro:Control
	{
		private String _texto;
		private int _repetición=1;
		private Formateador _formato=new Formateador(2,"Orange");

		public Formateador Formato
		{
			get{return _formato;}
		}

		public String Texto
```

```
            {
                  get{return _texto;}
                  set{_texto=value;}
            }

            public int Repetición
            {
                  get{return _repetición;}
                  set{_repetición=value;}
            }

            protected override void OnInit(EventArgs evento)
            {
                  base.OnInit(evento);
                  if(_texto==null)
                  {
                        _texto="Texto por defecto";
                  }
            }

            protected override void Render(HtmlTextWriter salida)
            {
                  int i;
                  for(i=0;i<_repetición;i++)
                  {
                        salida.AddAttribute("color",Formato.Color);
                        salida.AddAttribute("size",Formato.Tamaño.ToString());
                        salida.RenderBeginTag("font");
                        salida.Write(_texto);
                        salida.RenderEndTag();
                        salida.RenderBeginTag("br");
                  }
            }
      }
}
```

Código fuente 215

En el código del fichero .CS se puede observar que antes de la declaración de la clase del control de servidor se ha indicado el atributo ParseChildren asignándole el valor true, esto es necesario para que sea posible utilizar uno de los tipos de sintaxis de acceso a las subpropiedades, en concreto se trata de la sintaxis que permite anidar las subpropiedades con sus atributos correspondientes, en el ejemplo se verán los dos tipos de sintaxis.

Otro punto interesante del código del ejemplo es la utilización de los servicios de presentación del objeto HtmlTextWriter para generar la etiqueta font y añadirle dos propiedades, como se puede ver para la etiqueta de salto de línea (etiqueta <br> de HTML) sólo se ha aplicado el método de apertura de etiqueta, ya que esta etiqueta no posee cierre.

Una vez visto el código del control ASP .NET de servidor vamos a comentar la forma en la que se utilizaría dentro de una página ASP .NET. Como ya hemos adelantado, para hacer referencia a las subpropiedades del control EtiquetaCuatro (propiedades de la clase Formateador), disponemos de dos posibilidades.

La primera de estas posibilidades consiste en separar mediante un guión las propiedades de las subpropiedades, así para acceder a la propiedad Color tendríamos que escribir el siguiente atributo dentro de la etiqueta del control: Formato-Color.

La otra posibilidad es la de anidar la propiedad Formato dentro de las etiquetas de apertura y cierre del control y en la etiqueta de la propiedad Formato acceder a las subpropiedades como atributos de dicha

etiqueta, esta otra sintaxis es la que requiere que el atributo ParseChildren de la clase del control ASP .NET de servidor posea el valor true.

En el Código fuente 216 se muestra distintos usos del control EtiquetaCuatro, en el primero se utilizan los valores por defecto de las propiedades y subpropiedades, en el segundo se indican las subpropiedades utilizando la sintaxis de separación mediante guiones y en el tercero se anidan las etiquetas del control de servidor y de su propiedad de clase.

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<body>
<controles:EtiquetaCuatro runat="server" ID="Etiqueta1"/>
<controles:EtiquetaCuatro runat="server" ID="Etiqueta2"
Repetición="3" Texto="Hola" Formato-Color="Green" Formato-Tamaño="6"/>

<controles:EtiquetaCuatro runat="server" ID="Etiqueta3" Repetición="2"
Texto="Mundo">
      <Formato Color="Red" Tamaño="4"/>
</controles:EtiquetaCuatro>
</body>
</html>
```

Código fuente 216

En la Figura 118 se puede ver el resultado de la ejecución de esta página.

Figura 118

El Código fuente 217 es el código HTML que genera la ejecución de la página ASP .NET.

```
<html>
<head><title>Controles de servidor</title></head>
<body>
<font color="Orange" size="2">Texto por defecto</font><br>

<font color="Green" size="6">Hola</font><br>
             <font color="Green" size="6">Hola</font><br>
                    <font color="Green" size="6">Hola</font><br>


<font color="Red" size="4">Mundo</font><br>
                                 <font color="Red" size="4">Mundo</font><br>

</body>
</html>
```

Código fuente 217

En el siguiente enlace se encuentra disponible el código del fichero .CS que define las clases y el código de la página .ASPX que hace uso del control de servidor definido en la clase.

Heredando de la clase System.Web.IU.WebControl

Para controles de servidor similares a nuestra etiqueta que precisan de la gestión de un estilo para su aspecto, es recomendable que la clase que define dicho control herede de la clase System.Web.UI.WebControl, que nos va a ofrecer una gran funcionalidad en forma de propiedades.

Al heredar de la clase System.Web.IU.WebControl nuestro control ASP .NET de servidor obtiene de forma automática la siguiente funcionalidad:

- Ofrece propiedades que permiten manejar los estilos del control, como pueden ser ForeColor, BackColor o Font.

- Ofrece consistencia con los controles ASP .NET estándar, ya que heredamos de su misma clase.

- De forma automática se almacena el estilo y estado del control durante sucesivas llamadas a la página, cosa que los controles se servidor de los ejemplos anteriores no hacían de forma automática.

Para realizar un uso correcto de la clase WebControl debemos tener en cuenta lo siguiente dentro del código de nuestra clase:

- Se debe heredar de la clase WebControl, en lugar de la clase Control. Para heredar de esta clase se debe importar el espacio de nombres System.Web.UI.WebControls.

- Se debe declarar un constructor público que llame al constructor de la padre, indicando el elemento HTML que se va mostrar, en nuestra caso va a ser una etiqueta H3.

- Se tiene que sobrescribir el método RenderContents() para indicar el contenido que va a existir en el elemento H3. La clase WebControl de encargará de aplicar los atributos sobre las etiquetas así como los estilos, por lo tanto se puede eliminar el código dedicado a estas tareas.

- Se deben eliminar las propiedades del control de servidor que implementen estilos, ya que la clase WebControl los implentará por nosotros de forma automática.

A continuación, en el Código fuente 218, se ofrece una nueva versión de nuestra clase que define un control de servidor que realiza las funciones de una etiqueta. Como se puede ver el código es muy sencillo, hemos tenido en cuenta las recomendaciones indicadas.

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Drawing;
namespace Controles.Ejemplos
{
	public class EtiquetaWeb:WebControl
	{
		private String _texto;
		private int _repetición=1;
		public EtiquetaWeb():base("H3")
		{
		}
		public String Texto
		{
			get{return _texto;}
			set{_texto=value;}
		}
		public int Repetición
		{
			get{return _repetición;}
			set{_repetición=value;}
		}
		protected override void OnInit(EventArgs evento)
		{
			base.OnInit(evento);
			if(_texto==null)
			{
				_texto="Texto por defecto";
			}
		}
		protected override void Render(HtmlTextWriter salida)
		{
			int i;
			for(i=0;i<_repetición;i++)
			{
				base.Render(salida);
			}
		}
		protected override void RenderContents(HtmlTextWriter salida)
		{
			salida.Write(_texto);
		}
	}
}
```

Código fuente 218

En el código de nuestra clase se debe destacar la forma en la que se ha implementado la propiedad Repetición, como se puede ver se llama al método Render() de la clase padre el número de veces indicado por la variable privada _repetición.

El Código fuente 219 es el código de una página ASP .NET que hace uso de este nuevo control ASP .NET de servidor, como se puede comprobar se asignan valores a atributos que son comunes a todos lo controles Web, como puede ser Font-Size, BackColor, etc.

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<body>
<controles:EtiquetaWeb runat="server" ID="Etiqueta1"/>
<controles:EtiquetaWeb  runat="server" ID="Etiqueta2" Repetición="3"
ForeColor="Green" Font-Size="14" BackColor="Orange" Font-Name="Courier"
Font-Italic="True" Texto="Hola"/>
</body>
</html>
```

Código fuente 219

En la Figura 119 se puede observar la ejecución de esta página.

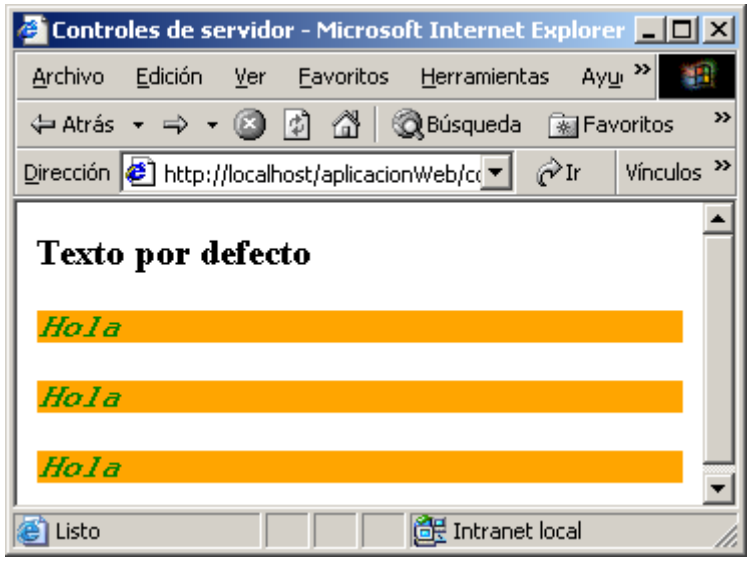


Figura 119

En el siguiente enlace se puede obtener el fichero de la clase y el fichero de la página.

Controles compuestos

Podemos crear nuevos controles de servidor a partir de la combinación de controles existentes, los controles de servidor compuestos son similares a los controles de usuario.

Por lo tanto un control de servidor puede ser contenedor de otros controles, la clase Control ofrece una propiedad llamada Controls, de la clase ControlsCollection, que contiene una colección con los controles hijos del control de servidor actual. Cuando se muestra un control de servidor compuesto,

cada uno de sus controles hijo también se mostrarán, si un control hijo tiene a su vez hijos se realiza el mismo proceso para ellos, así hasta que todos los controles se hayan mostrado.

Por defecto, al heredar nuestra clase de la clase Control, los elementos anidados dentro de las etiquetas del control de servidor se añaden de forma automática a la colección Controls del control de servidor.

Si tenemos el Código fuente 220 que se corresponde con una definición de una clase que básicamente no hace nada, y la utilizamos para generar un control ASP .NET de servidor, podemos utilizar este control de servidor mediante el siguiente código de una página ASP .NET (Código fuente 221).

```
using System;
using System.Web;
using System.Web.UI;

namespace Controles.Ejemplos
{
	public class Simple:Control
	{
	}
}
```

Código fuente 220

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<body>
<controles:Simple runat="server">
 <asp:Button Text="Un botón" Runat="server"/>
 Texto
</controles:Simple>
</body>
</html>
```

Código fuente 221

Tanto el control Button como el texto (de forma automática los literales se transforman en controles de la clase LiteralControl), se mostrarán en la página, ya que se llamará de forma automática a sus métodos Render(), a través de la colección Controls del control de servidor que hemos desarrollado. La página ASP .NET que se genera tiene el aspecto de la Figura 120.

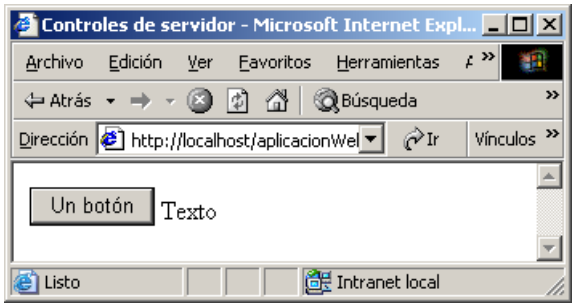


Figura 120

Normalmente cuando se va a crear un control compuesto, es el propio control quién decide qué controles hijos debe tener, es decir, es el encargado de crear sus controles hijos, y no lo va a ser el usuario del control como sucedía en el ejemplo anterior. Si el usuario (página ASP .NET) va a ser quién va crear todo el interfaz con los controles hijos se debe recurrir mejor a un control de usuario en lugar de a un control ASP .NET de servidor compuesto.

Los controles compuestos van a sobrescribir el método CreateChildControls() para añadir controles hijos a la colección Controls, el tipo de estos controles dependerá del interfaz de usuario que deseemos crear con nuestro control compuesto.

En el Código fuente 222 se ofrece el código correspondiente a una clase que crea un control de servidor compuesto, este control va estar formado por dos controles hijos, un objeto de la clase LiteralControl un objeto de la clase TextBox, es decir, un texto y una caja de texto. Además este control ASP .NET de servidor ofrece una propiedad llamada Valor, de tipo entero, que se corresponde con la propiedad Text del objeto TextBox que es uno de los hijos del control.

```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Controles.Ejemplos
{

	public class ControlCompuesto : Control, INamingContainer
	{

		public int Valor
		{
			get
			{
				this.EnsureChildControls();
				return Int32.Parse(((TextBox)Controls[1]).Text);
			}
			set
			{
				this.EnsureChildControls();
				((TextBox)Controls[1]).Text = value.ToString();
			}
		}

		protected override void CreateChildControls()
		{
			this.Controls.Add(new LiteralControl("<b>Valor: </b>"));
			TextBox caja = new TextBox();
			caja.Text = "0";
			this.Controls.Add(caja);
		}
	}
}
```

Código fuente 222

Como se puede comprobar en el código anterior, la clase además de heredar de la clase Control también hereda del interfaz INamingContainer, de esta forma le indicamos que nuestra clase va a ser una contenedora de otros controles.

Al asignar un valor a la propiedad Valor se recupera la propiedad Text del segundo control, para ello se accede al segundo índice de la colección Controls, y para poder acceder a la propiedad Text se debe aplicar el mecanismo de casting con la clase TextBox.

La llamada que se hace al método EnsureChildControls() nos asegura que los controles hijos se crearán solo una vez, y que estarán siempre creados cuando vayamos a acceder a ellos dentro de nuestro código de la clase del control compuesto.

Otro aspecto a tener en cuenta de esta clase es que no se sobrescribe el método Render() de la clase Control, ya que al ser un control compuesto se llamarán de forma automática todos los métodos Render() de cada uno de los controles hijo de la propiedad Controls.

Y en el método CreateChildControls(), se puede comprobar algo que ya habíamos adelantado, es en este método en el que se va creando el interfaz de usuario del control compuesto mientras que se van añadiendo los controles hijos a la colección Controls.

Ahora nos queda utilizar este control compuesto desde una página ASP .NET, en este caso se va a utilizar en un Web Form con dos objetos de la clase Button. Al pulsar cada uno de los botones de la página se va a ir añadiendo o restando, según sea el caso, una unidad a la propiedad Valor de nuestro control. El Código fuente 223 es el código de esta página ASP .NET.

```
                                <%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<script runat=server>
      private void Suma(Object sender, EventArgs e) {
            MiControl.Valor++;
    }

    private void Resta(Object sender, EventArgs e) {
          MiControl.Valor--;
    }

</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<Controles:ControlCompuesto id="MiControl" runat=server/>
<br>
<asp:button text="Suma" OnClick="Suma" runat="server" ID="boton1"/> |
<asp:button text="Resta" OnClick="Resta" runat="server" ID="boton2"/>

</form>

</html>
```

Código fuente 223

En la Figura 121 se puede ver un ejemplo de ejecución de la página.

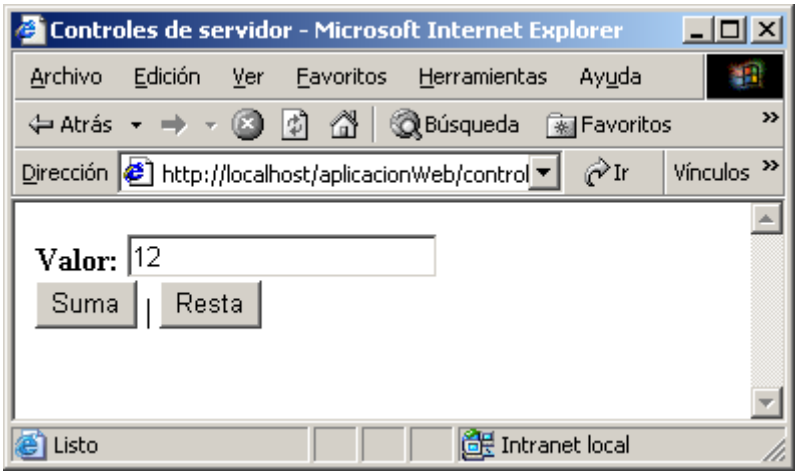


Figura 121

Tratando los datos enviados

Por defecto los controles ASP .NET de servidor no manejan los datos enviados al servidor cuando se envía un Web Form. Así si tenemos una caja de texto, que es un control de servidor desarrollado por nosotros, dentro de un formulario Web con un objeto Button que realiza un envío al servidor de dicho formulario, al pulsar sobre dicho botón el texto que se haya introducido en la caja de texto se pierde.

Si deseamos que nuestro control de servidor recupere los datos enviados por el usuario debemos implementar el interfaz IPostBackDataHandler. Este interfaz posee dos métodos: LoadPostData() y RaisePostDataChangedMethod().

El método LoadPostData() se ejecuta cuando un control, en nuestro caso un control Button, envía datos al servidor, este método se lanzará en todos los controles de la página. Este método recibe como parámetro todos los datos enviados al servidor utilizando un objeto de la clase NameValueCollection. Además de este parámetro que posee todos los valores en pares clave/valor, este método recibe como parámetro un objeto de la clase String que va a ser el identificador del control actual, así que podremos utilizar este parámetro como clave dentro de la colección NameValueCollection.

El método LoadPostData() devuelve un valor de tipo Boolean. Si se devuelve el valor true se ejecutará el método RaisePostDataChangedMethod(), si se devuelve el valor false este método no se ejecutará. Un control deberá devolver true si desea lanzar un evento como resultado de ciertos datos que se han enviado, los eventos deberán ser lanzados en el método RaisePostDataChangedMethod().

Teniendo todo lo anterior en cuenta, vamos a crear un sencillo control de servidor que consiste en una caja de texto que va a recuperar los datos enviados por el cliente y asignárselos a su propiedad Texto, que va a ser el texto que va a contener. En el Código fuente 224 se puede ver el código de la clase del control.

```
using System;
using System.Web;
using System.Web.UI;
using System.Collections.Specialized;
using System.Web.UI.WebControls;

namespace Controles.Ejemplos
{
```

```
	public class ControlPostBack: Control, IPostBackDataHandler
	{

		private String _texto="Texto por defecto";

		public String Texto
		{
			get {return _texto;}
			set {_texto = value;}
		}

		public bool LoadPostData(String clave, NameValueCollection valores)
		{
			_texto = valores[clave];
			return false;
		}

		public void RaisePostDataChangedEvent()
		{
		}

		protected override void Render(HtmlTextWriter salida)
		{
			salida.AddAttribute("type","text");
			salida.AddAttribute("name",this.UniqueID);
			salida.AddAttribute("value",this.Texto);
			salida.RenderBeginTag("input");
			salida.RenderEndTag();

		}
	}
}
```

Código fuente 224

Como se puede comprobar se han implementado los métodos del interfaz IpostBackDataHandler, y también se ha sobrescrito el método Render() para mostrar en pantalla la caja de texto, aquí se debe prestar atención a la creación de los atributos de la misma.

Para que el mecanismo de PostBack y recuperación de los datos enviados funcione correctamente, se debe utilizar el atributo name, asignándole el valor devuelto por la propiedad UniqueID de nuestro control de servidor, esta propiedad va a contener el identificador de nuestro control, este identificador se indica en la página mediante el atributo ID o en caso contrario es asignado automáticamente por ASP .NET.

En el método LoadPostData() se ha recuperado el valor de nuestro control accediendo a la colección de la clase NameValueCollection utilizando como índice el primer parámetro de este método. Para poder utilizar esta colección se debe importar el NameSpace System.Collections.Specialized.

Para demostrar que este control va a conservar los datos que el envíe el usuario, vamos a utilizar en una página ASP .NET que posee un objeto Button que enviará los datos del formulario al servidor, el Código fuente 225 es el código de la página.

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<body>
```

```
<form method="POST" runat="server" ID="Formulario">
<Controles:ControlPostBack id="MiControl" runat="server"/>
<br>
<asp:button text="Enviar" runat="server" ID="boton"/>
</form>

</html>
```

Código fuente 225

En la Figura 122 se puede ver en acción este control ASP .NET de servidor.

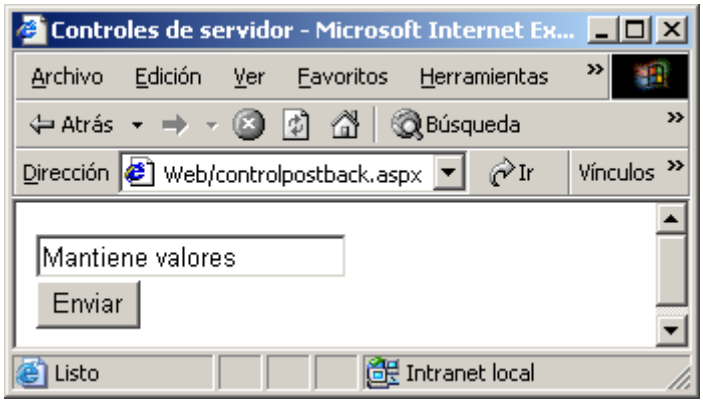


Figura 122

En el Código fuente 226 se puede ver el código HTML generado en este ejemplo.

```
<html>
<head><title>Controles de servidor</title></head>
<body>
<form name="Formulario" method="POST" action="controlpostback.aspx"
id="Formulario">
<input type="hidden" name="__VIEWSTATE" value="dDw4OTA1NzQ0NDM7Oz4=" />

<input type="text" name="MiControl" value="Mantiene valores" />
<br>
<input type="submit" name="boton" value="Enviar" id="boton" />
</form>

</html>
```

Código fuente 226

En el siguiente enlace se puede obtener el código de la clase del control y el código de la página ASP .NET que lo utiliza.

Manteniendo el estado de los controles

En el capítulo dedicado a los Web Forms vimos que una característica interesante que ofrecían es que permitían mantener el estado de un control de forma automática entre distintas recargas de una misma

página ASP .NET, para ello se utilizaba el mecanismo VIEWSTATE, mediante un campo oculto que almacenaba toda la información de los controles de formulario.

Todos los controles, que heredan de la clase Control, poseen una propiedad llamada ViewState de la clase Sytem.Web.UI.StateBag que permite almacenar el estado del control. Esta clase implementa el interfaz IDictionary, por lo que podemos almacenar información en forma de clave/valor, la clave va a ser una cadena y el valor va a ser un objeto de la clase System.Object, por lo que al recuperar los valores de la propiedad ViewState se debe utiliza el mecanismo de casting.

Vamos mostrar la utilización de la propiedad ViewState mediante un control ASP .NET de servidor muy sencillo, se trata de una etiqueta que posee dos propiedades, Texto y Tamaño, que indican el texto que se va a mostrar y su tamaño correspondiente, vamos utilizar la propiedad ViewState para almacenar los valores de estas propiedades, de esta forma no tendremos que utiliza variables privadas como en casos anteriores.

El Código fuente 227 es el código de la clase del control de servidor.

```
using System;
using System.Web;
using System.Web.UI;

namespace Controles.Ejemplos
{

	public class EtiquetaEstado: Control
	{

		public String Texto
		{
			get {return (String) ViewState["valorTexto"];}
			set {ViewState["valorTexto"] = value;}
		}

		public int Tamaño
		{
			get {return (int) ViewState["valorTamaño"];}
			set {ViewState["valorTamaño"] = value;}
		}

		protected override void Render(HtmlTextWriter salida)
		{
			salida.Write("<font size=" + this.Tamaño + ">" + this.Texto
                                                        + "</font>");
		}
	}
}
```

Código fuente 227

Para comprobar que los valores de las propiedades se mantienen entre distintas llamadas y recargas de una página ASP .NET, vamos a utilizar este control en una página ASP .NET (Código fuente 228) que posee un Web Form con dos objetos Button, y según el botón que se pulse se incrementará y decrementará el tamaño del texto del control de servidor.

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
```

```
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<script runat="server">
      private void Suma(Object sender, EventArgs e) {
            MiControl.Tamaño++;
            MiControl.Texto="Aumenta el tamaño";
    }

    private void Resta(Object sender, EventArgs e) {
          MiControl.Tamaño--;
          MiControl.Texto="Disminuye el tamaño";
    }

</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<Controles:EtiquetaEstado id="MiControl" runat="server"
 Texto="Etiqueta" Tamaño="1"/>
<br>
<asp:button text="Suma" OnClick="Suma" runat="server" ID="boton1"/> |
<asp:button text="Resta" OnClick="Resta" runat="server" ID="boton2"/>
</form>

</html>
```

Código fuente 228

En la Figura 123 se puede ver un ejemplo de ejecución de esta página.

Figura 123

El mantenimiento del estado de un control de servidor también nos será útil en el siguiente subapartado, que se encuentra dedicado a la forma en la que se lanzan los eventos desde los controles de servidor.

Lanzando eventos

ASP .NET ofrece un potente modelo de eventos del lado del servidor, mientras se va creando una página ASP .NET, los controles contenidos en la misma pueden lanzar eventos. Estos eventos pueden ser capturados por la página ASP .NET o bien por otros controles.

Para demostrar la utilización de eventos dentro de controles ASP .NET de servidor vamos a construir un control que sea una caja de texto que al cambiar el texto que contiene lance un evento. Este ejemplo además de utilizar eventos también vamos a hacer referencia a lo visto en subapartados anteriores, ya que como veremos se encuentra muy relacionados.

Para este apartado vamos a partir de la clase del Código fuente 224, que la veíamos en el apartado dedicado a recuperar lo datos enviados por el cliente.

Para incorporar un evento en nuestro componente debemos declarar un evento público de la clase EvenHandler. Debemos recordar que nuestro control implementa el interfaz IPostBackDataHandler y por lo tanto implementa los métodos LoadPostData() y RaisePostDataChangedEvent(), y va a ser en el método LoadPostData() dónde indicamos que se desea lanzar un evento y en el método RaisePostDataChangedEvent() es dónde se lanza realmente.

El método LoadPostData() devolverá true para indicar que deseamos que se ejecute el método RaisePostDataChangedEvent(). Pero sólo devolveremos true si el valor de la caja de texto ha cambiado respecto a su valor actual, para ello compararemos el valor de la propiedad Texto con el valor indicado por el usuario.

Para almacenar la propiedad Texto vamos a hacer uso de la propiedad ViewState, tal como veíamos en el apartado anterior, de esta forma podremos saber el valor anterior que tenía esta propiedad antes de ser modificada por el usuario y enviada al servidor. El valor indicado por el usuario lo tenemos disponible a través de la colección NameValueCollection que el método LoadPostData() recibe como parámetro.

Si el método LoadPostData() devuelve true, se ejecutará el método RaisePostDataChangedEvent() y será dentro de este método dónde se lance el evento que indica que el texto ha sido modificado.

Pero se recomienda no lanzar el evento directamente dentro del método RaisePostDataChangedEvent(), sino que será dentro de este método dónde se llame a otro método que si es el que lanza el evento, de esta forma la clases que hereden de la actual pueden sobrescribir el método que lanza el evento si lo desean.

Teniendo en cuenta todas las consideraciones anteriores, el código que presentaría la clase de nuestro control de servidor que lanza un evento cuando se cambia el texto que contiene, es el que se puede observar en el Código fuente 229.

```
using System;
using System.Web;
using System.Web.UI;
using System.Collections.Specialized;
using System.Web.UI.WebControls;

namespace Controles.Ejemplos
{

       public class ControlEvento: Control, IPostBackDataHandler
       {

              public event EventHandler TextoCambiado;

              public String Texto
              {
                     get {return (String) ViewState["valorTexto"];}
                     set{ViewState["valorTexto"]=value;}
              }
              public bool LoadPostData(String clave, NameValueCollection valores)
```

```
            {
                  bool lanzaEvento=false;
                  if(!Texto.Equals(valores[clave])) lanzaEvento=true;
                  Texto = valores[clave];
                  return lanzaEvento;
            }

            public void RaisePostDataChangedEvent()
            {
                  OnTextoCambiado(EventArgs.Empty);
            }

            protected void OnTextoCambiado(EventArgs evento)
            {
                  if(TextoCambiado!=null) TextoCambiado(this,evento);
            }

            protected override void Render(HtmlTextWriter salida)
            {
                  salida.AddAttribute("type","text");
                  salida.AddAttribute("name",this.UniqueID);
                  salida.AddAttribute("value",this.Texto);
                  salida.RenderBeginTag("input");
                  salida.RenderEndTag();

            }
      }
}
```

Código fuente 229

Como se puede ver en el código anterior el evento se ha denominado TextoCambiado, por lo tanto a la hora de tratar este evento dentro de la página ASP :NET deberemos utilizar como atributo de la etiqueta del control el atributo OnTextoCambiado.

Ahora ya podemos utilizar este control dentro de una página ASP .NET. Debemos tener en cuenta que la página debe tener un método que trate el evento TextoCambiado, este método se invocará desde el atributpo OnTextoCambiado, y consiste únicamente en indicar que el texto se ha cambiado, asignándole una cadena a un objeto Label.

Como se puede comprobar en el código de la página ASP .NET (Código fuente 230) se ha asignado a la propiedad EnableViewState del objeto Label el valor falso, esto es así para que en sucesivas llamadas no se mantenga el valor de la etiqueta, ya que el texto sólo deberá aparecer cuando el control de servidor lance el evento TextoCambiado.

```
<%@ Page language="C#"%>
<%@ Register TagPrefix="Controles" NameSpace="Controles.Ejemplos"
Assembly="ControlesServidor"%>
<html>
<head><title>Controles de servidor</title></head>
<script runat="server">
      void IndicaCambio(Object sender, EventArgs e) {
            etiqueta.Text="El texto ha cambiado";
    }
</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<Controles:ControlEvento id="MiControl" runat="server"
OnTextoCambiado="IndicaCambio"/>
<br>
```

```
<asp:button text="Enviar" runat="server" ID="boton"/><br>
<asp:Label Runat="server" ID="etiqueta" EnableViewState="False"/>
</form>

</html>
```

Código fuente 230

En la Figura 124 se puede ver un ejemplo de ejecución de esta página.

Figura 124

En el siguiente enlace se puede obtener la clase del control de servidor y la página ASP .NET que lo utiliza.

Con este subapartado damos por terminada esta introducción a la creación de controles ASP .NET de servidor, ya que el tema es tan extenso y tiene tantos matices y complejidades que podríamos dedicar prácticamente un curso completo al desarrollo de controles ASP .NET de servidor.

En el próximo capítulo vamos a comentar el tratamiento de errores dentro de ASP .NET, así como la utilización de los mecanismos de trazas y de depuración.

15

Tratamiento de errores

Introducción

En este capítulo y en el siguiente además de comentar el tratamiento de errores en ASP .NET, también vamos a ver otros temas muy relacionados con los errores, como son el mecanismo de trazas y el de depuración que nos ofrece ASP .NET.

En este capítulo veremos que el tratamiento de errores dentro de ASP .NET está mucho mejor cuidado que en sus versiones anteriores ofreciendo un tratamiento de errores estructurado a través del CLR (Common Language Runtime), recordamos al lector que el Entorno de Ejecución Común de Lenguajes o CLR, representa el alma de .NET Framework y es el encargado de la ejecución del código de las aplicaciones.

Anteriormente sólo disponíamos de la sentencia On Error Resume Next, y no teníamos la posibilidad de especificar una etiqueta para el tratamiento de errores, lo que suponía una dificultad a la hora de crear rutinas centralizadas para el tratamiento de errores.

Otro aspecto que se debe destacar del tratamiento de errores en ASP :NET respecto a ASP 3.0, es que el objeto ASPError, que nos ofrecía una descripción detallada del error que se había producido, ha desaparecido en ASP .NET, ahora los errores se representan con excepciones, como ya veremos más adelante.

Además del sistema estructurado para el tratamiento de errores que nos ofrece el CLR, ASP .NET nos ofrece otra serie de mecanismos propios para el tratamiento de errores que también comentaremos en el presente capítulo.

Tratamiento de errores estructurado

El tratamiento de errores estructurado es una parte fundamental del CLR, y ofrece a los programadores en .NET una forma muy interesante de gestión de errores, además de las siguientes características:

- Es independiente del lenguaje, por lo que las excepciones se pueden crear y ser lanzadas en un lenguaje y ser atrapadas en otro.
- Es independiente del proceso y de la máquina, podemos atrapar excepciones localmente que son lanzadas por componentes .NET remotos.
- Es un sistema jerárquico, permitiendo ordenar en capas los distintos tipos de excepciones.
- No es necesario comprobar valores de retorno en las llamadas a métodos, las excepciones se lanzarán de forma automática siempre que se produzcan.

Este tratamiento de errores se basa en capturar excepciones, una excepción va a ser una respuesta a situaciones anormales o erróneas que se producen en el tiempo de ejecución de la aplicación. Todos los tipos de excepciones heredan de una clase del .NET Framework llamada System.Exception, esta clase nos va a ofrecer la información detallada sobre el error que se ha producido, si lo comparamos con la versión anterior de ASP, viene a sustituir al objeto integrado ASPError.

La clase Exception

La clase Exception posee las siguientes propiedades para informarnos sobre el error que se ha producido:

- HelpLink: es una cadena que contiene una URL al fichero de ayuda asociado por dicho error.
- HResult: devuelve un código numérico para identificar la excepción que se ha producido, se utiliza para interoperatividad con COM.
- InnerException: esta propiedad tiene sentido cuando hay excepciones anidadas, devuelve un objeto Exception que representa la excepción interna.
- Message: devuelve una cadena con la descripción textual del error que se ha producido.
- Source: devuelve una cadena que contiene el nombre de la aplicación u objeto que ha lanzado el error.
- StackTrace: esta propiedad devuelve una cadena con las trazas de la pila de llamadas. Indica el lugar dentro del código en el que se ha producido la excepción.
- TargetSite: devuelve el método que lanzó la excepción.

En el siguiente apartado, dónde veremos los bloques try/catch veremos ejemplos que utilizan un objeto de la clase System.Exception para mostrar información sobre el error que se ha producido.

Try/Catch (Tratando las excepciones)

En este apartado vamos a ver ejemplos de tratamiento de excepciones dentro de páginas ASP .NET utilizando los bloques try/catch, también recordaremos algunos de los conceptos que comentábamos en el capítulo dedicado al lenguaje C#.

El tratamiento de errores es muy similar al del lenguaje Java, desde C# vamos a utilizar las sentencias try{...}catch{...}finally{...}, que se utilizan para atrapar excepciones dentro de nuestro código.

problemas en la programación, al acceder por ejemplo a un elemento de un array que no existe.

En C#, cuando se produce un error en un método, el método crea un objeto excepción y se lo pasará al entorno de ejecución. Este objeto contendrá información acerca del error que se ha producido.

Los candidatos a manejar o tratar el error que se ha producido son los métodos que se encuentran en la pila de llamadas del método en el que se ha producido el error, es decir, los métodos que se han invocado antes. El entorno de ejecución va buscando hacia atrás en la pila de llamadas desde el método en el que se ha producido el error, hasta que encuentra el método que tiene el tratamiento de la excepción adecuado.

Un tratamiento de excepción adecuado se produce cuando la excepción que se trata es del mismo tipo que la que se ha lanzado al producirse el error. Cuando se trata la excepción en la terminología C# se dice que se ha atrapado la excepción.

Para tratar excepciones, el primer paso es declarar un bloque try que englobe a todas las sentencias susceptibles de producir una excepción. Si se produce alguna excepción en los métodos comprendidos dentro del bloque try, se ejecutará el manejador de excepciones asociado al bloque try.

El manejador de excepciones asociado a un bloque try se indica mediante uno o más bloques catch. En su forma más sencilla el bloque catch puede ir vacío, sin ninguna sentencia. Los parámetros de catch declaran una variable del tipo de excepción que se desea atrapar. Mediante la variable de la excepción se recogerá el objeto correspondiente a la excepción que se ha producido y se utilizará para consultar la información que ofrece.

Podemos utilizar para un mismo bloque try varios bloques catch, de esta forma podremos indicar diferentes tipos de excepciones a tratar. Se debe ir desde la excepción más particular a la más general, para que no se oculten entre sí, cuando se produzca la excepción se ejecutará el bloque catch que corresponda al tipo de excepción en cada caso.

Como hemos visto a un bloque try le deben corresponder uno o más bloques catch, y hay veces en las que también le puede corresponder un bloque finally, aunque no de forma obligatoria.

En un bloque finally escribiremos todo el código de limpieza (liberar objetos, ficheros, etc.) en el caso de que se produzca una excepción, finally nos asegura que todas las sentencias de su bloque se ejecutarán cuando se produzca la excepción, una vez tratada por el bloque catch correspondiente. Se debe tener en cuenta que el bloque finally siempre se ejecutará, aunque no se haya producido una excepción.

En el Código fuente 231 ofrecemos un esquema de código que relaciona estos tres tipos de bloques, así un bloque try tendrá uno o más bloques catch y uno o ningún bloques finally.

```
try {
      ...
```

```
} catch (ClaseExcepción nombreVariable) {
      ...
} [catch (ClaseExcepción nombreVariable) {
      ...
}
[....]
[finally{
      .....
}]
```

Código fuente 231

Para lanzar una excepción desde nuestro código utilizaremos la palabra reservada throw, su sintaxis es muy sencilla únicamente debe ir seguida de un objeto que representa la excepción que se ha producido, que será un objeto que hereda de la clase System.Exception.

Una vez comentada la teoría sobre el tratamiento de excepciones con los bloques try/catch, vamos a ver un ejemplo muy sencillo de utilización de estos bloques dentro de una página ASP .NET.

En este ejemplo vamos a mostrar en un objeto TextBox el contenido del fichero de texto que nos indique el usuario a través de otra caja de texto, al pulsar el botón del Web Form se abrirá el fichero y se mostrará su contenido.

Dentro del bloque try se va a encontrar el código encargado de abrir el fichero y de mostrar su contenido, ya que es el código que la operación de apertura de un fichero y de su lectura puede producir excepciones.

Para el bloque try se han utilizado tres bloques catch, es decir, se van a tratar tres tipos de excepciones distintas, estas excepciones deben ordenarse desde la más particular a la más general. La excepción más particular es de la clase ArgumentException, esta excepción de lanzará cuando no indiquemos ningún nombre de fichero, es decir, cuando se realice la llamada al método constructor FileStream() faltándole el parámetro del nombre de fichero.

La segunda excepción es de la clase FileNotFoundException y se producirá cuando no se localice el fichero indicado. El siguiente tipo de excepción es de la clase IOException y se lanzará cuando se produzca un error de entrada/salida genérico.

Y por último tenemos la excepción más general, que es de la clase Exception, y que representa a cualquier tipo de excepción que se produzca. Este último bloque catch únicamente se ejecutará si la excepción que se ha producido no es de ninguno de los tipos de las anteriores.

En los bloques catch aparece un código muy similar en todos los casos, que consiste en mostrar el mensaje de error correspondiente, en ocultar la caja de texto que contiene le texto del fichero y en eliminar su contenido.

En el bloque finally únicamente se muestra un texto para demostrar que este bloque siempre se va ejecutar.

El Código fuente 232 es el código completo de la página ASP .NET que incorpora el tratamiento de errores comentado.

```
<%@ Page language="C#"%>
<%@ Import NameSpace="System.IO"%>
<html>
```

```
<head><title>Tratamiento de excepciones</title></head>
<script runat="server">
void MostrarFichero(Object sender, EventArgs evento) {

    try{
        FileStream fs = new FileStream(nombreFichero.Text, FileMode.Open);
        StreamReader sr = new StreamReader(fs);
        String contenido;
        contenido = sr.ReadToEnd();
        resultado.Visible=true;
        resultado.Text=contenido;
        fs.Close();
    }
    catch(ArgumentException excepcion){
             error.Text="Debe indicar el nombre del fichero";
             resultado.Text="";
             resultado.Visible=false;
    }
    catch(FileNotFoundException excepcion){
             error.Text="No se ha encontrado el fichero";
             resultado.Text="";
             resultado.Visible=false;
    }
    catch(IOException excepcion){
             error.Text="Excepción de entrada/salida";
             resultado.Text="";
             resultado.Visible=false;
    }
    catch(Exception excepcion){
             error.Text="Se ha producido la excepción: "+excepcion.Message;
             resultado.Text="";
             resultado.Visible=false;
    }
    finally{
             Response.Write("siempre se ejecuta");
    }
}
</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<asp:TextBox Runat="server" ID="nombreFichero"/>
<asp:button text="Enviar" runat="server" ID="boton" OnClick="MostrarFichero"/><br>
<asp:Label ID="error" Runat="server" EnableViewState="False" Text=""/><br>
<asp:TextBox TextMode="MultiLine" Runat="server" ID="resultado"
Columns="40" Rows="5" EnableViewState="False" Text="" Visible="False"/>
</form>
</html>
```

Código fuente 232

En el código anterior se puede ver que se ha importado el espacio de nombres System.IO, este NameSpace es necesario para utilizar las clases de acceso a ficheros, como son FileStream y StreamReader.

En la Figura 125 se muestra un ejemplo de ejecución de la página ASP .NET cuando todo es correcto y no se ha producido ninguna excepción.

Figura 125

Y en la Figura 126 se muestra un ejemplo de ejecución de la página ASP .NET cuando se produce una excepción, en este caso cuando el fichero indicado no se ha podido localizar.

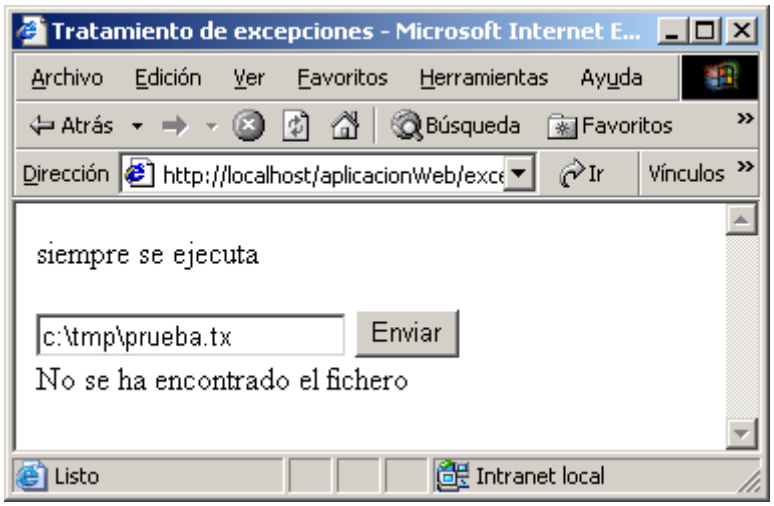


Figura 126

En el siguiente apartado vamos a comentar como podemos lanzar excepciones desde nuestro código.

Lanzando excepciones

Como ya hemos comentado anteriormente, para lanzar excepciones utilizaremos la sentencia throw, a continuación debe aparecer un objeto instancia de la clase de excepción que deseamos lanzar.

Una técnica muy común es la de atrapar excepciones y luego relanzarlas añadiendo nuevos detalles con nuestros propios mensajes de error.

Para mostrar como se pueden lazar excepciones vamos a retomar el ejemplo del apartado anterior, en el que dado un nombre de fichero mostrábamos su contenido.

En este caso al atrapar la excepción la vamos a relanzar indicando una descripción propia de la misma. El código resultante se puede ver en el Código fuente 233.

```
<%@ Page language="C#"%>
<%@ Import NameSpace="System.IO"%>
<html>
<head><title>Tratamiento de excepciones</title></head>
<script runat="server">
void MostrarFichero(Object sender, EventArgs evento) {
    try{
        FileStream fs = new FileStream(nombreFichero.Text, FileMode.Open);
        StreamReader sr = new StreamReader(fs);
        String contenido;
        contenido = sr.ReadToEnd();
        resultado.Visible=true;
        resultado.Text=contenido;
        fs.Close();
    }
    catch(ArgumentException excepcion){
             throw new ArgumentException("Debe indicar el nombre del fichero");
    }
    catch(FileNotFoundException excepcion){
             throw new FileNotFoundException("No se ha encontrado el fichero");
    }
    catch(IOException excepcion){
             throw new IOException("Excepción de entrada/salida");
    }
    catch(Exception excepcion){
       throw new Exception("Se ha producido la excepción: "+excepcion.Message);
    }
}
</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<asp:TextBox Runat="server" ID="nombreFichero"/>
<asp:button text="Enviar" runat="server" ID="boton" OnClick="MostrarFichero"/><br>
<asp:TextBox TextMode="MultiLine" Runat="server" ID="resultado"
Columns="40" Rows="5" EnableViewState="False" Text="" Visible="False"/>
</form>
</html>
```

Código fuente 233

Ahora cuando se produzca una excepción se interrumpirá la ejecución de la página pero aparecerá la descripción ofrecida desde nuestro código. En la Figura 127 se puede ver el resultado de la ejecución de esta página ASP .NET cuando se lanza una excepción al no indicar ningún nombre de fichero.

Pero si queremos seguir manteniendo que la página se ejecute y no se muestre el volcado de la pila y el error al usuario, podemos atrapar la excepción en un nivel superior, es decir, utilizar un bloque try/catch que agrupe a todo el código anterior, el Código fuente 234 es el código resultante de esta modificación.

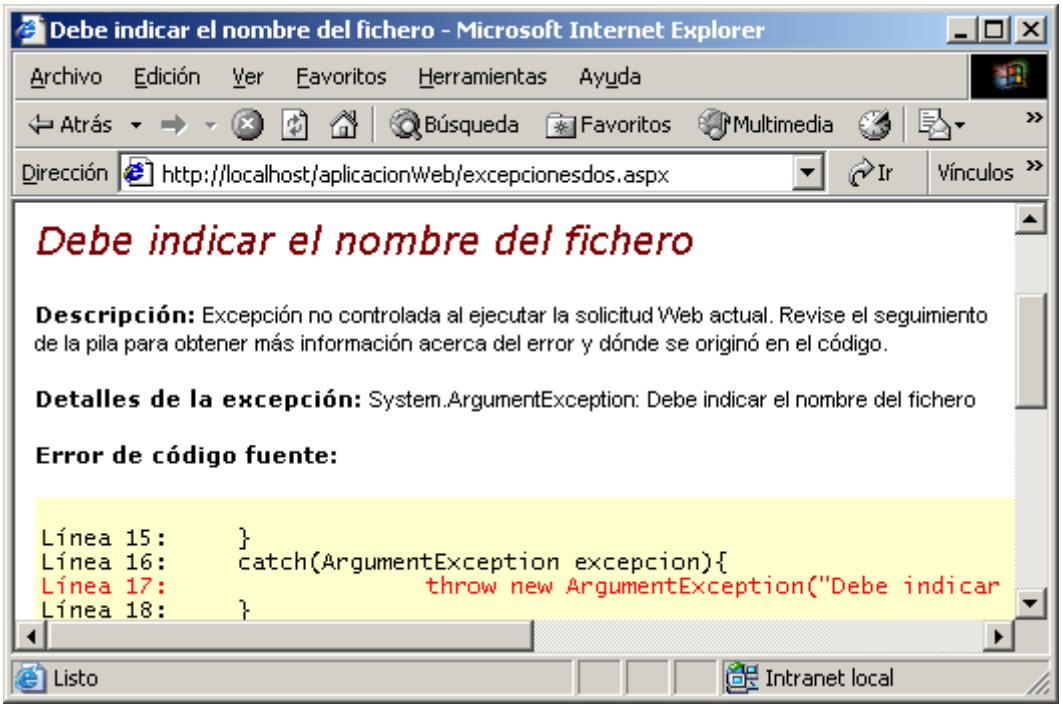


Figura 127

```
<%@ Page language="C#"%>
<%@ Import NameSpace="System.IO"%>
<html>
<head><title>Tratamiento de excepciones</title></head>
<script runat="server">
void MostrarFichero(Object sender, EventArgs evento) {
 try{
    try{
        FileStream fs = new FileStream(nombreFichero.Text, FileMode.Open);
         StreamReader sr = new StreamReader(fs);
         String contenido;
         contenido = sr.ReadToEnd();
         resultado.Visible=true;
         resultado.Text=contenido;
         fs.Close();
    }
    catch(ArgumentException excepcion){
              throw new ArgumentException("Debe indicar el nombre del fichero");
    }
    catch(FileNotFoundException excepcion){
              throw new FileNotFoundException("No se ha encontrado el fichero");
    }
    catch(IOException excepcion){
              throw new IOException("Excepción de entrada/salida");
    }
    catch(Exception excepcion){
              throw new Exception("Se ha producido la excepción: "
                                                    +excepcion.Message);
    }
 }catch(Exception excepcion){
       Response.Write("Se ha producido una excepción: "+excepcion.Message);
 }
}
</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<asp:TextBox Runat="server" ID="nombreFichero"/>
```

```
<asp:button text="Enviar" runat="server" ID="boton" OnClick="MostrarFichero"/><br>
<asp:TextBox TextMode="MultiLine" Runat="server" ID="resultado"
Columns="40" Rows="5" EnableViewState="False" Text="" Visible="False"/>
</form>
</html>
```

Código fuente 234

En la Figura 128 se puede ver la ejecución de esta nueva página ASP .NET cuando no indicamos un nombre de fichero.

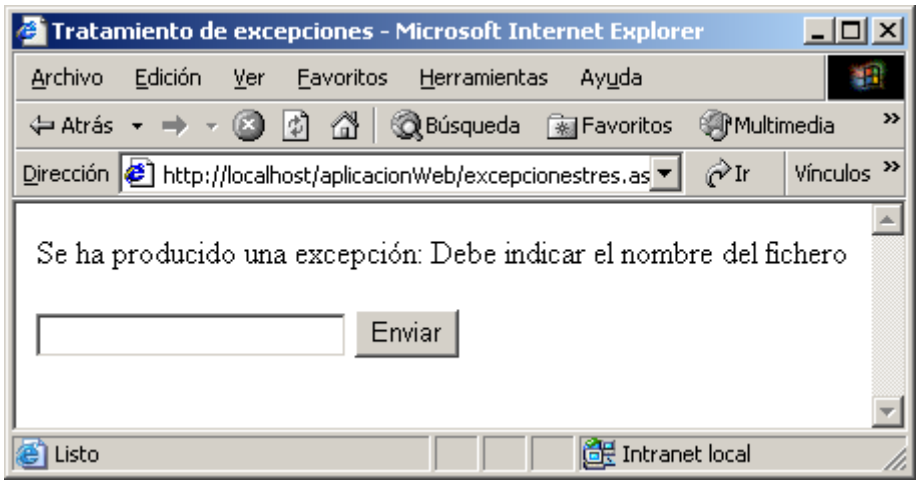


Figura 128

Excepciones personalizadas

Podemos llevar un poco más allá la personalización del mensaje de error mostrando, podemos crear nuestra propia clase de excepción.

Para crear nuestra propia excepción debemos heredar de la clase ApplicationException, en este caso vamos a crear una excepción llamada SinContenidoExcepcion, esta excepción está relacionada con los ejemplos anteriores y se lanzará cuando el contenido de la caja de texto que indica el nombre del fichero se encuentre vacía.

Para crear esta nueva clase de excepción, la vamos a crear como un componente .NET, es decir como una clase dentro de una librería de clases, para más información sobre componentes .NET se debe acudir al capítulo correspondiente.

Para la clase de la excepción hemos creado tres constructores, todos ellos deben llamar al constructor de la clase padre. También se ha incorporado un método, llamado Descripcion(), que devuelve una cadena con el mensaje de error y la traza de la pila.

El Código fuente 235 es el código de nuestra excepción.

```
using System;

namespace Componente.Ejemplos
{
```

```
      public class SinContenidoException:ApplicationException
      {
            public SinContenidoException()
            {
            }

            public SinContenidoException(String mensaje):base(mensaje)
            {
            }

            public SinContenidoException(String mensaje,
                                         Exception interna):base(mensaje,interna)
            {
            }

            public String Descripcion()
            {
                  return this.Message+"--->"+this.StackTrace;
            }

      }
}
```

Código fuente 235

Para utilizar esta excepción dentro de una página ASP .NET, únicamente debemos importar su espacio de nombres. En el Código fuente 236 se ofrece un ejemplo de utilización de esta nueva excepción. Seguimos con el ejemplo del fichero, pero en este caso no se atrapan excepciones, únicamente se lanza la excepción SinContenidoException cuando la caja de texto que indica el nombre del fichero está vacía.

```
<%@ Page language="C#"%>
<%@ Import NameSpace="System.IO"%>
<%@ Import NameSpace="Componente.Ejemplos"%>
<html>
<head><title>Tratamiento de excepciones</title></head>
<script runat="server">
void MostrarFichero(Object sender, EventArgs evento) {
      if(nombreFichero.Text.Equals(""))
            throw new SinContenidoException("Debe indicar un nombre de fichero");

      FileStream fs = new FileStream(nombreFichero.Text, FileMode.Open);
    StreamReader sr = new StreamReader(fs);
    String contenido;
    contenido = sr.ReadToEnd();
    resultado.Visible=true;
    resultado.Text=contenido;
    fs.Close();
}
</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<asp:TextBox Runat="server" ID="nombreFichero"/>
<asp:button text="Enviar" runat="server" ID="boton" OnClick="MostrarFichero"/><br>
<asp:TextBox TextMode="MultiLine" Runat="server" ID="resultado"
Columns="40" Rows="5" EnableViewState="False" Text="" Visible="False"/>
</form>
</html>
```

Código fuente 236

En la Figura 129 se puede ver el resultado de lanzar esta excepción.

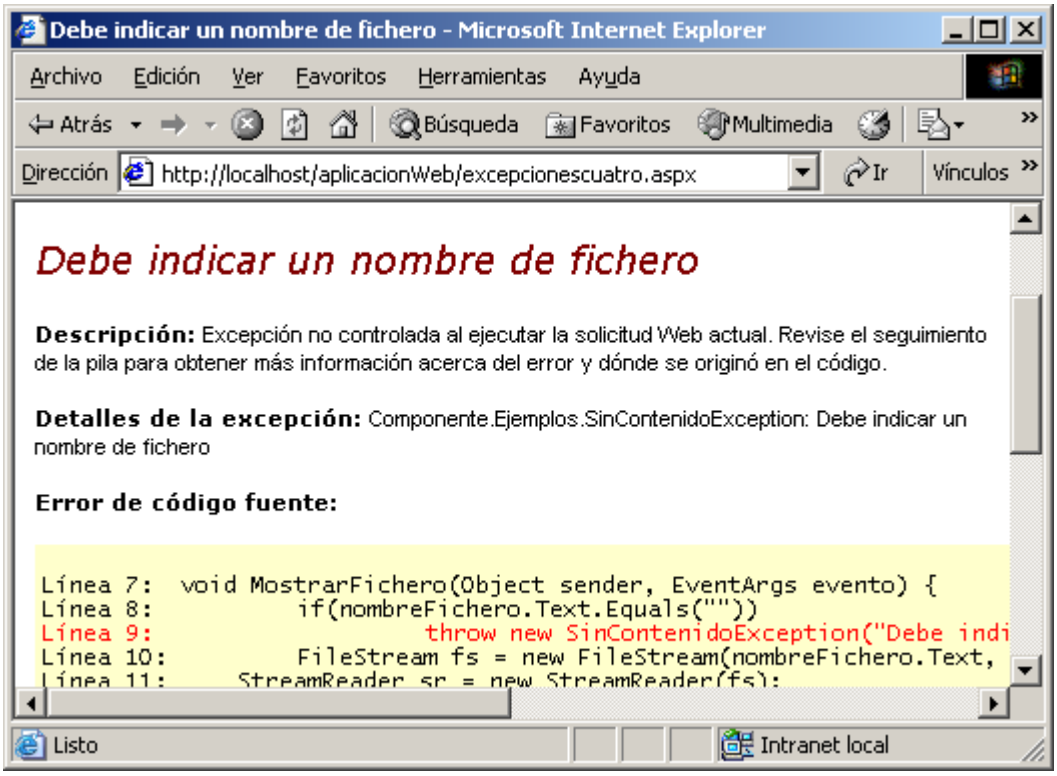


Figura 129

Si queremos tratar esta excepción en bloques try/catch, tal como hacíamos en otros ejemplos, no tenemos más que escribir el Código fuente 237 que se corresponde con el código de una página ASP .NET que trata las excepciones SinContenidoException, y además hace uso del método Descripcion().

```
<%@ Page language="C#"%>
<%@ Import NameSpace="System.IO"%>
<%@ Import NameSpace="Componente.Ejemplos"%>
<html>
<head><title>Tratamiento de excepciones</title></head>
<script runat="server">
void MostrarFichero(Object sender, EventArgs evento) {
      try{
        if(nombreFichero.Text.Equals(""))
             throw new SinContenidoException("Debe indicar un nombre de fichero");
         FileStream fs = new FileStream(nombreFichero.Text, FileMode.Open);
        StreamReader sr = new StreamReader(fs);
        String contenido;
        contenido = sr.ReadToEnd();
        resultado.Visible=true;
        resultado.Text=contenido;
        fs.Close();
    }
    catch(SinContenidoException excepcion){
             error.Text=excepcion.Descripcion();
             resultado.Text="";
             resultado.Visible=false;
    }
    catch(FileNotFoundException excepcion){
             error.Text="No se ha encontrado el fichero";
```

```
                resultado.Text="";
                resultado.Visible=false;
        }
        catch(IOException excepcion){
                error.Text="Excepción de entrada/salida";
                resultado.Text="";
                resultado.Visible=false;
        }
        catch(Exception excepcion){
                error.Text="Se ha producido la excepción: "+excepcion.Message;
                resultado.Text="";
                resultado.Visible=false;
        }
}
</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<asp:TextBox Runat="server" ID="nombreFichero"/>
<asp:button text="Enviar" runat="server" ID="boton" OnClick="MostrarFichero"/><br>
<asp:Label ID="error" Runat="server" EnableViewState="False" Text=""/><br>
<asp:TextBox TextMode="MultiLine" Runat="server" ID="resultado"
Columns="40" Rows="5" EnableViewState="False" Text="" Visible="False"/>
</form>
</html>
```

Código fuente 237

En la Figura 130 se puede ver un ejemplo de ejecución de la página ASP .NET, en este caso se ha atrapado la excepción SinContenidoException.

Figura 130

Hasta aquí hemos visto el tratamiento de excepciones que nos ofrece C# como lenguaje dentro del CLR (Common Language Runtime), pero también tenemos otras posibilidades a la hora de manejar excepciones, aunque se debe señalar que estas posibilidades no son excluyentes respecto de las vistas hasta ahora, sino que las podemos utilizar si lo deseamos como complemento.

El resto de posibilidades para el tratamiento y manejo de excepciones y errores nos lo ofrece el entorno de ASP .NET y los siguientes apartados se encuentran dedicados a ello.

Tratamiento de errores en ASP .NET

Como ya hemos adelantado, en los siguientes apartados vamos a tratar el tratamiento de errores que nos ofrece el entorno de las páginas ASP .NET.

Dentro de una página ASP .NET se pueden producir los siguientes tipos de errores:

- Errores de configuración: estos errores ocurren cuando la sintaxis o la estructura de un fichero Web.config, dentro de la jerarquía de configuración de la aplicación ASP .NET es incorrecto. El fichero Web.config es un fichero especial que nos permite configurar la aplicación ASP .NET mediante etiquetas XML, y además se puede definir una completa jerarquía de configuración utilizando distintos ficheros Web.config. En el capítulo correspondiente veremos el formato de este fichero de configuración , que permite que la configuración de páginas ASP .NET se encuentre basada en un sencillo mecanismo de ficheros XML.

- Errores del analizador: estos errores se producen cuando la sintaxis de la página ASP .NET está mal construida.

- Errores del compilador: este error acontece cuando las sentencias del lenguaje de la página son incorrectas.

- Errores en tiempo de ejecución: este último tipo de error se produce durante el proceso de ejecución de la página, estos errores no se pueden detectar en tiempo de compilación.

Por defecto, cuando se produce un error en una página ASP :NET se muestra la pila de llamadas, que es una serie encadenada de llamadas a métodos que nos llevan hasta el método que causó la excepción. Si el modo de depuración de ASP .NET está activado se nos mostrará el número de línea en el que se ha producido el error y su código fuente.

Para activar o desactivar el modo de depuración en una página ASP .NET, utilizaremos el atributo Debug de la directiva @Page, por defecto esta propiedad tiene el valor True. También se puede configurar el modo de depuración para una aplicación ASP .NET completa, esto se podrá realizar dentro del fichero de configuración Web.config, este fichero lo trataremos en siguientes capítulos, aunque más adelante realizaremos alguna referencia al mismo.

Dentro de ASP .NET tenemos varias formas de tratar los errores:

- A nivel de página mediante el evento de error, este tratamiento de errores se debe hacer por cada página ASP .NET.

- A nivel de aplicación mediante el evento de error, en este caso el tratamiento de errores es para la aplicación ASP .NET completa, y se debe programar en el fichero GLOBAL.ASAX de la aplicación. Este fichero tiene la misma funcionalidad que el fichero GLOBAL.ASA de las versiones anteriores de ASP, es decir, la de definir eventos y objetos globales a la aplicación, este fichero lo veremos con detenimiento cuando tratemos las aplicaciones ASP .NET y su configuración.

- Indicando una página de error para el tratamiento de los mismos, a través del atributo ErrorPage de la directiva @Page.

- En el fichero de configuración de la aplicación, el fichero Web.config vamos a poder declarar el tratamiento de errores de la aplicación.

A continuación veremos cada una de estas posibilidades de tratamiento de errores dentro de ASP .NET.

El método Page_Error

Esta es una de las posibilidades que nos ofrece ASP .NET para tratar los errores a nivel de página. En este caso se van a tratar los errores a través de código, cuando se produzca una excepción que no es tratada en la página se ejecutará el método Page_Error.

La información sobre el error que se ha producido la podemos obtener a través del método GetLastError() de la propiedad Server de la página, debemos recordar que algunas propiedades de la clase Page se corresponden con los objetos integrados de versiones anteriores de ASP. Este método devolverá un objeto de excepción.

Vamos a seguir utilizando el código de ejemplo de la página ASP .NET que muestra el contenido de un fichero, en este caso no vamos a utilizar los bloques try/catch, sino que vamos a hacer uso del evento Page_Error.

Dentro del método Page_Error será necesario llamar al método ClearError() del objeto Server, instancia de la clase HttpServerUtility, de esta forma el error se eliminará y no se mostrará la página de error típica de las aplicaciones ASP .NET.

En el método Page_Error nos vamos a limitar a mostrar la descripción de la excepción que se ha producido. En el Código fuente 238 se puede ver el código de la página ASP .NET correspondiente.

```
<%@ Page language="C#"%>
<%@ Import NameSpace="System.IO"%>
<html>
<head><title>Tratamiento de errores</title></head>
<script runat="server">
void Page_Error(Object obj, EventArgs evento){
      Response.Write("Se ha producido un error<br>");
      Response.Write(Server.GetLastError().ToString());
      Server.ClearError();
}

void MostrarFichero(Object sender, EventArgs evento) {
      FileStream fs = new FileStream(nombreFichero.Text, FileMode.Open);
    StreamReader sr = new StreamReader(fs);
    String contenido;
    contenido = sr.ReadToEnd();
    resultado.Text=contenido;
    fs.Close();
}
</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<asp:TextBox Runat="server" ID="nombreFichero"/>
<asp:button text="Enviar" runat="server" ID="boton" OnClick="MostrarFichero"/><br>
<asp:TextBox TextMode="MultiLine" Runat="server" ID="resultado"
Columns="40" Rows="5"/>
</form>
</html>
```

Código fuente 238

Si indicamos un nombre de fichero que no existe, la página ASP .NET mostrará un aspecto similar al de la Figura 131, como se puede comprobar se muestra la pila de llamadas hasta que se ha producido el error.

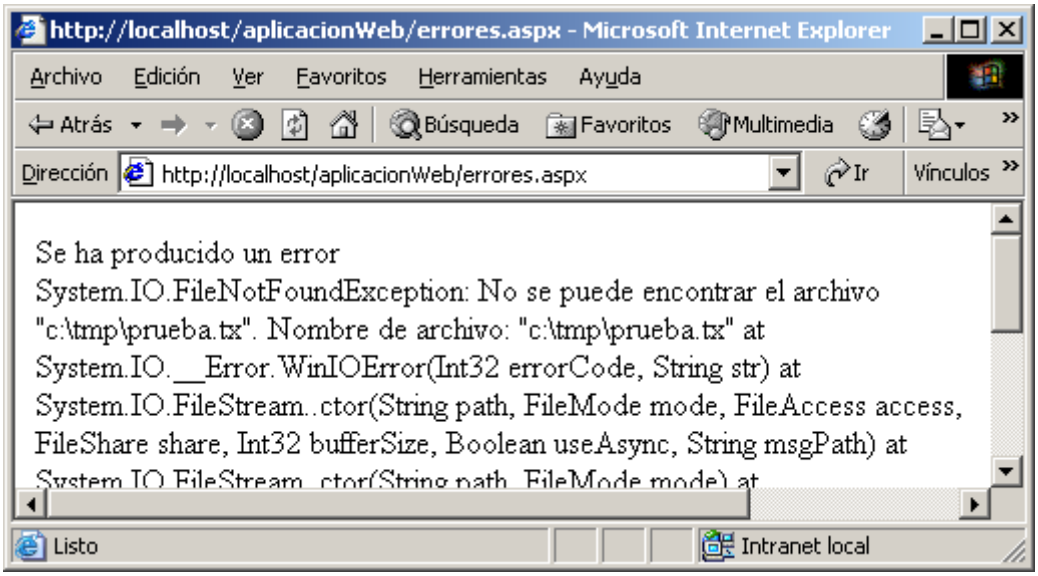


Figura 131

El método Application_Error

Este método, similar en funcionalidad al anterior, se va a ejecutar cuando se produzca un error que se corresponde con una excepción no tratada en una página ASP .NET, y además esta página no utiliza el método Page_Error para tratar el error.

El método Application_Error se debe incluir en el fichero GLOBAL.ASAX de la aplicación ASP .NET actual, este fichero especial que vamos a utilizar para definir métodos que tratan eventos a nivel de aplicación, así como para definir objetos a nivel de aplicación lo veremos con más detalle más adelante en este mismo texto, de momento nuestro fichero GLOBAL.ASAX va a contener únicamente el código encargado de tratar un error en el método Application_Error (Código fuente 239).

```
<script language="C#" runat="server">

void Application_Error(Object obj, EventArgs evento){
      Response.Write("Se ha producido un error en la página "
                                    +Request.Url.ToString()+"<br>");
      Response.Write(Server.GetLastError().ToString());
      Server.ClearError();
}

</script>
```

Código fuente 239

Para comprobar que se ejecuta el método Application_Error, únicamente tenemos que generar un error dentro de una página ASP .NET y no tratarlo dentro de la misma, si seguirmos con el ya famoso

ejemplo del Web Form que muestra el contenido del fichero que le indicamos, únicamente debemos eliminar el código de tratamiento de errores, y obtendremos el Código fuente 240.

```
<%@ Page language="C#"%>
<%@ Import NameSpace="System.IO"%>
<html>
<head><title>Tratamiento de errores</title></head>
<script runat="server">
void MostrarFichero(Object sender, EventArgs evento) {
       FileStream fs = new FileStream(nombreFichero.Text, FileMode.Open);
    StreamReader sr = new StreamReader(fs);
    String contenido;
    contenido = sr.ReadToEnd();
    resultado.Text=contenido;
    fs.Close();
}
</script>
<body>
<form method="POST" runat="server" ID="Formulario">
<asp:TextBox Runat="server" ID="nombreFichero"/>
<asp:button text="Enviar" runat="server" ID="boton" OnClick="MostrarFichero"/><br>
<asp:TextBox TextMode="MultiLine" Runat="server" ID="resultado"
Columns="40" Rows="5"/>
</form>
</html>
</html>
```

Código fuente 240

Ahora si ejecutamos la página ASP .NET anterior e indicamos un nombre de fichero que no existe, se ejecutará el método Application_Error,y aparecerá un mensaje como el de la Figura 132.

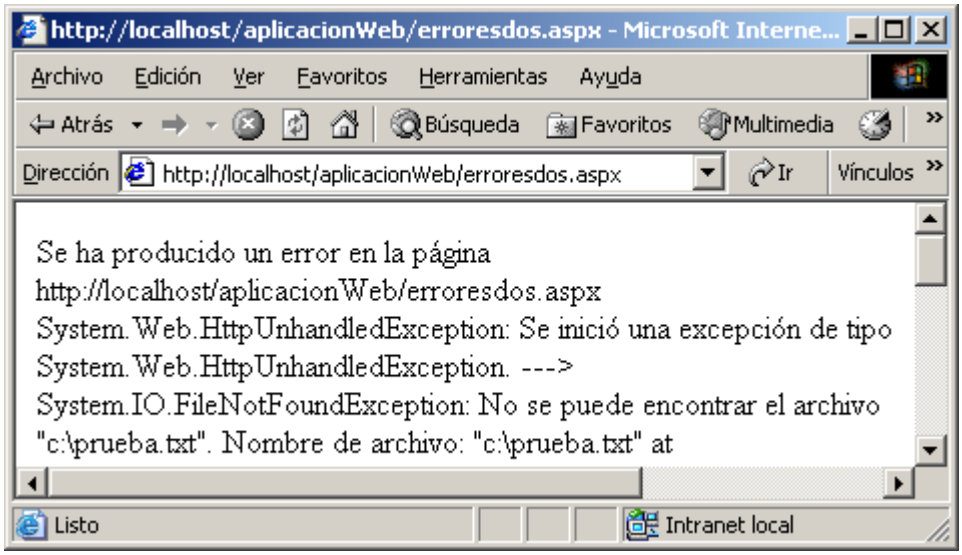


Figura 132

El atributo ErrorPage de la directiva @Page

Cuando comentamos la directiva @Page, en el capítulo correspondiente, ya comentamos este atributo, que nos va a permitir indicar una URL a una página ASP .NET que se mostrará cuando se produzca un error en la página actual.

Este atributo se suele utilizar cuando no deseamos dar una descripción detallada del error al usuario, sino que únicamente deseamos mostrarle una página de error sencilla.

Para poder utilizar este atributo de la directiva @Page con éxito debemos activar los errores personalizados, para ello debemos acudir al fichero de configuración de la aplicación ASP .NET, es decir, al fichero Web.config. Este fichero ya va siendo familiar para el lector, pero será en un próximo capítulo cuando veamos en profundidad dicho fichero de configuración.

Para indicar que los errores personalizados se encuentran activados, únicamente debemos incluir en el fichero Web.config, entre las etiquetas <system.web></system.web> el fragmento de código que aparece en el Código fuente 241.

```
<customErrors mode="On"/>
```

Código fuente 241

En el siguiente apartado veremos con más detenimiento la sección <customErrors> del fichero Web.config, ya que el siguiente método para tratar errores se basa en este fichero de configuración, en el que se indicarán páginas de error personalizadas.

Para probar esta otra forma de tratamiento de errores vamos a crear una página ASP .NET que consiste en un Web Form que nos va a permitir sumar dos cantidades y nos va a mostrar su resultado. El Código fuente 242 es el código de esta página.

```
<%@Page Language="C#" ErrorPage="PaginaError.aspx"%>
<html>
<head><title>Tratamiento de errores</title></head>
<script runat="server">
void Suma (Object obj,EventArgs e){
       int res;
       res=int.Parse(valor1.Text)+int.Parse(valor2.Text);
       resultado.Text=res.ToString();
}
</script>
<body>
<form id="formulario" runat="server">
<asp:TextBox id="valor1" runat="server" width="40"></asp:TextBox>+
<asp:TextBox id="valor2" runat="server" width="40"></asp:TextBox>=
<asp:TextBox id="resultado" runat="server" width="40"></asp:TextBox>
<asp:Button id="boton" runat="server" text="Suma" onclick="Suma"></asp:Button>
</form>
</body></html>
```

Código fuente 242

Como se puede ver esta página indica que su página de error es la página PaginaError.aspx. El Código fuente 243 es el código de la página de error, únicamente se limita a indicar el nombre del fichero en el que se ha producido el error.

```
<%@ Page Language="C#"%>
<html>
<head>
<title>Página de error</title>
</head>
<body>
Se ha producido un error en la página:
<i><b><%=Request.QueryString["aspxerrorpath"]%></b></i>
</body>
</html>
```

Código fuente 243

Si indicamos unos datos incorrectos para la suma, por ejemplo que uno de los sumandos sea una letra, se cargará en el navegador la página de error, que mostrará un aspecto como el de la Figura 133.

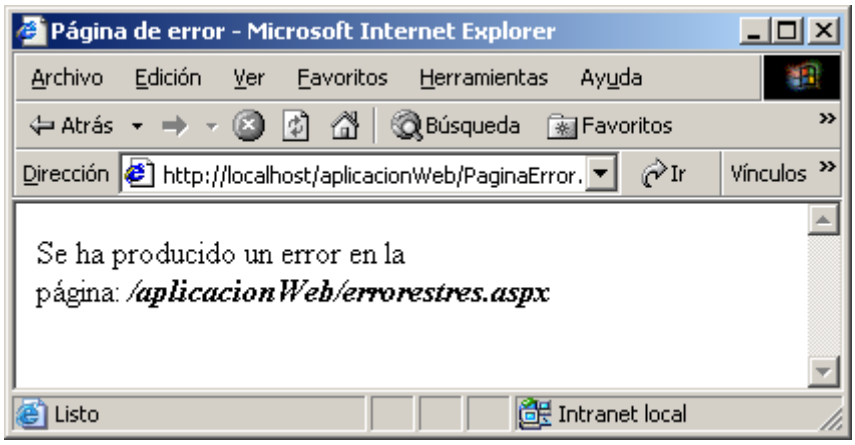


Figura 133

Definiendo páginas de error en el fichero Web.config

Los métodos anteriores para el tratamiento de errores en páginas ASP .NET se realizaban a través de código, pero también existe la posibilidad de utilizar un método declarativo de errores que consiste en definir las páginas de error que se van a utilizar, para ello se debe hacer uso del fichero de configuración de la aplicación, es decir, del fichero Web.config.

Por defecto, cuando se produce un error no tratado, se muestra una página de error similar a la de la Figura 134.

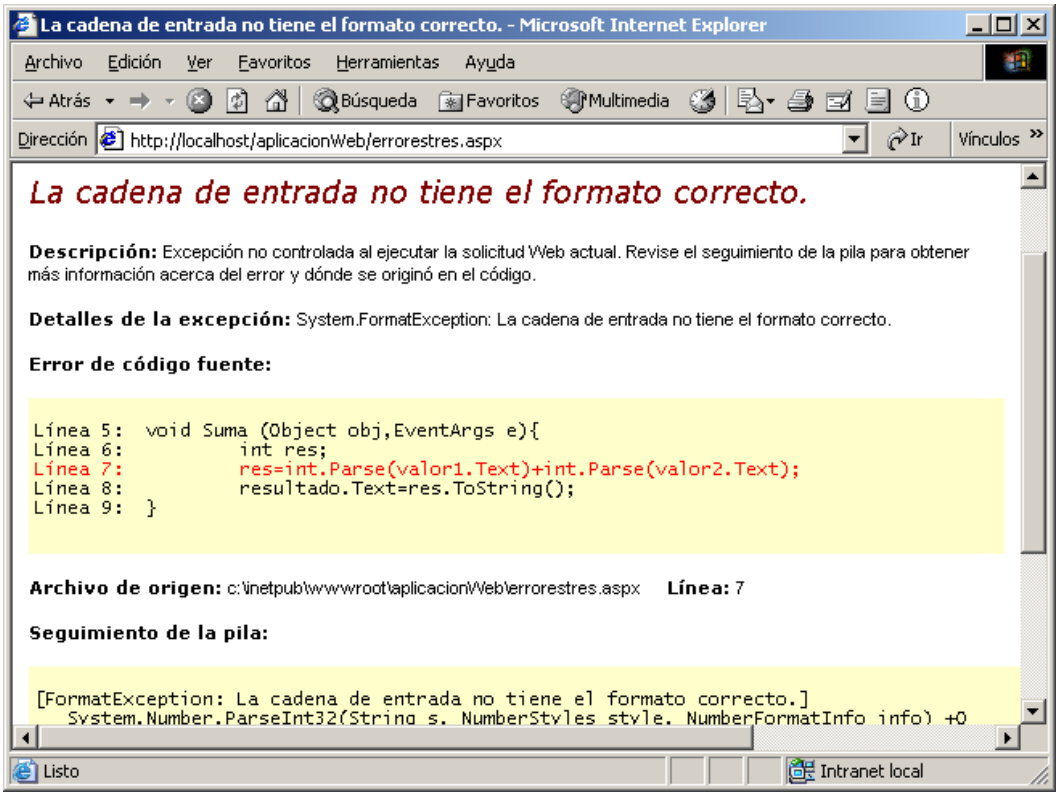


Figura 134

Este método es utilizado para tratar errores del protocolo HTTP y también para mostrar páginas de error personalizadas. Estas páginas de error personalizadas van a permitir que los usuarios vean errores sencillos y no engorrosos volcados de pila o fragmentos de código (aunque por otro lado estos datos son muy útiles para los desarrolladores), esto se debe tener en cuenta cuando vamos a pasar a nuestra aplicación ASP .NET a un entorno de producción.

Para configurar las páginas de error desde el fichero de Web.config de la aplicación ASP .NET debemos utilizar la sección <customErrors>, que se encuentra dentro de la sección más general <system.web>. La sección <customErrors> del fichero de configuración Web.config nos va a permitir especificar como se comporta ASP .NET cuando se produce un error no tratado por una página ASP .NET o por la aplicación. Tenemos varias opciones para indicar como se deben mostrar las páginas de error.

Dentro de la sección <customErrors> disponemos del atributo mode, que determinará si se mostrará una página de error personalizada o por el contrario la típica página de error de ASP .NET. Para indicar las distintas posibilidades este atributo posee los siguientes valores:

- RemoteOnly: cuando asignamos este valor al atributo mode, la página de error típica de ASP .NET con la descripción completa de los errores, se mostrará únicamente a los usuarios que accedan al servidor desde la propia máquina, es decir, aquellos que utilicen la dirección IP 127.0.0.1 o el nombre de dominio localhost. A las peticiones que no sean locales se les mostrará la página de error personalizada que haya sido indicada en el atributo defaultRedirect, este atributo también pertenece a la sección <customErrors>.

- On: en este caso la página de error de ASP .NET no se mostrará nunca, sino que se mostrará la página de error personalizada que corresponda. Si no se indica ninguna página de error, aparecerá una página de error indicando como debemos configurar nuestro fichero Web.config

para que se muestren las páginas de error personalizadas, en la Figura 135 se puede ver un ejemplo de esta página de error.

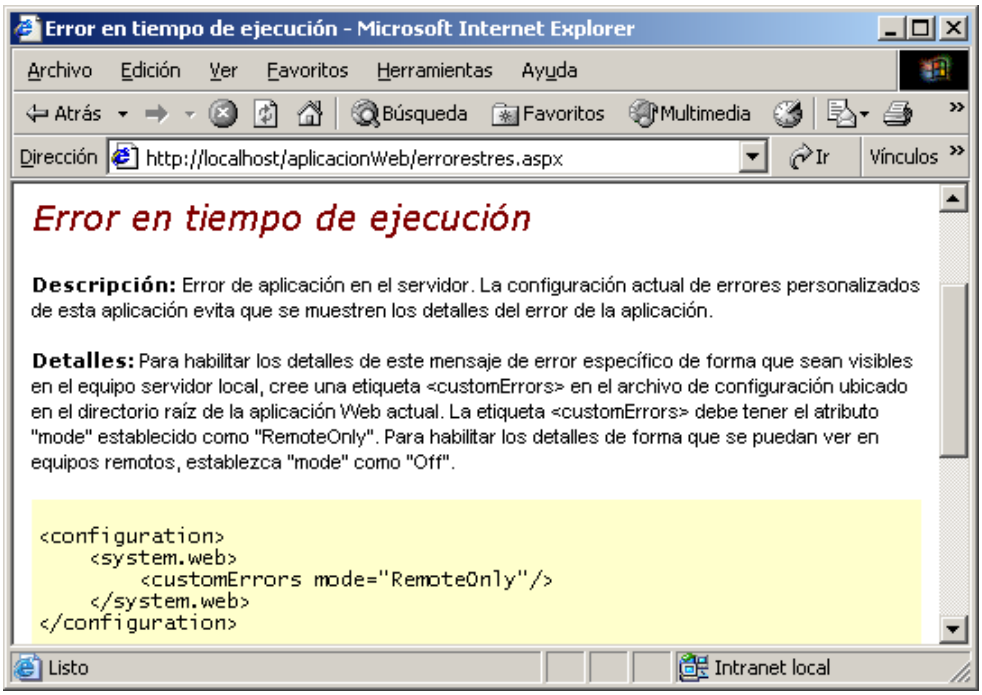


Figura 135

- Off: ASP .NET mostrará siempre la página de error ya conocida por todos, que contendrá las líneas de código que han producido el error, el volcado de pila, el número de línea del fichero de origen del error, etc.

En el caso de estar trabajando en un entorno de pruebas o de desarrollo, para que el equipo de desarrollo posea toda la información posible acerca de los errores que se producen en la aplicación ASP .NET, se establecerá el valor de la propiedad mode a Off. Así el podrá aparecer en el fichero Web.config.

En un entorno de producción será más conveniente utilizar el valor RemoteOnly o el valor On para el atributo mode, de esta forma los errores detallados nunca llegarán a los usuarios finales. En este caso también deberemos indicar las páginas de error personalizadas que deseamos mostrar.

Se debe indicar que la página de error indicada mediante el mecanismo del apartado anterior, es decir, la utilización del atributo ErrorPage de la directiva @Page, sobrescribe los valores indicados de página de error personalizadas dentro del fichero Web.config, ya que estas páginas de error son para la aplicación ASP .NET, y el atributo ErrorPage nos va a permitir indicar una página de error para una página ASP .NET concreta.

Como ya hemos comentado, en los entornos de producción siempre será conveniente indicar páginas de error personalizadas para nuestras aplicaciones ASP .NET, dentro del fichero Web.config tenemos dos formas de indicar páginas de error:

- Redirecciones por defecto: de esta forma cuando se produzca un error, sea del tipo que sea, el cliente será redirigido a la página de error indicada.

- Redirecciones personalizadas: el cliente será redirigido a una página de error personalizada determinada, cuando se produce un error del protocolo HTTP de un tipo específico, por ejemplo el error 404 cuando no se encuentra una página determinada.

Para las redirecciones por defecto utilizaremos el atributo defaultRedirect de la etiqueta customErrors, este atributo contendrá la URL a la página de error personalizada por defecto. La utilización de este atributo se puede ver en el Código fuente 244.

```
<configuration>
  <system.web>
     <customErrors mode="On" defaultRedirect="/aplicacionWeb/PaginaError.aspx"/>
  </system.web>
</configuration>
```

Código fuente 244

En este caso los cada vez que se produzca un error, independientemente del tipo que sea, los clientes remotos verán siempre la página de error llamada PaginaError.aspx.

Pero en la personalización de errores podemos ir más allá e indicar una página de error determinada que se mostrará cuando se produzca un tipo de error específico, este mecanismo se puede utilizar en combinación con el anterior, es decir, si el tipo de error no coincide con ninguno de los indicados se mostrará la página de error indicada en el atributo defaultRedirect de la etiqueta customErrors.

Para indicar las páginas de error atendiendo al tipo de error que se produce, se debe utilizar el subelemento <error> dentro de la sección customErrors. El elemento error va a poseer dos atributos que son:

- statusCode: en este atributo indicaremos el código de estado del protocolo HTTP que deseemos que coincida con el error que se ha dado. Si se encuentra una coincidencia con este código de estado al cliente se mostrará la página de error personalizada que corresponda.

- redirect: este atributo indica la página ala que se enviará al cliente en el caso de que se produzca el error coincidente con el código de estado del protocolo http indicado en el atributo statusCode.

En el Código fuente 245 se muestra un ejemplo de utilización de la etiqueta <error>, en este caso los clientes remotos verán la página PaginaNoEncontrada.html en el caso de que se produzca el código de estado 404 del protocolo HTTP, este código indica que una página determinada no ha podido ser encontrada. Si no el código de estado no es el 404 al cliente se le mostrará la página de error personalizada por defecto, es decir la indicada en el atributo defaultRedirect.

```
<configuration>
  <system.web>
     <customErrors mode="On" defaultRedirect="/aplicacionWeb/PaginaError.aspx">
       <error statusCode="404" redirect="/aplicacionWeb/PaginaNoEncontrada.html"/>
     </customErrors>
  </system.web>
</configuration>
```

Código fuente 245

Con este apartado finalizamos las distintas formas que tenemos de tratar los errores dentro de ASP .NET en el siguiente capítulo veremos los mecanismos de trazas y de depuración que nos ofrece ASP :NET

16

Trazas y depuración en ASP .NET

Introducción

En el presente capítulo vamos a ver otros dos temas muy relacionados con los errores, como son el mecanismo de trazas y el de depuración que nos ofrece ASP .NET. Este capítulo se divide por lo tanto en dos bloques temáticos. Vamos a pasar al primero de ellos.

El mecanismo de trazas

Este mecanismo es una de las nuevas características que nos ofrece ASP .NET, en versiones anteriores el mecanismo más común de trazas era la utilización de la sentencia Response.Write, pero ASP .NET nos ofrece un mecanismo de trazas automático.

Las trazas se pueden habilitar o deshabilitar a nivel una página ASP .NET determinada o bien al nivel de una aplicación ASP .NET completa, en este último caso deberemos acudir de nuevo al ya cada vez más conocido fichero de configuración de la aplicación, es decir, al fichero Web.config.

A continuación vamos a ver las dos posibilidades que nos ofrece ASP .NET a la hora de utilizar trazas.

Trazas a nivel de página

Para hacer uso del mecanismo de trazas dentro de una página ASP .NET determinada no tenemos nada más que asignar el valor True al atributo Trace de la directiva @Page.

En el Código fuente 246 se ofrece una página ASP .NET muy sencilla, que únicamente pretende mostrar la utilización de las trazas dentro de una página.

```
<%@ Page Trace="True" %>
<html>
<head>
<title>Trazas</title>
</head>
<body>
<b>Se activan las trazas.</b>
</body>
</html>
```

Código fuente 246

La ejecución de esta página se puede ver en la Figura 136.

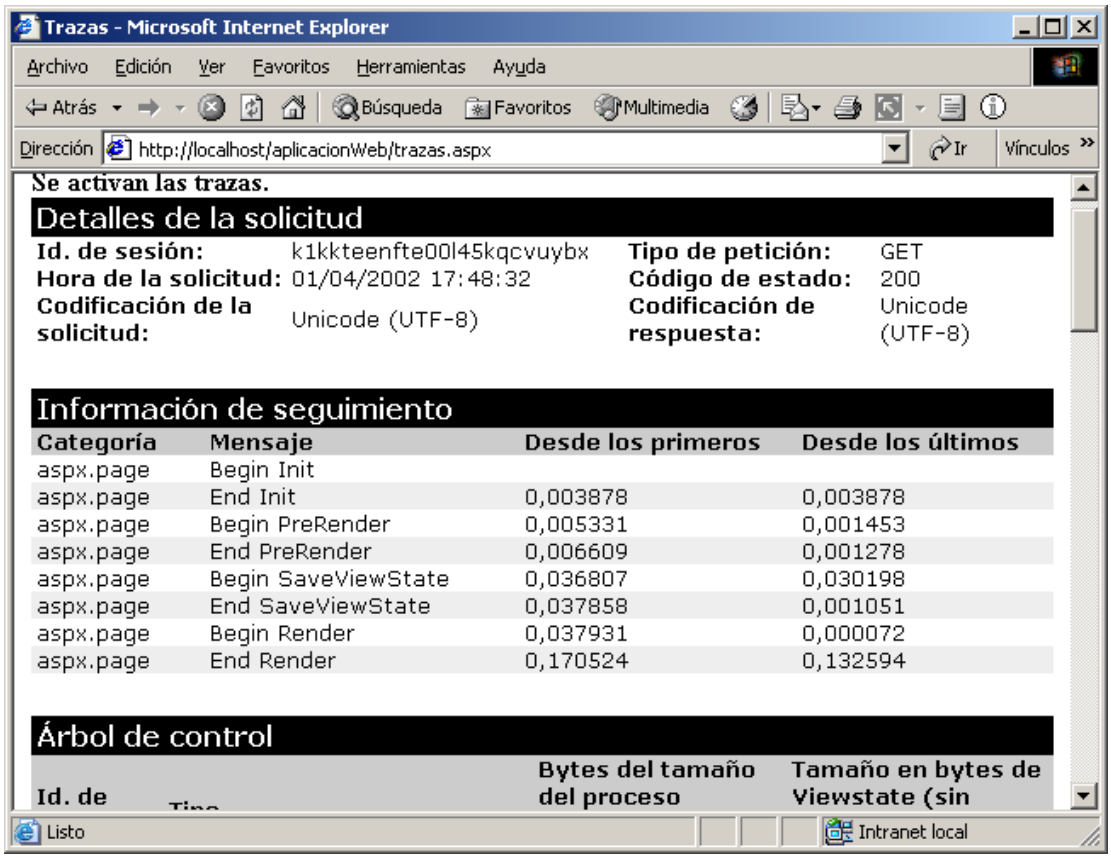


Figura 136

Como se puede comprobar, el entorno del .NET Framework nos ofrece una gran cantidad de información de forma automática con sólo activar el mecanismo de trazas. Esta información se encuentra dividida en varias secciones, que vamos a comentar a continuación:

- Detalles de la solicitud (Request Details): esta sección ofrece información sobre la petición realizada a la página ASP .NET, alguna de esta información es el tipo de petición, el código de estado, la hora de la petición, el identificador de sesión, etc.

- Información de seguimiento (Trace Information): en esta sección se puede observar el orden de ejecución de la página ASP .NET, se indica el tiempo empleado en la ejecución de cada una de las fases de la página ASP .NET. Si añadimos mensajes personalizados en nuestras trazas (algo que veremos en breve), parecerá también en esta sección.

- Árbol de control (Control Tree): aquí aparece una lista de la jerarquía de todos los controles incluidos en la página ASP .NET actual. Para cada control se indica el tamaño que ocupa en bytes.

- Estado de sesión (Session State): indica los detalles de cada elemento presente en el estado de la sesión actual. El objeto Session se verá más adelante en este mismo texto. Si no existe ningún elemento almacenado en la sesión esta sección no aparecerá.

- Estado de aplicación (Application State): esta sección tiene un cometido similar al anterior, pero este caso para los elementos almacenados en la aplicación.

- Colección de cookies (Cookies Collection): esta sección muestra los detalles de cada cookie existente en el momento actual.

- Colección de encabezados (Headers Collection): esta sección contiene los encabezados del protocolo HTTP utilizados para realizar la petición de la página.

- Colección de formularios (Forms Collection): contiene el nombre y valor de cada uno de los campos de los un formulario Web, sino existe ningún Web Form en la página esta sección no aparecerá.

- Colección del QueryString (QueryString Collection): contiene el nombre y valor de cada uno de los contenidos de la cadena de consulta (QueryString) que se ha utilizado para solicitar la página ASP .NET, si no existe el QueryString esta sección no aparecerá como información de traza de la página.

- Variables del servidor (Server Variables): esta última sección ofrece una completa lista de las variables de servidor y su contenido correspondiente.

Para desactivar las trazas de una página, únicamente debemos asignar el valor False al atributo Trace de la directiva @Page, o simplemente no indicar este atributo de la directiva @Page, ya que False es su valor por defecto.

En el siguiente apartado vamos a comentar como se puede escribir información en las trazas de una página ASP .NET.

Escribiendo en las trazas

Además de la información automática que nos ofrece ASP .NET mediante la activación del mecanismo de trazas, también podremos añadir nuestra propia información de trazas a la sección de Información de seguimiento.

Esto se puede hacer gracias a la propiedad Trace del objeto Page, esta propiedad es un objeto de la clase System.Web.TraceContext. Esta clase nos ofrece el método Write() para escribir información en las trazas de la página.

En el Código fuente 247 se puede ver una página ASP .NET que escribe información en las trazas indicando el evento de la página que se está produciendo en cada caso.

```
<%@ Page Language="C#" Trace="True" %>
<html>

<script runat="server">
void Page_Init(Object sender, EventArgs e) {
      Page.Trace.Write("Evento Init");
}

void Page_Load(Object sender, EventArgs e) {
      Page.Trace.Write("Evento Load");
}

void Page_PreRender(Object sender, EventArgs e) {
      Page.Trace.Write("Evento PreRender");
}
</script>

<head>
<title>Trazas</title>
</head>
<body>
<asp:Label runat="server" id="texto">Eventos de la página con trazas</asp:Label>
</body>
</html>
```

Código fuente 247

Estos mensajes se pueden apreciar en la sección Información se seguimiento tal como se puede comprobar en la Figura 137, que es un fragmento del resultado de la ejecución del código anterior.

Información de seguimiento

Categoría	Mensaje	Desde los primeros	Desde los últimos
aspx.page	Begin Init		
	Evento Init	0,000063	0,000063
aspx.page	End Init	0,000105	0,000042
	Evento Load	0,000139	0,000034
aspx.page	Begin PreRender	0,000173	0,000034
	Evento PreRender	0,000206	0,000033
aspx.page	End PreRender	0,000239	0,000033
aspx.page	Begin SaveViewState	0,000458	0,000219
aspx.page	End SaveViewState	0,000492	0,000034
aspx.page	Begin Render	0,000523	0,000031
aspx.page	End Render	0,000792	0,000269

Figura 137

La propiedad Trace del objeto Page nos va a ofrecer acceso al contexto de las trazas de la página ASP .NET actual, para ello la clase TraceContext, de la que es instancia la propiedad Trace, nos ofrece dos propiedades y dos métodos que son los siguientes:

- IsEnabled: esta propiedad de tipo booleano indica si el mecanismo de trazas se encuentra activado en la página ASP .NET. Es una propiedad de lectura/escritura y se corresponde con el valor del atributo Trace de la directiva @Page.

- TraceMode: propiedad que indica la forma de ordenación de los mensajes que aparecen dentro de la sección Información de seguimiento, esta propiedad puede tener los valores

SortByCategory (ordenación por categoría) y SortByTime (ordenado por el momento en el que se ha producido). Esta propiedad se corresponde también con un atributo de la directiva @Page, en este caso con el atributo TraceMode. Su valor por defecto es SortByTime.

- Write: este método, ya utilizado en el ejemplo anterior, permite escribir información en la sección Información de seguimiento.

- Warn: este método tiene la misma funcionalidad que el anterior, pero en este caso los mensajes aparecen en color rojo.

Los métodos Write() y Warn() se encuentran sobrecargados, presentando los parámetros del Código fuente 248 en cada una de sus versiones.

```
Trace.Write(Mensaje)
Trace.Write(Categoría, Mensaje)
Trace.Write(Categoría, Mensaje, InformaciónError)

Trace.Warn(Mensaje)
Trace.Warn(Categoría, Mensaje)
Trace.Warn(Categoría, Mensaje, InformaciónError)
```

Código fuente 248

A continuación pasamos a comentar los distintos parámetros de estos métodos.

- Categoría: cadena que se corresponde con la categoría que se le va a signar al mensaje. Esta categoría nos sirve para describir y agrupar mensajes que poseen un significado similar.

- Mensaje: cadena que representa el mensaje que se va a mostrar en la sección Información de seguimiento.

- InformaciónError: objeto de la clase System.Exception, permite mostrar información relativa a una excepción que se haya producido, esta información aparecerá a continuación del mensaje de la traza.

En el Código fuente 249 se ofrece una página ASP .NET que utiliza a modo de resumen práctico los métodos y propiedades de la clase TraceContext. Además esta página muestra como se puede activar y desactivar de forma dinámica el mecanismo de trazas.

```
<%@ Page Language="C#" %>
<html>

<script runat="server">

void Page_Load(Object sender, EventArgs e) {
      if (activar.Checked) {
            Trace.IsEnabled=true;
      }else{
            Trace.IsEnabled=false;
      }
      if (orden1.Checked) {
            Page.Trace.TraceMode=TraceMode.SortByCategory;
      }else if (orden2.Checked) {
            Page.Trace.TraceMode=TraceMode.SortByTime;
```

```
        }
        Page.Trace.Write("Eventos de la página","Evento Load");
}

void Page_PreRender(Object sender, EventArgs e) {
        Page.Trace.Write("Eventos de la página","Evento PreRender");
}

void PulsadoOrden(Object Sender, EventArgs e) {
        Trace.Warn("Eventos de los controles","Se ha pulsado un RadioButton");
}

void PulsadoActivar(Object Sender, EventArgs e) {
        Trace.Warn("Eventos de los controles","Se ha pulsado el CheckBox");
}

</script>

<head>
<title>Trazas</title>
</head>

<body>
<asp:Label runat="server" id="texto">Eventos de la página con
trazas</asp:Label><br>

<form runat="server" ID="Formulario">
<asp:RadioButton id="orden1" Text="Ordenar por categoría" runat="server"
GroupName="orden" OnCheckedChanged="PulsadoOrden" AutoPostBack="True"/><br>
<asp:RadioButton id="orden2" Text="Ordenar por tiempo" runat="server"
GroupName="orden"  OnCheckedChanged="PulsadoOrden" AutoPostBack="True"/><br>
<asp:CheckBox id="activar" Text="Activar trazas" runat="server"
OnCheckedChanged="PulsadoActivar" AutoPostBack="True"/>
</form>

</body>
</html>
```

Código fuente 249

Aunque el mecanismo de trazas es encuentre desactivado no es necesario eliminar el código que muestra información de trazas, es decir, la llamada a los métodos Write() y Warn() del objeto Trace no generará ningún error aunque las trazas se encuentren desactivadas, únicamente estos mensajes no se tendrán en cuenta.

La página ASP .NET del ejemplo muestra un formulario como el de la Figura 138, que permite indicar el orden en el que se va a mostrar la información presente en la sección Información de seguimiento, y además de estos objetos RadioButton también se ofrece un objeto CheckBox que permite activar y desactivar el mecanismo de trazas de la página.

Para mostrar información relativa a los eventos de la página se utiliza el método Write(), y para los eventos de los controles se utiliza el método Warn().

En la Figura 139 se puede ver un momento en la ejecución de esta página ASP .NET, en este caso se ha seleccionado que la información de las trazas se ordene por su categoría, y evidentemente en este caso se ha activado el mecanismo de trazas.

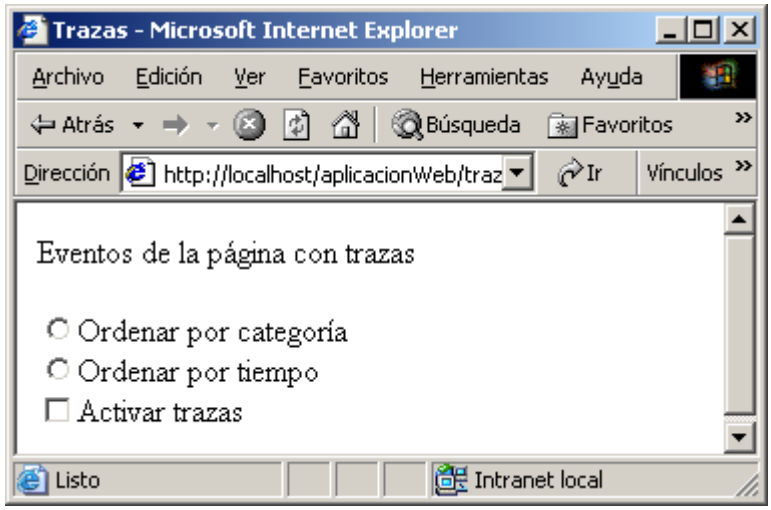


Figura 138

Información de seguimiento

Categoría	Mensaje	Desde los primeros	Desde los últimos
aspx.page	Begin ProcessPostData Second Try	0,000069	0,000069
aspx.page	End ProcessPostData Second Try	0,000144	0,000075
aspx.page	Begin Raise ChangedEvents	0,000181	0,000037
aspx.page	End Raise ChangedEvents	0,000261	0,000039
aspx.page	Begin Raise PostBackEvent	0,000297	0,000036
aspx.page	End Raise PostBackEvent	0,000339	0,000042
aspx.page	Begin PreRender	0,000376	0,000037
aspx.page	End PreRender	0,000466	0,000052
aspx.page	Begin SaveViewState	0,001253	0,000787
aspx.page	End SaveViewState	0,001359	0,000107
aspx.page	Begin Render	0,001409	0,000050
aspx.page	End Render	0,002087	0,000678
Eventos de la página	Evento Load		
Eventos de la página	Evento PreRender	0,000414	0,000038
Eventos de los controles	Se ha pulsado un RadioButton	0,000222	0,000041

Figura 139

En el siguiente enlace se puede obtener el código de esta página.

Trazas desde componentes .NET

El mismo método de interacción con el mecanismo de trazas de una página ASP .NET también se encuentra disponible desde un componente .NET.

Para poder acceder al entorno de trazas de una página ASP .NET desde un componente debemos importar el NameSpace System.Web y obtener una referencia al contexto en el que se está ejecutando el componente mediante la propiedad Current de la clase HttpContext. Esta propiedad nos va a devolver una instancia de la clase HttpContext que nos va a permitir acceder a la propiedad Trace como si nos encontráramos dentro de la propia página ASP .NET.

En el Código fuente 250 se puede ver la clase de un componente que muestra la fecha y hora actuales, como se puede comprobar en el método muestraHora() se hace uso del método Write() del objeto Trace para mostrar una traza en la página ASP .NET en la que se está ejecutando dicho componente.

```
using System;
using System.Web;


namespace Componentes.Ejemplos
{


	public class ComponenteSencilloTraza
	{
		private HttpContext contextoTrazas;


		public ComponenteSencilloTraza()
		{
			contextoTrazas=HttpContext.Current;
		}


		public String muestraHora()
		{
			DateTime ahora=DateTime.Now;
			contextoTrazas.Trace.Write("Componente .NET",
						"Se ha llamado al método muestraHora");
			return ahora.ToString();
		}
	}
}
```

Código fuente 250

El Código fuente 251 es el código que se corresponde con la página ASP .NET que hace uso de este componente.

```
<%@ Page Language="C#" Trace="True" TraceMode="SortByCategory" %>
<%@ Import Namespace="Componentes.Ejemplos" %>
<html>
<script runat="server">
ComponenteSencilloTraza componente=new ComponenteSencilloTraza();
void Pulsado(Object origen, EventArgs argumentos){
	etiqueta.Text =componente.muestraHora();
}
</script>
<head><title>Trazas</title></head>
<body>

<form id="formulario" method="post" runat="server">
<asp:button id="boton" text="Pulsar" onclick="Pulsado" runat="Server"/>
<asp:label id="etiqueta" runat="server"/>
</form>

</body>
</html>
```

Código fuente 251

Esta página resulta muy sencilla, únicamente muestra un Web Form con un objeto Button y un objeto Label, al pulsar sobre le objeto Button se mostrará la fecha y hora actuales en el objeto Label. Para poder hacer uso del componente .NET se ha tenido que utilizar la directiva @Import correspondiente.

Si pulsamos el botón que muestra el Web Form de la página del ejemplo, se obtendrá una sección de Información de seguimiento similar a la de la Figura 140.

Información de seguimiento

Categoría	Mensaje	Desde los primeros	Desde los últimos
aspx.page	Begin Init		
aspx.page	End Init	0,000057	0,000057
aspx.page	Begin LoadViewState	0,000102	0,000046
aspx.page	End LoadViewState	0,000183	0,000081
aspx.page	Begin ProcessPostData	0,000220	0,000037
aspx.page	End ProcessPostData	0,000269	0,000049
aspx.page	Begin ProcessPostData Second Try	0,000319	0,000051
aspx.page	End ProcessPostData Second Try	0,000354	0,000035
aspx.page	Begin Raise ChangedEvents	0,000388	0,000034
aspx.page	End Raise ChangedEvents	0,000422	0,000035
aspx.page	Begin Raise PostBackEvent	0,000456	0,000034
aspx.page	End Raise PostBackEvent	0,003302	0,000091
aspx.page	Begin PreRender	0,003342	0,000040
aspx.page	End PreRender	0,003386	0,000044
aspx.page	Begin SaveViewState	0,003799	0,000413
aspx.page	End SaveViewState	0,003884	0,000085
aspx.page	Begin Render	0,003925	0,000041
aspx.page	End Render	0,004373	0,000448
Componente .NET	Se ha llamado al método muestraHora	0,003211	0,002755

Figura 140

Como se puede comprobar los componentes .NET se encuentran perfectamente integrados con el mecanismo de trazas que nos ofrece ASP .NET.

En el siguiente enlace se puede obtener el código de la página ASP .NET y el código de la clase del componente .NET utilizados en el ejemplo.

En el siguiente apartado vamos a comentar la segunda alternativa de la que disponemos a la hora de utilizar el mecanismo de trazad que nos ofrece ASP .NET, en este caso se trata de las trazas a nivel de la aplicación ASP .NET.

Trazas a nivel de aplicación

En este caso el mecanismo de trazas se va a aplicar a toda la aplicación ASP .NET, para ello debemos editar y modificar el fichero de configuración de la aplicación, es decir, el fichero Web.config.

Para configurar el mecanismo de trazas de una aplicación ASP .NET debemos añadir un elemento <trace> dentro de la sección <system.web> del fichero de configuración Web.config.

La etiqueta trace, que nos va a permitir habilitar y configurar el mecanismo de trazas a nivel de una aplicación ASP .NET, presenta los siguientes atributos:

- enabled: este atributo indica si el mecanismo de trazas se encuentra activado a o no, tiene por lo tanto la misma funcionalidad que el atributo de mismo nombre de la directiva @Page. Su valor por defecto es false.

- requestLimit: en este atributo indicaremos el número máximo de peticiones HTTP de las que se va a almacenar información de trazas, las trazas van a ser almacenadas en un registro de

trazas mediante un mecanismo circular en las que permanecerán las últimas n peticiones. Por defecto el valor que presenta este atributo es 10.

- pageOutput: indica si la información de trazas se va a mostrar al final de cada página ASP .NET, tal como se hacía con las trazas a nivel de página. Aunque la información de las trazas no se muestre en la página, quedará constancia de las mismas en el registro de trazas. Su valor por defecto es false.

- traceMode: este atributo nos permite indicar el modo de ordenación de los mensajes de trazas en la sección Información de seguimiento, tiene por lo tanto la misma funcionalidad que el atributo de mismo nombre de la directiva @Page. Puede presentar los valores SortByCategory y SortByTime, su valor por defecto es SortByTime.

- localOnly: indica si la información de trazas se muestra únicamente a los clientes locales o por el contrario se muestra también a los clientes remotos. Su valor por defecto es false.

En el Código fuente 252 se muestra un fragmento del fichero Web.config de una aplicación ASP .NET, en este caso se está activando el mecanismo de trazas para la aplicación actual, pero la información de las trazas no se va a mostrar al final de cada página y únicamente se va a permitir acceder al registro de trazas de la aplicación a los clientes locales.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <trace
       enabled="true"
       requestLimit="10"
       pageOutput="false"
       traceMode="SortByTime"
      localOnly="true"
    />
  </system.web>
</configuration>
```

Código fuente 252

Debido a que existe la posibilidad de no mostrar la información de las trazas al final de cada página, como ocurre en el caso del ejemplo, debemos tener una herramienta que nos permita consultar la información de trazas de las distintas páginas ASP .NET que forman parte de la aplicación Web actual, para ello el entorno de ASP .NET nos ofrece una URL especial que nos va a permitir consultar el registro de trazas de la aplicación ASP .NET actual.

Para hacer uso de esta herramienta debemos invocarla navegando al recurso trace.axd desde el directorio raíz de la aplicación ASP .NET en la que se ha activado el mecanismo de trazas. Este fichero no existe realmente en la aplicación, sino que se trata de una URL especial que ASP .NET va a interceptar para mostrarnos el registro de trazas de la aplicación.

La llamada a este recurso nos va a mostrar una página similar a la de la Figura 141, en la que aparecen una serie de peticiones realizadas a las páginas ASP .NET de la aplicación actual. El número de peticiones dependerá del valor de la propiedad requestLimit de la etiqueta trace dentro del fichero Web.config.

Al pulsar sobre el enlace Ver detalles de cada una de las peticiones de las páginas ASP .NET, veremos la información de trazas de la página correspondiente. También se nos ofrece la posibilidad de eliminar el registro de trazas actual.

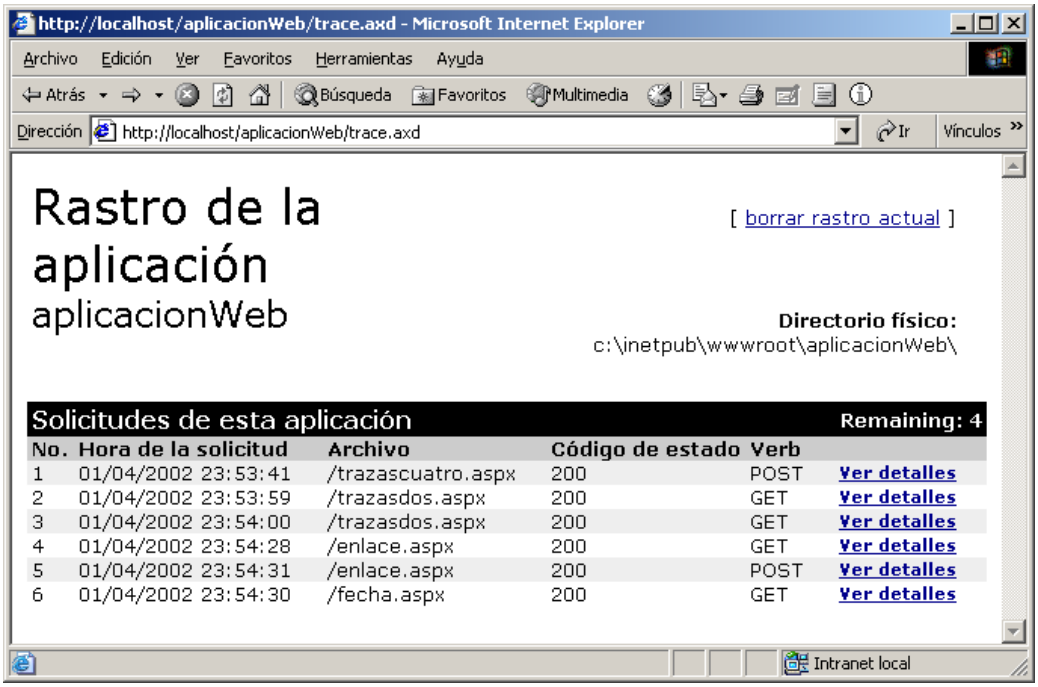


Figura 141

Mediante el mecanismo de trazas a nivel de aplicación podemos monitorizar toda la información de trazas de las páginas de la aplicación sin que los usuarios sean conscientes de ello.

Con este apartado finalizamos la parte del presente capítulo dedicado a la utilización de trazas dentro de ASP .NET, en lo que resta de capítulo comentaremos el mecanismo de depuración que ofrece ASP .NET.

Depuración en ASP .NET

Además de ofrecer un mecanismo de trazas, como novedad ASP .NET también ofrece herramientas para realizar la depuración de páginas. En versiones anteriores de ASP la mejor forma de depurar era, al igual que sucedía con las trazas, utilizar la instrucción Response.Write, aunque existía una herramienta para depurar llamada Script Debugger, conseguir depurar con la misma era bastante complejo y los resultados no eran los más satisfactorios.

Todo estos inconvenientes han terminado en .NET, ya que el CLR (Common Language Runtime), ofrece la depuración integrada para todo tipo de aplicaciones, incluidas las aplicaciones ASP .NET.

Dentro del entorno .NET Framework existen dos formas de depurar, la primera de ellas es utilizar la herramienta SDK debugger, esta herramienta se incluye en el SDK del .NET Framework, y la segunda posibilidad es utilizar el entorno de desarrollo Visual Studio .NET.

Depurando con el SDK Debugger

Esta herramienta, que nos va a permitir la depuración de páginas ASP .NET, se encuentra incluida como parte de la instalación del SDK del .NET Framework. El ejecutable de este depurador el fichero DbgCLR.exe y lo podemos encontrar en el directorio C:\Archivos de programa\Microsoft.NET\ FrameworkSDK\GuiDebug. Este depurador también es llamado Depurador Microsoft CLR.

Este depurador ofrece unas funcionalidades muy similares al depurador de Visual Studio .NET, exceptuando la depuración remota y la edición y continuación con la ejecución del código.

Como ya vimos en la primera parte de este capítulo, si queremos utilizar las trazas dentro de una página ASP .NET determinada debemos utilizar un atributo de la directiva @Page, para activar la depuración de una página debemos hacer algo similar. En este caso se debe utilizar el atributo Debug de la directiva @Page, este atributo admite los valores True/False, por defecto presenta el valor False, es decir, por defecto no se encuentra habilitada la depuración de páginas.

Al activar la depuración de una página ASP .NET estamos indicando al compilador que genere los símbolos de depuración necesarios para realizar el proceso de depuración cuando se estime necesario.

Si además de habilitar la depuración en las páginas ASP .NET, también deseamos depurar los componentes .NET utilizados en las mismas, debemos indicar a la hora de compilar los componentes que vamos a depurarlos. Para ello se utiliza el parámetro /debug con el compilador, si estamos utilizando el compilador del Microsoft .NET Framework SDK (Software Development Kit), es decir, la herramienta CSC. EN capítulos anteriores ya vimos otros parámetros que podíamos utilizar con el compilador de C#.

Si compilamos nuestros componentes desde Visual Studio .NET, no debemos hacer nada adicional, ya que por defecto el entorno de desarrollo compila los componentes con la opción de depuración habilitada.

A continuación vamos ver como se puede depurar una página ASP .NET y los componentes utilizados en la misma mediante el depurador ofrecido por el SDK del .NET Framework, para ello vamos a utilizar un ejemplo. Se trata de una página ASP .NET, cuyo código es el Código fuente 253, que hace uso de un sencillo componente .NET que muestra la fecha y hora actuales.

```
<%@ Page Debug="True"%>
<%@ Import Namespace="Componentes.Ejemplos" %>
<html>
<script language="C#" runat="server">
ComponenteSencillo componente=new ComponenteSencillo();
void Pulsado(Object origen, EventArgs argumentos){
      etiqueta.Text =componente.muestraHora();
}
</script>
<head><title>Depuración en ASP .NET</title></head>
<body>

<form id="formulario" method="post" runat="server">
<asp:button id="boton" text="Pulsar" onclick="Pulsado" runat="Server"/>
<asp:label id="etiqueta" runat="server"/>
</form>

</body>
</html>
```

Código fuente 253

Como se puede ver en el código anterior se ha habilitado la depuración en la página, el código de la página es muy sencillo, presenta un Web Form con una etiqueta y un botón, la pulsar el botón se instanciará un objeto de la clase Componentes.Ejemplos.ComponenteSencillo y se llamará al método muestraHora(). En el Código fuente 254 se puede ver el código de la clase del componente, se supone que este componente se ha compilado con el parámetro debug, para así permitir su depuración.

```
using System;

namespace Componentes.Ejemplos
{

	public class ComponenteSencillo
	{
		public ComponenteSencillo()
		{
		}

		public String muestraHora()
		{
			DateTime ahora=DateTime.Now;
			return ahora.ToString();
		}
	}
}
```

Código fuente 254

En la Figura 142 se puede ver la ejecución de esta página que hace uso del componente.

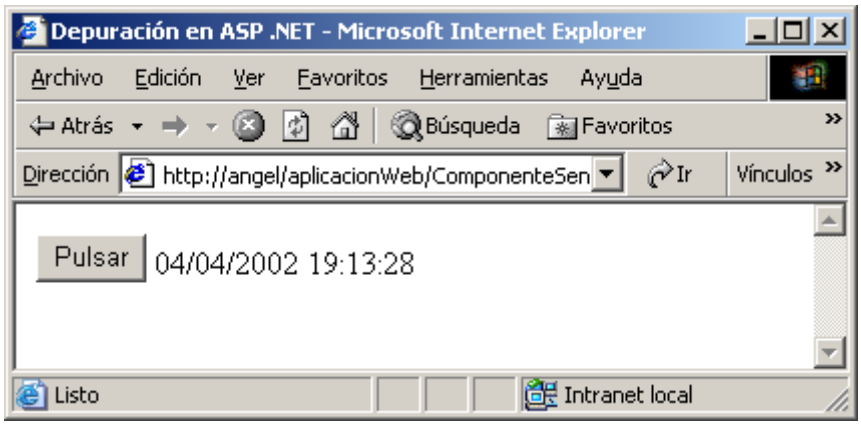


Figura 142

Se debe señalar que antes de utilizar el depurador es necesario ejecutar la página ASP :NET dentro del navegador correspondiente, de esta forma los símbolos de depuración para la página serán cargados para que los pueda utilizar el depurador en su proceso de depuración.

Ya tenemos preparada la página ASP .NET y el componente que deseamos depurar, el siguiente paso es ejecutar el depurador (fichero DbgCLR.exe), que se encuentra en la ruta C:\Archivos de programa\Microsoft.NET\FrameworkSDK\GuiDebug.

Una vez que se ha iniciado el depurador y con la página ASP .NET cargada en el navegador debemos acceder ala opción de menú Herramientas|Procesos de depuración. En ese momento aparecerá un

diálogo similar al de la . En ese diálogo debemos marcar la casilla Mostrar procesos del sistema, de esta forma podremos seleccionar el proceso que se corresponde con la ejecución de la página ASP .NET que vamos a depurar.

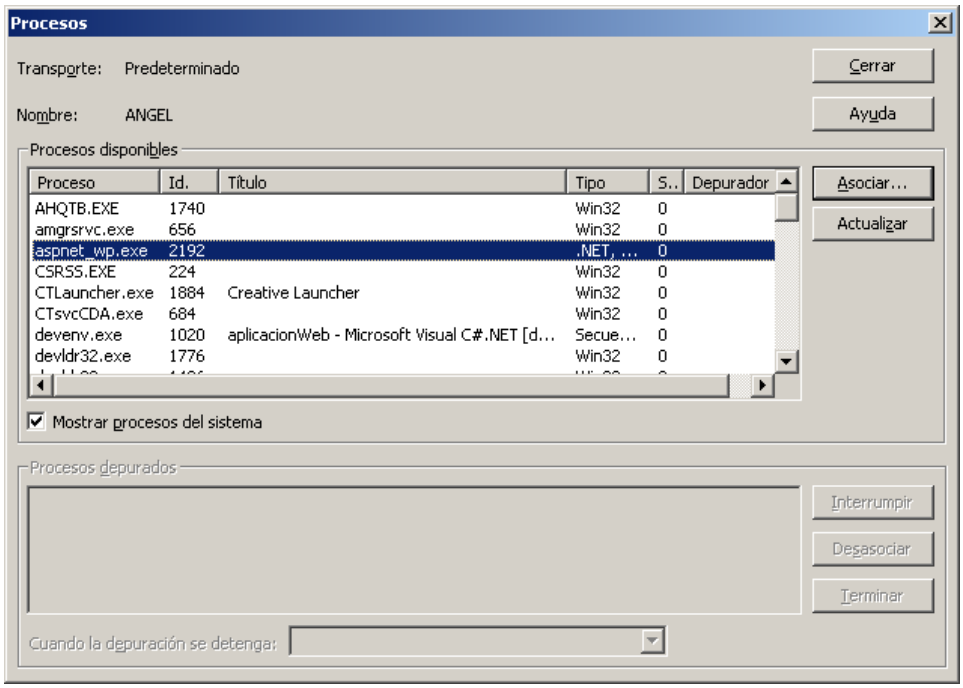


Figura 143

El proceso del sistema que se corresponde con la ejecución de las páginas ASP .NET es aspnet_wp.exe. Una vez seleccionado este proceso entre la lista de procesos del sistema, pulsamos el botón Asociar, de esta forma indicamos que el depurador va a depurar el proceso indicado, una vez hecho esto podemos cerrar el diálogo.

Una vez preparado el SDK debugger ya sólo tenemos que indicarle la página ASP .NET que deseamos depurar, para ello se utiliza la opción de menú Archivo|Abrir. En el diálogo que aparece indicaremos la página correspondiente.

Con la página ASP .NET cargada en el depurador podemos añadirle un punto de ruptura. Para añadir eun punto de ruptura en el código fuente, únicamente debemos pulsar en el margen gris de la línea correspondiente. En este caso vamos a añadir el punto de ruptura en la línea en la que se llama al método muestraHora() del objeto componente de la clase ComponenteSencillo. Al ejecutarse esta línea de código de la página ASP .NET se detendrá al ejecución de la misma en el punto de ruptura, es decir, el SDK debugger funciona de la misma forma que los depuradores convencionales de otros entornos. En la Figura 144 se puede ver el aspecto que tendría la página dentro del depurador CLR.

ComponenteSe...puracion.aspx

```
<%@ Page Debug="True"%>
<%@ Import Namespace="Componentes.Ejemplos" %>
<html>
<script language="C#" runat="server">
ComponenteSencillo componente=new ComponenteSencillo();
void Pulsado(Object origen, EventArgs argumentos){
    etiqueta.Text =componente.muestraHora();
```

En ComponenteSencilloDepuracion.aspx, línea 7 carácter 2 ('ASP.ComponenteSencilloDepuracion_aspx.Pulsado(System.Object, System.EventArgs)') en el programa '[604] aspnet_wp.exe'

```
<head><title>Depuración en ASP .NET</title></head>
<body>

<form id="formulario" method="post" runat="server">
<asp:button id="boton" text="Pulsar" onclick="Pulsado" runat="Server"/>
<asp:label id="etiqueta" runat="server"/>
</form>

</body>
</html>
```

Figura 144

Una vez que se ha detenido la ejecución en el punto de interrupción, podemos controlar lo que deseamos hacer a continuación. Podemos acceder al código fuente del método dentro del componente si pulsamos la opción Ir a instrucciones, en este caso se abrirá una nueva ventana de código, similar a la de la Figura 145 con el código fuente del componente .NET, y se indicará en color amarillo en que línea nos encontramos.

ComponenteSencilloDepuracion.aspx | ComponenteSencillo.cs

```
using System;

namespace Componentes.Ejemplos
{

    public class ComponenteSencillo
    {
        public ComponenteSencillo()
        {
        }

        public String muestraHora()
        {
            DateTime ahora=DateTime.Now;
            return ahora.ToString();
        }
    }
}
```

Figura 145

Una vez que se termina de ejecutar el código del componente .NET se vuelve a la página, es decir, tenemos un control completo sobre los pasos en la ejecución de la página, ya que podemos indicar también salir del componente mediante la opción Paso a paso para salir.

Para aquellos lectores que hayan utilizado versiones anteriores de ASP comprobarán las grandes facilidades que nos ofrece este depurador, la depuración de páginas ASP .NET supone todo un avance a la hora de localizar errores en nuestras aplicaciones.

El depurador CLR ofrece una serie de características muy similares a las de otros depuradores de otros entornos de desarrollo, por lo que no nos vamos a entretener mucho más comentando este depurador, aunque si merece la pena enumerar las posibilidades que nos ofrece:

- Control completo sobre la ejecución de las instrucciones del código fuente, permitiendo Ir a instrucciones, Paso a paso por procedimientos y Paso a paso para salir.
- Establecimiento de puntos de ruptura en líneas específicas.
- Establecimiento de puntos de ruptura en excepciones específicas.
- Establecimiento de puntos de ruptura en expresiones.
- Visualización del contenido de variables.
- Visualización de la pila de llamadas.

Pero hay dos características, que ya avanzábamos anteriormente, que no las soporta este depurador, se trata de la depuración de procesos remotos y la posibilidad de editar el código y continuar. Por esto último se dice que el depurador CLR es un depurador de sólo lectura, debido a esto no podemos editar el código y continuar con el proceso de depuración. Si deseamos estas características adicionales debemos acudir al depurador de Visual Studio .NET.

Depurando con Visual Studio .NET

En el entorno de desarrollo Visual Studio .NET, la depuración es una parte del propio entorno, es decir, se encuentra integrada con le herramienta de desarrollo.

Para depurar desde Visual Studio .NET únicamente debemos establecer como página de inicio la página ASP .NET que deseamos depurar. Esta página debe asignar el valor True al atributo Debug de la directiva @Page, esto es algo común para cualquier depurador.

En la vista HTML del editor de Visual Studio .NET estableceremos los puntos de ruptura necesarios, una vez hecho esto acudimos a la opción de menú Depurar|Iniciar, y de forma automática se cargará la página ASP .NET indicada como página de inicio y se iniciará la depuración de la misma.

El aspecto y forma de trabajar que ofrece el depurador de Visual Studio .NET es muy similar a la ofrecida por el depurador CLR, pero además permite la depuración remota, es decir, permite depurar páginas ASP .NET que se encuentran en servidores Web remotos, y además permite modificar el código fuente de las páginas mientras éstas se están ejecutando.

En la Figura 146 se puede ver Visual Studio .NET en pleno proceso de depuración, en este caso se trata de la misma página que utilizamos para el depurador CLR con su correspondiente punto de ruptura.

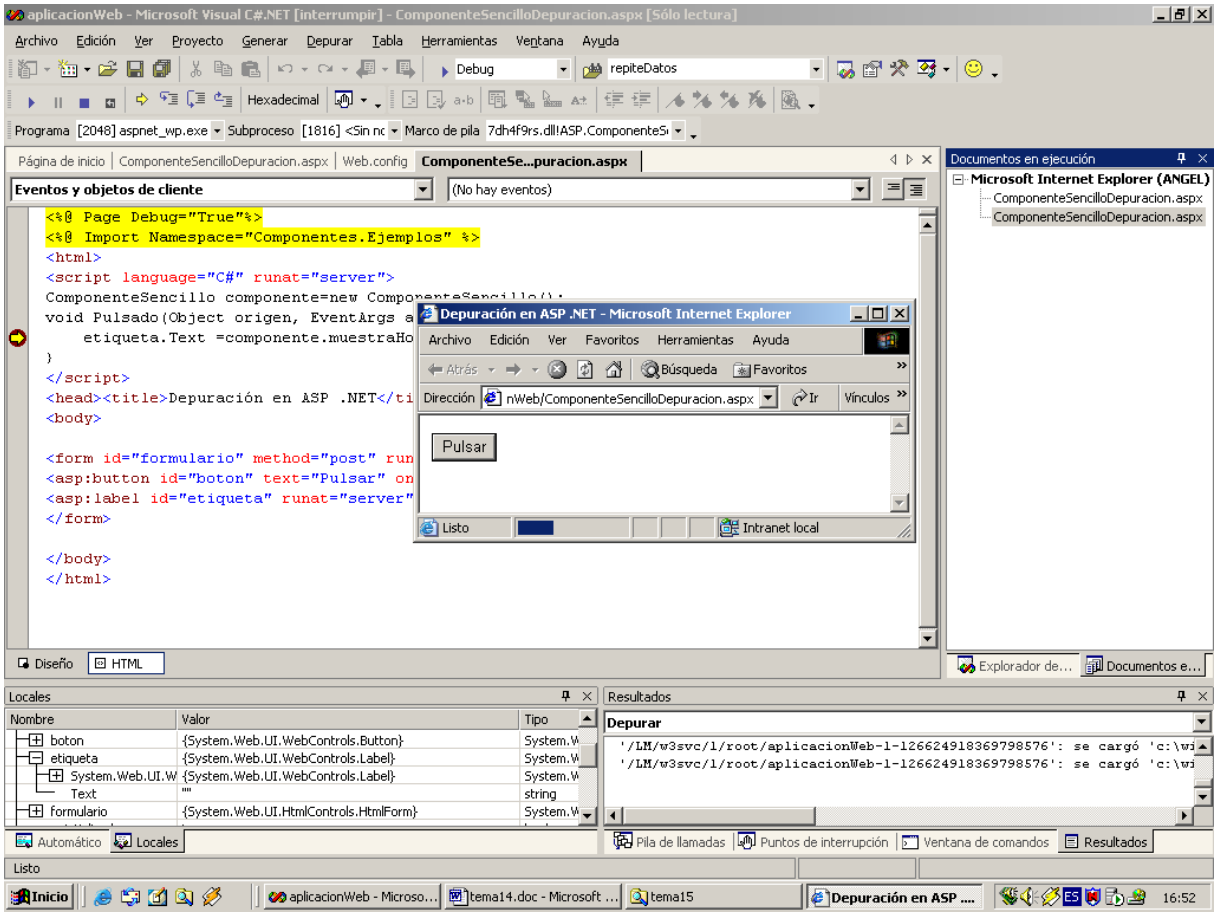


Figura 146

Con este apartado finalizamos el presente capítulo, el siguiente capítulo supone un paréntesis en el texto, ya que no nos vamos a ocupar de ASP .NET realmente, sino que vamos a comentar el servidor Web que alberga a nuestras aplicaciones ASP .NET, se trata del servidor Internet Information Server 5.0.

17

Internet Information Server 5.0

Introducción

Como ya advertíamos al final del capítulo anterior, este capítulo se va a salir un poco de las líneas que estábamos siguiendo, ya que no trata directamente sobre ASP .NET, sino que en este caso vamos a comentar el servidor Web sobre el que se ejecutan las páginas ASP .NET. Este servidor Web no es otro que Internet Information Server 5.0.

Internet Information Server (IIS) es un servidor Web que permite publicar información en una intranet de la organización o en Internet. Internet Information Server transmite la información mediante el protocolo de transferencia de hipertexto (HTTP).

Internet Information Server puede configurarse para proporcionar servicios de Protocolo de transferencia de archivos (FTP), y a partir de la versión 4.0, puede proporcionar también servicio de correo electrónico basado en el protocolo SMTP (Simple Mail Transfer Protocol) y servicio de noticias basado en el protocolo NNTP (Network News Transfer Protocol).

Las versiones anteriores de ASP forman parte de Internet Information Server desde la versión 3.0 del servidor Web. Al instalar IIS 5.0 se instala por defecto ASP 3.0. Se debe recordar que ASP .NET es completamente compatible con ASP, podemos tener aplicaciones Web basadas en ASP y ASP .NET funcionando en el mismo servidor Web Internet Information Server 5.0, las páginas ASP tienen la extensión .ASP y son procesadas por la DLL ASP.DLL y sin embargo las páginas ASP .NET poseen la extensión .ASPX y son procesadas por el entorno de ejecución .NET Framework

Cuando llega una petición al servidor Web, éste las procesa, desde páginas HTML a vídeo, el proceso sigue los métodos convencionales de un servidor Web, tal como enviar el fichero al navegador que ha realizado la petición.

En este capítulo vamos a tratar los temas de configuración y administración de IIS 5.0 que tienen que ver más o menos directamente con el entorno de ASP, no se trata de un capítulo que intente explicar la administración y configuración de IIS 5.0, ya que esto supondría un texto exclusivo completo.

El servidor web Internet Information Server 5.0

Internet Information Server 5.0 es un servidor Web para plataformas Windows 2000 completamente integrado con el sistema operativo. IIS 5.0 forma parte de la instalación de Windows 2000 y permite disponer de un servidor Web tanto en el entorno de Internet como en el entorno de Intranet.

IIS 5.0 se encuentra en todas las versiones de Windows 2000: Professional, Server, y Advanced Server, pero para implementar un servidor Web es más adecuado elegir la versión Server o Advanced Server, aunque para pruebas o desarrollo puede ser completamente válida la versión Professional.

IIS 5.0 ofrece una administración muy sencilla que se realizará mediante el Administrador de servicios de Internet.

La versión 5.0 de IIS permite que el desarrollo de aplicaciones Web sea mucho más robusto y la creación de sitios Web sea más configurable y completa. Ofrece un entorno escalable basado en los componentes cliente/servidor que se pueden integrar dentro de las aplicaciones Web.

Internet Information Server 5.0 es el servidor Web más rápido y recomendable para la plataforma Windows 2000, ya que se encuentra integrado completamente con el Servicio de Directorios de Windows 2000, esta combinación del servicio Web con los servicios del sistema operativo permite desarrollar aplicaciones basadas en la Web fiables y escalables.

Instalando IIS 5.0

Al instalar Windows 2000, en cualquiera de sus versiones, deberemos indicar que deseamos instalar también el componente de Windows 2000 llamado Servicios de Internet Information Server (IIS). Este componente va a convertir nuestro equipo en un servidor Web que soporte aplicaciones ASP, si deseamos ejecutar también aplicaciones ASP .NET debemos instalar el .NET Framework, tal como comentábamos en los capítulos de introducción.

Para averiguar si tenemos instalado IIS 5.0 en nuestra máquina podemos realizar una sencilla prueba que consiste en escribir la siguiente URL http://nombreEquipo en el navegador Web. Si IIS 5.0 está instalado aparece una ventana similar a la Figura 147.

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

Figura 147. IIS 5.0 está instalado

Figura 148. El Administrador de servicios de Internet en el menú de Inicio

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

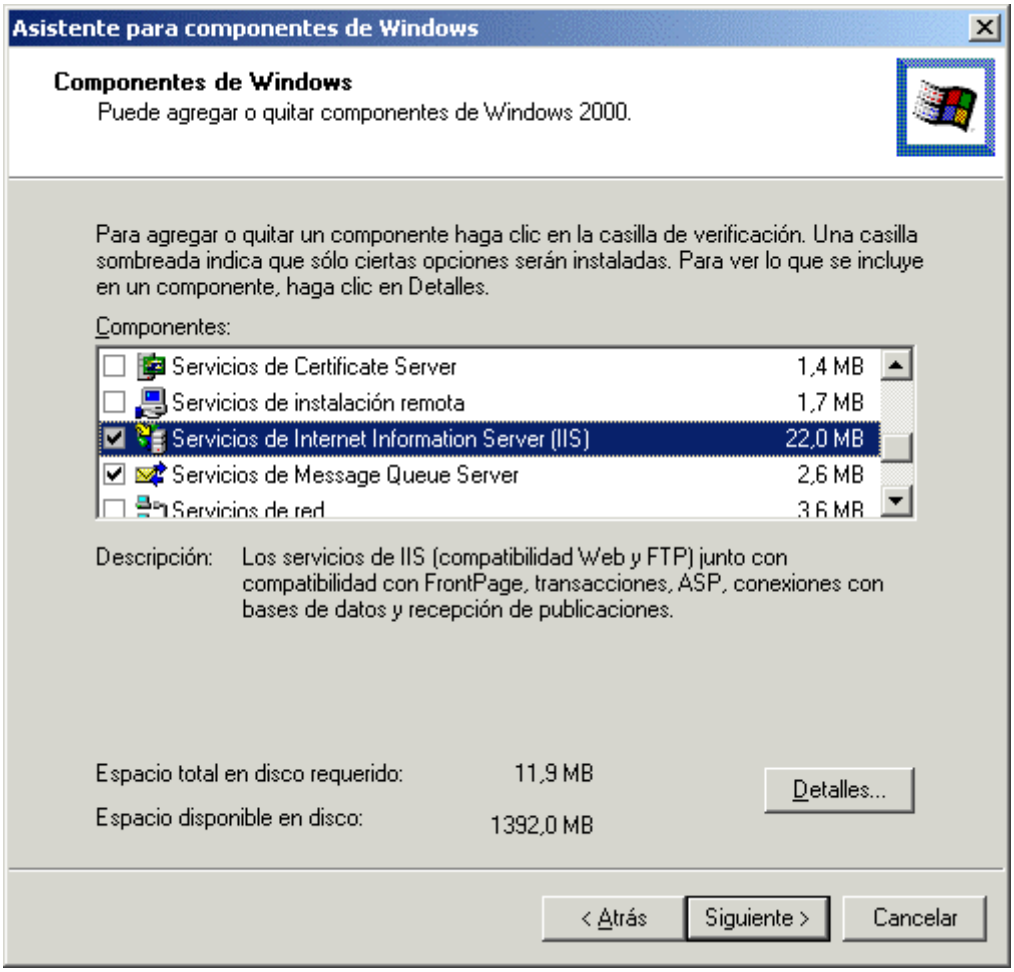


Figura 149. Instalando IIS 5.0

La instalación de IIS 5.0 es muy sencilla, lo que nos pedirá será la ruta del directorio de publicación en Internet y poco más, por defecto esta ruta es c:\Inetpub\wwwroot.

Novedades de IIS 5.0

Realmente no existen grandes diferencias ni novedades dramáticas respecto a la versión anterior de IIS. En nuestro caso sólo vamos a comentar las novedades que nos interesen desde nuestro punto de vista, es decir, desde ASP.

Un adelanto importante fue el que ofreció IIS 3.0 ya que fue el primer servidor Web que soportaba páginas ASP y el que introdujo esta tecnología. IIS 3.0 era poco robusto y su configuración y administración se encontraba bastante restringida a unas pocas opciones.

La versión 4.0 de IIS supuso un gran paso dentro de la política de Microsoft respecto a Internet, ya que IIS 4.0 era mucho más completo que su predecesor, permitiendo una administración y configuración bastante detallada. También resulta mucho más robusto y escalable.

IIS 4.0 implementaba el concepto de aplicación Web, permitiendo aislar las mismas y ofrecía también distintas formas de ejecución para una aplicación ASP. Las aplicaciones ASP se podían configurar desde IIS 4.0, cosa que no ocurría con IIS 3.0.

En la versión 4.0 de IIS se eliminó el anticuado servicio Gopher y se incluyó la versión 2.0 de las páginas activas de servidor. Gracias a la integración de IIS 4.0 con el servidor de transacciones MTS 2.0 (Microsoft Transaction Server) se podían crear páginas ASP transaccionales, fue en este momento cuando apareció el objeto integrado de ASP ObjectContext.

Y por fin llegamos a la versión actual de IIS, la versión 5.0. Esta versión al igual que las predecesoras ofrece una nueva versión de ASP, la versión 3.0 que es la versión anterior a ASP .NET, aunque como ya se ha comentado ASP .NET no es sólo una nueva versión de ASP, el lector a estas alturas del texto ya habrá comprendido tal afirmación.

Como ya adelantábamos IIS 5.0 no supone un gran cambio respecto a IIS 4.0, vamos a comentar alguna de las novedades que aporta la nueva versión del servidor Web de Microsoft. Con IIS 5.0 se permite una nueva forma de definir las aplicaciones ASP que consiste en definir aplicaciones ASP para que se ejecuten de forma aislada al servicio Web pero agrupadas.

Se ha eliminado el servidor de transacciones MTS 2.0, ahora ha pasado a formar parte de lo que se denomina Servicios de componentes. IIS 5.0 también se encuentra perfectamente integrado con Servicios de componentes aunque no se encuentran en la misma herramienta de administración.

IIS 5.0 es más robusto y escalable que su predecesor, pudiéndose incluso recuperar de fallos de forma automática.

Las extensiones de FrontPage, necesarias para la creación de proyectos Web desde Visual InterDev en el servidor, se encuentran integradas de forma completa en la herramienta administrativa de IIS 5.0 que se denomina Administrador de servicios de Internet, y que posee el mismo aspecto que la consola MMC de la versión 4.0. Ya no es posible desde el menú de Inicio tener acceso al sitio Web de administración de IIS 5.0, sin embargo este Web de administración sigue existiendo y lo veremos en próximos apartados.

El administrador de servicios de internet

Internet Information Server 5.0 ofrece dos herramientas de administración, por un lado podemos utilizar una herramienta denominada Administrador de servicios de Internet, que es similar a lo que ya existía en IIS 4.0, es decir, se trata de un complemento de la consola MMC (Microsoft Management Console), y por otro lado podemos administrar IIS 5.0 a través de una versión del administrador de servicios Internet basada en la administración a través de Web, utilizando lo que se denomina sitio Web de administración.

La administración remota a través de Web es posible gracias al sitio Web de administración que ofrece IIS 5.0 como parte de su instalación. Por defecto este sitio Web se encuentra restringido a la máquina local, es decir, a la dirección IP 127.0.0.1 o al nombre de equipo localhost (máquina local). Esta restricción es lógica, ya que no desearemos que la administración de nuestro servidor Web pueda ser realizada por cualquier usuario anónimo de Internet.

La administración remota puede ser muy útil cuando se produzca algún problema en el servidor y el administrador del mismo no se encuentre físicamente en el mismo lugar, el problema lo podrá resolver cómodamente desde el lugar en el que se encuentre.

Se debe tener en cuenta que la dirección IP de la máquina en la que se encuentre el administrador debe tener el acceso concedido al sitio Web de administración y debe pertenecer a los operadores de sitios Web.

El sitio Web de administración poseerá el mismo nombre que el sitio Web predeterminado del servidor en el que esté instalado IIS 5.0, pero se diferenciará en el número de puerto asignado al sitio Web de administración. Al instalar IIS 5.0 y crear el sitio Web de administración se le asigna un número de puerto aleatorio. Un poco más adelante veremos como averiguar este número de puerto y así poder conocer la dirección del sitio Web de administración y utilizarlo para administrar IIS 5.0 de forma remota.

Aunque la administración a través de este sitio Web ofrece muchas de las funciones recogidas por el Administrador de servicios de Internet, hay una serie de facilidades de administración que no se encuentran disponibles, ya que requieren coordinación con el sistema operativo.

Vamos a comentar la herramienta de Administración de servicios de Internet. La ejecutamos desde del menú Inicio|Herramientas administrativas|Administrador de servicios Internet. Esta herramienta de administración, nos va a permitir administrar y configurar en un mismo entorno todos los servicios del servidor IIS 5.0.el uso de esta herramienta es bastante sencillo e intuitivo.

La consola que ofrece el Administrador de servicios de Internet se divide en dos paneles, el panel de la izquierda se denomina panel de alcance (*scope pane*), y el de la derecha panel de resultados. El panel de alcance presenta una jerarquía en forma de árbol con los componentes que se pueden administrar en la consola.

En el panel de alcance se representan los servidores que se pueden administrar mediante un icono de un ordenador acompañado del nombre del servidor.

Cada uno de los nodos del árbol que aparece en el panel de alcance son instancias de servicios individuales y pueden contener a su vez un conjunto de subobjetos administrables. Podemos abrir las páginas de propiedades de cualquier nodo pulsando el botón derecho sobre le mismo y seleccionado la opción Propiedades del menú desplegable.

El segundo panel que aparece en la consola es el de resultados, este panel visualiza los contenidos o resultados referentes al nodo seleccionado en el panel de alcance, de manera que se puede acceder a todo tipo de información relacionada con cada nodo. En la Figura 150 se puede ver el aspecto que muestra el Administrador de servicios de Internet.

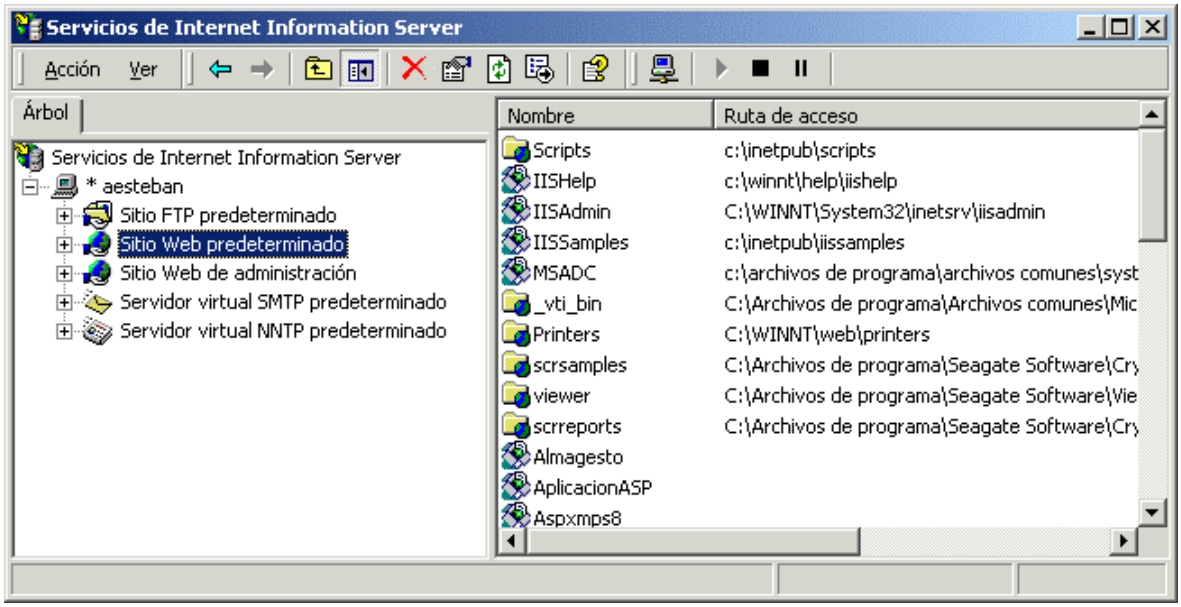


Figura 150. Administración de los servicios de IIS

Desde la consola no sólo podremos realizar la administración local de un servidor, sino que podremos realizar la administración remota de otros servidores IIS 5.0.

Para administrar un servidor remoto deberemos pulsar con el botón derecho sobre el nombre de nuestro servidor y seleccionar la opción conectar o bien sobre el icono de la barra de herramientas que representa aun equipo.

A continuación aparecerá una ventana en la que se nos pide el nombre del equipo al que nos queremos conectar, una vez indicado el nombre y pulsado el botón de aceptar, de forma automática se añadirá otro elemento servidor a la consola identificado con su nombre de equipo correspondiente.

En la Figura 151 se puede ver el aspecto que muestra un servidor local y dos servidores remotos, para distinguirlos el servidor local se señala con un asterisco (*) y la apariencia del icono que lo representa es diferente.

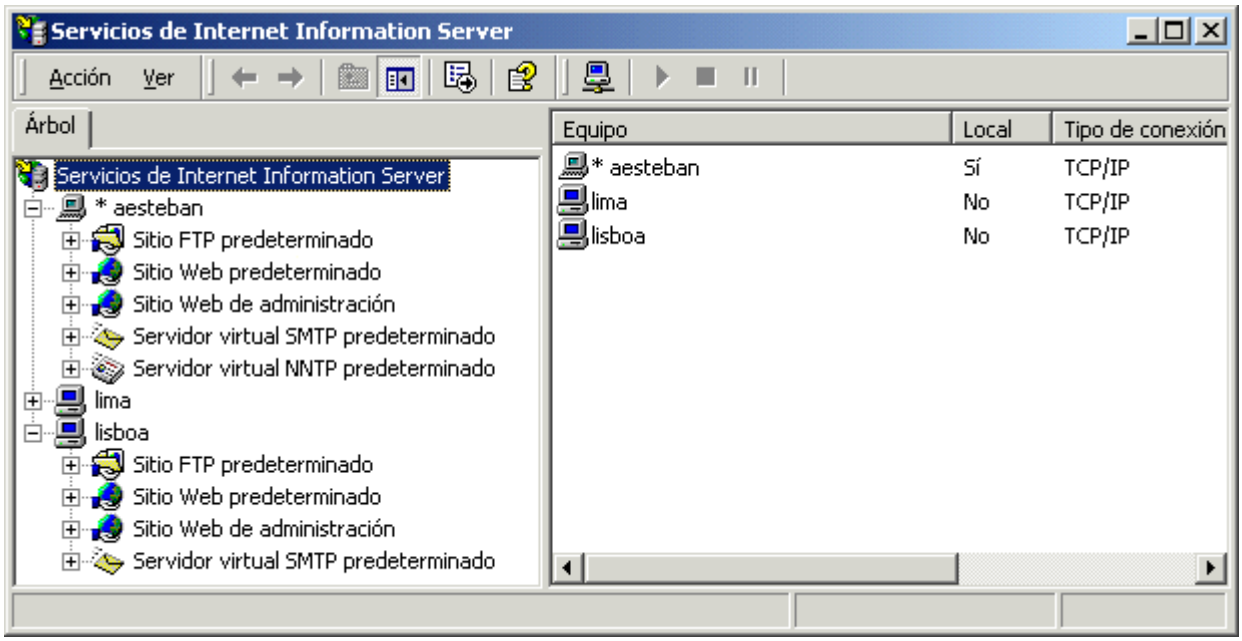


Figura 151. Administración remota de servidores IIS 5.0

Como ya se ha comentado, la utilización de la consola es muy sencilla lo único que hace falta es realizar una serie de pruebas para familiarizarse con ella.

La configuración de Internet Information Server se basa en la utilización de hojas de propiedades, un ejemplo se puede ver en la figura 6. A estas hojas de propiedades (*Property Sheets*) accederemos pulsando con el botón derecho del ratón, sobre el elemento a configurar, y seleccionando la opción Propiedade*s* que aparece en el menú contextual.

Cada hoja de propiedades mostrará una serie de parámetros para configurar, dependiendo del elemento seleccionado (directorio, sitio Web o fichero). Las hojas de propiedades tendrán también una serie de pestañas o fichas que intentan facilitarnos la administración de los servicios agrupando en cada ficha parámetros configurables relacionados. Se pueden definir propiedades para sitios Web, directorios e incluso ficheros. En la Figura 153 se puede apreciar el aspecto que presentaría la hoja de propiedades correspondiente a un fichero.

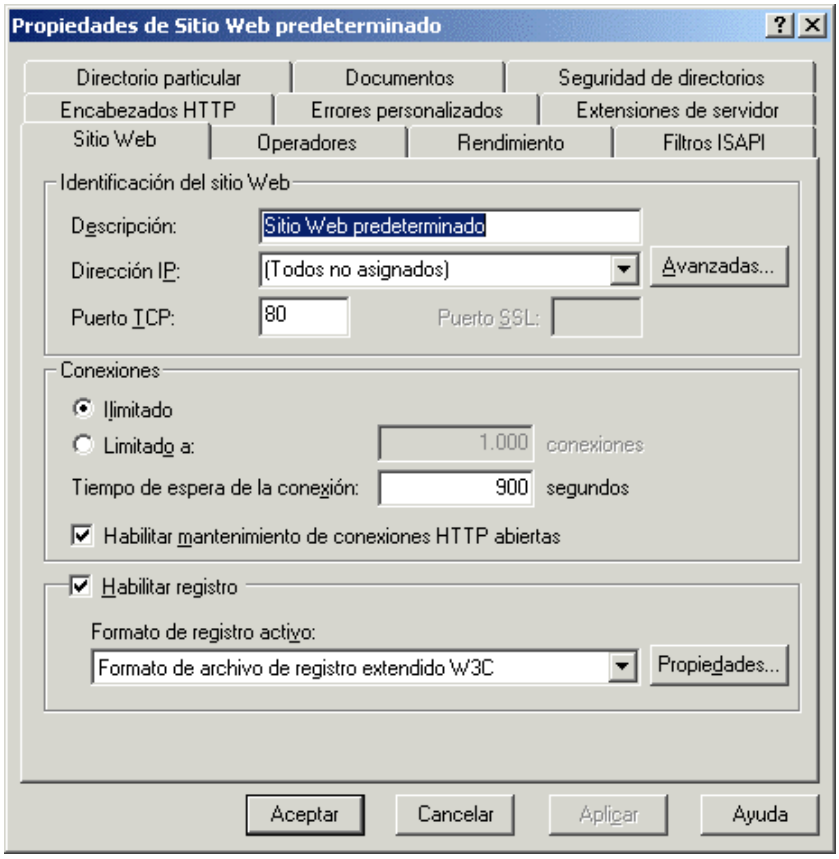


Figura 152. Hoja de propiedades de un sitio Web

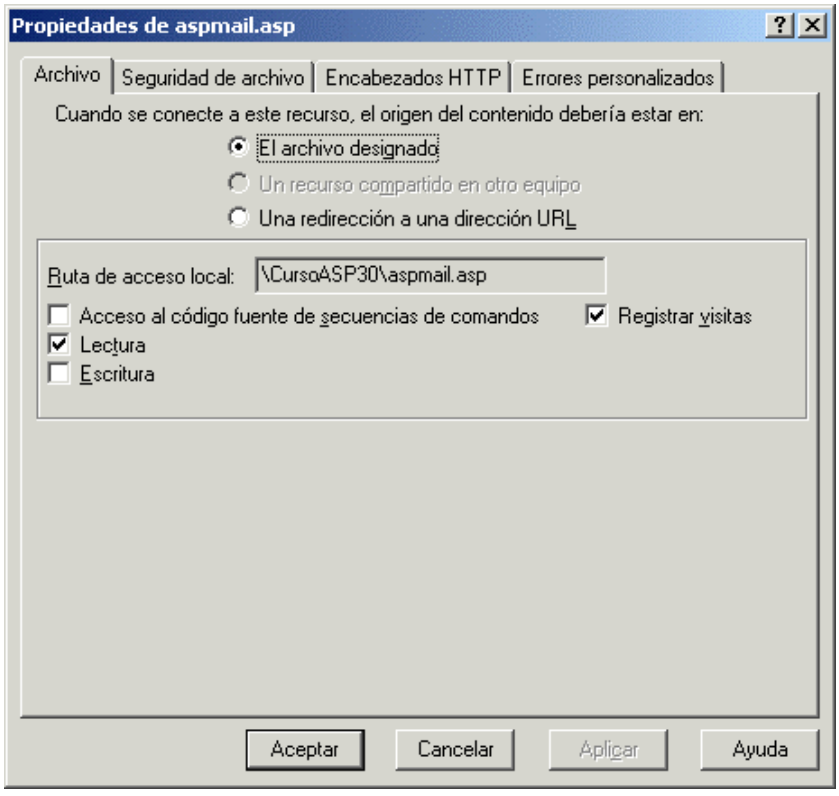


Figura 153. Hoja de propiedades de un fichero

Elementos de IIS 5.0

Dentro del Administrador de servicios de Internet se pueden distinguir varios nodos, dependiendo también de la instalación que se haya realizado de IIS 5.0. En la instalación más completa vemos lo siguiente:

- Sitio FTP predeterminado: es el sitio FTP del servidor, desde aquí configuraremos el servicio FTP que ofrece IIS 5.0.
- Sitio Web predeterminado: es el elemento que más nos va a interesar, desde aquí se configura el servicio Web que ofrece IIS 5.0.
- Sitio Web de administración: nos ofrece la posibilidad de administrar IIS 5.0 desde un sitio Web, como ya hemos comentado anteriormente.
- Servidor virtual SMTP predeterminado: representa el servicio de correo de IIS 5.0, se trata del servicio de correo saliente SMTP (Simple Mail Transfer Protocol).
- Servidor virtual NNTP predeterminado: representa el servicio de noticias de IIS 5.0, se trata del servicio NNTP (Network News Transport Protocol).

En este capítulo nos vamos a centrar en el sitio Web predeterminado, que representa el sitio Web por defecto del servidor IIS 5.0 y que se corresponde con http://nombreMaquina.

Para averiguar en que puerto se encuentra el sitio Web de administración y así poder utilizarlo de forma remota, pulsaremos con el botón derecho del ratón sobre este elemento y seleccionaremos la opción Propiedades. En la hoja de propiedades del sitio Web de administración parece el número de puerto (puerto TCP) que tiene asignado. Y para acceder desde el navegador al Web de administración simplemente escribiremos: http://localhost:numPuerto o si queremos administrar IIS desde otra máquina escribiremos: http://nombreServidor:numPuerto.

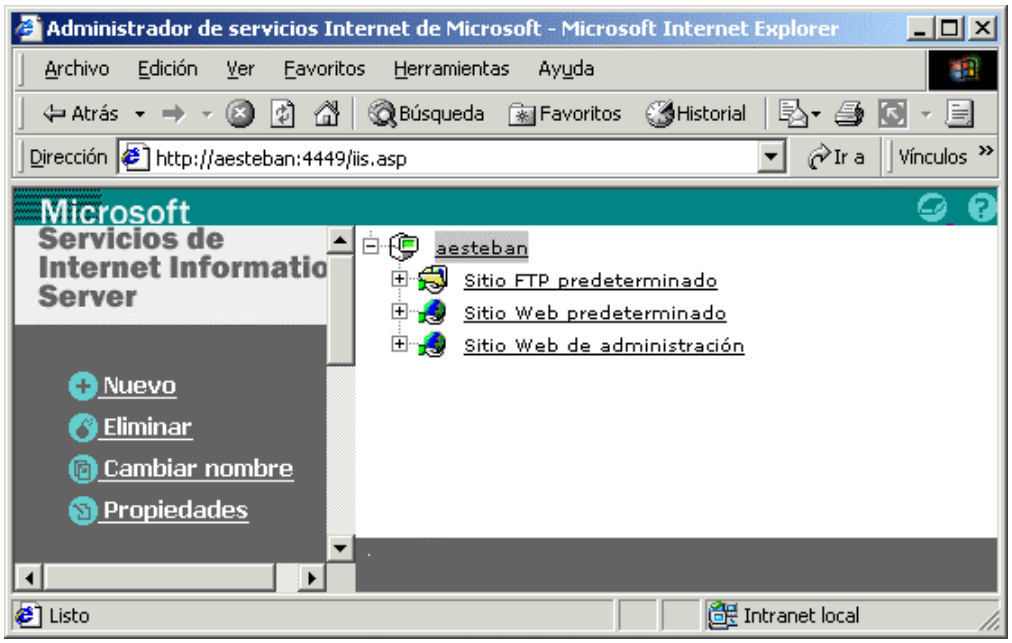


Figura 154. El sitio Web de administración

Administración del sitio web

Cuando creamos un sitio Web lo hacemos con la intención de hacer disponible una serie de información más o menos compleja (desde páginas HTML estáticas hasta aplicaciones Web basadas en páginas ASP o ASP .NET) para una serie de usuarios, ya sea para nuestra propia empresa u organización (Intranet) o para un grupo de usuarios más numeroso y general (Internet). A través del sitio Web ofrecemos la información y aplicaciones Web que se encuentren dentro de un directorio determinado en el servidor Web, es decir, en el servidor en el que tenemos instalado IIS 5.0.

Este directorio a partir del cual se publican los documentos que se encuentran en él en el entorno de Internet o Intranet se denomina directorio de publicación o particular (*home*), o también lo podemos encontrar como directorio raíz.

Para publicar documentos en el sitio Web, si este sitio Web únicamente consta de archivos ubicados en un disco duro del equipo (servidor IIS 5.0), simplemente deberá copiar los ficheros correspondientes al directorio de publicación del sitio Web. En el caso del sitio Web predeterminado, si se han respetado los valores por defecto en la instalación de IIS, su directorio de publicación (a partir de ahora vamos a utilizar siempre este término en lugar de directorio raíz o particular) será c:\Inetpub\wwwroot.

Cada sitio Web es necesario que tenga un directorio de publicación, en este directorio se suele ubicar la página que aparece por defecto cuando indicamos el nombre del sitio Web.

Cada sitio Web debe tener asignados una dirección IP y un número de puerto que servirán para identificar el sitio Web, junto con el nombre de encabezado de host, entre los diferentes sitios Web que tengamos hospedados en nuestro servidor.

Dentro del sitio Web podremos definir una serie de propiedades cuyos valores se aplicarán a todos los elementos contenidos en el sitio Web (ficheros o subdirectorios) o no, es decir, si modificamos una propiedad en el sitio Web tenemos la posibilidad de elegir si deseamos que el valor se propague al resto de los elementos o no.

A continuación vamos a proceder a comentar las propiedades más relevantes que podemos encontrar en la hoja de propiedades de un sitio Web. No se va a tratar de una descripción detallada, ya que no es el objetivo del presente texto.

Sitio Web

Si seleccionamos el sitio Web predeterminado y pulsamos sobre él con el botón derecho del ratón podremos seleccionar la opción Propiedades del menú que aparece. De esta forma podremos observar las diferentes propiedades que podemos configurar a nivel de sitio Web.

La primera hoja que podemos observar dentro de las hojas de propiedades de un sitio Web es la identificada mediante Sitio Web. En esta ficha, podremos configurar una serie de parámetros relacionados con la identificación del sitio Web, las conexiones al sitio Web y la forma de registrar los accesos a nuestro sitio Web.

En cuanto a la identificación del sitio Web tenemos cuatro parámetros: Descripción, Dirección IP, Puerto TCP y Puerto SSL. En Descripción indicaremos el nombre con el que identificaremos nuestro sitio dentro de la consola, no se debe confundir con el nombre de dominio o URL que utilizamos para acceder al sitio Web a través de Internet. De esta forma podremos cambiar la denominación de sitio Web Predeterminado por la que deseemos.

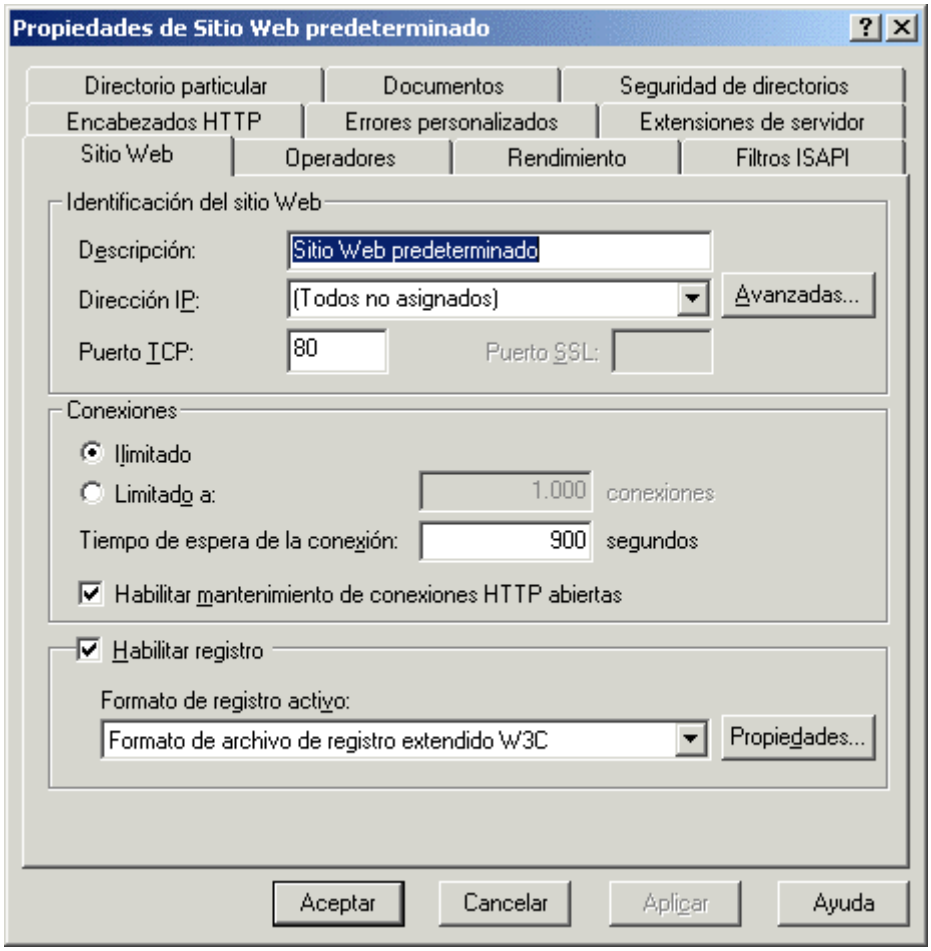


Figura 155. Propiedades del sitio Web

El siguiente parámetro es la dirección IP que tiene asociada el sitio Web, podemos elegir una de las que tenga asignadas el servidor o bien elegir la opción Todos no asignados, si elegimos esta última opción estaremos asignando al sitio Web todas las direcciones IP del servidor que se encuentren libres. De esta forma pasa a ser el sitio Web predeterminado.

El número de puerto normalmente no será necesario modificarlo, por defecto el servicio Web se encuentra en el puerto 80.

El puerto SSL (Secure Sockets Layer) sólo se debe utilizar para conexiones seguras, basados en certificados de cliente y servidor.

En la conexión a nuestro sitio Web podremos limitar el número de conexiones de clientes o bien indicar que aceptamos un número ilimitado de ellas. También podremos indicar el tiempo de espera máximo (time-out) utilizado para establecer una conexión, esto asegurará que se cerrarán todas las conexiones si el protocolo HTTP no puede establecer una conexión desde el cliente en el tiempo especificado, por defecto son 900 segundos.

Podremos activar el registro de nuestro sitio Web seleccionando la opción correspondiente. Tenemos diferentes formatos para guardar la información relativa los accesos a nuestro sitio Web. Cada uno de estos formatos definen que datos se van a almacenar y que estructura va a tener el fichero o tabla en la que van a ser almacenados.

Directorio particular

Esta hoja de propiedades es utilizada para configurar el directorio de publicación del sitio Web y se puede observar en la Figura 156.

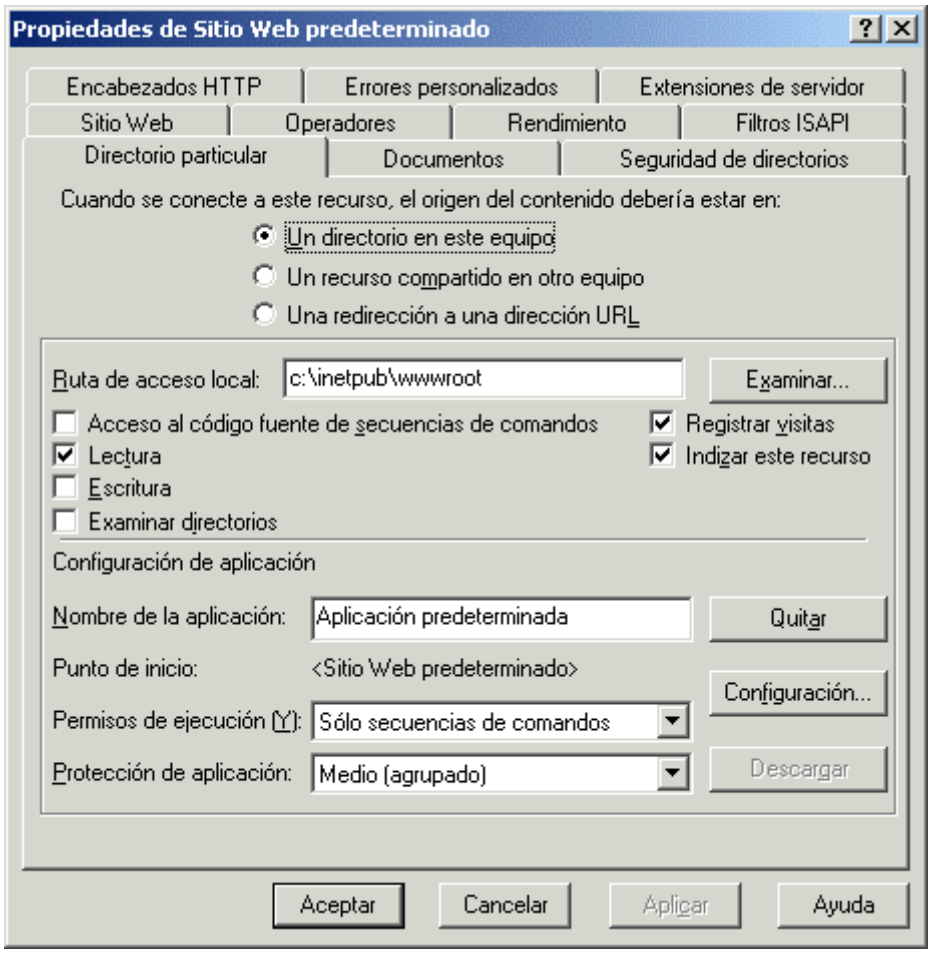


Figura 156. Propiedades del directorio de publicación en Internet

En esta hoja podremos modificar la ubicación del directorio de publicación del sitio Web, por defecto el directorio de publicación es c:\Inetpub\wwwroot. Se tomará el directorio de publicación indicado si está seleccionada la opción de Un directorio en este equipo, también podremos indicar que el directorio de publicación se encuentra en un recurso compartido de red (Un directorio ubicado en otro equipo) o bien podremos redirigir a todos nuestros clientes a una URL determinada.

También se pueden configurar una serie de parámetros para indicar si queremos registrar la actividad del sitio Web, indizar su contenido, y el tipo de acceso que va a tener el usuario

Para que el usuario pueda ver el listado de directorios que se corresponde con nuestro sitio Web deberemos activar la casilla Permitir examinar directorios, de esta forma si el usuario escribe el nombre de dominio en el que se encuentra nuestro sitio Web sin especificar ninguna página y si el documento predeterminado no está activado en el directorio especificado, aparecerá el listado de directorios en forma de una lista de vínculos de hipertexto correspondientes a los archivos y subdirectorios del directorio virtual actual. Está opción suele estar desactivada, ya que de esta forma revelaríamos a nuestros usuarios la estructura exacta de nuestro sitio Web, algo que puede llegar a ser peligroso desde el punto de vista de la seguridad.

Si queremos registrar el acceso al directorio raíz debemos activar la casilla Registrar visitas, se utilizará el formato de log que se encuentre definido para el sitio Web.

Podemos observar, también, que tenemos una opción llamada Permisos de ejecución, pero estos permisos son relativos a la ejecución de aplicaciones ASP/ASP .NET. Para no permitir la ejecución de programas ni secuencias de comandos en el directorio seleccionaremos la opción Ninguno. Si queremos que se ejecuten secuencias de scripts asignadas a un motor de secuencias de comandos en este directorio seleccionaremos la opción Sólo secuencias de comandos, este sería el caso para las páginas ASP y ASP .NET. Y si queremos ejecutar cualquier aplicación dentro de este directorio seleccionaremos la opción Sec. comandos y ejecutables.

Documentos

En esta hoja podremos indicar la página por defecto del sitio Web. Cuando un usuario no especifique en el navegador ningún fichero en concreto de nuestro sitio Web se le mostrará esta página por defecto. Podemos indicar diferentes documentos predeterminados dentro de la lista que se puede apreciar en la Figura 157, el orden en el que se indican los documentos es significativo, el servidor Web devolverá el primero que encuentre. El orden de prioridad de los documentos lo podemos modificar utilizando las flechas. El documento predeterminado podrá ser tanto una página HTML como una página ASP o una página ASP .NET.

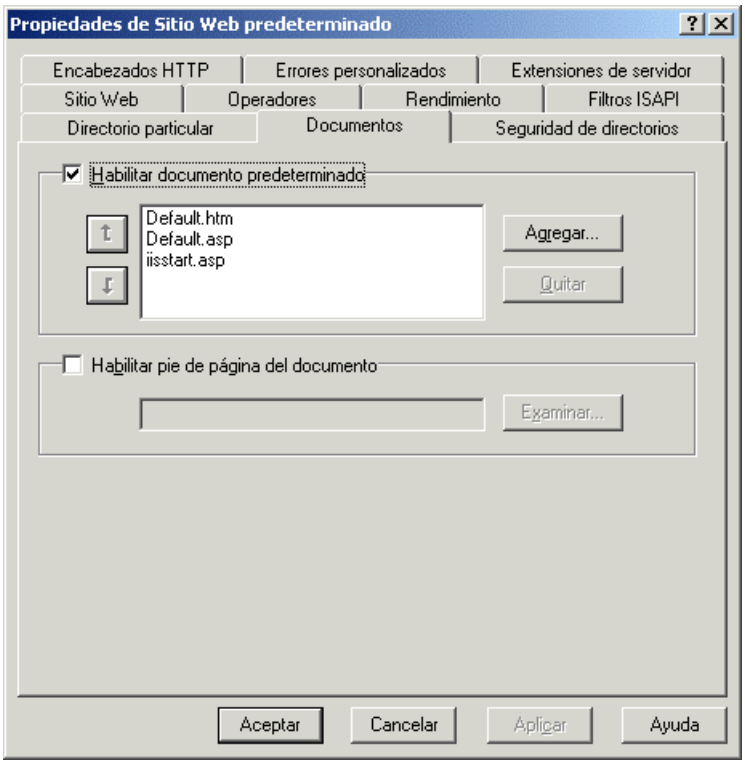


Figura 157. Hoja de propiedades Documentos

Mediante la opción Habilitar pie de página del documento podremos configurar el servidor Web para que inserte de manera automática un fichero en formato HTML en la parte inferior de todos los documentos del sitio Web.

Operadores

En esta hoja podemos definir qué cuentas de usuarios del dominio que pertenecen a la figura especial del operador Web. Un operador es una cuenta de usuario de Windows 2000 que tiene permisos para alterar la dinámica del servicio Web, es decir, iniciar, detener o pausar el sitio Web.

Para añadir una cuenta para que actúe de operador del sitio Web pulsaremos únicamente el botón Agregar y seleccionaremos la cuenta de usuario de Windows correspondiente, podremos indicar tanto cuentas como grupos de cuentas de usuarios. Para eliminar una cuenta de los operadores se selecciona de la lista y se pulsa el botón Quitar. La hoja de propiedades Operadores la podemos observar en la Figura 158.

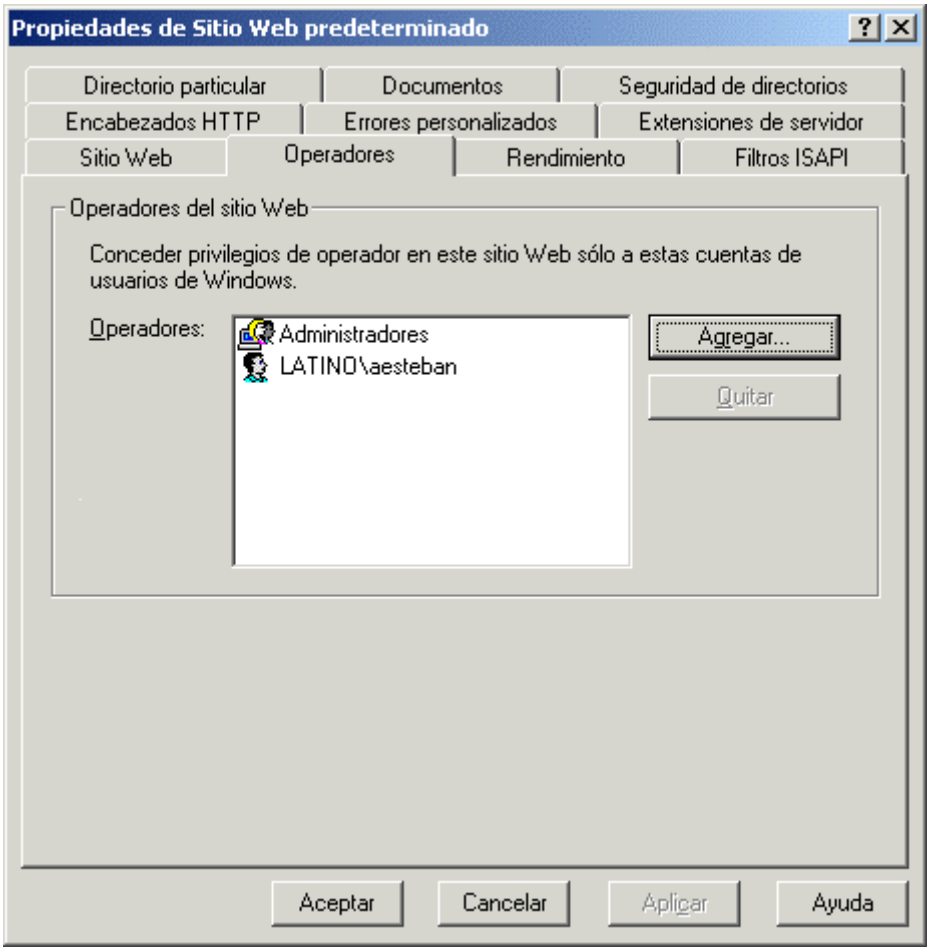


Figura 158. Operadores del sitio Web

Por defecto el grupo Administradores de dominio de Windows se encuentra dentro de los operadores del sitio Web.

Errores personalizados

IIS 5.0 permite personalizar los mensajes de error del protocolo HTTP que se envían a los clientes. A través de la hoja de propiedades Errores personalizados podemos indicar los diferentes mensajes de error que se enviarán al cliente. Estos mensajes de error personalizados pueden tener forma de fichero o de dirección URL.

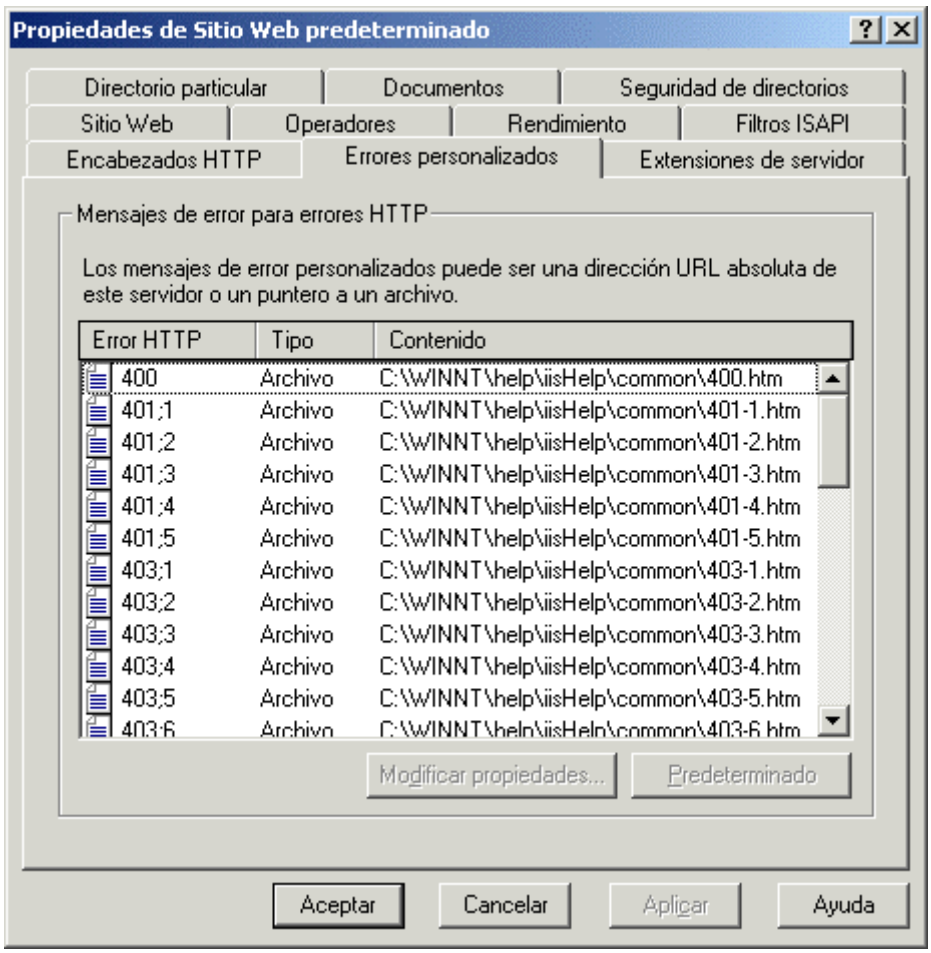


Figura 159. Errores personalizados

Rendimiento

Para ajustar el rendimiento de nuestro servidor Web podemos utilizar la hoja de propiedades Rendimiento. En esta hoja podemos configurar el ajuste del rendimiento de nuestro servidor atendiendo al número de accesos que se espera por día, disponemos de tres opciones: menos de 10.000, menos de 100.000 y más de 100.000. De esta forma el servidor podrá ajustar los recursos de una manera más apropiada.

También podemos limitar el ancho de banda a los KB/s que deseemos. Esta opción nos permite racionalizar el ancho de banda y repartirlo entre diversos servicios de Internet (sitios Web, FTP...).

Y como novedad de IIS 5.0 también se puede asignar un límite de proceso de uso de la CPU expresado en tanto por ciento, también se puede forzar a cumplir los límites establecidos.

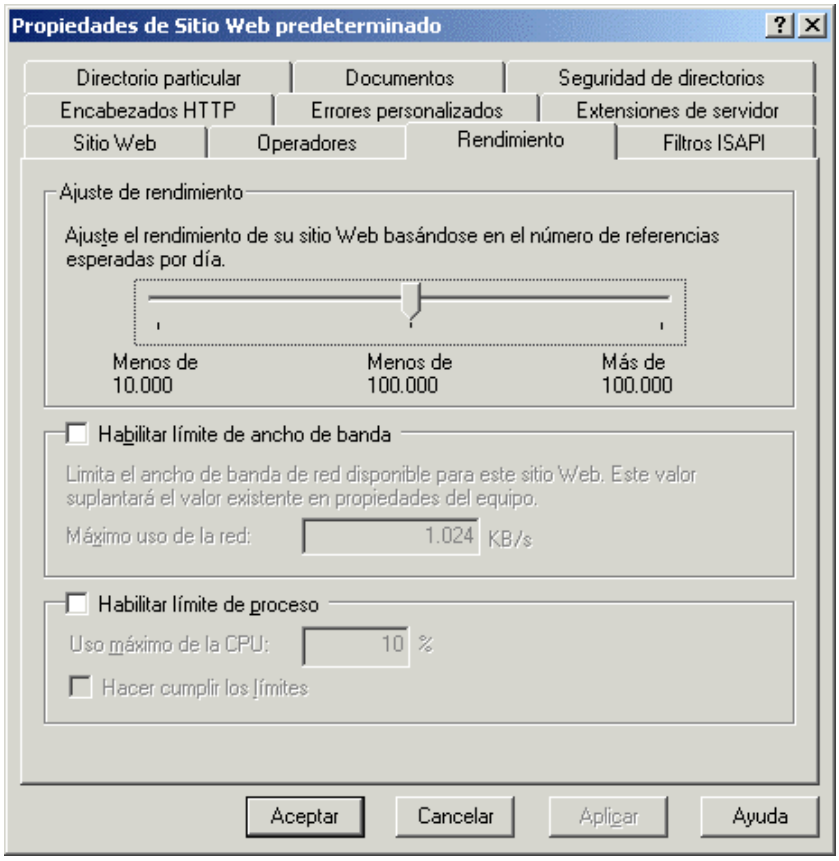


Figura 160. Rendimiento del sitio Web

Seguridad de directorios

En la hoja de propiedades etiquetada como Seguridad de directorios, configuraremos la seguridad de nuestro sitio Web. Se pueden definir distintos niveles de seguridad atendiendo a diferentes criterios, también se puede tener una combinación de la seguridad ofrecida por IIS 5.0 y Windows 2000 a través de NTFS, no debemos olvidar que IIS se encuentra integrado con el sistema operativo. Esta hoja se divide en tres partes, pasamos a comentar cada una de ellas.

En la opción Control de autenticación y acceso anónimo si pulsamos el botón Modificar podremos configurar las características de acceso anónimo y autenticación del servidor Web. Para que el servidor pueda autenticar los usuarios, primero se deben crear cuentas de usuario válidas de Windows 2000 y luego asignar permisos a los ficheros a través de NTFS.

A través de la opción Restricciones de nombre de dominio y dirección IP, podremos permitir o denegar el acceso a diferentes equipos o grupos de equipos atendiendo a sus direcciones IP.

Y en la última opción Comunicaciones seguras crearemos un par de claves SSL (Secure Sockets Layer) y la petición de un certificado de servidor.

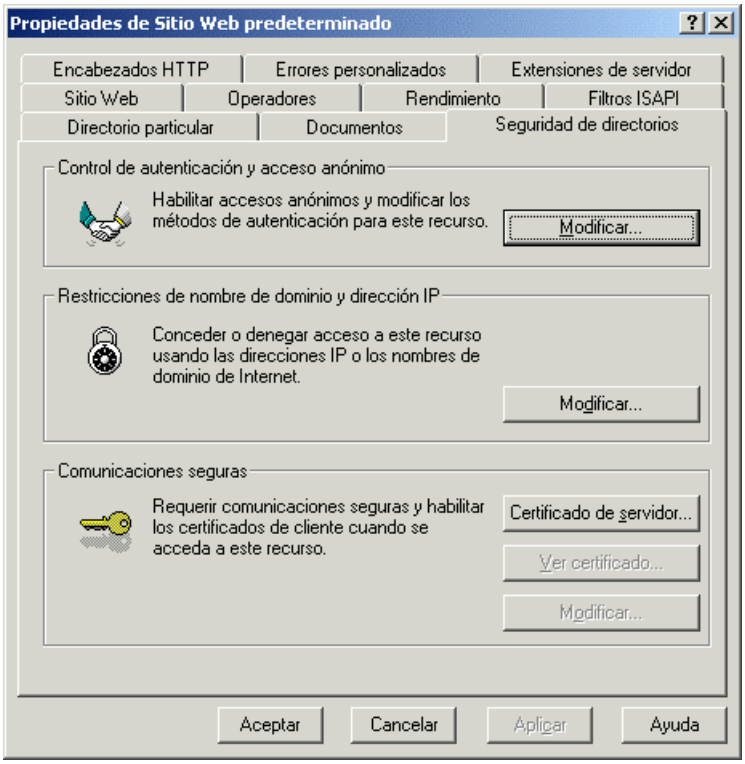


Figura 161. Seguridad de directorios

Filtros ISAPI

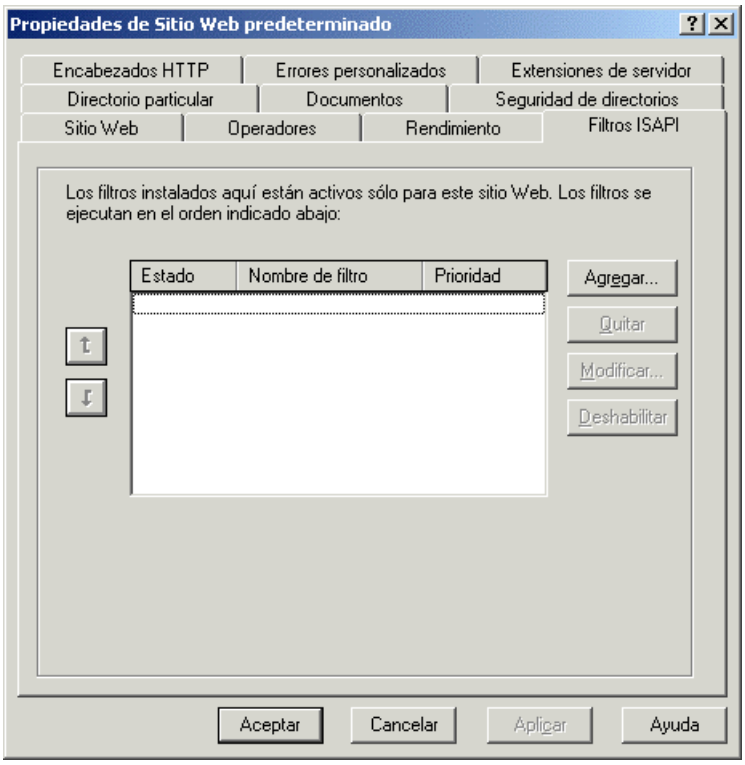


Figura 162. Filtros ISAPI

Para instalar filtros ISAPI (Internet Server Application Program Interface) utilizaremos la hoja de propiedades Filtros ISAPI. Un filtro ISAPI es un programa que responde a sucesos durante el procesamiento de una petición HTTP y está cargado siempre en la memoria del sitio Web. Físicamente son ficheros .DLL. Por defecto no hay ningún filtro ISAPI instalado.

Encabezados HTTP

En esta hoja de propiedades vamos a poder establecer los valores de las cabeceras HTTP que se envíen a nuestros clientes (navegadores Web).

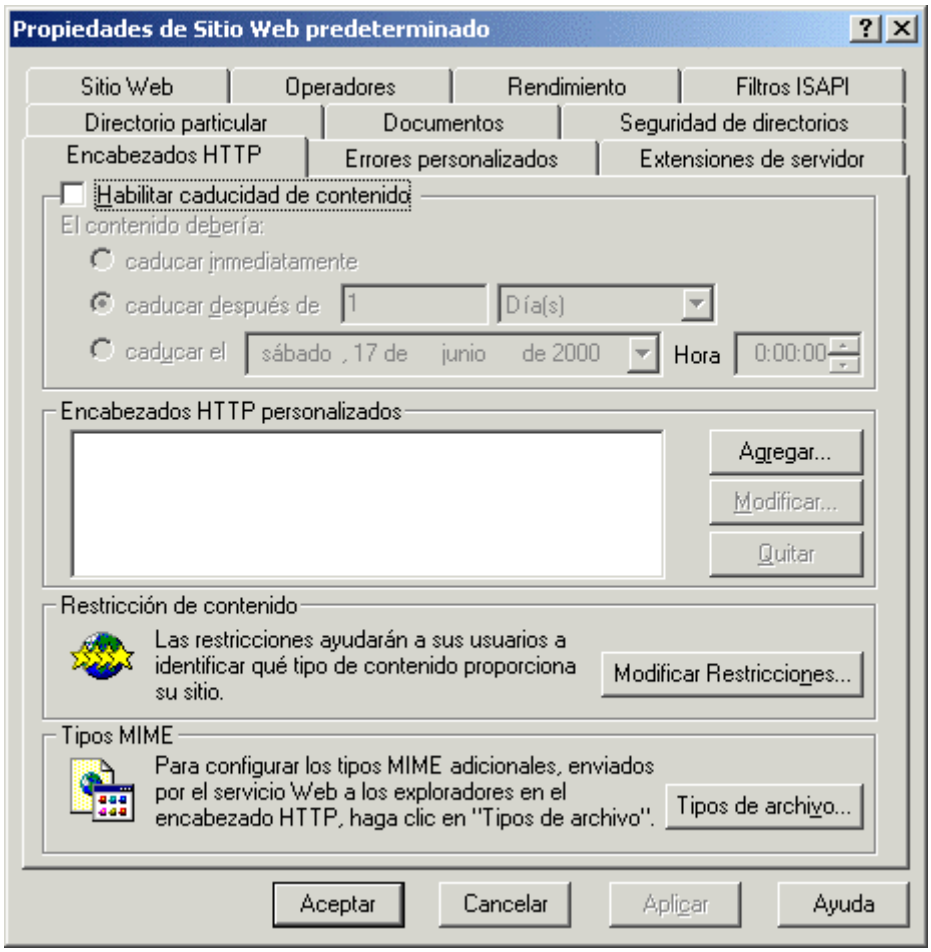


Figura 163. Encabezados http

Si activamos la casilla de verificación Habilitar caducidad de contenido nos permitirá establecer el momento en el que la información de la página se considera no válida u obsoleta (ha caducado), el navegador Web comparará la fecha del sistema con la de la página que le llega y determinará si se visualiza la página almacenada en la caché o bien pedirá una actualización de la misma.

En la propiedad *Encabezados HTTP personalizados* podremos enviar un encabezado HTTP personalizado del servidor Web al navegador del equipo cliente.

Las páginas Web pueden incorporar cabeceras con información que identifique sus contenidos desde un punto de vista moral, la configuración de estas cabeceras la realizaremos a través del apartado Restricción de contenido. De esta manera los usuarios pueden filtrar las páginas en función de ese

contenido. IIS utiliza un método desarrollado por el organismo RSAC (Recreational Software Advisory Council) que clasifica los contenidos atendiendo a diversos grados de violencia, pornografía y lenguaje ofensivo. Para establecer estas restricciones de contenido se debe pulsar el botón Modificar Restricciones.

En la opción Tipos MIME (Multipurpose Internet Mail Extensions) podemos determinar los tipos de archivos que se enviarán al cliente Web.

Extensiones de servidor

Desde esta hoja de propiedades, que es novedad en IIS 5.0, podemos configurar algunos de los parámetros relativos a las extensiones de servidor de FrontPage.

Las extensiones de FrontPage nos van a servir para conectarnos al sitio Web con clientes como el Explorador de FrontPage, el Editor de FrontPage y la herramienta de desarrollo Visual InterDev, para desarrollar páginas HTML o páginas ASP, y manipular los sitios Web.

En la versión 5.0 de IIS estas extensiones se encuentran completamente integradas con el Administrador de servicios de Internet. Las extensiones de servidor de FrontPage se instalan conjuntamente con IIS 5.0.

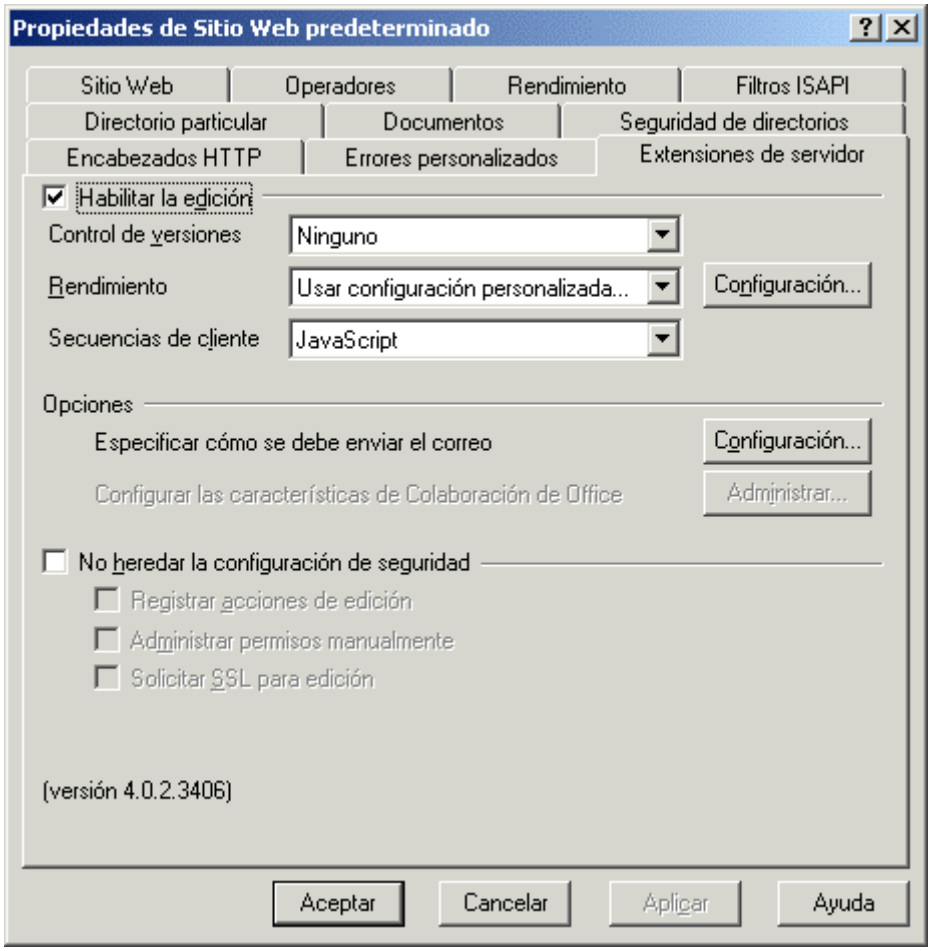


Figura 164. Extensiones de servidor de FrontPage

En el caso de no estar instaladas en el sitio Web predeterminado no parecerá esta hoja de propiedades. Para instalar las extensiones de servidor acudiremos al sitio Web predeterminado y pulsando con el botón derecho del ratón seleccionaremos la opción de menú Todas las tareas| Configurar las Extensiones de servidor. Esta opción lanzará un wizard que nos guiará en el proceso de instalación que resulta sencillo.

En la figura 18 podemos ver la hoja de propiedades que nos permite configurar parte de las extensiones de servidor de FrontPage. Para permitir que los autores modifiquen contenidos del sitio Web deberemos activar la opción Habilitar la edición, es recomendable desactivar esta opción una vez que el sitio Web se haya finalizado y ya se encuentre en producción. También podemos indicar que se lleve un control de versiones sobre las diferentes modificaciones realizadas sobre las páginas por los autores, el tipo de script de cliente por defecto y también opciones relativas al rendimiento del sitio Web.

Otros parámetros de esta hoja de propiedades permiten especificar la configuración del correo del sitio Web, que se utilizará para algunos formularios, para indicar un dirección de contacto etc. Otro parámetro nos permite especificar si un subWeb va a heredar o no la configuración de seguridad del Web padre. Dentro de estas opciones de seguridad podemos especificar si queremos registrar las acciones realizadas por los autores, si es necesario o no la conexión segura a través de SSL para proteger los datos enviados y si se va a permitir o no administrar manualmente los permisos a través de las listas de acceso (ACL).

Además de configurar las extensiones de FrontPage desde esta hoja de propiedades podemos manipularlas pulsando son el botón derecho del ratón sobre el directorio o sitio Web deseado y en la opción Todas las tareas vemos que aparecen acciones relacionadas con la administración de las extensiones de FrontPage (Configurar extensiones de servidor, Comprobar extensiones de servidor, etc.), también aparecen algunas opciones relacionadas con las extensiones dentro del menú Nuevo (Administrador de extensiones de servidor, Web de extensiones de servidor, etc.).

La aplicación web

Este apartado sirve como enlace al siguiente capítulo ya que en este apartado veremos como se puede crear una aplicación ASP .NET desde IIS.

Podemos definir una aplicación Web como un conjunto de páginas Web en combinación con scripts de servidor que permiten realizar una serie de tareas como si de una aplicación convencional cliente/servidor se tratase.

Desde el punto de vista físico y de administración del servidor Web, una aplicación es un conjunto de ficheros que se ejecutan dentro de un conjunto definido de directorios del sitio Web, es decir, una aplicación se corresponde con una estructura de directorios del sitio Web, y desde la consola de administración de IIS 5.0 deberemos indicar el punto de arranque de la aplicación y hasta dónde se extiende.

Se puede decir que una aplicación ASP/ASP .NET es un tipo de aplicación Web determinado.

Para distinguir los directorios de arranque de una aplicación el Administrador de servicios de Internet los muestra con un icono especial, se representan como una pequeña bola con un aspa metida en una caja, esto se puede observar en la Figura 165, no se deben confundir con los directorios virtuales, que se representan con una pequeña bola azul.

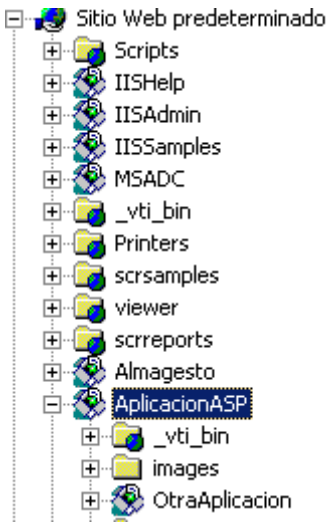


Figura 165. Aplicaciones Web

Se puede observar en la figura anterior que existen varias aplicaciones Web diferentes, de esta forma todos los ficheros del directorio AplicacionASP forman parte de una misma aplicación menos los ficheros que se encuentran en el directorio OtraAplicacion que forman parte de una aplicación Web distinta.

Podemos anidar tantas aplicaciones Web como deseemos teniendo en cuenta que debe existir una lógica a la hora de crear distintas aplicaciones Web, aunque no es recomendable anidar aplicaciones Web.

Las aplicaciones ASP/ASP .NET ofrecen un estado de aplicación, de esta forma una aplicación puede compartir información entre los diferentes ficheros que se encuentren incluidos en la aplicación.

Desde el punto de vista de las páginas ASP/ASP .NET podemos distinguir dos tipos de contextos en una aplicación ASP/ASP .NET, lo que se denomina estado de la aplicación y estado de la sesión. No es posible intercambiar información entre aplicaciones distintas, cada aplicación tiene un estado de aplicación único con todos sus estados de sesión que se corresponden con los clientes que en ese momento se encuentran conectados a la aplicación Web. Tampoco es posible intercambiar información a nivel de sesión aunque las sesiones se encuentren en la misma aplicación, el intercambio de información se encuentra restringido a nivel del estado de la aplicación y dentro de una misma aplicación.

Para crear el punto de inicio de una aplicación Web, o crear una aplicación ASP/ASP .NET, desde nuestro punto de vista es lo mismo, debemos seleccionar en el Administrador de servicios de Internet el directorio que va a hacer las veces de punto de inicio, y en la ficha de propiedades llamada Directorio pulsaremos el botón Crear dentro del apartado Configuración de aplicación. De forma automática se creará la aplicación y aparecerán dos nuevos botones Quitar y Configurar. El primero de ellos se utilizará para eliminar la aplicación creada y el segundo para configurarla. También podremos indicar un nombre para la aplicación, aunque este nombre es únicamente descriptivo.

Esta hoja de propiedades desde la que creamos la aplicación Web se puede ver en la Figura 166.

La creación de aplicaciones Web e indicación de directorios de inicio de las mismas, se hace de la misma forma para aplicaciones basadas en ASP .NET, que para aplicaciones basadas en ASP. Aunque, como veremos en un futuro capítulo dedicado al a configuración de aplicaciones ASP .NET, ASP .NET no hace uso de la configuración de la aplicación indicado dentro de IIS, pero si que resulta necesario crear el inicio de aplicación Web.

ASP .NET ofrece un sistema de configuración de aplicaciones basados en ficheros XML, la aplicación ASP .NET poseerá un fichero de configuración denominado Web.config.

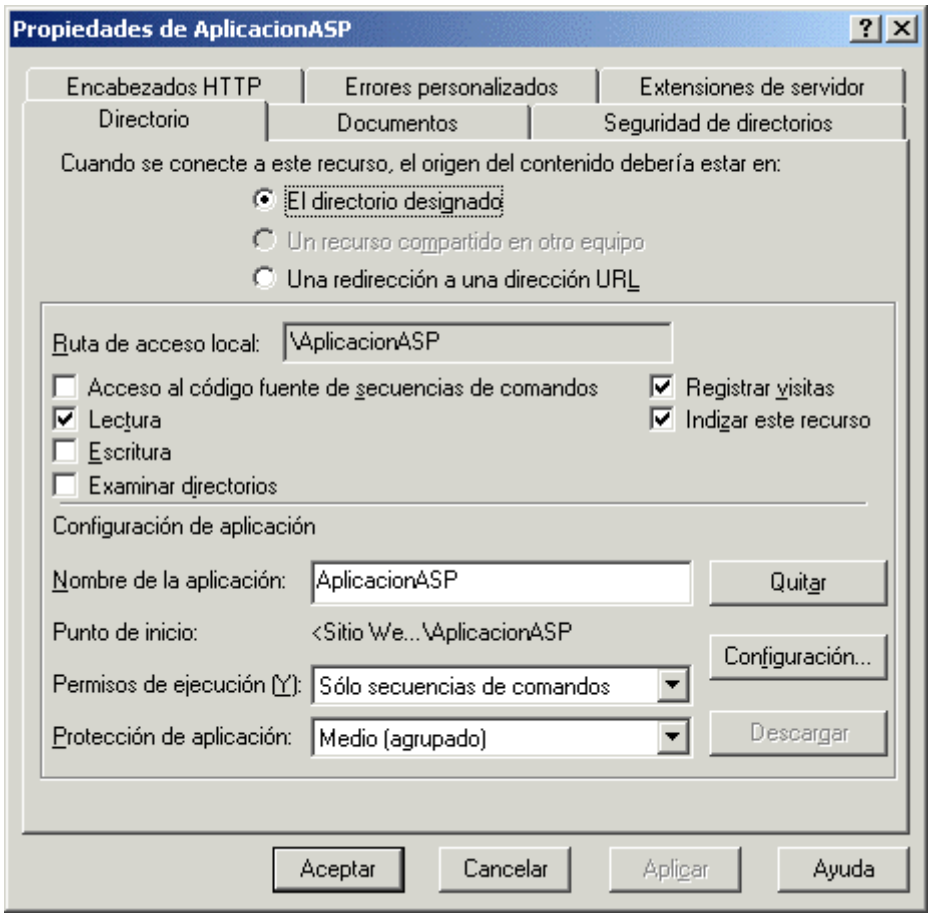


Figura 166. Creando una aplicación

Para las aplicaciones Web basadas en ASP 3.0 el servidor IIS 5.0 permitía la posibilidad de definir tres niveles de protección distintos para dichas aplicaciones, pero esta configuración tampoco va a tener ningún efecto sobre las aplicaciones ASP .NET, ya que ASP .NET se ejecutará en su propio proceso, que se encuentra separado de el servidor IIS.

Debido a que nuestro tema principal es ASP .NET no vamos a entrar en demasiado detalle en los modos de protección de la aplicación, únicamente los vamos a comentar a título informativo en la Tabla 11.

Protección	**Descripción**
Baja (proceso IIS)	Las aplicaciones ASP se ejecutan todas en el mismo espacio de memoria que el servidor Web IIS 5.0. Si una aplicación ASP falla afectará a todo el servidor Web.
Media (agrupado)	Esta es la protección por defecto, todas las aplicaciones ASP se ejecutan agrupadas en un espacio de memoria distinto que el del servidor Web. Si se produce un fallo en una aplicación ASP no afecta al servidor Web, pero sí a resto de las aplicaciones ASP.

Alta (aislado)	Cada aplicación ASP se ejecuta en un espacio de memoria distinto. De esta forma si una aplicación falla no afectará al resto de las aplicaciones ASP ni tampoco al servidor Web, ya que se ejecuta en su propio espacio de memoria.

Tabla 11

Una vez creada la aplicación Web, si pulsamos sobre el botón Configuración aparecerán tres hojas de propiedades que nos permitirán configurar nuestra aplicación Web. Al igual que sucedía con la protección de la aplicación, la configuración de la aplicación no es válida para ASP .NET, ya que ASP .NET va a utilizar su propio sistema de configuración. La única excepción es la asignación de extensiones a la aplicación, por lo tanto no vamos a entrar en más detalles en la configuración de aplicaciones Web, únicamente comentaremos la pestaña de Asignaciones para la aplicación (Figura 167), en el capítulo correspondiente veremos en detalle la configuración de las aplicaciones ASP .NET.

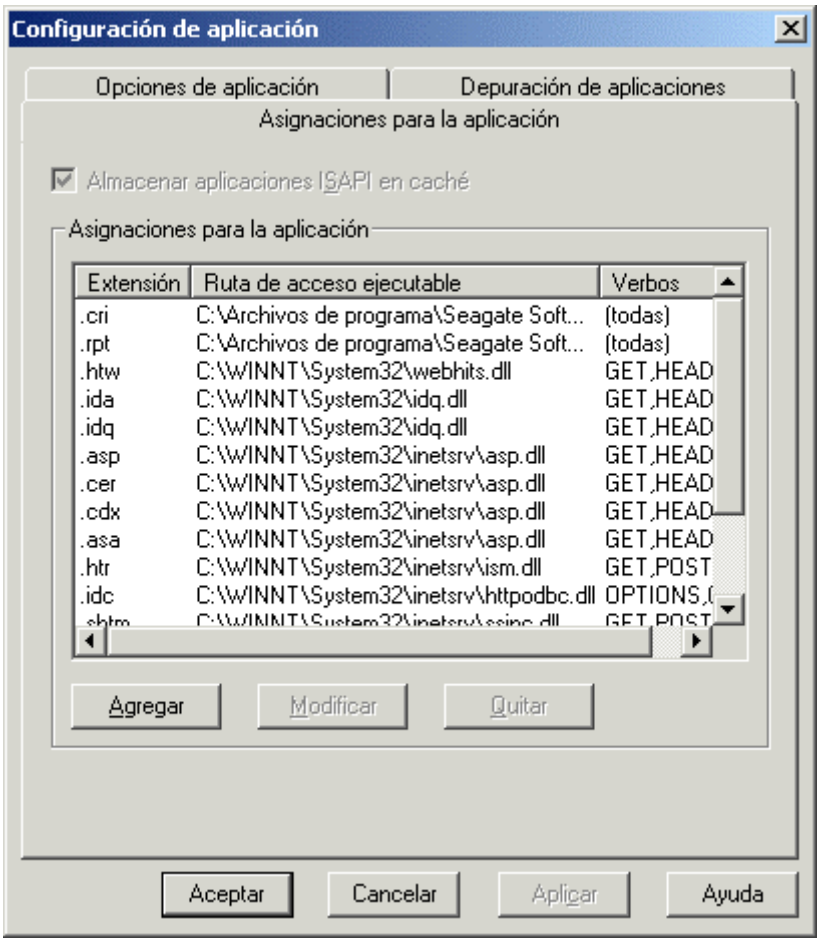


Figura 167. Asignación de aplicaciones

Para agregar una nueva asignación de una extensión a un ejecutable en el servidor Web debemos pulsar el botón Agregar, aparecerá un nuevo diálogo en el que deberemos indicar varios parámetros. También podemos quitar y modificar asignaciones existentes, en la Figura 168 se puede ver editada una asignación de los ficheros ASP.

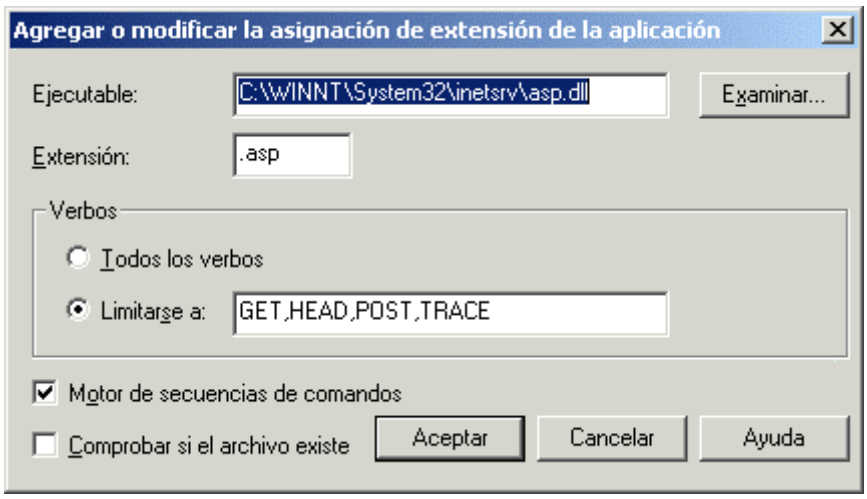


Figura 168. Agregando una asignación de extensión

En la caja de texto Ejecutable debemos indicar la ruta de acceso al programa que procesará el fichero con la extensión correspondiente, en el ejemplo que aparece en la imagen indicamos una DLL (ASP.dll) que se corresponde con el intérprete de ASP y que nos va a permitir ejecutar secuencias de comandos ASP. En la caja de texto Extensión indicaremos la extensión que queremos asignar al ejecutable, en este caso es la extensión .asp.

En el epígrafe verbos podemos indicar los métodos del protocolo HTTP que no deben transmitirse a una aplicación debemos indicarlo en la caja de texto Limitarse a, por ejemplo, podremos prohibir los métodos PUT y DELETE del protocolo HTTP, al indicar varios verbos debemos separarlos mediante comas.

Para indicar al servidor Web que compruebe la existencia del fichero solicitado por el cliente y asegurarse de que el usuario que ha realizado la petición tiene permiso de acceso a ese fichero se debe activar la casilla de verificación Comprobar si el archivo existe.

Para permitir el procesamiento de ficheros con la extensión indicada el directorio debe tener permisos que permitan la ejecución de secuencias de comandos y se debe activar la casilla de verificación Motor de secuencias de comandos.

Además de realizar esta asignación de la extensión dentro de IIS también debemos realizar una configuración adicional en el fichero de configuración Web.config, esta configuración la veremos más adelante.

En el siguiente capítulo seguiremos con un tema relacionado con el de este apartado, trataremos en profundidad las aplicaciones ASP .NET, comentaremos las distintas formas que tenemos de almacenar el estado dentro de una aplicación, haciendo uso de los objetos Application, Session y Cache. También veremos los eventos de la aplicación y el fichero especial GLOBAL.ASAX, para aquellos lectores conocedores de versiones anteriores de ASP les adelanto que este fichero tiene una funcionalidad similar al antiguo fichero GLOBAL.ASA.

18

Aplicaciones ASP .NET

Introducción

En el capítulo anterior ya comentábamos el concepto de aplicación Web (o aplicación ASP .NET), en este capítulo vamos a retomar el concepto y vamos a comentar aspectos adicionales sobre las aplicaciones ASP .NET, también veíamos anteriormente como podíamos crear una aplicación ASP .NET desde el servidor Web IIS 5.0.

En versiones anteriores de ASP ya existía el concepto de aplicación Web y se definía como un conjunto de ficheros .ASP y el fichero de la aplicación GLOBAL.ASA, ASP .NET amplía este concepto, ya que las aplicaciones ASP .NET se van a encontrar compuestas de diferentes tipos de recursos como pueden ser páginas ASP .NET, componentes .NET, Web Services, controles de usuario, controles de servidor, ficheros de configuración, el fichero GLOBAL.ASAX, etc.

Cada aplicación ASP .NET dentro de un mismo servidor Web se ejecutará en su propio dominio de aplicación del .NET Framework. Estos dominios se encuentran aislados de forma segura, de esta forma no existirán conflictos en nombres de clases, no se podrá compartir ni intercambiar información entre distintas aplicaciones ASP .NET.

Una aplicación ASP .NET se puede ver como un proceso, cuando este proceso falla no hace fallar al proceso del servidor Web, sí un fallo en una aplicación ASP .NET no tiene porque repercutir en otras aplicaciones ASP .NET ejecutándose en el mismo servidor Web.

Elementos básicos de una aplicación ASP .NET

Además de todos los recursos que comentábamos en el apartado anterior que puede contener una aplicación ASP .NET, como pueden ser las páginas ASP .NET y componentes .NET, existen otros recursos de notable interés que merece la pena destacar y que son los siguientes:

- Directorio bin: este directorio se encuentra inmediatamente debajo del directorio raíz de la aplicación, y es utilizado para contener los assemblies utilizados en la aplicación actual. En el apartado correspondiente ampliaremos la definición de este directorio de la aplicación.

- Fichero GLOBAL.ASAX: este fichero es el reemplazo que hace ASP .NET del antiguo fichero GLOBAL.ASA de las versiones anteriores de ASP. Este fichero nos va a permitir tratar eventos a nivel de aplicación, así como a crear o inicializar variables a nivel de aplicación. No es obligatorio que exista este fichero en la aplicación, aunque Visual Sutido .NET lo crea de forma automática al crear una aplicación ASP .NET. Este recurso también tiene un apartado reservado en este capítulo.

- El fichero Web.config: este fichero en formato XML va a permitir indicar la configuración de nuestra aplicación ASP .NET. En el siguiente capítulo veremos en detalle los distintos parámetros de la aplicación que podemos configurar, también trataremos el formato que presenta este fichero de configuración.

Como se puede comprobar estos recursos de la aplicación ASP .NET ya los hemos ido mencionando en algunas ocasiones a lo largo del presente texto.

A continuación vamos a pasar a comentar en profundidad el directorio bin de la aplicación.

El directorio BIN de la aplicación

Si creamos un aplicación ASP .NET desde Visual Studio .NET de forma automática nos creará el directorio bin de la aplicación, pero si no es así deberemos crearlo nosotros mismos. Este directorio no debe tener ningún tipo de permiso de ejecución ni tampoco de lectura, ya que lo va a utilizar internamente el CLR para utilizar los assemblies presentes en dicho directorio.

En la Figura 169 se ofrece la hoja de propiedades que presenta el directorio bin dentro de los Servicios de Internet Information Server. Con estos permisos se evita que un cliente pueda acceder al directorio y descargar los assemblies contenidos en el mismo.

Para registrar un componente en ASP .NET debemos situarlo en un directorio determinado de la aplicación ASP .NET, los componentes se van a distribuir de una manera muy sencilla, simplemente se debe copiar al directorio BIN de la aplicación, como ya hemos comentado, este directorio debe encontrarse inmediatamente debajo del directorio raíz de la aplicación ASP .NET.

A continuación se comentan las ventajas de almacenar los componentes en el directorio BIN:

- No es necesario registrar el componente en el servidor Web. No es necesario el registro de componentes para hacerlos disponibles para una aplicación ASP .NET. Como ya hemos comentado los componentes se pueden distribuir copiando la DLL del componente (assembly) al directorio BIN de la aplicación correspondiente, esta operación puede ser realizada a través del protocolo FTP. En este punto cabe hacer una aclaración sobre el término DLL y assembly o ensamblado. Podemos establecer una analogía entre un ensamblado y una DLL, ya que ambos contienen clases, que se exponen a otras aplicaciones. Por dicho motivo, a un

ensamblado también se le da el nombre de *DLL lógica*; el término *DLL* se emplea porque tiene un comportamiento similar al de las DLL's tradicionales, y el término *lógica* porque un ensamblado es un concepto abstracto, ya que se trata de una lista de ficheros que se referencian en tiempo de ejecución, pero que no se compilan para producir un fichero físico, a diferencia de lo que ocurre con las DLL's tradicionales.

- No es necesario reiniciar el servidor Web. Cuando cambiamos un componente reemplazando la antigua DLL del mismo por una nueva versión únicamente tenemos que sobrescribir el fichero y de forma automática la aplicación o aplicaciones ASP .NET que estén utilizando el componente comenzará a hacer uso de la nueva versión modificada del componente .NET. No es necesario reiniciar el servidor Web, ni el servicio Web ni descargar de memoria las aplicaciones ASP .NET que hacen uso del componente.

- No existen conflictos de espacies de nombre. Cada componente cargado desde el directorio BIN se encuentra limitado al ámbito de la aplicación que se está ejecutando. Esto quiere decir que muchas aplicaciones pueden potencialmente utilizar diferentes componentes con el mismo nombre de NameSpace o de clase, sin ocasionarse por ello ningún tipo de conflicto.

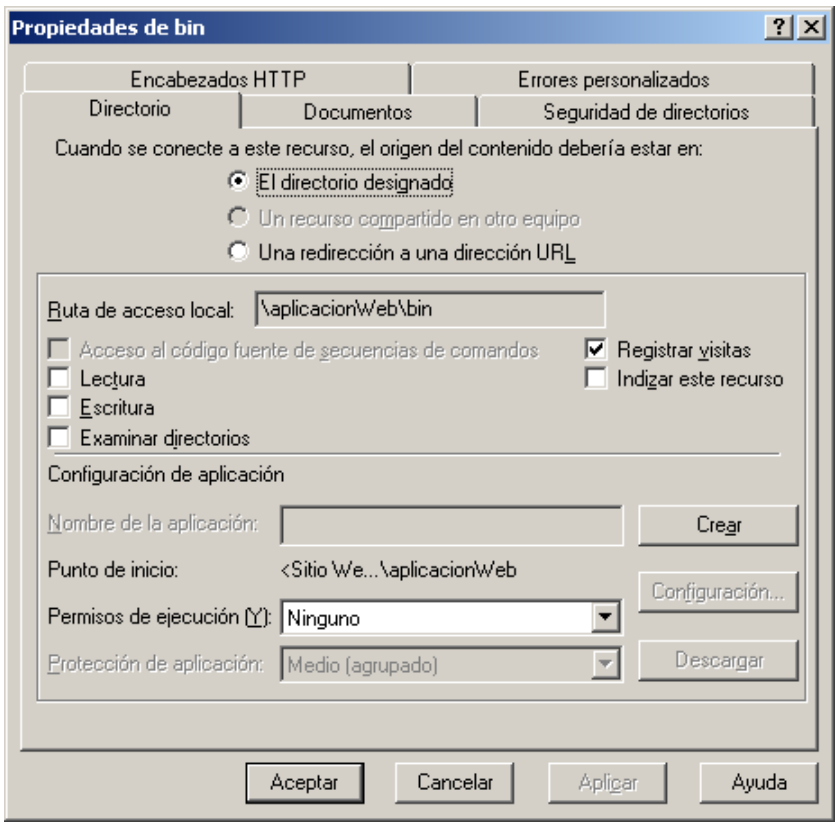


Figura 169

El fichero GLOBAL.ASAX

Aquellos lectores que ya conozcan ASP habrán reconocido este fichero de la aplicación ASP .NET como la versión del fichero GLOBAL.ASA de ASP .NET, y están en lo cierto, ya que es muy similar al fichero de ASP. Este fichero nos va a permitir definir el tratamiento de eventos a nivel de la aplicación, así como la creación de objetos y variables globales a toda la aplicación.

Únicamente puede existir un fichero GLOBAL.ASAX dentro de una misma aplicación ASP .NET, este fichero va a contener código fuente que constituye parte de nuestra aplicación, no sucede como con el fichero Web.config, que únicamente posee información de configuración de la aplicación.

Debido a la distinta extensión que presenta este fichero respecto de su versión anterior, es posible tener en un mismo directorio raíz una aplicación ASP y una nueva aplicación ASP .NET. Microsoft se ha preocupado bastante de no interferir en aplicaciones ASP existentes, debido a esto las aplicaciones ASP y las aplicaciones ASP .NET no pueden intercambiar ningún tipo de información ni de recursos, ni siquiera a través del objeto Application, este objeto junto con los objetos Session y Cache (que son propiedades del objeto Page) se tratarán en detalle.

El fichero GLOBAL.ASAX sigue un formato similar al de las páginas ASP .NET, en el Código fuente 255 se ofrece un esquema general que suele presentar este fichero.

```
<%@ [Directiva] [atributo]=[valor]%>

<script ruant="server">
  [Métodos para el tratamiento de eventos]
</script>
```

Código fuente 255

Como mecanismo de seguridad no está permitido realizar una petición directamente al fichero GLOBAL.ASAX en el navegador. Si intentamos acceder a este fichero se nos mostrará un error como el de la Figura 170.

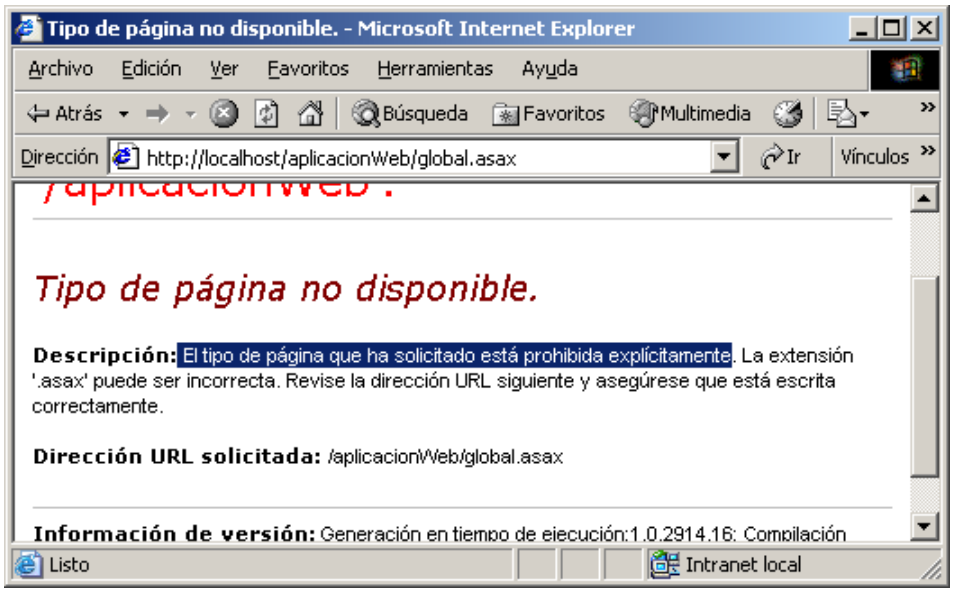


Figura 170

Por lo general un fichero GLOBAL.ASAX va a incluir directivas, eventos y código fuente del usuario. A continuación vamos a ir comentando cada uno de los elementos que podemos encontrar dentro del fichero GLOBAL.ASAX.

Directivas

De forma similar a las directivas vistas para las páginas ASP .NET, el fichero GLOBAL.ASAX posee sus propias directivas que indican a ASP .NET como se debe compilar este fichero.

El fichero GLOBAL.ASAX presenta tres directivas distintas: Application, Import y Assembly, cada una de ellas tiene sus respectivos atributos. Vamos a comentar estas directivas:

- Application: esta directiva nos permite indicar la clase de la que va a heredar el fichero GLOBAL.ASAX, y de forma adicional ofrece una opción de documentación del fichero. Para ello ofrece dos atributos:
    - Inherits: en este atributo indicaremos el nombre de la clase padre de la que va a heredar la instancia compilada del fichero GLOBAL.ASAX. Esta posibilidad será de utilidad cuando deseemos añadir nuestros propios métodos y propiedades al fichero GLOBAL.ASAX. Por defecto la clase padre del fichero GLOBAL.ASAX va a ser la clase System.Web.HttpApplication, esta clase representa una aplicación ASP .NET.
    - Description: este atributo permite indicar una descripción al fichero GLOBAL.ASAX. Este texto es descartado cuando el fichero es compilado, por lo tanto tiene valor únicamente a nivel de documentación.
- Import: esta directiva tiene el mismo significado y funcionalidad que la directiva Import de las páginas ASP .NET. Esta directiva es utilizada para importar de manera explícita un espacio con nombre (NameSpace) determinado. Al importar un NameSpace determinado tendremos acceso a todas las clases e interfaces definidos en el mismo. Se puede indicar un espacio con nombre de la plataforma .NET Framework o bien un espacio con nombre definido por nosotros mismos. Esta directiva posee un único atributo:
    - NameSpace: el valor de este atributo será el de un espacio con nombre válido. Únicamente se puede indicar un espacio con nombre por directiva Import, si deseamos importar o utilizar más de un espacio con nombre deberemos utilizar una directiva Import por cada NameSpace.
- Assembly: al igual que sucedía con la directiva anterior, esta otra directiva del fichero GLOBAL.ASAX tiene el mismo significado que la directiva Assembly de las páginas ASP .NET. .Esta directiva nos va a permitir utilizar un assembly directamente en el fichero GLOBAL:ASAX. Para indicar el assembly que deseamos utilizar, esto implica instanciar objetos de las clases contenidas en el assembly y utilizar sus propiedades y métodos como los de cualquier clase del .NET Framework, disponemos de los siguientes atributos excluyentes entre sí:
    - Name: en este atributo se indicará el nombre del assembly compilado, este nombre no va a contener ninguna extensión de fichero.
    - Src: en este caso se debe indicar el camino al fichero de código fuente que define el assembly, en este caso si se debe indicar la extensión correspondiente.

Declaración de código

Este es el segundo elemento que podemos encontrar dentro del fichero GLOBAL.ASAX. Los bloques de código se van a declarar de la misma forma que lo hacíamos en las páginas ASP .NET, haciendo uso de la etiqueta <script>.

Normalmente estos bloques de código van a contener los métodos que van a tratar los eventos a nivel de aplicación.

En el Código fuente 256 se puede ver un sencillo ejemplo de fichero GLOBAL.ASAX, que contiene una directiva @Application para realizar una descripción del fichero, y un bloque de código que va a tratar los eventos de inicio de sesión y de inicio de petición de una página. Los eventos que se producen de forma global en la aplicación y que pueden ser tratados en el fichero GLOBAL.ASAX los comentaremos en el apartado correspondiente.

```
<%@ Application Description="Ejemplo sencillo de GLOBAL.ASAX" %>

<script language="C#" runat="server">

    void Session_OnStart(Object obj, EventArgs e) {
        Response.Write("La sesión se ha iniciado...<br>");
    }

    void Application_OnBeginRequest(Object obj, EventArgs e) {
        Response.Write("Se ha iniciado la petición...<br>");
    }

</script>
```

Código fuente 256

El resultado del código anterior, al ejecutar por primera vez un usuario una página ASP .NET de la aplicación correspondiente, se puede ver en la Figura 171.

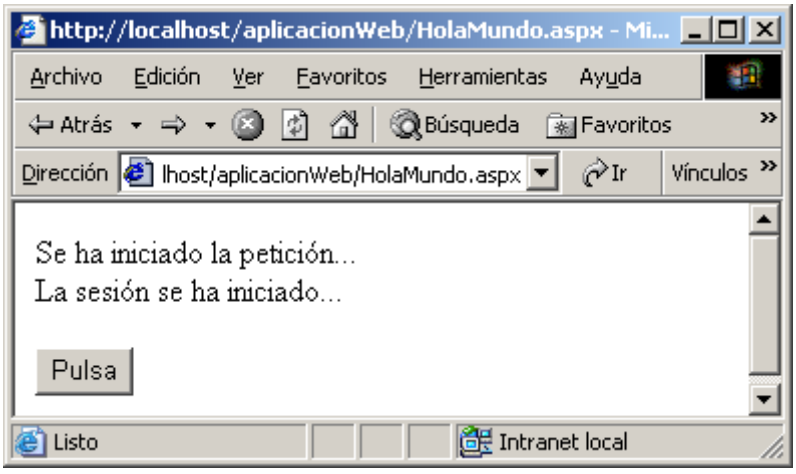


Figura 171

Inclusiones del lado del servidor

Dentro del fichero GLOBAL.ASAX podemos utilizar la instrucción include de versiones anteriores de ASP, vamos a realizar inclusiones de archivos del servidor, es decir, se puede insertar información en un fichero GLOBAL.ASAX antes de su compilación. Para ello se utiliza la sentencia especial include que ofrece la siguiente sintaxis del Código fuente 257.

```
<!--#include Virtual|File="nombrearchivo"-->
```

Código fuente 257

Virtual y File indican el tipo de ruta de acceso que se utiliza para incluir el archivo, virtual y relativa respectivamente. Por ejemplo, si el archivo a incluir, llamado incluido.inc, se encuentra en el directorio virtual /miWeb se debería escribir el Código fuente 258.

```
<!--#include Virtual="/miWeb/incluido.inc"-->
```

Código fuente 258

Pero si queremos expresar la inclusión del archivo utilizando su ruta relativa utilizaremos File, la ruta relativa comienza en el directorio en el que se encuentra la aplicación ASP .NET a la que pertenece el fichero GLOBAL .ASAX, así por ejemplo, si la aplicación se encuentra en el directorio miWeb y el fichero incluido.inc se encuentra en el directorio miWeb\subdirectorio deberemos escribir el Código fuente 259.

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

Código fuente 259

Los archivos incluidos no requieren una extensión especial. Los ficheros incluidos puede ser muy útiles, sobre todo aquellos que se incluyen utilizando la opción Virtual, ya qu enos permite acceder a un fichero que puede localizarse en un directorio común que puede ser compartido por diversas aplicaciones ASP .NET.

Si se modifica el contenido de un fichero de inclusión utilizado en un fichero GLOBAL.ASAX, la aplicación ASP .NET se reiniciará automáticamente para realizar la nueva compilación del fichero GLOBAL.ASAX con los cambios realizados en el fichero incluido.

Etiquetas de declaración de objetos

Las etiquetas de declaración de objetos nos van a permitir declarar e instanciar objetos que van a tener un ámbito de sesión o de aplicación. Las objetos con ámbito de aplicación son visibles para todos los usuarios que están utilizando la misma aplicación ASP .NET, es decir son compartidos por varios usuarios. En contraposición al ámbito de sesión, cuyas variables son para cada uno de los usuarios conectados, es decir, no se comparten y son propias de cada sesión. Podremos acceder a un objeto com ámbito de aplicación en cualquiera de las páginas ASP .NET contenidas en la aplicación ASP .NET actual.

En ASP .NET el concepto de ámbito de aplicación y sesión no ha cambiado respecto a anteriores versiones de ASP. Más adelante en este mismo capítulo veremos las distintas formas que tenemos disponibles de almacenar información en la aplicación, es lo que se va a denominar gestionar el estado de la aplicación.

Para realizar esta declaración de objetos dentro del fichero GLOBAL.ASAX se debe hacer uso de la etiqueta <object>. Esta forma de crear objetos ya se encontraba disponible en ASP para el fichero GLOBAL.ASA de las aplicaciones.

A través de la etiqueta object, además de indicar el ámbito que va a poseer el objeto correspondiente, también podemos indicar el tipo de objeto que se va a crear, ya que es posible crear instancias de assemblies .NET, o de objetos COM especificando su ProgID o su CLSID, aunque por coherencia se deberían utilizar assemblies del .NET Framework.

La sintaxis general de la etiqueta object se ofrece en el Código fuente 260.

```
<object id="identificador" runat="server" class|progid|classid="clase|progID|CLSID"
scope="session|application"/>
```

Código fuente 260

Dónde id es el identificador único que se va a utilizar para el objeto dentro de la aplicación ASP .NET; scope va a ser el ámbito que va a tener el objeto dentro de la aplicación, sus valores pueden ser Application o Session, para indicar el ámbito de aplicación o de sesión, respectivamente.

Para indicar el nombre de una clase del .NET Framework (o de un assembly) utilizaremos el atributo class, para indicar el ProgID (identificador de programa) de un componente COM utilizaremos lógicamente el atributo progid y para indicar el identificador de clase (CLSID) de un componente COM se hará uso del atributo classid.

Si deseamos crear un objeto de la clase System.Date con ámbito de aplicación y queremos que se identifique mediante el nombre fecha escribiremos el Código fuente 261.

```
<object id="fecha" runat="server" class="System.Date" scope="application"/>
```

Código fuente 261

El objeto no será instanciado hasta que no sea utilizado por primera vez dentro de la aplicación ASP .NET correspondiente.

Una vez descritos los distintos elementos que podemos encontrar en el fichero GLOBAL.ASAX de una aplicación ASP .NET, vamos a ocuparnos de las funcionalidades que ofrece este fichero global de la aplicación. En primer lugar vamos a comentar los distintos eventos de la aplicación que podemos tratar dentro del fichero GLOBAL.ASAX, y en segundo lugar se detallarán las distintas alternativas que nos ofrece la aplicación ASP .NET para mantener los objetos o valores de variables a través de la vida de la aplicación.

Eventos de la aplicación

Los eventos de la aplicación son una serie de eventos que se lanzan de forma global a la aplicación y que van a ser tratados en los métodos correspondientes dentro del fichero GLOBAL.ASAX. Estos eventos tienen el mismo significado que los eventos que eran atrapados en el fichero GLOBAL.ASA

de las aplicaciones ASP, pero en ASP .NET se ha ampliado el número de eventos, de cuatro eventos disponibles en ASP se han pasado a dieciocho eventos.

Los métodos encargados de tratar estos eventos tienen todos un prototipo común, es decir, todos ellos presentan los mismos tipos de parámetros. El primer parámetro es un objeto de la clase Object que va a contener el objeto que ha lanzado el evento y el segundo parámetro es un objeto de la clase EventArgs que va a representar el evento que se ha producido, permitiendo recuperar detalles adicionales sobre el evento.

En el Código fuente 262 se puede observar el prototipo que poseen los métodos del fichero GLOBAL.ASAX que nos van a permitir tratar los eventos de la aplicación.

```
void MétodoQueTrataEvento(Object obj, EventArgs e) {
}
```

Código fuente 262

También es posible utilizar una versión reducida de estos prototipos (Código fuente 263), en este caso se suprimen los parámetros de los métodos. Se considera una buena práctica hacer uso de la primera alternativa.

```
void MétodoQueTrataEvento() {
}
```

Código fuente 263

A continuación vamos a describir los distintos eventos de la aplicación, estos eventos se dividen en dos categorías, los eventos que se lanzan en cada petición de una página ASP .NET y los eventos que se lanzan de forma condicional.

Eventos por petición

Los eventos por petición de la aplicación (per-request) son aquellos eventos que se lanzan durante cada petición realizada a una aplicación ASP .NET. Vamos a describir brevemente los distintos eventos que se encuentran dentro de este grupo, el orden en el que comentan los distintos evento ses el orden en el que se ejecutan:

- Application_OnBeginRequest: este evento se produce cada vez que se realiza una petición de una página ASP .NET o de un Web Service a la aplicación ASP .NET. Este evento se puede utilizar para ejecutar un código determinado antes de ejecutar una página o servicio Web.

- Application_OnAuthenticateRequest: este evento se lanza cuando ASP .NET está preparado para realizar una autenticación de la petición. ASP .NET posee una característica avanzada que permite utilizar distintos métodos de autenticación. ASP .NET puede colaborar con el servidor Web IIS para establecer mecanismos de autenticación.

- Application_OnAuthorizeRequest: este método es similar al anterior, en este caso este evento indica que la petición está lista para ser autorizada. Este evento es utilizado también el sistema de seguridad que nos ofrece ASP .NET.

- Application_OnResolveRequestCache: este evento, como su nombre indica, está relacionado con los mecanismos de caché ofrecidos por ASP .NET, concretamente este evento se lanzará cuando ASP .NET esta preparado para determinar si la petición realizada debería ser servida desde la caché.

- Application_OnAcquireRequestState: este evento es lanzado cuando ASP .NET se dispone a obtener información almacenada a nivel de sesión.

- Application_OnPreRequestHandlerExecute: el evento se lanzará antes de que el servicio encargado de tratar la petición sea llamado.

- Application_OnPostRequestHandlerExecute: este evento es le primero que se lanza una vez que se ha terminado de servir la solicitud, en ese momento el objeto Response ya posee datos para devolver al cliente.

- Application_OnRealeaseRequestState: este evento se lanza cuando se liberan los datos de la sesión.

- Application_OnUpdateRequestCache: este otro evento se lanza cuando ASP .NET actualiza la caché con la petición actual.

- Application_OnEndRequest: este evento se lanzará cuando la solicitud ha sido completada, es el último evento que se lanza antes de enviar al cliente las cabeceras de respuesta del protocolo HTTP.

En la Figura 172 se muestra un esquema que relaciona estos eventos por petición.

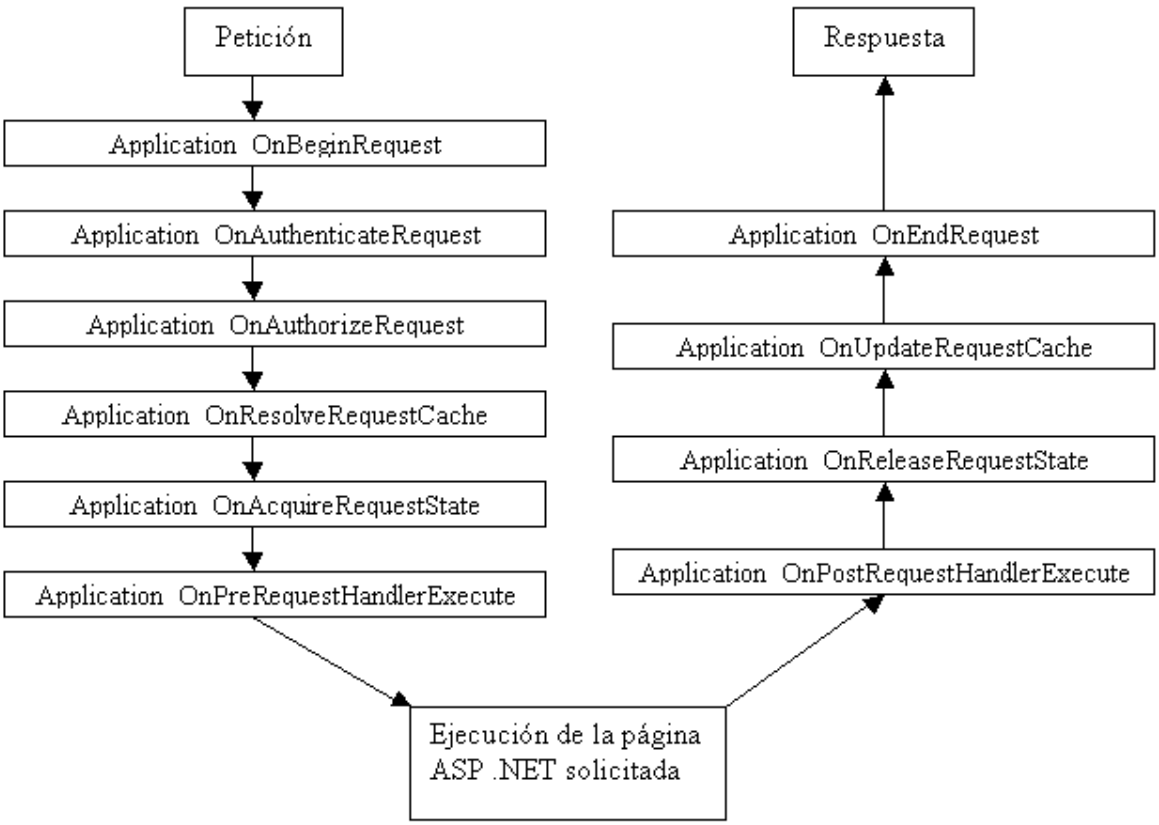


Figura 172

En este esquema se muestra que el servidor Web IIS recibe una petición que es pasada automáticamente al entorno de ejecución de ASP .NET. Los eventos de la aplicación ASP .NET se lanzan comenzando con el evento Application_OnBeginRequest. Inmediatamente antes de que sea tratada la petición por se lanza el evento Application_OnPreRequestHandlerExecute. Inmediatamente después de procesar la página se ejecuta lanza el evento Application_OnPostRequestHandlerExecute. Y finalmente antes de que la respuesta sea devuelta por el servidor IIS se lanzará el evento Application_OnEndRequest.

El fichero GLOBAL.ASAX que aparece en el Código fuente 264 nos muestra el orden de ejecución de estos eventos que se producen por cada petición realizada.

```
<%@ Application Description="Ejemplo sencillo de GLOBAL.ASAX" %>

<script language="C#" runat="server">

    void Application_OnBeginRequest(Object obj, EventArgs e) {
        Response.Write("evento <b>Application_OnBeginRequest</b><br>");
    }

       void Application_OnAuthenticateRequest(Object obj, EventArgs e) {
         Response.Write("evento <b>Application_OnAuthenticateRequest</b><br>");
    }

       void Application_OnAuthorizeRequest(Object obj, EventArgs e) {
         Response.Write("evento <b>Application_OnAuthorizeRequest</b><br>");
    }

    void Application_OnResolveRequestCache(Object obj, EventArgs e) {
        Response.Write("evento <b>Application_OnResolveRequestCache</b><br>");
    }

       void Application_OnAcquireRequestState(Object obj, EventArgs e) {
        Response.Write("evento <b>Application_OnAcquireRequestState</b><br>");
    }

       void Application_OnPreRequestHandlerExecute(Object obj, EventArgs e) {
        Response.Write("evento <b>Application_OnPreRequestHandlerExecute</b><br>");
    }

    void Application_OnPostRequestHandlerExecute(Object obj, EventArgs e) {
        Response.Write(
              "evento <b>Application_OnPostRequestHandlerExecute</b><br>");
    }

    void Application_OnReleaseRequestState(Object obj, EventArgs e) {
        Response.Write("evento <b>Application_OnReleaseRequestState</b><br>");
    }

    void Application_OnUpdateRequestCache(Object obj, EventArgs e) {
        Response.Write("evento <b>Application_OnUpdateRequestCache</b><br>");
    }

    void Application_OnEndRequest(Object obj, EventArgs e) {
        Response.Write("evento <b>Application_OnEndRequest</b><br>");
    }
</script>
```

Código fuente 264

En la Figura 173 se puede ver el resultado de ejecutar una página ASP .NET, si volvemos a ejecutar la página se ejecutarán de nuevo todos los eventos.

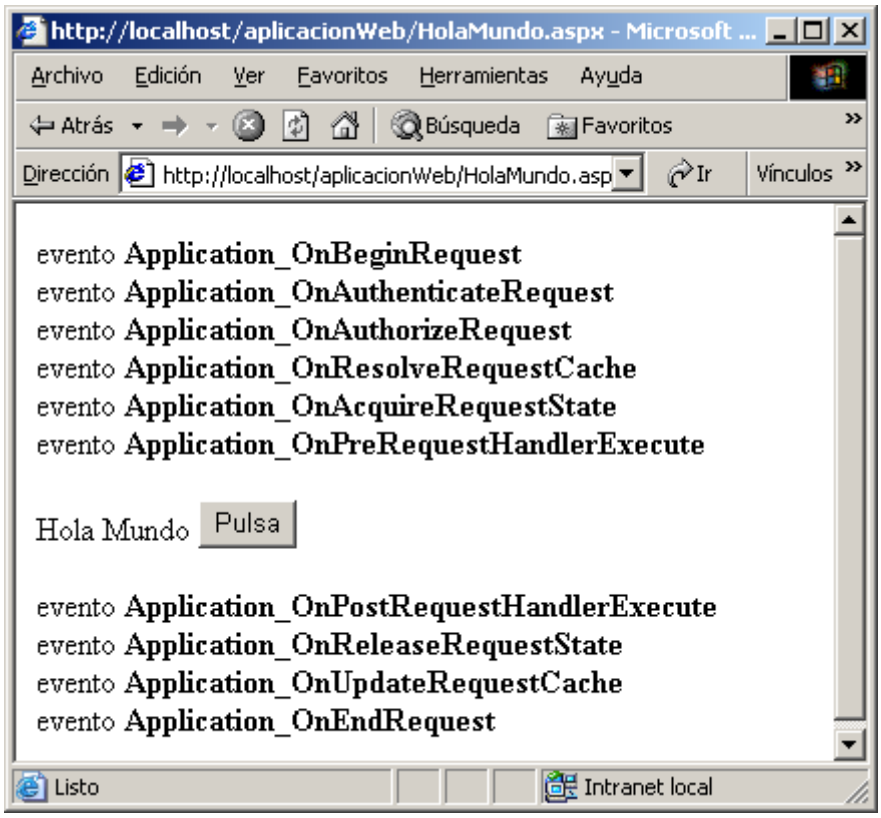


Figura 173

En el código anterior se debe realizar una aclaración, como se puede comprobar se ha utilizado el objeto Response para escribir información como resultado de la ejecución de una página, pero este objeto Response no es la propiedad Response de la clase Page (como habíamos visto en capítulos anteriores), ya que no nos encontramos dentro de una página ASP .NET, sino que nos encontramos dentro del fichero GLOBAL.ASAX.

Este objeto Response es una propiedad de la clase HttpApplication, una instancia de esta clase va a representar a la aplicación ASP .NET que se está ejecutando actualmente. La clase HttpApplication es la clase de la que hereda el fichero GLOBAL.ASAX. Debido a la estrecha relación que guarda esta clase con el fichero GLOBAL.ASAX, y a que representa a la propia aplicación ASP .NET le vvamos a dedicar un apartado en este capítulo.

Dentro del grupo de los eventos de aplicación que se producen por cada solicitud, hemos extraído dos de ellos, ya que estos no se lanzan en un orden determinado, sino que se lanzan en el momento en que es posible devolver datos al cliente.

Por defecto ASP .NET activa el mecanismo de búfer a la hora de enviar la respuesta al cliente, es decir la respuesta no se envía inmediatamente al cliente sino que ASP .NET procesará la página en el servidor antes de enviar algo al usuario. El contenido del búfer no es enviado al navegador hasta que no se haya terminado de procesar la página o hasta que los métodos End o Flush del objeto Response no son invocados. Una vez que ya se ha enviado contenido al navegador del cliente, no se puede modificar su valor.

Para activar a odesactivar el búfer de una página determinada utilizaremos el atributo Buffer de la directiva @Page. Este atributo posee los valores true/false y se utilizará para activa o desactivar el búfer de la página ASP .NET actual. El mecanismo de búfer ya existía en versiones anteriores de ASP, y en ASP .NET está estrechamente relacionado con la propiedad Response de la clase Page, de hecho se puede acceder también a la propiedad Buffer del objeto Response de la clase HttpResponse.

Cuando el mecanismo de búfer está activado estos dos eventos se producen después del evento Application_OnEndRequest, sin embargo si el búfer se encuentra desactivado, estos eventos se producirán cuando la información de respuesta vaya siendo disponible, es decir, cuando la información se haya enviado al cliente.

Vamos a describir estos dos eventos:

- Application_OnPreSendRequestHeaders: este evento es lanzado antes de que las cabeceras de respuesta del protocolo HTTP sean enviadas al cliente. Una vez que se han enviado las cabeceras, ya no podemos modificar el contenido de la respuesta.

- Application_OnPreSendRequestContent: este segundo evento es lanzado antes de que el cuerpo de la respuesta HTTP sea enviado al cliente que ha realizado la solicitud.

Si queremos utilizar estos eventos en el fichero GLOBAL.ASAX, únicamente debemos añadir las líneas del Código fuente 265, en este caso en el evento Application_OnPreSendRequestContent no podemos hacer una instrucción Response.Write, ya que el cuerpo de la respuesta ya no se encuentra disponible.

```
void Application_OnPreSendRequestHeaders(Object obj, EventArgs e) {
    Response.Write("evento <b>Application_OnPreSendRequestHeaders</b><br>");
}

void Application_OnPreSendRequestContent(Object obj, EventArgs e) {
    //Response.Write("evento <b>Application_OnPreSendRequestContent</b><br>");
}
```

Código fuente 265

A continuación vamos a pasar a comentar el segundo grupo de eventos de la aplicación, se trata los eventos condicionales, es decir, aquellos que se lanzarán o no en el proceso de una petición a ASP .NET, dependerá de unas condiciones determinadas que vamos a pasar comentar.

Eventos condicionales

Los eventos del grupo anterior eran fijos para cada solicitud, es decir, siempre se producían, pero este otro grupo de eventos se producen cuando se cumple una condición determinada en cada caso, vamos a pasar a comentar estos otros eventos, muchos de ellos ya existían en el fichero GLOBAL.ASA de ASP:

- Application_OnStart: este evento se lanza cuando se inicia la ejecución de la aplicación ASP .NET correspondiente. La ejecución de la aplicación se produce cuando entra el primer usuario en la aplicación, es decir, cuando el primer usuario carga una página ASP .NET perteneciente a una aplicación. Dentro de este evento se deberá indicar el código de inicialización de la aplicación ASP .NET. Este evento tiene el mismo significado que en

versiones anteriores de ASP. Si este evento no se ha producido ya cuando se realiza una petición, se ejecutará antes del evento Application_OnBeginRequest.

- Application_OnEnd: este evento es el evento contrario al anterior, este evento se lanzará cuando finalice la ejecución de la aplicación ASP .NET. La finalización de la ejecución de la aplicación ASP .NET se puede producir por varios motivos: cuando se apaga el servidor Web, cuando finaliza la última de las sesiones de un usuario con la aplicación o se modifica el código fuente del fichero GLOBAL.ASAX, forzando de esta forma una nueva compilación. Dentro de este evento deberá escribir el código de limpieza de la aplicación, es decir, el código necesario para liberar la memoria y recursos utilizados por la aplicación ASP .NET. Aunque la mayoría de este código no será necesario, ya que el CLR se encargará de liberar recursos de forma automática, es una buena práctica incluir ese código en el evento Application_OnEnd.

- Session_OnStart: este evento es lanzado cuando se inicia una nueva sesión dentro de una aplicación ASP .NET. El inicio de sesión se produce cuando el usuario carga la primera página de una aplicación ASP .NET. Dentro de este evento se deberán indicar las acciones a llevar a cabo antes de que se cargue la primera página de la aplicación. Este evento se suele utilizar para inicializar las variables para toda la sesión, así por ejemplo, si la aplicación ASP .NET va a manipular una serie de tablas de una base de datos, se podrá crear una conexión con la base de datos en el evento de inicio de sesión y almacenarla en el objeto Session para que esté disponible para ese usuario mientras su sesión permanezca activa. Más adelante veremos las distintas posibilidades que nos ofrece la aplicación ASP .NET de almacenar información de forma persistente a lo largo de toda la aplicación y a lo largo de toda la sesión. También se puede utilizar para cargar una serie de variables que definan el perfil del usuario que se acaba de conectar. Para se ejecute este evento siempre se debe haber lanzado previamente el evento Application_OnStart.

- Session_OnEnd: este evento es el evento contrario al evento Session_OnStart, este evento se lanza cuando finaliza una sesión de un usuario. Una sesión de un usuario finaliza cuando se haya caducado la sesión al permanecer inactiva el tiempo indicado por su propiedad Timeout, o bien, se puede forzar mediante a una llamada al método Abandon() de la clase System.Web.SessionState.HttpSessionState. Cuando el usuario cierra el navegador no se ejecuta el evento Session_OnEnd de forma inmediata, sino que se ejecutará cuando se cumpla el Timeout correspondiente, para la aplicación ASP .NET es igual que el usuario se encuentre con la página cargada en el navegador sin realizar ninguna petición o que el usuario haya cerrado su sesión del navegador, lo que se tiene en cuenta es el tiempo de inactividad de la sesión del usuario. Si finaliza la ejecución de la aplicación ASP .NET se ejecutarán todos los eventos Session_OnEnd de cada sesión de usuario antes de lanzarse el evento Application_OnEnd.

- Application_OnError: este evento se lanzará cuando se produzca un error no tratado dentro de la aplicación ASP .NET. Este evento lo vimos en detalle en el capítulo dedicado al tratamiento de errores desde ASP .NET.

- Application_OnDisposed: este evento es lanzado cuando la aplicación ASP .NET inicia su descarga de memoria, este evento se producirá antes del evento Application_OnEnd.

Para mostrar el orden de ejecución de estos eventos vamos a crear un fichero GLOBAL.ASAX como el que aparece en el Código fuente 266. En este caso se ha incluido un método dentro del GLOBAL.ASAX que nos permite escribir el evento que se ha producido en la aplicación, en este caso no se utiliza la instrucción Response.Write() debido a que el objeto Response no está disponible en algunos de estos eventos. Para este ejemplo no se ha utilizado el evento Application_OnError, ya que se comentó en detalle en capítulos anteriores.

```
<%@ Application Description="Ejemplo sencillo de GLOBAL.ASAX" %>
<%@ Import NameSpace="System.IO" %>

<script language="C#" runat="server">
    void EscribeEventoEnFichero(String texto){
       FileStream fs = new FileStream("EventosAplicacion.txt",
                                          FileMode.Append,FileAccess.Write);
       StreamWriter sw = new StreamWriter(fs);
       sw.WriteLine(texto);
       sw.Close();
       fs.Close();
    }

    void Application_OnStart(Object obj, EventArgs e) {
             EscribeEventoEnFichero("Evento Application_OnStart");
    }

    void Application_OnEnd(Object obj, EventArgs e) {
             EscribeEventoEnFichero("Evento Application_OnEnd");
    }

    void Application_OnDisposed(Object obj, EventArgs e) {
             EscribeEventoEnFichero("Evento Application_OnDisposed");
    }

       void Session_OnStart(Object obj, EventArgs e) {
             EscribeEventoEnFichero("Evento Session_OnStart");
    }

       void Session_OnEnd(Object obj, EventArgs e) {
             EscribeEventoEnFichero("Evento Session_OnEnd");
    }

</script>
```

Código fuente 266

Para que se ejecuten todos los presentes en nuestro fichero GLOBAL.ASAX tenemos que seguir los siguientes pasos: nos aseguramos que la aplicación no se esté ejecutando, a continuación cargamos una página ASP .NET, en ese momento se ejecutará primero el evento Application_OnStart y a continuación el evento Sesson_OnStart; para forzar los tres eventos que nos quedan podemos modificar el código fuente del fichero GLOBAL.ASAX, al modificar su código se fuerza a la compilación de dicho fichero de la aplicación, lo que supone antes finalizar la ejecución de la aplicación, lanzar los eventos Session_OnEnd, Application_OnDisposed y Application_OnEnd, en este mismo orden.

Si abrimos el fichero EventosAplicacion.txt (que se encontrará en el directorio c:\winnt\system32, ya que no hemos indicado ninguna ruta determinada) podremos observar que presenta el contenido del Código fuente 267.

```
Evento Application_OnStart
Evento Session_OnStart
Evento Session_OnEnd
Evento Application_OnDisposed
Evento Application_OnEnd
```

Código fuente 267

A continuación vamos a mostrar un esquema (Figura 174) que muestra a modo de resumen el orden en el que se producen los eventos condicionales de la aplicación, y también indica la circunstancia que provoca el lanzamiento de dicho evento. Los eventos Application_OnStart, Application_OnDisposed y Application_OnEnd se lanzan una única vez durante toda la vida de la aplicación ASP .NET, sin embargo los eventos Session_OnStart y Session_OnEnd se ejecutarán repetidas veces durante la vida de la aplicación, dependiendo de las sesiones que hayan iniciado los usuarios con la aplicación ASP .NET.

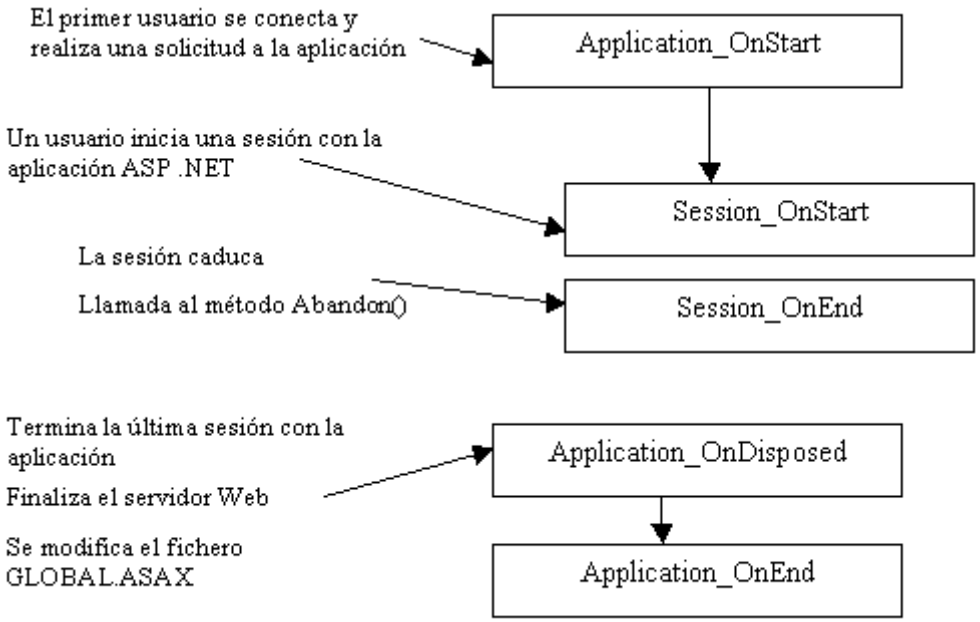


Figura 174

Con este esquema finalizamos el apartado dedicado a los eventos de la aplicación ASP .NET que podemos tratar en el fichero GLOBAL.ASAX. El siguiente apartado va a describir de forma breve la clase que representa a las aplicaciones ASP .NET, es decir, la clase System.Web.HttpApplication.

La clase HttpApplication

Como ya hemos comentado en varios puntos de esta capítulo, esta es la clase que va representar a la aplicación ASP .NET y es por defecto la clase padre utilizada para compilar el fichero GLOBAL.ASAX.

Esta clase presenta los eventos que ya hemos comentado en el apartado anterior, además de estos eventos nos ofrece el acceso a varios objetos, que se corresponden con los objetos integrados de anteriores versiones de ASP, a través de propiedades. Vamos comentar las propiedades que ofrece la clase HttpApplication:

- Application: esta propiedad nos va a ofrecer una referencia a un objeto de la clase System.Web.HttpApplicationState, este objeto nos va a permitir almacenar y acceder a información que va a ser común a toda la aplicación Web, es decir, es una información compartida por todos los clientes de una aplicación Web determinada. Esta propiedad es equivalente al objeto integrado Application de anteriores versiones de ASP.

- Request: esta propiedad ofrece una referencia a un antiguo objeto integrado del modelo de objetos de versiones anteriores de ASP, en este caso nos devuelve una instancia de un objeto de la clase System.Web.HttpRequest. Este objeto va a permitir acceder a todo la información necesaria de la petición del protocolo HTTP que ha sido utilizada para demandar la página de el servidor Web.

- Response: este objeto de la clase System.Web.HttpResponse nos va a permitir manipular la respuesta devuelta al cliente que ha realizado la petición sobre la página ASP .NET actual, este objeto representa la respuesta HTTP que se envía al cliente.

- Server: ofrece una referencia a un objeto de la clase System.Web.HttpServerUtility, este objeto tiene la misma funcionalidad que el objeto Server de anteriores versiones de ASP, es decir, es un compendio de utilidades que permiten realizar una serie de operaciones sobre las peticiones recibidas. Una de las funcionalidades que se ofrecía como principal en las versiones anteriores de ASP, ahora en ASP .NET no tiene dicha importancia se trata de la posibilidad de crear componentes existentes en el servidor, ya hemos visto que en ASP .NET los objetos se crean de manera distinta, ahora los instanciamos mediante el operador new e importando el espacio con nombre de sea necesario.

- Session: en este caso se obtiene una instancia de la clase System.Web.SessionState. HttpSessionState, este objeto nos va a permitir almacenar información entre diferentes páginas ASP incluidas en una misma aplicación ASP .NET. La diferencia con el objeto Application se encuentra en el ámbito de las variables, cada variable del objeto Session es particular a una sesión de un usuario determinado, no a toda la aplicación. De esta forma, cada usuario tendrá sus variables y sus valores, sin dar lugar a problemas de concurrencia, tampoco se podrá acceder a distintas variables de sesión, cada usuario tiene su espacio de almacenamiento. Las variables de aplicación son valores globales y comunes a toda la aplicación, y las variables de sesión son particulares para cada usuario de la aplicación. Los objetos Session y Application los vremos en detalle en el siguiente apartado.

- User: propiedad que devuelve información sobre el usuario que ha realizado la petición de la página ASP .NET.

- Context: esta propiedad nos devuelve un objeto de la clase HttpContext, que encapsula toda la información relativa a una solicitud del protocolo HTTP.

En el siguiente apartado vamos comentar las posibilidad que nos ofrece la aplicación ASP .NET a la hora de mantener información entre distintas peticiones realizadas a la aplicación.

Gestión del estado de la aplicación ASP .NET

La gestión del estado de la aplicación ASP .NET consiste en la persistencia de objetos o valores a través de la vida de la aplicación o bien a través de una sesión de usuario con la aplicación ASP .NET.

Se debe recordar que el protocolo HTTP es un protocolo sin estado, es decir, no se puede almacenar información entre diferentes conexiones HTTP. No se puede mantener el estado entre diferentes páginas Web a través del protocolo HTTP, sino que se deben utilizar otros mecanismos como las cookies. Pero el objeto Application junto con el objeto Session y el objeto Cache nos permite de forma sencilla y directa almacenar información abstrayéndonos del uso de cookies y de encabezados HTTP.

ASP .NET nos ofrece cuatro posibilidades a la hora de almacenar información entre diversas solicitudes dentro de una aplicación ASP .NET, dos de estas posibilidades ya se encontraban

disponibles en ASP a través de los objetos integrados Application y Session, en ASP .NET estos objetos integrados se encuentran en las dos propiedades de mismo nombre del objeto Page, y también a en la clase HttpApplication a través de las propiedades Application y Session.

A continuación vamos a pasar a describir brevemente las distintas posibilidades que nos ofrece una aplicación ASP .NET para almacenar de forma persistente información:

- El objeto Session: este objeto nos va a permitir almacenar el estado del usuario que ha iniciado una sesión con la aplicación ASP .NET, la información se mantendrá durante la vida de cada sesión particular. Cada objeto almacenado en el objeto Session es particular a una sesión de un usuario determinado, no a toda la aplicación. De esta forma, cada usuario tendrá sus variables y sus valores, sin dar lugar a problemas de concurrencia, tampoco se podrá acceder a distintas variables de sesión, cada usuario tiene su espacio de almacenamiento. Las variables de aplicación son valores globales y comunes a toda la aplicación, y las variables de sesión son particulares para cada usuario de la aplicación ASP .NET.

- El objeto Application: este objeto nos va a permitir almacenar información que va a ser común para toda la aplicación ASP .NET y que se mantendrá durante toda la vida de la aplicación. Las variables almacenadas dentro del objeto Application son visibles para todos los usuarios que están utilizando la misma aplicación ASP .NET, es decir son compartidas por varios usuarios. En contraposición al objeto Session, cuyas variables son para cada uno de los usuarios conectados, es decir, no se comparten y son propias de cada sesión. Podremos acceder a una variable a nivel de aplicación en cualquiera de las páginas ASP .NET contenidas en la aplicación ASP .NET actual.

- El objeto Cache: en anteriores capítulos ya vimos una de las utilidades de este objeto, que permitía almacenar en caché páginas ASP .NET sin tener que volver a ejecutarlas, pero este objeto ofrece una funcionalidad adicional. El otro tipo de caché, denominado caché ASP .NET, que ofrece ASP .NET es la que permite almacenar nuestros propios valores y objetos para ser reutilizados entre distintas páginas ASP .NET dentro de una misma aplicación. La caché es global a toda la aplicación, por lo tanto la información se puede acceder desde todas las páginas. Los métodos utilizados para acceder a la caché son seguros en lo que a accesos concurrentes se refiere, no es necesario utilizar bloqueos a la hora de modificar o recuperar valores. Para utilizar esta caché ASP .NET nos ofrece el nuevo objeto Cache, este objeto lo utilizaremos de forma muy similar a los objetos Application y Session. Al igual que sucedía con el objeto Application, los objetos presentes en el objeto Cache van a conservarse durante toda la vida de la aplicación.

- Variables estáticas: además de mantener el estado común a toda la aplicación ASP .NET mediante los objetos Application y Cache, existe una tercera posibilidad para almacenar objetos y mantener su estado para toda la aplicación, esta es mediante la utilización de variables estáticas. Gracias a las técnicas de orientación a objetos podemos definir variables estáticas de las que existirá una única copia en toda la aplicación.

A continuación vamos a comentar en detalle cada una de estas posibilidades de la aplicación ASP .NET a la hora de almacenar objetos de forma persistente.

El objeto Session

Este objeto, instancia de la clase System.Web.SessionState.HttpSessionState, es muy similar al objeto integrado Session de versiones anteriores de ASP, pero en ASP .NET se ofrecen una serie de características que suponían problemas en ASP. Estos dos problemas eran la utilización del estado de

sesión en escenarios Web farm y la utilización del objeto Session cuando el cliente tenía deshabilitadas las cookies en su navegador.

En ASP, el estado se sesión se solía perder en escenarios de Web farm en los que participaban diversos servidores, pero ahora en ASP .NET los objetos almacenados en el objeto Session se pueden acceder desde los distintos servidores Web existentes en el Web farm, el desarrollador no tiene que hacer nada en especial, ya que este mecanismo es automático.

La otra mejora del objeto Session es la posibilidad de almacenar la información en la sesión del usuario sin la necesidad de utilizar el mecanismo auxiliar de cookies del protocolo http. Se debe señalar que en las anteriores versiones de ASP se podían almacenar variables dentro del objeto Session si el navegador acepta cookies, ya que el objeto Session para almacenar información se basaba en la cookie ASPSESSIONID, aunque la utilización de esta cookie era completamente transparente para el programador.

Antes de seguir con la exposición del capítulo vamos a definir lo que es una cookie. Una cookie es utilizada para mantener información entre diferentes conexiones HTTP. Se debe recordar que el protocolo HTTP es un protocolo sin estado, es decir, no se retiene información entre las diferentes conexiones que se realicen. Por esta razón, ni el cliente ni el servidor pueden mantener información entre diferentes peticiones o a través de diferentes páginas Web.

Este mecanismo para mantener información entre diferentes conexiones HTTP fue propuesto e implementado en un principio por la compañía Netscape. Existen varios usos prácticos de las cookies, a continuación se van a comentar los más destacados:

- Para almacenar información acerca de las preferencias del cliente que se conecta a nuestro sitio Web, por ejemplo el color seleccionado de la página, el tipo de letra, etc.

- Para conservar información personal, no sensible, del usuario, como puede ser el nombre, el país de origen, código postal, el número de veces que ha accedido a nuestro sitio Web, etc.

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

Siempre debe haber una primera ocasión en la que el cliente conectado especifique el valor de la cookie, una vez especificado este valor ya puede ser utilizada la cookie en las diferentes conexiones que realice ese cliente, ya que la información ha quedado almacenada en la cookie. Esta información se almacena físicamente en un fichero del disco duro local del cliente. En el caso del navegador

Internet Explorer de Microsoft cada cookie se corresponde con un fichero.txt que se encuentra en el directorio Documents and Settings (en el caso de Windows 2000) y en el subdirectorio Cookies.

Una vez comentadas estas dos novedades del objeto Session vamos a pasar a describir su utilización básica.

Algo básico que podemos hacer con el objeto Session es almacenar un objeto a nivel de sesión y luego recuperarlo, el almacenamiento de estos objetos se basa en una estructura en forma de pares clave/valor, el patrón de uso del objeto Session dentro de ASP .NET es muy similar al de ASP. En el Código fuente 268 se ofrece la sintaxis para almacenar un objeto en la sesión del usuario, en el objeto Session siempre almacenaremos objetos, y estos objetos quedarán almacenados como instancias de la clase Object, la clase raíz de los objetos.

```
Session["CadenaClave"]=objeto;
```

Código fuente 268

Una vez almacenado un objeto con una clave determinada en el objeto Session, podremos recuperarlo haciendo uso de esa misma clave.

En el Código fuente 269 se ofrece una página ASP .NET muy sencilla que almacena un objeto String dentro del objeto Session y a continuación recupera su valor para mostrarlo en pantalla.

```
<%@ Page language="c#" %>
<html>
<head><title>El objeto Session</title></head>
<body>
<%Session["clave"]="valor";
Response.Write(Session["clave"]);%>
</body>
</html>
```

Código fuente 269

En este caso la recuperación del objeto desde la sesión presenta una sintaxis muy sencilla, debido a que el objeto es de la clase String. Pero si el objeto fuera de otro tipo, por ejemplo de la clase DateTime, deberíamos realizar la conversión correspondiente mediante el mecanismo de casting adecuado, ya que el objeto Session almacena la información en forma de instancias del objeto Object, es decir, se almacenan los objetos como instancias de la clase raíz de la jerarquía de objetos del .NET Framework.

En el Código fuente 270 se ofrece un ejemplo de página ASP .NET que utiliza el objeto Session para almacenar un objeto de la clase DateTime.

```
<html>
<body>
<script language="C#" runat="server">
public void Page_Load(Object sender, EventArgs args){
      DateTime ahora;
      Session["fecha"]=DateTime.Now;
      ahora=(DateTime)Session["fecha"];
      texto.Text=ahora.ToString();
}
</script>
<form id="formulario" method="post" runat="server">
<asp:Label runat="server" ID="texto"></asp:Label>
</form>
</body>
</html>
```

Código fuente 270

Como se puede comprobar el manejo del objeto Session de ASP .NET es muy similar a como se hacía en ASP, pero debemos tener en cuenta las dos novedades que ofrecía este objeto dentro de ASP .NET. Debido a que el objeto Session no está asociado al mismo proceso que ASP .NET, nos permite utilizar

el estado de sesión en escenarios de Web farm, sin tener que preocuparnos de si el cliente está accediendo a través de un servidor proxy. La segunda novedad, que vamos a ver con detenimiento un poco más adelante en este mismo apartado, permite almacenar el estado de la sesión del usuario sin hacer uso de las cookies del usuario.

Como hemos comentado anteriormente, el objeto Session lo podemos obtener como una propiedad dela clase HttpApplication, dentro del fichero GLOBAL.ASAX de la aplicación, o bien como una propiedad de la clase Page dentro de una página ASP .NET. En cualquier caso este objeto es una instancia de la clase System.Web.SessionState.HttpSessionState, a continuación vamos a describir brevemente las propiedades y métodos principales de esta clase.

Entre las propiedades de la clase HttpSessionState podemos encontrar las siguientes:

- SessionID: devuelve un objeto de la clase String que contiene la identificación de la sesión para el usuario. Cada sesión tiene un identificador único que genera el servidor al crearla. No se debe utilizar esta propiedad como clave de una tabla de una base de datos, ya que, al reiniciar el servidor Web, algunos de los valores de SessionID pueden coincidir con los generados antes de que se apagase el servidor. Es únicamente de lectura.

- Timeout: la propiedad Timeout especifica el intervalo de inactividad para el objeto Session en minutos. Si el usuario no actualiza o solicita una página durante ese intervalo, la sesión termina. El valor por defecto de esta propiedad es de 20 minutos, es decir, por defecto la sesión permanecerá inactiva 20 minutos. Una sesión se dice que está inactiva mientras el navegador cliente no realice una petición. El valor de esta propiedad se puede modificar dinámicamente a lo largo de la ejecución de la aplicación ASP .NET.

- CodePage: indica la página de códigos de caracteres que va a ser utilizado. A esta propiedad se le puede asignar un valor entero par especificar la página de códigos que se va a utilizar. Así si queremos utilizar el juego de caracteres del alfabeto turco le asignaremos a esta propiedad el valor 1254. Esta propiedad es también de lectura/escritura.

- LCID (Locale Identifier): propiedad de lectura/escritura, es una abreviatura estándar e internacional que identifica de forma única la localización de los sistemas. Esta localización determina la forma en la que se le da formato a las horas y fechas, como se tratan las cadenas de caracteres y los diferentes elementos del alfabeto. Así si queremos establecer la propiedad LCID para la región de Rusia, le debemos asignar a esta propiedad el valor 1049.

- IsCookieless: devuelve los valores True o False para indicar si la sesión está utilizando o no el mecanismo de cookies para mantener el identificador de la sesión (SessionID). Por defecto su valor es False, es decir, por defecto la sesión utiliza las cookies. Se trata de una propiedad de sólo lectura.

- IsReadOnly: propiedad que indica si el objeto Session se encuentra en modo de sólo lectura, esta es una optimización que ofrece ASP .NET cuando no es necesario actualizar o modificar el estado de la sesión. Por defecto esta propiedad de sólo lectura presenta el valor False, en el siguiente capítulo, dedicado a la configuración de la aplicación ASP .NET, veremos como indicar que una sesión es de sólo lectura, en general veremos la configuración del objeto Session desde el fichero Web.config.

- Mode: propiedad de sólo lectura que indica el modo de almacenamiento utilizado para el estado de la sesión. Los valores que puede devolver esta propiedad son: InProc (el estado de sesión se encuentra con el proceso actual de ASP .NET), Off (el estado de sesión se encuentra desactivado), SQLServer (se utiliza un proceso de SQL Server para almacenar el estado), StateServer (se utiliza un proceso de Windows para almacenar el estado). El valor por defecto

es InProc. Esta característica de la sesión también se configura a través del fichero Web.config.

A continuación se ofrece un sencillo código (Código fuente 271) de una página ASP .NET que nos muestra los valores de algunas de las propiedades de una sesión.

```
<%@ Page language="c#" %>
<html>
<head><title>El objeto Session</title></head>
<body>
IsReadOnly=<%=Session.IsReadOnly%><br>
IsCookieless=<%=Session.IsCookieless%><br>
Timeout=<%=Session.Timeout%><br>
SessionID=<%=Session.SessionID%><br>
LCID=<%=Session.LCID%>
</body>
</html>
```

Código fuente 271

El resultado de la ejecución de la página ASP .NET se puede ver en la Figura 175.

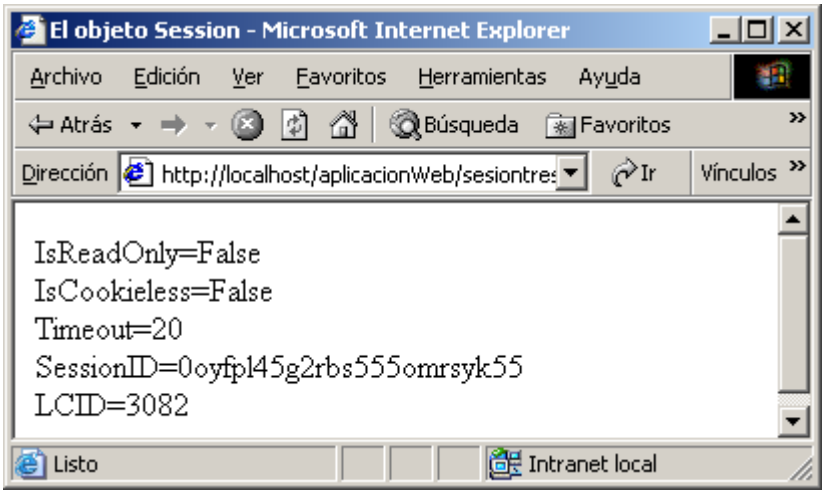


Figura 175

Una vez comentadas las principales propiedades de la clase HttpSessionState vamos a pasar a los métodos más destacados:

- Abandon: al lanzar sobre el objeto Session el método Abandon(), se destruyen todos los objetos de la sesión y se liberan sus recursos, finalizando la misma. Si no se llama explícitamente al método Abandon(), el servidor destruirá los objetos cuando la sesión caduque, atendiendo al valor de la propiedad Timeout. La destrucción del objeto Session no se hará efectiva hasta que el servidor no haya terminado de ejecutar la página ASP .NET. Por lo tanto las variables del objeto Session existirán mientras no se cargue otra página. La ejecución de este método provocará el lanzamiento del evento Session_OnEnd.

- Add: añade un nuevo elemento a la sesión, es equivalente a la sintaxis vista anteriormente para incluir un objeto en la sesión. Como primer parámetro este método presenta un objeto String

que representa el nombre de la clave y el segundo parámetro es un objeto de la clase Object, y será el objeto que deseamos almacenar en la sesión.

- Clear: método sin parámetros que elimina todos los objetos almacenados en la sesión actual.
- Remove: elimina el objeto de la sesión cuya clave coincida con la cadena que se pasa como parámetro.

Como ya habíamos comentado, hay ocasiones en las que los usuarios deshabilitan el mecanismo de cookies de sus navegadores, esto ocasionaba en ASP que no se pudiera mantener el estado de sesión, ya que internamente ASP utilizaba una cookie.

Con ASP .NET esto ha cambiado, ya que posee un mecanismo auxiliar no basado en cookies para mantener el estado de sesión, podemos configurar el objeto Session para que el identificador de sesión no se almacene en una cookie, sino que vaya incluido en la URL que hace referencia a la página ASP .NET correspondiente.

Para indicar que el objeto Session no va a utilizar cookies debemos acceder al fichero de configuración dela aplicación ASP .NET, es decir, al fichero web.config (este fichero lo veremos en detalle en el siguiente capítulo), y dentro del elemento <sessionState> se debe asignar a su propiedad cookieless el valor true, tal como se muestra en el Código fuente 272.

```
<configuration>
  <system.web>
    <sessionState
            mode="InProc"
            stateConnectionString="tcpip=127.0.0.1:42424"
            sqlConnectionString="data source=127.0.0.1;user id=sa;password="
            cookieless="true"
            timeout="20"
    />
  </system.web>
</configuration>
```

Código fuente 272

Al guardar estos cambios realizados en el fichero de configuración de la aplicación ASP .NET, se aplicarán de forma inmediata sobre la aplicación, ahora nuestra sesión está configurada para trabajar sin cookies, veamos los resultados mediante un ejemplo sencillo.

Tenemos una página ASP .NET con un Web Form que permite almacenar el contenido de un objeto TextField en un objeto de la sesión, además permite ver el contenido de dicho objeto y borrar el estado de la sesión. El Código fuente 273 es el código de esta página.

```
<%@ Page language="c#"%>

<HTML>
<script runat="server">
      void AlmacenaEstado(Object sender, EventArgs args){
            Session["texto"]=texto.Text;
      }

      void BorraEstado(Object sender, EventArgs args){
            Session.Clear();
```

```
      }

      void MuestraEstado(Object sender, EventArgs args){
            if(Session["texto"]!=null)
                  etiqueta.Text="El contenido de la sesión es: "+Session["texto"];
            else
                  etiqueta.Text="La sesión no tiene contenido";
      }
</script>
<head><title>El objeto Session</title></head>
<body>
<form id="WebForm" method="post" runat="server">
<asp:TextBox Runat="server" ID="texto"/><br>
<asp:Button Runat="server" ID="almacena" OnClick="AlmacenaEstado"
                           Text="Almacenar en la sesión"/><br>
<asp:Button Runat="server" ID="muestra" OnClick="MuestraEstado"
                           Text="Muestra contenido sesión"/><br>
<asp:Button Runat="server" ID="borra" OnClick="BorraEstado"
                           Text="Borra contenido sesión"/><br>
<asp:Label id="etiqueta" runat="server" EnableViewState="False"/><br>
<br>
<asp:HyperLink Runat="server" ID="enlace" NavigateUrl="sesion.aspx" Text="Enlace"/>
</form>
</body>
</HTML>
```

Código fuente 273

En este caso no se está utilizando un cookie, sino que para ir teniendo constancia del identificador de sesión se incluye de forma automática en la URL de la página, tal como se puede apreciar en la Figura 176.

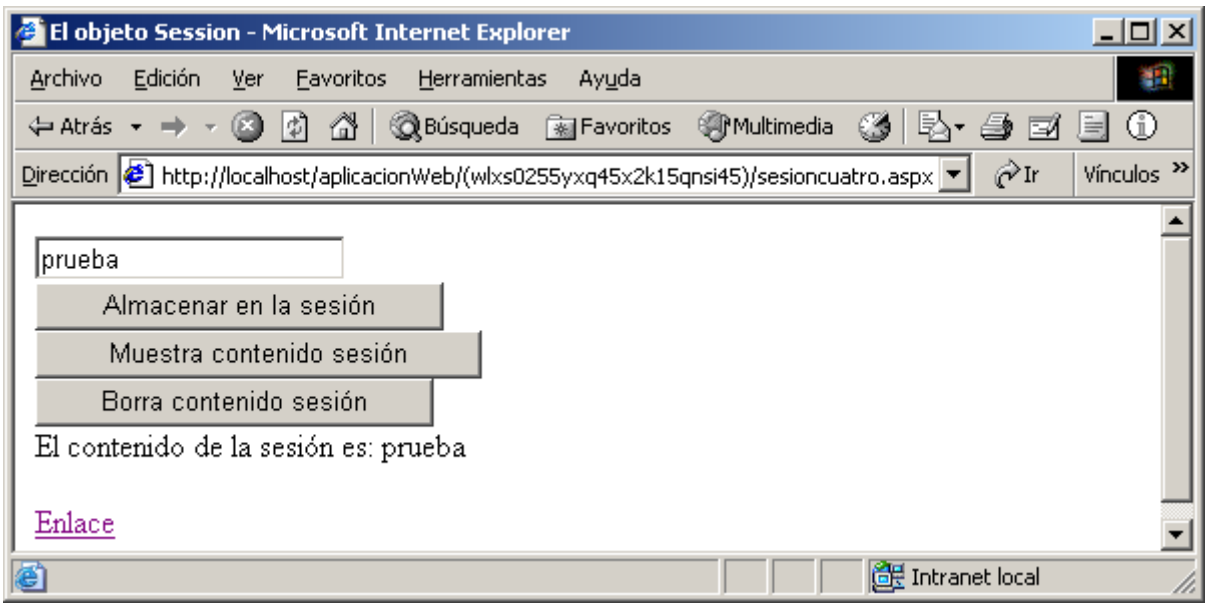


Figura 176

Si incluimos un enlace en la página ASP .NET a otra página, al pulsar sobre el enlace, de forma automática se incluirá también el identificador de sesión dentro del propio enlace.

Las aplicaciones ASP .NET se pueden configurar para que el objeto Session haga uso de cookies o bien no haga uso de ellas, pero los dos mecanismos son exclusivos, no se pueden indicar las dos opciones al mismo tiempo.

En el siguiente apartado trataremos otras de las posibilidades que ofrece la aplicación ASP .NET a la hora de almacenar el estado, en este caso se va a tratar del objeto Application.

El objeto Application

Esta segunda forma de gestionar el estado de la aplicación también se encontraba disponible en ASP a través del objeto integrado Application, en ASP .NET el objeto Application se va a obtener de dos formas distintas, como propiedad de la clase HttpApplication, dentro del fichero GLOBAL.ASAX, y como propiedad del objeto Page dentro de las páginas ASP .NET, en cualquier caso se trata de una instancia de la clase System.Web.HttpApplicationState. El almacenamiento de los objetos se basa en una estructura en forma de pares clave/valor.

A diferencia de lo que sucedía con el objeto Session, en el objeto Application vamos almacenar objetos que van a ser comunes para todos los usuarios de la aplicación ASP .NET. Estos objetos se mantendrán durante la vida de la aplicación.

Otra diferencia respecto al objeto Session, es que los objetos almacenado en el objeto Application no permiten almacenarse en un proceso separado, sino que se almacenarán en el mismo proceso que ASP .NET.

La sintaxis para almacenar objetos en el estado de aplicación y recuperarlos es igual a la sintaxis utilizada con el objeto Session, como se puede apreciar en el Código fuente 274.

```
Application[“CadenaClave”]=objeto;
```

Código fuente 274

A la hora de modificar un objeto dentro del estado de la aplicación debemos tener en cuenta que se pueden dar accesos concurrentes al objeto, ya que es compartido por todos los usuarios de la aplicación, para evitar problemas de concurrencia el objeto Application posee dos métodos Lock() y UnLock() el primero bloquea el acceso al objeto almacenado en el estado de la aplicación, para que únicamente pueda acceder un usuario, y el segundo desbloquea el objeto. Estos métodos tiene el mismo cometido que en ASP.

Para ver un ejemplo de utilización del objeto Application vamos a crear un fichero GLOBAL.ASAX, como el del Código fuente 275, que vaya contando las solicitudes que tiene una aplicación ASP .NET, para ello utilizaremos los eventos Application_OnStart, para inicializar el contador.

```
<script language="C#" runat="server">

    void Application_OnStart(Object obj, EventArgs e) {
             Application["solicitudes"]=0;
    }

    void Application_OnBeginRequest(Object obj, EventArgs e) {
             Application.Lock();
             Application["solicitudes"]=(int)Application["solicitudes"]+1;
```

```
            Application.UnLock();
    }

</script>
```

Código fuente 275

Como se puede observar, los métodos de bloqueo y desbloqueo de la aplicación no se han utilizado en el evento Application_OnStart, ya que este evento se ejecutará una única vez durante la vida de la aplicación ASP.

Si después queremos visualizar el valor del contador almacenado en el estado de la aplicación, únicamente tenemos que escribir el Código fuente 276, que pertenece a una página ASP .NET de la aplicación.

```
<%@ Page language="c#" %>

<html>

<head><title>El objeto Application</title></head>
<body>
<%Response.Write(Application["solicitudes"]);%>
</body>

</html>
```

Código fuente 276

En el siguiente apartado vamos a mostrar la tercera forma de almacenar información en la aplicación ASP .NET, en este caso no hay un equivalente con ASP, ya que se va a hacer uso del nuevo objeto Cache de ASP .NET

El objeto Cache

El objeto Cache va a almacenar objetos que se van a poder utilizar en toda la aplicación ASP .NET por todos sus usuarios, es decir, el ámbito de la información almacenada en este objeto es igual que la del objeto Application, y se va a mantener durante toda la vida de la aplicación, pero el objeto Cache, de la clase System.Web.Caching.Cache, presenta una serie de diferencias respecto al objeto Application, nos ofrece una serie de funcionalidades que el objeto Application no presenta.

En este caso el objeto Cache lo vamos a utilizar para almacenar datos en forma de objetos, no debemos confundir este tipo de caché, llamada también caché de ASP .NET, con el mecanismo de caché que nos ofrece ASP .NET a través de la directiva @OutputCache, es este otro caso se trata de la caché de salida de las página ASP .NET, y ya la comentamos un capítulo anterior, concretamente el dedicado a la clase Page.

Este objeto lo obtenemos como una propiedad de la clase Page, a través de la propiedad Cache. Además de la posibilidad de almacenar objetos en forma de pares clave/valor, al igual que el objeto Application, ofrece las siguientes características adicionales:

- Establecimiento de dependencias basadas en ficheros y claves: estas dependencias de los datos almacenados en el objeto Cache se pueden establecer respecto de ficheros y claves, así por

ejemplo si un elemento de la caché de ASP .NET depende de un fichero que es modificado este elemento será invalidado y eliminado de la caché. Las dependencias de ficheros se pueden utilizar en el siguiente escenario: una aplicación ASP .NET que lee información financiera almacenada en un fichero XML, este fichero es actualizado de forma periódica; la aplicación procesa los datos del fichero y crea un gráfico con los mismos, par almacenar este fichero la aplicación utiliza el objeto Cache estableciendo una dependencia respecto del fichero XML; cuando el fichero es modificado los datos se eliminan de la caché y la aplicación vuelve a leerlos para recuperar en el objeto Cache una copia actualizada de los datos.

- Establecimiento de dependencias basadas en caducidad: en este caso la validez de los datos almacenados en el objeto Caché dependerán de un tiempo determinado, esto permite al desarrollador establecer el tiempo de vida de los datos de los elementos de la caché, puede ser basado en un tiempo explícito o absoluto, por ejemplo a las cinco de la mañana, o bien basado en un tiempo relativo, por ejemplo cuando han pasado 20 minutos del último acceso al elemento. El establecimiento de dependencias basado en ficheros o claves se puede combinar con el basado en caducidades.

- Mantenimiento de recursos: los elementos almacenados en el objeto Cache que no se usan de forma frecuente son eliminados de la caché liberando la memoria que ocupaban. Debido a esto cuando deseemos recuperar un elemento almacenado en el objeto Caché deberemos comprobar si existe o ya ha sido eliminado.

- Gestión de bloqueos: al igual que sucedía con el objeto Application, los objetos almacenados en el objeto Cache se pueden acceder desde cualquier página ASP .NET de la aplicación, por lo tanto se pueden dar modificaciones concurrentes de estos objetos, pero en le objeto Cache no hay que utilizar métodos similares a Lock() y UnLock(), como sucedía en el objeto Application, sino que el objeto Cache controla de forma automática estos accesos concurrentes.

- Callbacks: cuando eliminamos un elemento almacenado en el objeta Cache, se nos va a permitir ejecutar un código determinado.

El objeto Cache ofrece dos formas distintas para insertar elementos, la forma implícita y la forma explícita, a continuación vamos a comentar ambas posibilidades.

La sintaxis que presenta la forma implícita de almacenar un objeto en la caché es similar a la que presentaban los objetos Session y Application, es decir, es encuentra basada en el almacenamiento de pares clave/valor, en el Código fuente 277 se puede ver esta sintaxis.

```
Cache["cadenaClave"]=objeto;
```

Código fuente 277

La forma de explícita de almacenar elementos en el objeto Cache es más potente, ya que es la que nos va a permitir establecer dependencias respecto de ficheros, claves o de una caducidad, para ellos se hace uso del método Insert(). Este método se encuentra sobrecargado, presentando por lo tanto diversas versiones que nos van a permitir establecer distintas políticas de dependencias, incluso permiten una combinación de ellas.

A continuación vamos a describir brevemente las distintas posibilidades que ofrece este método de la clase System.Web.Caching.Cache:

- public void Insert(string clave, Object objeto): inserta un elemento en el objeto Cache, sin ninguna política de caducidad o de dependencia, en este caso se eliminará el elemento cuando el sistema lo crea conveniente.

- public void Insert(string clave, Object objeto, CacheDependency dependencia): inserta un objeto en la caché con dependencias de fichero o de clave.

- public void Insert(string clave, Object objeto, CacheDependency dependencia, DateTime tiempoExplícito, TimeSpan tiempoRelativo): inserta un objeto en la caché indicando una dependencia de fichero o de clave y también una dependencia de caducidad, no es obligatorio indicar los dos tipos de dependencias.

- public void Insert(string clave, Object objeto, CacheDependency dependencia, DateTime tiempoExplícito, TimeSpan tiempoRelativo, CacheItemPrority prioridad, CacheItemPriorityDecay prioridad, CacheItemRemovedCallBack callBack): además de permitir indicar las dependencias de fichero, clave y caducidad, se indican las prioridades del elemento a la hora de ser eliminado del objeto Cache. En el último parámetro si indica un método que se ejecutará cuando el elemento sea eliminado de la caché.A continuación vamos a utilizar el objeto Cache mediante un ejemplo, en este caso se va a almacenar el contenido de una tabla de una base de datos. Para almacenar el objeto que va a representar la base de datos (objeto DataView de ADO .NET) dentro del objeto Cache vamos a utilizar el método Insert(), es decir, la forma explícita, ya que como hemos dicho es la que más posibilidades nos ofrece.

En este ejemplo vamos a establecer una dependencia basada en la caducidad de los datos, podemos establecer que la tabla permanezca en el objeto Cache durante un minuto, una vez pasado este minuto la tabla se destruirá, por lo que tendremos que volver a recuperarla de la base de datos para volver a almacenarla en el objeto Cache. El Código fuente 278 es la página ASP .NET que ilustra este ejemplo.

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<head><title>El objeto Cache</title></head>
<script runat="server">

void CargaEmpleados(){
      SqlConnection conexion =
        new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      SqlDataAdapter comando =
         new SqlDataAdapter("select firstname, lastname,city from Employees",
                                                                  conexion);
      DataSet ds = new DataSet();
      comando.Fill(ds, "Employees");
      Cache.Insert("empleados",ds.Tables["Employees"].DefaultView,null,
                                  DateTime.Now.AddMinutes(1),TimeSpan.Zero);
}

void Page_Load(Object sender, EventArgs e) {
      DataView datos;
      if(Cache["empleados"]==null){
            estado.Text="Recuperando datos de la base de datos...";
            CargaEmpleados();
      }else{
            estado.Text="Recuperando datos del objeto Cache...";
      }
      datos=(DataView)Cache["empleados"];
      tabla.DataSource=datos;
```

```
      tabla.DataBind();
}
</script>

<body>
<form id="formulario" runat="server">
      <asp:Label ID="estado" Runat="server"/>
      <asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
      HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow">
      </asp:DataGrid>
      <asp:Button ID="boton" runat="server" Text="Recargar página"/>
</form>
</body>
</html>
```

Código fuente 278

Como se puede comprobar se ha incluido un Web form con varios controles Web, uno de ellos es un objeto Label que vamos a utilizar para indicar si los datos se van a obtener de la caché o bien no se encuentran presentes y se tiene que recuperar de la base de datos, esta situación se dará en la primera ejecución de la página y cada vez que pase un minuto. Otro control Web que ofrece este formulario es un control DataGrid que nos va a permitir ver los datos, y por último tenemos un botón para poder recargar la página ASP .NET y comprobar en que situación se encuentra la caché.

Para comprobar si existe la tabla dentro del objeto Cache se compara el elemento correspondiente con el valor null.

En el ejemplo se ha utilizado una versión del método Insert() del objeto Cache que permite especificar una caducidad basada en un tiempo explícita y una caducidad basada en un tiempo relativo, en nuestro paso hemos utilizado una caducidad explícita ya que estamos indicando que el elemento del objeto Cache que vamos a añadir caducará pasado un minuto, para ello hemos utilizado la propiedad Now de la clase DateTime, y le hemos aplicado el método AddMinutes.

Y en el parámetro en el que podemos indicar el tiempo relativo, como no lo vamos a utilizar, le hemos asignado el valor TimeSpan.Zero. En los tres primeros parámetros de esta versión del método Insert() se indica la clave del objeto que vamos a añadir a la caché, el objeto que se va a añadir y por último las dependencias de ficheros o claves, que como en nuestro ejemplo no existen se ha asignado el valor null.

En la Figura 177 se puede ver un ejemplo de ejecución de la página ASP .NET del ejemplo.

Si deseamos variar el comportamiento de la caducidad, utilizando por ejemplo un tiempo relativo de un minuto, es decir, si en un minuto no hemos accedido al elemento que nos ocupa del objeto Cache, será invalidado del mismo, teniéndolo que recuperar de nuevo de la base de datos, deberíamos sustituir la llamada al método Insert() del ejemplo anterior con la que aparece en el Código fuente 279.

```
Cache.Insert("empleados",ds.Tables["Employees"].DefaultView,null,
                                        DateTime.Maxvalue,TimeSpan.FromMinutes(1));
```

Código fuente 279

Para indicar que no se va a utilizar una caducidad absoluta se asigna al parámetro correspondiente el valor DateTime.MaxValue.

En el siguiente apartado comentaremos la última de las opciones que nos ofrece una aplicación ASP .NET para mantener el estado.

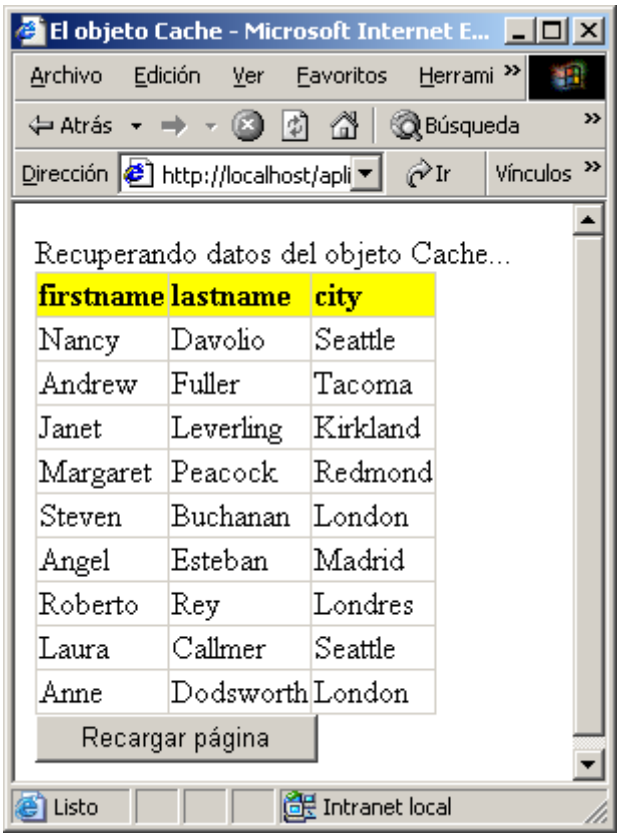


Figura 177

Variables estáticas

Si declaramos una variable con el modificador de contenido static mantiene su contenido para todas las instancias de la clase que se hagan, así como para las clases hijas que de ella se hereden. A estas variables se les denomina variables estáticas.

Mientras que los variables de una clase se inicializan para cada nueva instancia que se haga de la clase, es decir, existe una copia por cada instancia de la clase, de las variables estáticas existe una sola instancia, independientemente del número de instanciaciones que de la clase se realicen. De este modo, todos los objetos comparten un lugar de almacenamiento común.

Mediante la declaración de variables estáticas en el fichero GLOBAL.ASAX podremos compartir el valor de las variables a través de todas las páginas de la aplicación.

Pero para poder hace uso de variables y métodos estáticos declarados en el fichero GLOBAL.ASAX de la aplicación Web, debemos utilizar el atributo Classname de la directiva @Application. Este atributo no le habíamos comentado con anterioridad, y nos va a servir para indicar el nombre de al clase que se le va a asignar al fichero GLOBAL.ASAX cuando se compile.

En el resto de páginas ASP .NET de la aplicación tendremos acceso a las variables y métodos estáticos declarados en el fichero GLOBAL.ASAX a través del nombre de la clase indicado en el atributo Classname de la directiva @Application.

En el Código fuente 280 se muestra la declaración de una variable estática dentro del fichero GLOBAL.ASAX de una aplicación ASP .NET, y en el Código fuente 281 se muestra la utilización de esta variable desde una página ASP .NET de la misma aplicación.

```
<%@ Application Classname="Aplicacion" %>
<script language="C#" runat="server">
      public static String variableEstatica="Hola Mundo!";
</script>
```

Código fuente 280

```
<%@ Page language="c#" %>
<html>
<head><title>El objeto Application</title></head>
<body>
<%Response.Write(Aplicacion.variableEstatica);%>
</body>
</html>
```

Código fuente 281

La utilización de variables estáticas, en muchos pasos, suele ser más rápido que acceder a objetos almacenados en el objeto Application.

Con este apartado terminamos con las distintas formas que nos ofrece una aplicación ASP .NET de gestionar el estado de la misma.

Para finalizar este capítulo lo haremos con el siguiente aparado, en el que se comenta como podemos utilizar nuestra propia clase de la que va a heredar el fichero GLOBAL.ASAX.

Utilizando nuestra propia clase para el fichero GLOBAL.ASAX

Mediante la utilización del atributo Inherits de la directiva @Application podemos indicar una clase de la que va a heredar el fichero GLOBAL.ASAX de la aplicación ASP .NET. Esto es de gran utilidad cuando deseamos añadir nuestros propios métodos y propiedades como parte del fichero GLOBAL.ASAX.

La clase de la que va heredar el fichero GLOBAL.ASAX debe cumplir únicamente un requisito, heredar de la clase System.Web.HttpApplication, esta clase la veíamos en un apartado dentro de este mismo capítulo.

Una vez construida la clase la debemos compilar generando el assembly correspondiente, que como ya vimos en anteriores capítulos copiaremos dentro del directorio BIN de la aplicación ASP .NET. Vamos a ver todo esto a través de un sencillo ejemplo.

En este ejemplo deseamos añadir al fichero GLOBAL.ASAX un método que nos devuelva la fecha y hora actuales, para ello tenemos que heredar de una clase creada por nosotros mismos, cuyo código fuente se ofrece a continuación (Código fuente 282).

```
using System;
using System.Web;

namespace Componentes.Ejemplos
{
	public class MiAplicacion:HttpApplication
	{
		public String muestraHora()
		{
			DateTime ahora=DateTime.Now;
			return ahora.ToString();
		}
	}
}
```

Código fuente 282

Una vez compilada la clase y copiado el assembly al directorio BIN de la aplicación, ya podemos utilizar dicha clase dentro del fichero GLOBAL.ASAX. Simplemente tenemos que indicar el nombre de la clase dentro del atributo Inherits de la directiva @Application, de esta forma ya podemos hacer uso del método muestraHora().

En el Código fuente 283 se muestra un fichero GLOBAL.ASAX que hereda de la clase Componentes.Ejemplos.MiAplicacion, y en el evento Application_OnBeginRequest se hace uso del método muestraHora(), para indicar en cada solicitud la fecha y hora actuales.

```
<%@ Application Inherits="Componentes.Ejemplos.MiAplicacion" %>
<script language="C#" runat="server">
	void Application_OnBeginRequest(Object obj, EventArgs e) {
		Response.Write(muestraHora());
    }
</script>
```

Código fuente 283

En la Figura 178 se puede ver un ejemplo de ejecución de una página ASP .NET incluida dentro de la aplicación ASP .NET que posee el fichero GLOBAL.ASAX anterior.

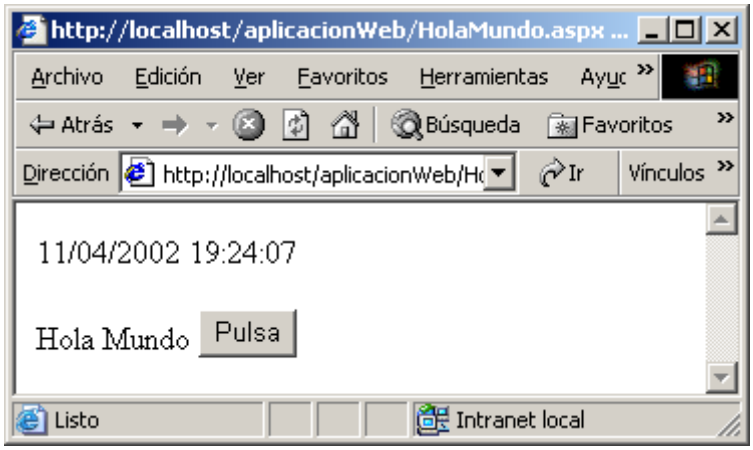


Figura 178

En el siguiente capítulo vamos a tratar en profundidad la configuración de las aplicaciones ASP .NET. Esta configuración se va a realizar a través del fichero, basado en XML, Web.config, este fichero ya le hemos tratado en algunas ocasiones, pero será en el capítulo siguiente cuando veamos todos sus usos, y también veremos algunos que ya hemos tratado en algunas partes del texto.

19

Configuración de aplicaciones ASP .NET

Introducción

Las aplicaciones Web, al igual que el resto de aplicaciones, requieren de unos medios de almacenamiento para poder almacenar los detalles que describen los comportamientos y propiedades de la aplicación, estos detalles son conocidos como información de configuración.

En este capítulo vamos a comentar el sistema de configuración que nos ofrece el .NET Framework para configurar nuestras aplicaciones ASP .NET. Algunos de los aspectos de la configuración de aplicaciones ASP .NET ya los hemos adelantado puntualmente en algunos apartados de capítulos anteriores, de todas formas en este capítulo los volveremos a tratar.

En versiones anteriores de ASP la configuración de aplicaciones Web se realizaba a través del Administrador de servicios de Internet, ya que la información relativa a la configuración de aplicaciones ASP se almacenaba en un repositorio binario denominado metabase de Internet Information Server. Este sistema de configuración de aplicaciones Web basado en el servidor Web Internet Information Server no es válido para las aplicaciones ASP .NET.

El sistema de configuración de aplicaciones ASP .NET está basado en ficheros XML, es decir, se utilizan ficheros XML de configuración. Tal como comentan algunos autores, podemos decir que el sistema de configuración de ASP .NET es un sistema simple y potente basado en XML.

Este sistema de configuración ofrecido por ASP .NET soporta dos tipos de ficheros de configuración:

- Configuración del servidor: la configuración del servidor se almacena en un fichero denominado machine.config. Este fichero va a representar la configuración por defecto de todas las aplicaciones ASP .NET existentes en el servidor. Este fichero de configuración del servidor lo podemos localizar en el directorio [WinNT]\Microsoft .NET\Framework\ [versión]\config.

- Configuración de la aplicación: la información relativa a configuración de una aplicación ASP .NET determinada se almacena en el fichero web.config. Un servidor Web pude contener varios ficheros web.config, cada uno de ellos dentro del directorio raíz de cada una de las aplicaciones ASP .NET del servidor. La configuración del fichero web.config sobrescribe los valores especificados en el fichero machine.config. Si el fichero web.config se encuentra en un directorio en un nivel superior, del que dependen varias aplicaciones ASP .NET, este fichero indicará la configuración de todas estas aplicaciones, aunque después cada aplicación puede sobrescribir estos valores con su propio fichero web.config.

Como ya se ha comentado la configuración de las aplicaciones ASP .NET no van a depender de la metabase de Internet Information Server, sino que se basa en ficheros XML, de lo que se desprende las siguientes ventajas:

- Valores de configuración en formato reconocible: es muy sencillo abrir un fichero XML y leer o modificar la configuración.

- Actualizaciones inmediatas: al contrario que sucedía con las aplicaciones ASP, en ASP .NET las modificaciones realizadas en la configuración de las aplicaciones se aplican de forma inmediata sin necesidad de reiniciar el servidor Web o parar los servicios. Los cambios en la configuración afectan de forma inmediata al sistema y son completamente transparentes para el usuario.

- No es necesario el acceso local al servidor: ya que como hemos comentado, los cambios realizados en la configuración se aplican de forma automática.

- Configuraciones fácilmente repetibles: para tener una aplicación ASP .NET en un servidor con la misma configuración que otra aplicación distinta, únicamente debemos copiar los ficheros de configuración a la aplicación ASP .NET correspondiente.

- Bloqueo de valores de configuración: podemos bloquear los valores de configuración que deseemos para que no sean sobrescritos.

Pero el sistema de configuración ofrecido por ASP .NET no elimina de forma completa la utilización de la metabase de Internet Information Server, existen dos excepciones que son las siguientes:

- Creación de aplicaciones Web: en el capítulo dedicado al servidor Web Internet Information Server ya adelantábamos la forma de crear una aplicación Web (aplicación ASP .NET), este proceso que hacíamos en ASP, mediante la herramienta administrativa Administrador de servicios de Internet, lo vamos a tener que seguir realizando en ASP .NET para indicar el directorio de inicio de una aplicación ASP .NET. El proceso es el mismo y posee el mismo significado. Para más detalles se puede acudir al capítulo dedicado a IIS 5.0.

- Asignación personalizada de extensiones: para asignar de forma personalizada una extensión a un ejecutable determinado lo debemos hacer también a través de IIS.

El resto de las acciones de configuración que deseemos aplicar a nuestra aplicación ASP .NET lo haremos a través de los ficheros de configuración en formato XML.

A continuación vamos a comentar la forma que tiene ASP .NET de aplicar la configuración a las aplicaciones.

Aplicando la configuración

Cuando ASP .NET aplica los valores de configuración a una solicitud determinada, se produce una unión entre los ficheros machine.config y web.config. Los valores de configuración se heredan entre distintas aplicaciones ASP .NET, siendo el fichero machine.config la configuración raíz de la que heredan el resto.

En la Figura 179 se muestra un esquema que describe la forma en la que se aplica la configuración.

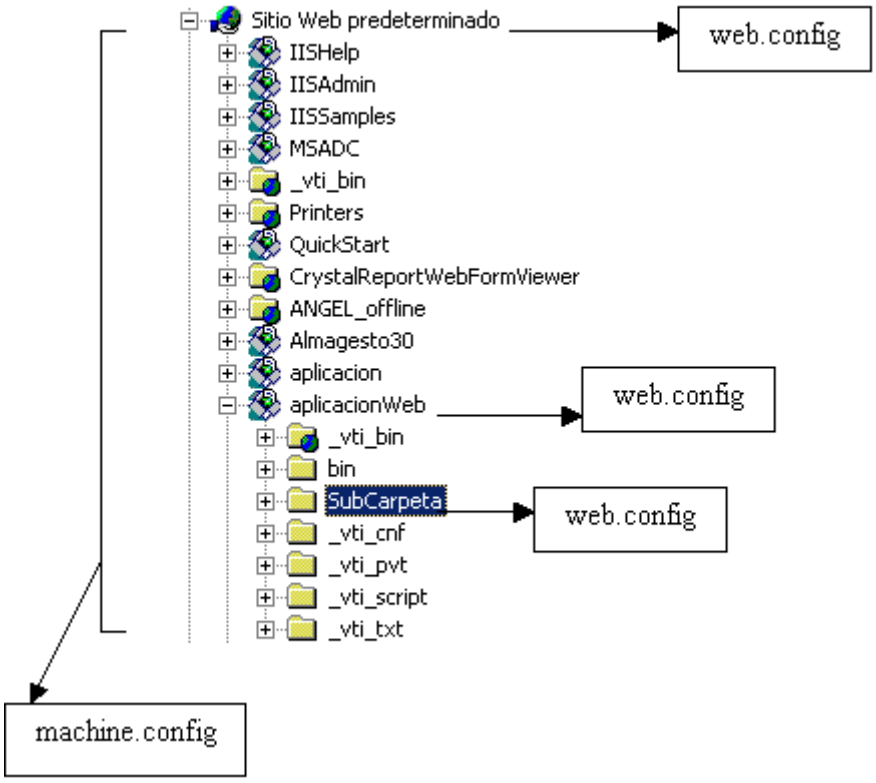


Figura 179

En primer lugar tenemos el fichero machine.config que afectará a todas las aplicaciones ASP .NET existentes en el servidor Web. A los valores de configuración indicados en este fichero se le añadirían o sobrescribirían los presentes en el fichero web.config que posee el sitio Web predeterminado, que actúa como una aplicación ASP .NET.

Si seguimos la cadena nos encontramos con la aplicación ASP .NET llamada aplicacionWeb, que ofrece su propio fichero web.config que se combinaría con el presente en el sitio Web predeterminado. Y como final de esta cadena de configuración tenemos un fichero web.config dentro de un subdirectorio de la aplicación ASP .NET antes mencionada, en este subdirectorio se volverán a sobrescribir o añadir valores de configuración a las páginas ASP .NET presentes en el mismo.

Los cambios realizados a cualquiera de los ficheros de configuración que participan en la definición de una configuración de una aplicación ASP .NET se aplican de manera inmediata y automática. Esto es

posible gracias a que el sistema operativo ofrece una serie de eventos en forma de notificaciones cuando uno de estos ficheros XML de configuración en modificado. EN ningún caso será necesario intervenir en el funcionamiento del servicio Web.

ASP .NET configura de manera automática el servidor Web IIS para que no sea posible que un cliente realice una petición a un fichero web.config, y de esta forma pueda ver la configuración de nuestra aplicación ASP .NET, con el consiguiente peligro para la seguridad de nuestra aplicación y sistema. Si un usuario realiza una solicitud al fichero web.config se le mostrará un mensaje de acceso denegado, tal como aparece en la Figura 180.

Figura 180

Una vez comentado como aplica ASP .NET la configuración sobre sus aplicaciones, vamos a pasar a describir el formato de los ficheros de configuración.

Formato de los ficheros de configuración

Como ya se ha mencionado en varias ocasiones, existen dos tipos de ficheros XML dentro del sistema de configuración de ASP .NET, el fichero machine.config y el fichero web.config. Estos dos ficheros se diferencian únicamente en el nombre, ya que internamente presentan el mismo formato XML.

Se debe tener en cuenta que el sistema de configuración de ASP .NET distingue entre mayúsculas y minúsculas.

Cada fichero de configuración contiene una jerarquía anidada de etiquetas y subetiquetas XML con atributos que especifican los valores de configuración. Debido a que las etiquetas XML deben encontrarse bien construidas, distinguen entre mayúsculas y minúsculas, siguiendo el siguiente criterio:

- Los nombres de etiquetas y atributos comienzan con letra minúscula y las primeras letras de cada una de las siguientes palabras que la formen irán en mayúscula.

- Los valores de los atributos de las etiquetas comienzan siempre por letra mayúscula y las primeras letras de cada una de las siguientes palabras que la formen irán también en

mayúscula. Una excepción son los valores true y false, que se presentan con todas las letras minúsculas.

El elemento raíz de los ficheros de configuración es siempre el elemento <configuration> (Código fuente 284), dentro de este elemento podemos encontrar dos secciones generales: sección de los manejadores y sección de los valores de configuración.

```
<configuration>
<!-- Configuración para aplicar a la aplicación ASP .NET -->
</configuration>
```

Código fuente 284

Antes de seguir con la descripción del formato de los ficheros de configuración de ASP .NET vamos a mostrar en el Código fuente 285 un fichero de configuración web.config que presenta las dos secciones generales, la que aparece en negrita es la sección de los manejadores y la que aparece en cursiva es la sección de los valores de configuración. En concreto, desde el punto de vista de las aplicaciones ASP .NET, a nosotros nos va a interesar la sección <system.web> que se encuentra en la sección general de los valores de configuración.

```
<configuration>
   <configSections>
      <section name="appSettings" type=
         "System.Web.Configuration.NameValueSectionHandler" />
      <sectionGroup name="system.web">
         <section name="sessionState" type=
            "System.Web.Configuration.SessionStateConfigHandler" />
      </sectionGroup>
   </configSections>

   <appSettings>
      <add key="dsn" value="localhost;uid=MyId;pwd=;" />
      <add key="msmqserver" value="server\myqueue" />
   </appSettings>

    <system.web>
       <sessionState cookieless="true" timeout="10" />
    </system.web>
</configuration>
```

Código fuente 285

En los siguientes subapartados vamos a comentar en detalle las dos grandes secciones que presentan los ficheros de configuración.

Sección de manejadores

La sección de los manejadores, dentro de los ficheros de configuración de ASP .NET, identifican las clases del .NET Framework que se utilizarán cuando el sistema de configuración se carga. Esta sección se encuentra entre las etiquetas <configSections>.

La función de estas clases es la de leer los valores de la sección de los valores de configuración que les corresponda.

El atributo name de la etiqueta <section> define el nombre de la etiqueta del elemento de la sección de los valores de configuración del que se va a encargar el manejador, cuya clase especificamos en el atributo type, dentro de este atributo además se indica el assembly en el que se encuentra la clase junto con su versión correspondiente.

Si se desea definir manejadores para una sección de valores de configuración que a su vez va a tener varias secciones, las distintas etiquetas <section> irán incluidas entre las etiquetas <sectionGroup>.

En el Código fuente 286 se muestra un fragmento del fichero machine.config que se encarga de la configuración general de todas las aplicaciones ASP .NET del servidor. Este fragmento se corresponde con la definición de dos manejadores para las secciones sessionState y trace, que a su vez pertenecen al grupo <system.web>, también se define el manejador para la sección appSettings.

```
<?xml version="1.0" encoding="UTF-8" ?>

<configuration>
    <configSections>
        <section name="appSettings"
         type="System.Configuration.NameValueFileSectionHandler,System,
         Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
        <sectionGroup name="system.web">
            <section name="sessionState"
              type="System.Web.SessionState.SessionStateSectionHandler, System.Web,
              Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
              allowDefinition="MachineToApplication" />
            <section name="trace"
              type="System.Web.Configuration.TraceConfigurationHandler, System.Web,
              Version=1.0.2411.0, Culture=neutral,
              PublicKeyToken=b03f5f7f11d50a3a" />
        </sectionGroup>
    </configSections>

… … … …

</configuration>
```

Código fuente 286

El grupo <system.web> va a ser de gran interés, ya que nos va a permitir configurar los distintos aspectos de nuestras aplicaciones ASP .NET.

Una vez que se ha declarado la sección de los manejadores no es necesario volver a declararla en los fichero web.config, ya que si se encuentran en el fichero machine.config, o en un fichero web.config de nivel superior, la heredarán de manera automática.

Podemos indicar que algunos de los valores presentes en el fichero machine.config no puedan ser sobrescritos por los valores de configuración de los ficheros web.config.

Sección de valores de configuración

Esta es la segunda sección principal que nos podemos encontrar dentro de un fichero de configuración de ASP .NET. Mientras que la sección de los manejadores definen clases, esta sección identifican las

propiedades que afectan al comportamiento de la aplicación ASP .NET. En muchos de los casos necesitaremos saber únicamente el significado de la opción que vamos a modificar, por ejemplo los valores para el elemento <sessionState> (este elemento lo comentamos en el capítulo anterior cuando tratamos el objeto Session).

En el Código fuente 287 se muestra un fragmento del fichero machine.config, en este caso se define el manejador para la sección sessionState, y dentro de la sección sessionState se definen los valores de las propiedades que van a permitir configurar el estado de la sesión para la aplicación ASP .NET.

```
<?xml version="1.0" encoding="UTF-8" ?>

<configuration>
    <configSections>
        <sectionGroup name="system.web">
            <section name="sessionState"
              type="System.Web.SessionState.SessionStateSectionHandler, System.Web,
              Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
              allowDefinition="MachineToApplication" />
        </sectionGroup>
… … … …

    </configSections>
… … … …
    <system.web>
        <sessionState
            mode="InProc"
            stateConnectionString="tcpip=127.0.0.1:42424"
            sqlConnectionString="data source=127.0.0.1;user id=sa;password="
            cookieless="false"
            timeout="20"
        />
    </system.web>
</configuration>
```

Código fuente 287

Normalmente los valores de configuración de la aplicación ASP .NET no los vamos a indicar en el fichero machine.config, ya que estos valores afectarán a todas las aplicaciones ASP .NET del servidor, sino que utilizaremos un fichero web.config particular para una aplicación ASP .NET determinada. En este fichero web.config heredaremos la sección de los manejadores indicada en el fichero machine.config.

Así si queremos indicar la misma configuración que en el ejemplo anterior para el elemento <sessionState>, el Código fuente 288 sería el contenido de nuestro fichero web.config.

```
<configuration>
  <system.web>
    <sessionState
            mode="InProc"
            stateConnectionString="tcpip=127.0.0.1:42424"
            sqlConnectionString="data source=127.0.0.1;user id=sa;password="
            cookieless="false"
            timeout="20"
    />
  </system.web>
</configuration>
```

Código fuente 288

Una vez que ya hemos visto el formato de los ficheros de configuración, vamos a pasar a comentar las tareas de configuración más comunes que podemos realizar sobre nuestras aplicaciones ASP .NET.

Tareas comunes de configuración

Si examinamos detenidamente el fichero de configuración machine.config que tenemos en nuestro equipo, podemos encontrar alrededor de treinta opciones de configuración distintas. En este texto no vamos a ver todas estas tareas de configuración, sino que vamos a ver las más comunes y las que más se pueden usar en un entorno real de trabajo.

Todas estas opciones de configuración se encontrarán dentro de la sección de los valores de configuración del fichero de configuración correspondiente, que por lo general será el fichero web.config.

Algunas de estas tareas de configuración ya las hemos comentado en distintos capítulos del presente texto.

A continuación se describen brevemente estas tareas de configuración:

- Configuración general (<httpRuntime>): en este apartado podemos indicar el tiempo de espera que se va a aplicar a un recurso solicitado de la aplicación ASP .NET.

- Configuración de la página (<pages>): en esta tarea se nos permite indicar algunas características de las páginas ASP .NET de la aplicación: activación del búfer o del mantenimiento de estado automático (VIEWSTATE).

- Configuración de la aplicación (<appSettings>): en esta sección podemos almacenar información en pares clave/valor, estos datos quedarán almacenados en el sistema de configuración y luego podrán ser recuperados en la aplicación ASP .NET.

- Estado de la sesión (<sessionState>): nos permite configurar las distintas opciones para la utilización del objeto Session dentro de la aplicación. Esta sección ya la comentamos en un capítulo anterior.

- Configuración de trazas (<trace>): al igual que sucedía en el caso anterior, esta opción de configuración de la aplicación ASP .NET ya la vimos en un capítulo previo, y nos va a permitir configurar el mecanismo de trazas de la aplicación.

- Errores personalizados (<customErrors>): sección que ya hemos visto con anterioridad y que nos permite definir las páginas de error que se van a utilizar en la aplicación ASP .NET.

- Web Services (<webServices>): esta sección nos va a permitir configurar algunos de los aspectos de los Web Services (servicios Web) que utilizará nuestra aplicación ASP .NET, no vamos a profundizar en ello ya que el uno de los siguientes capítulos va estar dedicado por completo a los servicios Web.

- Globalización (<globalization>): en esta otra sección vamos a poder indicar el conjunto de caracteres que se van a utilizar en la aplicación.

- Compilación (<compilation>): en este apartado vamos a indicar algunos de los comportamientos de compilación de las páginas ASP .NET, como puede ser la definición del lenguaje a utilizar.

- Identidad (<identity>): nos permite configurar el usuario que va a ejecutar el proceso de ASP .NET.

- Manejadores HTTP (<httpHandlers>): los manejadores HTTP son los responsables de servir las solicitudes de una extensión particular dentro de ASP .NET, como pueden ser .ASPX o .ASMX. Dentro de esta sección se pueden añadir manejadores personalizados o bien eliminar manejadores existentes.

- Módulos HTTP (<httpModules>): los módulos HTTP son los responsables de filtrar cada solicitud/respuesta en una aplicación ASP .NET, así por ejemplo determinarán si una solicitud determinada debería ser servida desde la caché o bien dirigida al manejador HTTP correspondiente.

- Modelo de proceso (processModel): las opciones disponibles en esta sección nos van a permitir configurar el modelo de proceso aplicado al proceso ASP .NET en ejecución.

Una vez descritos brevemente algunos de las opciones de configuración de una aplicación ASP .NET, vamos a entrar en detalle en cada uno de ellos, mostrando en cada caso el código XML que debemos incorporar en nuestro fichero web.config para realizar la labor de configuración correspondiente.

Configuración general

En esta sección se va a indicar una serie de parámetros de configuración genéricos para la aplicación ASP .NET. Para ello se hace uso de la etiqueta <httpRuntime>. Esta etiqueta presenta los siguientes atributos:

- executionTimeout: en este atributo indicaremos en segundos el tiempo de espera que se aplicará a la ejecución de un recurso solicitado, una vez sobrepasado este tiempo de espera la aplicación ASP .NET finalizará la ejecución del recurso. El valor por defecto de este atributo es de 90 segundos. Este atributo es similar al parámetro, disponible en las opciones de configuración de una aplicación ASP dentro de IIS, Tiempo de espera de archivo de comandos de ASP (ScriptTimeout).

- maxRequestLength: indica en KB el tamaño máximo de una petición, por defecto el valor que toma este atributo es de 4099 KB.

- UserFullyQualifiedRedirectUrl: indica si al cliente se le va a devolver una URL completa o una URL relativa, por defecto tiene el valor false, por lo que se enviará una URL relativa.

En el Código fuente 289 se ofrece un ejemplo de utilización de esta sección.

```
<configuration>
  <system.web>
    <httpRuntime
      executionTimeout="90"
      maxRequestLength="4096"
      useFullyQualifiedRedirectUrl="false"
    />
  </system.web>
</configuration>
```

Código fuente 289

Estos son los valores que presenta por defecto esta sección dentro del fichero machine.config.

A continuación vamos a tratar una sección más específica de la configuración de las páginas ASP .NET dentro de una aplicación.

Configuración de la página

Mediante la etiqueta <pages> podemos controlar algunos de los comportamientos de las páginas ASP .NET presentes en una aplicación.

La etiqueta <pages> ofrece los siguientes atributos:

- autoEventWireup: este atributo puede presentar los valores true/false, por defecto tiene el valor true, e indica si los eventos de la página (UnLoad, Load, Init, etc.) se van a lanzar de forma automática.

- buffer: este atributo al igual que el anterior posee los valores true/false y se utilizará para activa o desactivar el búfer de las páginas ASP .NET de la aplicación actual. El mecanismo de búfer ya existía en versiones anteriores de ASP. Si le asignamos el valor true (valor por defecto) conseguiremos que ASP .NET ejecute por completo las páginas antes de enviar algo al usuario. El contenido del búfer no es enviado al navegador hasta que no se haya terminado de ejecutar la página.

- enableSessionState: este atributo permite activar o desactivar el estado de sesión, es decir, permite o no la utilización del objeto Session para almacenar información común a la sesión actual del usuario con la aplicación Web. Este atributo puede tener los valores true/false, para indicar si está disponible el estado de sesión a través del objeto Session (instancia de la clase System.Web.HttpSessionState), y también puede tener el valor readOnly, cuando se le asigna este valor las páginas podrán leer variables de sesión pero no modificarlas ni crear variables nuevas. Por defecto este atributo tiene el valor true.

- enableViewState: permite activar o desactivar el mantenimiento automático de los valores de los controles Web dentro de los formularios Web, por defecto tiene el valor true.

- enableViewStateMac: indica si a la hora de mantener el estado de los controles Web se debe realizar mediante un mecanismo de autenticación de la máquina cliente. El valor por defecto de este atributo es false.

Estos atributos se corresponden con los atributos del mismo nombre de la directiva @Page, por lo que utilizando esta directiva podemos sobrescribir estos valores de configuración para una página ASP .NET en concreto. En el Código fuente 290 se ofrece el fragmento del fichero machine.config que se corresponde con la sección que estamos tratando.

```
<configuration>
  <system.web>
    <pages buffer="true" enableSessionState="true" enableViewState="true"
     enableViewStateMac="false" autoEventWireup="true" />
  </system.web>
</configuration>
```

Código fuente 290

Si en el fichero web.config de una aplicación concreta deseamos sobrescribir algunos de los valores anteriores, no tenemos nada más que incluir la etiqueta con sus atributos correspondientes. Así por ejemplo si en nuestra aplicación ASP .NET deseamos desactivar el búfer nuestro fichero web.config debería contener al menos el Código fuente 291.

```
<configuration>
  <system.web>
    <pages buffer="false" />
  </system.web>
</configuration>
```

Código fuente 291

Esta forma de sobrescribir valores de configuración va a ser lo común en las aplicaciones ASP .NET.

Configuración de la aplicación

En esta sección vamos a poder almacenar valores de detalles de configuración de la aplicación, sin necesidad de tener que definir nuestros propio manejador de sección de configuración. Para almacenar estas parámetros dentro del fichero de configuración vamos a utilizar pares clave/valor.

Estos valores de configuración definidos en la sección <appSettings> los vamos a poder recuperar en las páginas ASP .NET de la aplicación. Esto lo podemos hacer gracias a que la clase System.Configuration.ConfigurationSettings nos ofrece una propiedad en forma de colección, llamada AppSettings, que nos va a permitir acceder a los parámetros definidos en el fichero de configuración.

Dentro de la etiqueta <appSettings> tenemos unas subetiquetas <add>, existirá una etiqueta <add> por cada parámetro o valor que queremos indicar en la aplicación, este etiqueta posee dos atributos: key, que es la clave con la que después vamos a poder acceder a este parámetro a través de la colección AppSettings y value que va a ser el valor que le vamos a asignar al parámetro.

En el Código fuente 292 se muestra el contenido de un fichero web.config en el que se definen dos parámetros para la aplicación, se indica una cadena de conexión y una sentencia SQL.

```
<configuration>
  <appSettings>
      <add key="conexion" value="server=angel;database=northwind;uid=sa;pwd="/>
      <add key="sentencia" value="select firstname, lastname,city from Employees"/>
  </appSettings>
</configuration>
```

Código fuente 292

Como se puede comprobar esta etiqueta de configuración de la aplicación no se encuentra dentro de la sección <system.web>.

En el Código fuente 293 se muestra el código de una página ASP .NET dentro de la aplicación que recupera y utiliza los parámetros almacenados en el fichero web.config. Los valores almacenados los utiliza para establecer la conexión con un base de datos y ejecutar la sentencia SQL para mostrar los resultados de la misma en un control Web DataGrid.

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script runat="server">
void Page_Load(Object sender, EventArgs e) {
      String cadenaConexion=ConfigurationSettings.AppSettings["conexion"];
      String sentencia=ConfigurationSettings.AppSettings["sentencia"];
      SqlConnection conexion =
            new SqlConnection(cadenaConexion);
      SqlDataAdapter comando =
            new SqlDataAdapter(sentencia, conexion);
      DataSet ds = new DataSet();
      comando.Fill(ds, "Employees");
      tabla.DataSource = ds.Tables["Employees"].DefaultView;
      tabla.DataBind();
}
</script>

<body>
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow">
</asp:DataGrid>
</body>
</html>
```

Código fuente 293

Si intentamos acceder en la colección AppSettings un elemento que no ha sido declarado como parámetro de la aplicación en el fichero de configuración de la misma nos devolverá el valor null.

Por defecto en el fichero machine.config no se almacena ningún valor en esta sección y la misma aparece vacía.

Configuración de la sesión

En el apartado anterior ya vimos en que consistía el estado de sesión y como se podría utilizar desde las páginas ASP .NET. Desde el fichero de configuración de la aplicación ASP .NET también tenemos la posibilidad de configurar la forma en la que se va a utilizar el estado de sesión mediante la etiqueta <sessionState>, para ello presenta los siguientes atributos:

- mode: indica el modo de almacenamiento utilizado para el proceso que se corresponde con el estado de la sesión. Los valores que podemos asignar a este atributo son: InProc (el estado de sesión se encuentra en el proceso actual de ASP .NET), Off (el estado de sesión se encuentra desactivado), SQLServer (se utiliza un proceso de SQL Server para almacenar el estado), StateServer (se utiliza un proceso en forma de servicio de Windows para almacenar el estado). El valor por defecto es InProc. El valor de este atributo lo podemos consultar a través de la propiedad Mode del objeto Session.

- stateConnectionString: en este atributo se indica la dirección IP y el número de puerto utilizados para comunicarse con el servicio de Windows que ofrece las facilidades de almacenamiento, este atributo únicamente tiene sentido utilizarlo cuando el atributo mode tiene el valor de StateServer, es decir, cuando el proceso de la sesión se va a encontrar en un servicio de Windows.

- SqlConnectionString: identifica la cadena de conexión de la base de datos utilizada para almacenar el estado cuando el atributo mode posee el valor SQLServer. Esta cadena debe incluir la dirección IP y el nombre y contraseña de usuario para conectar a la base de datos de SQL Server.

- cookieless: atributo que indica si el objeto Session utiliza para almacenar el identificador de sesión el mecanismo de cookies, o por el contrario no utiliza cookies y el identificador de sesión lo va manteniendo a través del mecanismo de URLs, el mecanismo de URLs será utilizado cuando el valor de este atributo sea true. Por defecto el valor de esta atributo es false. El valor de esta atributo se puede consultar a través de la propiedad IsCookieless del objeto Session.

- Timeout: el atributo timeout especifica el intervalo de inactividad para el objeto Session en minutos. Si el usuario no actualiza o solicita una página durante ese intervalo, la sesión termina. El valor por defecto de este atributo es de 20 minutos, es decir, por defecto la sesión permanecerá inactiva 20 minutos. Una sesión se dice que está inactiva mientras el navegador cliente no realice una petición. El valor de esta atributo se puede modificar dinámicamente a lo largo de la ejecución de la aplicación ASP .NET, para ello podemos utilizar Timeout del objeto Session.

En el Código fuente 294 se ofrece el fragmento que se corresponde con la sección <sessionState> y que podemos encontrar el fichero machine.config. Incluso se nos ofrece un comentario acerca de los atributos que podemos utilizar en esta etiqueta junto con sus correspondientes valores válidos.

```
<configuration>
  <system.web>
     <!-- sessionState Attributes:
          mode="[Off|InProc|StateServer|SQLServer]"
          stateConnectionString="tcpip=server:port"
          sqlConnectionString="valid System.Data.SqlClient.SqlConnection string,
                               minus Initial Catalog"
          cookieless="[true|false]"
          timeout="timeout in minutes"
     -->
     <sessionState
            mode="InProc"
            stateConnectionString="tcpip=127.0.0.1:42424"
            sqlConnectionString="data source=127.0.0.1;user id=sa;password="
            cookieless="false"
            timeout="20"
     />
  </system.web>
</configuration>
```

Código fuente 294

Si deseamos aumentar el timeout y desactivar el mecanismo de cookies para mantener el identificador de sesión para una aplicación ASP .NET en particular, no tenemos nada más que incluir el Código fuente 295 dentro del fichero web.config de dicha aplicación.

```
<configuration>
  <system.web>
     <sessionState
            cookieless="false"
            timeout="20"
```

```
    />
  </system.web>
</configuration>
```

Código fuente 295

Configuración de trazas

Para poder configurar el mecanismo de trazas que nos ofrece ASP .NET a nivel de aplicación haremos uso de la etiqueta <trace>.

En el capítulo correspondiente veíamos con detenimiento el funcionamiento del mecanismo de trazas de ASP .NET, en este apartado vamos a limitarnos a su configuración desde los ficheros de configuración de la aplicación.

La etiqueta trace, que nos va a permitir habilitar y configurar el mecanismo de trazas a nivel de una aplicación ASP .NET, presenta los siguientes atributos:

- enabled: este atributo indica si el mecanismo de trazas se encuentra activado a o no, tiene por lo tanto la misma funcionalidad que el atributo de mismo nombre de la directiva @Page. Su valor por defecto es false.

- requestLimit: en este atributo indicaremos el número máximo de peticiones HTTP de las que se va a almacenar información de trazas, las trazas van a ser almacenadas en un registro de trazas mediante un mecanismo circular en las que permanecerán las últimas n peticiones. Por defecto el valor que presenta este atributo es 10.

- pageOutput: indica si la información de trazas se va a mostrar al final de cada página ASP .NET, tal como se hace con las trazas a nivel de página. Aunque la información de las trazas no se muestre en la página, quedará constancia de las mismas en el registro de trazas. Su valor por defecto es false.

- traceMode: este atributo nos permite indicar el modo de ordenación de los mensajes de trazas en la sección Información de seguimiento, tiene por lo tanto la misma funcionalidad que el atributo de mismo nombre de la directiva @Page. Puede presentar los valores SortByCategory y SortByTime, su valor por defecto es SortByTime.

- localOnly: indica si la información de trazas se muestra únicamente a los clientes locales o por el contrario se muestra también a los clientes remotos. Su valor por defecto es false.

El Código fuente 296 es el código que se corresponde con los valores por defecto que presenta la sección de trazas dentro del fichero machine.config. En este caso también se incluye el comentario que se ofrece en el fichero machine.config, como se puede comprobar el mecanismo de trazas se encuentra por defecto deshabilitado para la aplicación.

```
<configuration>
  <system.web>
       <!--
        trace Attributes:
          enabled="[true|false]" - Enable application tracing
          localOnly="[true|false]" - View trace results from localhost only
          pageOutput="[true|false]" - Display trace ouput on individual pages
          requestLimit="[number]" - Number of trace results available in trace.axd
```

```
          traceMode="[SortByTime|SortByCategory]" - Sorts trace result displays
                                                        based on Time or Category
        -->
        <trace
            enabled="false"
            localOnly="true"
            pageOutput="false"
            requestLimit="10"
            traceMode="SortByTime"
        />
  </system.web>
</configuration>
```

Código fuente 296

En el Código fuente 297 se muestra un fragmento del fichero web.config de una aplicación ASP .NET, en este caso se está activando el mecanismo de trazas para la aplicación actual, pero la información de las trazas se va a mostrar al final de cada página y únicamente se va a permitir acceder al registro de trazas de la aplicación a los clientes locales, las trazas se ordenarán por categoría, también hemos aumentado el número máximo de peticiones que se almacenan.

```
<configuration>
  <system.web>
    <trace
       enabled="true"
       requestLimit="20"
       pageOutput="true"
       traceMode="SortByCategory"
      localOnly="true"
    />
  </system.web>
</configuration>
```

Código fuente 297

Errores personalizados

Al igual que sucedía en la sección anterior, ya hemos abordado el tema del tratamiento de errores dentro de ASP .NET y de su personalización a través de páginas de error definidas por el usuario.

Para configurar las páginas de error desde el fichero de web.config de la aplicación ASP .NET debemos utilizar la sección <customErrors>, que nos va a permitir especificar como se comporta ASP .NET cuando se produce un error no tratado por una página ASP .NET o por la aplicación. Tenemos varias opciones para indicar como se deben mostrar las páginas de error.

Dentro de la sección <customErrors> disponemos del atributo mode, que determinará si se mostrará una página de error personalizada o por el contrario la típica página de error de ASP .NET. Para indicar las distintas posibilidades este atributo posee los siguientes valores:

- RemoteOnly: cuando asignamos este valor al atributo mode, la página de error típica de ASP .NET con la descripción completa de los errores, se mostrará únicamente a los usuarios que accedan al servidor desde la propia máquina, es decir, aquellos que utilicen la dirección IP 127.0.0.1 o el nombre de dominio localhost. A las peticiones que no sean locales se les mostrará la página de error personalizada que haya sido indicada en el atributo

defaultRedirect, este atributo también pertenece a la sección <customErrors>, tal como veremos a continuación. Este es le valor por defecto que posee el atributo mode.

- On: en este caso la página de error de ASP .NET no se mostrará nunca, sino que se mostrará la página de error personalizada que corresponda. Si no se indica ninguna página de error, aparecerá una página de error indicando como debemos configurar nuestro fichero web.config para que se muestren las páginas de error personalizadas.

- Off: ASP .NET mostrará siempre la página por defecto, que contendrá las líneas de código que han producido el error, el volcado de pila, el número de línea del fichero de origen del error, etc.

En el Código fuente 298 se muestra el aspecto que tiene por defecto dentro del fichero machine.config la etiqueta <customErrors>.

```
<configuration>
  <system.web>
    <customErrors mode="RemoteOnly" />
  </system.web>
</configuration>
```

Código fuente 298

Dentro del fichero web.config tenemos dos formas de indicar páginas de error personalizadas:

- Redirecciones por defecto: de esta forma cuando se produzca un error, sea del tipo que sea, el cliente será redirigido a la página de error indicada.

- Redirecciones personalizadas: el cliente será redirigido a una página de error personalizada determinada, cuando se produce un error del protocolo HTTP de un tipo específico, por ejemplo el error 404 cuando no se encuentra una página determinada.

Para las redirecciones por defecto utilizaremos el atributo defaultRedirect de la etiqueta customErrors, este atributo contendrá la URL a la página de error personalizada por defecto. La utilización de este atributo se puede ver en el Código fuente 299.

```
<configuration>
  <system.web>
     <customErrors mode="On" defaultRedirect="/aplicacionWeb/PaginaError.aspx"/>
  </system.web>
</configuration>
```

Código fuente 299

En este caso los cada vez que se produzca un error, independientemente del tipo que sea, los clientes remotos verán siempre la página de error llamada PaginaError.aspx.

Pero en la personalización de errores podemos ir más allá e indicar una página de error determinada que se mostrará cuando se produzca un tipo de error específico, este mecanismo se puede utilizar en combinación con el anterior, es decir, si el tipo de error no coincide con ninguno de los indicados se mostrará la página de error indicada en el atributo defaultRedirect de la etiqueta customErrors.

Para indicar las páginas de error atendiendo al tipo de error que se produce, se debe utilizar el subelemento <error> dentro de la sección customErrors. El elemento error va a poseer dos atributos que son:

- statusCode: en este atributo indicaremos el código de estado del protocolo HTTP que deseemos que coincida con el error que se ha dado. Si se encuentra una coincidencia con este código de estado al cliente se mostrará la página de error personalizada que corresponda.

- redirect: este atributo indica la página a la que se enviará al cliente en el caso de que se produzca el error coincidente con el código de estado del protocolo HTTP indicado en el atributo statusCode.

En el Código fuente 300 se muestra un ejemplo de utilización de la etiqueta <error>, en este caso los clientes remotos verán la página PaginaNoEncontrada.html en el caso de que se produzca el código de estado 404 del protocolo HTTP, este código indica que una página determinada no ha podido ser encontrada. Si no el código de estado no es el 404 al cliente se le mostrará la página de error personalizada por defecto, es decir la indicada en el atributo defaultRedirect.

```
<configuration>
  <system.web>
     <customErrors mode="On" defaultRedirect="/aplicacionWeb/PaginaError.aspx">
       <error statusCode="404" redirect="/aplicacionWeb/PaginaNoEncontrada.html"/>
     </customErrors>
  </system.web>
</configuration>
```

Código fuente 300

Se debe señalar que la utilización del atributo ErrorPage de la directiva @Page, sobrescribe los valores indicados de página de error personalizadas dentro de los ficheros de configuración, ya que estas páginas de error son para la aplicación ASP .NET, y el atributo ErrorPage nos va a permitir indicar una página de error para una página ASP .NET concreta.

Web Services

Una de las ideas centrales en los que descansa .NET es el concepto de "Web programable". De la misma forma que un componente instalado en un servidor y gestionado mediante Transaction Server o Component Server puede dar servicio a todas las máquinas del dominio de ese servidor, la idea es que los sistemas construidos utilizando datos y servicios provean de múltiples servicios Web. Los servicios Web suministran los elementos básicos para los sistemas, la Web dota de los sistemas de acceso a ellos, y los desarrolladores funden todos esos elementos en un modo útil.

Los servicios Web pueden ser específicos a una aplicación en concreto, sin embargo existen un buen número de servicios horizontales que la mayoría de las aplicaciones Web necesitan.

El consumidor de un servicio Web puede ser un navegador de Internet, otro dispositivo con conexión a Internet, o una aplicación. Incluso un servicio Web puede ser consumidor de otro servicio Web (de la misma forma que un componente COM puede utilizar otro componente como parte de su implementación y funcionamiento).

ASP .NET ofrece un entorno muy flexible a la hora de desarrollar servicios Web, y una de las posibilidades que ofrece es la de configurar algunos de los aspectos de los servicios web a través de los ficheros de configuración de la aplicación.

De momento no vamos a entra más detalles con los servicios Web, ya que un futuro capítulo va a tratar en profundidad los mismos, lo único que vamos a ver es la posibilidad que no ofrece el sistema de configuración de ASP .NET para indicar la página de ayuda que se muestra al invocar un servicio Web.

Para poder indicar la anterior opción de configuración se debe utilizar la etiqueta <wsdlHelpGenerator>, que se encontrará dentro de la sección <webservices>. En el Código fuente 301 se puede ver el valor por defecto de esta etiqueta dentro del fichero machine.config.

```
<configuration>
  <system.web>
        <webServices>
            <wsdlHelpGenerator href="DefaultWsdlHelpGenerator.aspx" />
        </webServices>
  </system.web>
</configuration>
```

Código fuente 301

Globalización

Los valores indicados en la sección <globalization> nos va a permitir configurar las opciones de codificación y cultura. La etiqueta <globalization> nos ofrece cinco atributos para indicar diversos aspectos de la codificación utilizada en nuestra aplicación, veamos una breve descripción de los mismos:

- requestEncoding: mediante este atributo podemos indicar la codificación utilizada en cada solicitud, su valor por defecto es utf-8, es decir se utiliza la codificación basada en 8 bits para aceptar todos los caracteres unicode.

- responseEncoding: este atributo tiene el mismo significado que el anterior pero aplicado una respuesta enviada al cliente. Por defecto también tiene la codificación utf-8.

- fileEncoding: permite indicar el tipo de codificación aplicado a los ficheros, también posee por defecto la codificación utf-8.

- culture: en este otro atributo podemos especificar el lugar en el que nos encontramos para que se aplique a las cadenas el idioma adecuado, así como también a las fechas y su formato, así por ejemplo en-US representa al idioma inglés de Estados Unidos y fr-FR francés en Francia.

- uiculture: en este caso se indica la información igual que en el atributo anterior, pero se va a utilizar para realizar búsquedas en las cadenas del idioma correspondiente.

En el Código fuente 302 se muestra el fragmento que se corresponde con la sección <globalization> dentro del fichero machine.config.

```
<configuration>
```

```
  <system.web>
        <globalization
                requestEncoding="utf-8"
                responseEncoding="utf-8"
        />
  </system.web>
</configuration>
```

Código fuente 302

Si deseamos indicar el idioma español localizándolo en España para un aplicación ASP .NET en el fichero web.config de la aplicación deberíamos tener el Código fuente 303.

```
<configuration>
  <system.web>
    <globalization
        culture="es-ES"
        uiCulture="es-ES"
    />
  </system.web>
</configuration>
```

Código fuente 303

Para probar esta configuración podemos intentar mostrar la fecha y hora actuales indicando también el nombre del mes y del día de la semana, esto lo conseguimos con la siguiente página ASP .NET (Código fuente 304).

```
<%@ Page language="C#"%>

<html>
<head><title>Globalización</title></head>
<script runat="server">
      void Page_Load(Object sender, EventArgs e) {
            fecha.Text=DateTime.Now.ToString("D");
      }
</script>

<body>
<asp:Label Runat="server" ID="fecha"></asp:Label>
</body>
</html>
```

Código fuente 304

En la Figura 181 se puede ver el resultado de la ejecución de la página anterior.

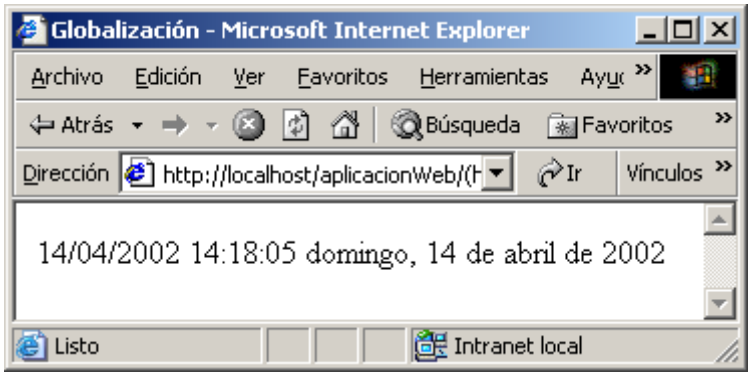


Figura 181

Compilación

No debemos olvidar que, a diferencia con sus versiones anteriores, las páginas ASP .NET se compilan de forma automática la primera vez que realizamos una solicitud de las mismas. Mediante la sección <compilation> vamos a poder indicar algunos de los parámetros utilizados a la hora de compilar una página ASP .NET.

La etiqueta compilation nos ofrece los siguientes atributos:

- debug: atributo que puede presentar los valores true/false y que permite activar la depuración en la aplicación ASP .NET actual. El mecanismo de depuración lo vimos anteriormente en detalle. Por defecto este atributo presenta el valor false.

- defaultLanguage: en este atributo indicaremos el lenguaje por defecto de la plataforma .NET que se utilizará para compilar las páginas ASP .NET. Este lenguaje puede ser sobrescrito por el atributo Language de la directiva @Page o bien por la declaración del lenguaje en los bloques <script>. El valor por defecto de este atributo es vb, para indicar que se compliará la página en el lenguaje Visual Basic .NET.

- strict: indica si se va utilizar el modo Option Strict de Visual Basic a la hora de compilar las páginas ASP .NET. Su valor por defecto es false. El modo Option Strict de Visual Basic indica que las conversiones de tipos deben ser estrictas, y por lo tanto no se realizar las conversiones automáticas.

- explicit: mediante este atributo indicaremos si la declaración de variables es obligatoria o no, es equivalente al modo Option Explicit de Visual Basic. Su valor por defecto es true.

Dentro de la etiqueta <compilation> nos podemos encontrar dos subelementos <compilers> y <assemblies>. Veamos estos dos nuevos elementos.

En el elemento compilers podemos definir los distintos compiladores que se utilizarán para compilar las páginas ASP .NET. Para indicar cada compilador utilizaremos un subelemento <compiler>. Por defecto en el fichero machine.config se indican tres compiladores, que son con los que se corresponden con los lenguajes estándar de la plataforma .NET: Visual Basic .NET, C# y JScript.

La etiqueta compiler presenta los siguientes atributos:

- language: en este atributo se indica el nombre del lenguaje al que pertenece el compilador, esta cadena es la que se utilizará en la propiedad Language de la directiva @Page y en los bloques de <script>.

- extension: define las extensiones de los ficheros que se van a utilizar para el mecanismo de Code-Behind.

- type: indica el nombre de la clase que representa al compilador que se va a encargar de compilar las páginas ASP .NET.

El segundo subelemento que encontrábamos dentro de la etiqueta <compilation> era la etiqueta <assemblies>, esta etiqueta nos va permitir indicar los assemblies que se deben añadir en la compilación de las páginas ASP .NET.

Para cada assembly se debe utilizar un sublemento <add>, esta nueva etiqueta presenta el atributo assembly para indicar el nombre del assembly sin incluir la extensión. Por defecto en el fichero machine.config se indican una serie de assemblies, por lo tanto estos assemblies se encontrarán disponibles para todas las aplicaciones ASP :NET. A continuación se comentan estos assemblies:

- mscorlib.dll: contiene las clases base, como pueden ser String, Object, int, etc., también incluye la raíz del espacio de nombres System.

- System.dll: contiene los generadores de código para C#, VB .NET y JScript. Amplia la definición del espacio con nombre System, e incluye espacios de nombre adicionales, como puede ser NET, que contiene las clases para la red.

- System.Web.dll: contiene las clases y los NameSpaces utilizados y requeridos por ASP .NET, como puede ser HttpRequest, Page o bien el especio de nombres System.Web.UI que contendrá los controles de servidor de ASP .NET.

- System.Data.dll: contiene las clases y los NameSpaces que pertenecen a ADO .NET.

- System.Web.Services.dll: contiene las clases y espacios de nombres necesarios para construir servicios Web desde ASP .NET. En el capítulo correspondiente comentaremos como crear y utilizar servicios Web desde las aplicaciones ASP .NET.

- System.Xml.dll: contiene las clases y los espacios con nombre relacionados con el lenguaje XML.

- System.Drawing.dll: contiene las clases y los NameSpaces para trabajar con imágenes.

- *: el carácter especial del asterisco indica a ASP .NET que incluya todos los assemblies que se encuentren en el directorio BIN de las aplicaciones ASP .NET. La importancia y utilización de este directorio ya la comentamos en capítulos anteriores.

En el Código fuente 305 se puede ver la sección <compilation> que nos ofrece por defecto el fichero machine.config.

```
<configuration>
  <system.web>
    <compilation debug="false" explicit="true" defaultLanguage="vb">
      <compilers>
          <compiler language="c#;cs;csharp" extension=".cs"
            type="Microsoft.CSharp.CSharpCodeProvider, System, Version=1.0.2411.0,
            Culture=neutral, PublicKeyToken=b77a5c561934e089" />
          <compiler language="vb;visualbasic;vbscript" extension=".vb"
            type="Microsoft.VisualBasic.VBCodeProvider, System, Version=1.0.2411.0,
            Culture=neutral, PublicKeyToken=b77a5c561934e089" />
```

```
          <compiler language="js;jscript;javascript" extension=".js"
            type="Microsoft.JScript.JScriptCodeProvider, Microsoft.JScript" />
      </compilers>

      <assemblies>
        <add assembly="mscorlib"/>
        <add assembly="System, Version=1.0.2411.0, Culture=neutral,
                                              PublicKeyToken=b77a5c561934e089"/>
        <add assembly="System.Web, Version=1.0.2411.0, Culture=neutral,
                                              PublicKeyToken=b03f5f7f11d50a3a"/>
        <add assembly="System.Data, Version=1.0.2411.0, Culture=neutral,
                                              PublicKeyToken=b77a5c561934e089"/>
        <add assembly="System.Web.Services, Version=1.0.2411.0, Culture=neutral,
                                              PublicKeyToken=b03f5f7f11d50a3a"/>
        <add assembly="System.Xml, Version=1.0.2411.0, Culture=neutral,
                                              PublicKeyToken=b77a5c561934e089"/>
        <add assembly="System.Drawing, Version=1.0.2411.0, Culture=neutral,
                                              PublicKeyToken=b03f5f7f11d50a3a"/>
        <add assembly="*"/>
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

Código fuente 305

Identidad

En la sección <identity> vamos a poder configurar la identidad del proceso que ejecuta ASP .NET en el servidor. Para ello esta etiqueta nos ofrece los siguientes atributos:

- impersonate: este atributo puede presentar los valores true/false. Si posee el valor true indica que el proceso de ASP .NET se va a ejecutar bajo la identidad ofrecida por Internet Information Server, que por defecto es el usuario IUSR_NombreServidor, o bien bajo el usuario que nosotros indiquemos en los siguientes atributos de la etiqueta identity. El valor por defecto de esta propiedad es false.

- name: este atributo estará disponible cuando el atributo impersonate tenga el valor true, y lo vamos a utilizar cuando deseemos indicar una cuenta de usuario de Windows específica para representar al proceso de ejecución de ASP .NET.

- password: se utiliza en combinación con el atributo anterior y en el indicaremos la contraseña del usuario que se va a utilizar en el proceso.

En el Código fuente 306 se ofrece el fragmento del fichero machine.config que se corresponde con la sección de identidad.

```
<configuration>
  <system.web>
    <identity impersonate="false" />
  </system.web>
</configuration>
```

Código fuente 306

Si deseamos indicar un usuario determinado de Windows para una aplicación ASP .NET en concreto, deberíamos incluir el Código fuente 307 en nuestro fichero web.config.

```
<configuration>
  <system.web>
    <identity impersonate="true" user="aesteban" password="xxx" />
  </system.web>
</configuration>
```

Código fuente 307

Manejadores HTTP

ASP .NET se basa en una arquitectura ampliable denominada entorno de ejecución HTTP (HTTP Runtime). Este entorno de ejecución es el responsable de manejar las solicitudes y enviar las respuestas. Son los manejadores los que van a determinar el trabajo que se debe realizar al recibir una solicitud determinada.

Para indicar estos manejadores haremos uso de la etiqueta <httpHandlers>. Para indicar cada uno de los manejadores HTTP de los que se va a hacer uso se utiliza una subetiqueta <add>, esta etiqueta posee los siguientes atributos:

- verb: este atributo indica el tipo de verbo HTTP para el que el manejador va a procesar la solicitud. Esta atributo puede tener los valores Get, Post, Head, etc., y también el carácter especial de asterisco (*) para indicar que el manejador tratarán todos los verbos del protocolo HTTP.

- path: atributo que indica la ruta del recurso que se debe tratar, al igual que en el caso anterior acepta también el carácter especial de asterisco. Por lo general se suele indicar únicamente el asterisco seguido de la extensión de los ficheros que se van a tratar.

- type: en este atributo vamos a indicar la clase que implementa el manejador HTTP. El valor indicado para este atributo sigue el formato [NameSpace].[Calse],[Nombre del assembly].

En el fichero machine.config se indican una serie de manejadores HTTP, que son lo básicos para ASP .NET. Entre ellos podemos encontrar los manejadores para páginas ASP .NET (extensión .aspx), para servicios Web (.asmx), controles de usuario (.ascx) y ficheros de configuración (.config).

En los casos de los servicios Web y de las páginas ASP .NET se indica la clase correspondiente que se va a encargar de tratar la solicitud, mientras que el caso de los controles de usuario y de los fichero de configuración se les asigna una manejador especial de la clase HttpForbiddenHandler. Este manejador deniega el acceso a esas extensiones cuando son solicitadas de forma directa.

En el Código fuente 308 se ofrece el contenido de la sección <httpHandlers> que nos ofrece el fichero machine.config.

```
<configuration>
  <system.web>
   <add verb="*" path="trace.axd" type="System.Web.Handlers.TraceHandler,
     System.Web, Version=1.0.2411.0, Culture=neutral,
     PublicKeyToken=b03f5f7f11d50a3a" />
```

```
    <add verb="*" path="*.aspx" type="System.Web.UI.PageHandlerFactory, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.ashx" type="System.Web.UI.SimpleHandlerFactory,
      System.Web, Version=1.0.2411.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.asmx"
      type="System.Web.Services.Protocols.WebServiceHandlerFactory,
      System.Web.Services, Version=1.0.2411.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" validate="false"/>
    <add verb="*" path="*.vsdisco"
      type="System.Web.Services.Discovery.DiscoveryRequestHandler,
      System.Web.Services,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      validate="false"/>
    <add verb="*" path="*.rem"
      type="System.Runtime.Remoting.Channels.Http.HttpRemotingHandlerFactory,
      System.Runtime.Remoting, Version=1.0.2411.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089" validate="false"/>
    <add verb="*" path="*.soap"
      type="System.Runtime.Remoting.Channels.Http.HttpRemotingHandlerFactory,
      System.Runtime.Remoting, Version=1.0.2411.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089" validate="false"/>
    <add verb="*" path="*.asax" type="System.Web.HttpForbiddenHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.ascx" type="System.Web.HttpForbiddenHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.config" type="System.Web.HttpForbiddenHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.cs" type="System.Web.HttpForbiddenHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.csproj" type="System.Web.HttpForbiddenHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.vb" type="System.Web.HttpForbiddenHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.vbproj" type="System.Web.HttpForbiddenHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.webinfo" type="System.Web.HttpForbiddenHandler,
      System.Web, Version=1.0.2411.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*.asp" type="System.Web.HttpForbiddenHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="GET,HEAD" path="*" type="System.Web.StaticFileHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    <add verb="*" path="*" type="System.Web.HttpMethodNotAllowedHandler, System.Web,
      Version=1.0.2411.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
  </system.web>
</configuration>
```

Código fuente 308

En el código anterior se puede ver también el manejador encargado de mostrarnos toda la información de trazas a nivel de aplicación, solicitar el recurso trace.axd se cargará el manejador que nos ofrece la información de las trazas.

Podemos añadir nuestros propios manejadores incluyendo una nueva etiqueta <add> en la sección de los manejadores HTTP, que haga referencia a la clase que implementa nuestro manejador.

Vamos a construir como ejemplo un manejador muy sencillo, cuya función es la de mostrar un saludo junto al nombre que se para por el QueryString a una página específica. En el Código fuente 309 se muestra el contenido de esta clase.

```
using System;
using System.Web;

namespace Componentes.Ejemplos
{

	public class ManejadorHolaMundo:IHttpHandler
	{

		public void ProcessRequest(HttpContext contexto)
		{
			HttpRequest peticion=contexto.Request;
			HttpResponse respuesta=contexto.Response;
			respuesta.Write("<html><body><h1>");
			respuesta.Write("Hola " + peticion.QueryString["nombre"]);
			respuesta.Write("</h1></body></html>");

		}

		public bool IsReusable
		{
			get{return true;}
		}
	}
}
```

Código fuente 309

Para crear una clase que realice las funciones de un manejador HTTP debemos implementar el interfaz System.Web.IHttpHandler. Y de este interfaz debemos implementar únicamente el método ProcessRequest(), que será el método responsable de procesas la petición a la que se ha asociado el manejador a través del fichero web.config. También es necesario implementar la propiedad de sólo lectura IsReusable, que simplemente devolverá el valor true.

En nuestro caso, en el método ProcessRequest() vamos a generar código HTML sencillo y vamos a recuperar mediante la colección QueryString del objeto Request, el parámetro que se ha pasado a la llamada, y a este parámetro le vamos a añadir una cadena de texto que representa el saludo.

Una vez compilada la clase y generado el assembly correspondiente, copiaremos el assembly al directorio BIN de la aplicación. Ahora sólo nos queda añadir el manejador HTTP en el fichero web.config de la aplicación ASP .NET, para ello utilizaremos el Código fuente 310.

```
<configuration>
  <system.web>
    <httpHandlers>
      <add verb="*" path="HolaManejador.aspx"
          type="Componentes.Ejemplos.ManejadorHolaMundo,ComponentesNET" />
    </httpHandlers>
  </system.web>
</configuration>
```

Código fuente 310

En este caso la clase del manejador es la clase ManejadorHolaMundo, que se encuentra en el assembly ComponentesNET.dll. Si realizamos la solicitud del recurso HolaManejador.aspx obtendremos un resultado similar al de la Figura 182.

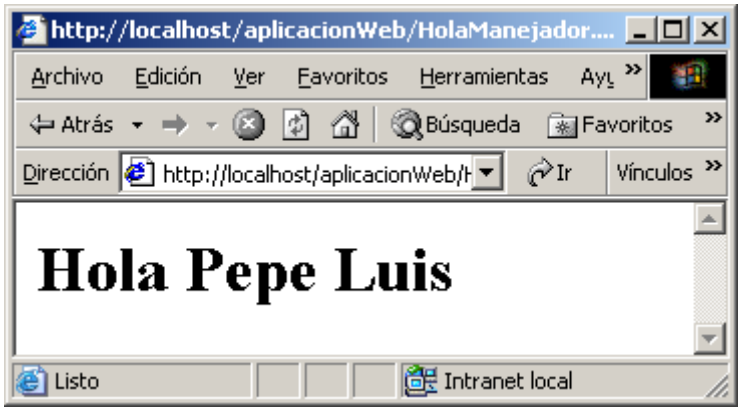


Figura 182

Modelo de proceso

A diferencia de ASP, ASP .NET se va a ejecutar en un proceso distinto al del servidor IIS, este proceso es el proceso del sistema aspnet_wp.exe. ASP .NET va a utilizar el servidor IIS únicamente para recibir solicitudes y devolver respuestas, IIS no va a ejecutar el código de las páginas ASP .NET, el encargo va a ser el proceso de trabajo de ASP .NET, es decir, aspner_wp.exe (wp, working process).

Podemos ver el proceso de trabajo de ASP .NET dentro del Administrador de tareas de Windows, tal como aparece en la Figura 183.

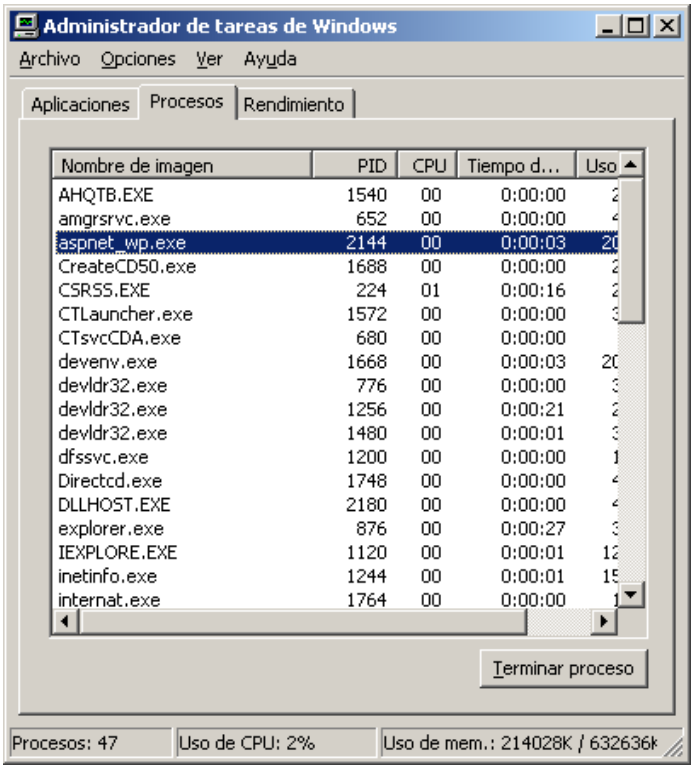


Figura 183

La sección <processModel> del fichero de configuración de la aplicación nos va a permitir configurar el proceso de trabajo de ASP .NET. El establecimiento de los valores de esta sección pude hacerse únicamente dentro del fichero machine.config, ya que los valores indicados van a aplicarse completamente a todas las aplicaciones ASP .NET presentes en el servidor.

La etiqueta processModel ofrece un gran número de atributos que nos van a permitir configurar el proceso de trabajo de ASP .NET, estos atributos son los siguientes:

- enable: presenta un valor booleano para indicar si ASP .NET se va a ejecutar en un proceso separado. Por defecto presenta el valor true.

- timeout: determina cuanto tiempo vivirá el proceso de ASP .NET antes de que un nuevo proceso sea creado y ocupe su lugar. El valor por defecto es infinite, pero también podemos indicar un tiempo determinado utilizando el formato HH:MM:SS.

- idleTimeout: este atributo nos va a permitir indicar cuando queremos que el proceso de ASP .NET finalice su ejecución de forma automática, en este atributo indicaremos el tiempo que esperará el proceso antes de apagarse si haber recibido ninguna solicitud. Su valor por defecto es infinite, es decir, no se apagará de forma automática, esperará un tiempo indeterminado hasta que se produzca la siguiente solicitud. También podemos indicar un tiempo determinado en el formato HH:MM:SS.

- shutDownTimeout: nos permite indicar el tiempo que esperará ASP .NET antes de terminar con el proceso de trabajo en el caso de que se encuentre bloqueado. El valor por defecto es de 5 segundos (00:00:05). También utiliza el formato HH:MM:SS.

- requestLimit: indica el número de solicitudes que se sirven antes de que el proceso de trabajo de ASP .NET se finalice y vuelva a iniciarse, es decir, es lo que se denomina reciclar el proceso. Por defecto no hay un límite de solicitudes (valor infinite).

- requestQueueLimit: indica el número máximo de peticiones encoladas que se sirven antes de que el proceso de ASP .NET se recicle de forma automática. El valor por defecto es de 5000.

- restartQueueLimit: indica el número de peticiones que pueden permanecer en cola mientras el proceso se está reiniciando.

- memoryLImit: indica en tanto por ciento la cantidad máxima de memoria física que el proceso puede consumir, si supera esta cantidad el proceso de trabajo de ASP .NET será reciclado. El valor por defecto es de 60, es decir, el 60% de la memoria física.

- webGarden: este atributo nos va a permitir indicar si deseamos activar el modo WebGarden. Este modo consiste en tener varios procesos de trabajo de ASP .NET ejecutándose en un mismo servidor. El valor por defecto de este atributo es false.

- cpuMask: valor hexadecimal que se utiliza para determinar que procesador se debe asignar a cada proceso de trabajo de ASP .NET cuando se está utilizando el modo WebGarden, este atributo únicamente tiene sentido cuando el atributo webGarden tiene el valor true.

- userName: permite indicar el usuario con el que se ejecutará el proceso de ASP .NET, por defecto es la cuenta SYSTEM (cuenta del sistema), pero podemos indicar un usuario determinado de Windows cuando lo consideremos necesario.

- password: en este otro atributo indicaremos la contraseña del usuario indicado en el atributo userName. El valor por defecto de este atributo es AutoGenerate, es decir, la contraseña se genera de forma automática para la cuenta del sistema.

- logLevel: indica el nivel de registro del proceso de ASP .NET, por defecto únicamente registra los errores, es decir, por defecto tiene el valor Errors. Podemos configurarlo para que registre todo los eventos (All) o para que no se registre ninguno (None).

- clientConnectedCheck: intervalo de tiempo en el que el proceso de ASP .NET comprobará si un cliente se encuentra conectado antes de que se procese la solicitud que ha realizado. El valor por defecto de este atributo es de 5 segundos (00:00:05).

- comAuthenticationLevel: indica el nivel de autenticación de la seguridad de DCOM.

- comImpersonationLevel: indica el nivel de autenticación de la seguridad de COM.

En el Código fuente 311 aparece el fragmento de código que se corresponde con la sección <processModel> del fichero machine.config.

```
<configuration>
  <system.web>
        <processModel
            enable="true"
            timeout="Infinite"
            idleTimeout="Infinite"
            shutdownTimeout="0:00:05"
            requestLimit="Infinite"
            requestQueueLimit="5000"
            restartQueueLimit="10"
            memoryLimit="60"
            webGarden="false"
            cpuMask="0xffffffff"
            userName="SYSTEM"
            password="AutoGenerate"
            logLevel="Errors"
            clientConnectedCheck="0:00:05"
            comAuthenticationLevel="Connect"
            comImpersonationLevel="Impersonate"
        />
  </system.web>
</configuration>
```

Código fuente 311

Este ha sido el último apartado que describe una de las tareas de configuración que podemos realizar sobre nuestras aplicaciones ASP .NET.

Antes de finalizar este capítulo vamos a comentar un par de aspectos avanzados acerca de la configuración de aplicaciones ASP .NET.

Indicando la localización

Ya hemos visto a lo largo de los distintos apartados que tenemos la posibilidad de configurar de forma general todas las aplicaciones del servidor mediante el fichero de configuración machine.config, y la posibilidad de configurar una aplicación en concreto mediante el fichero web.config.

Pero existe una opción más, mediante el uso de la etiqueta <location> vamos a poder indicar valores específicos para una aplicación ASP .NET determinada desde el fichero machine.config, de esta forma no será necesario incluir el fichero web.config dentro de la aplicación correspondiente.

Para indicar dentro de la etiqueta location la aplicación ASP .NET que deseamos configurar se debe utilizar el atributo path. En este atributo indicaremos la ruta a la aplicación, en el formato Sitio Web/directorio de la aplicación, siempre se utilizará la descripción del sitio Web que aparece en el Administrador de servicios de Internet.

Así si tenemos una aplicación ASP .NET en el directorio virtual aplicacionWeb que se encuentra en el sitio Web con la descripción Sitio Web predeterminado, y deseamos indicar desde el fichero machine.config una configuración específica para esa aplicación, en concreto deseamos modificar el atributo del timeout de la sesión. Para ello escribiríamos el siguiente código en el fichero machine.config (Código fuente 312).

```
<configuration>
  <location path="Sitio Web predeterminado/aplicacionWeb">
    <system.web>
      <sessionState timeout="30"/>
    </system.web>
  </location>
</configuration>
```

Código fuente 312

Entre las etiquetas de inicio y fin de location indicaremos las secciones de configuración que vamos a aplicar a una aplicación ASP .NET determinada. Como se puede comprobar las etiquetas location funcionan como una sección más dentro del fichero de configuración.

Bloqueando valores de configuración

Para terminar este capítulo vamos a tratar otro aspecto avanzado de la configuración de aplicaciones ASP .NET, en este caso se trata de bloquear valores en el fichero machine.config para que no exista la posibilidad de que sean sobrescritos en los ficheros web.config de las aplicaciones ASP .NET particulares.

Para bloquear secciones de configuración específicas y evitar sobrescritura por parte de una aplicación ASP .NET determinada, vamos hacer uso de la ya conocida etiqueta location. Esta etiqueta ofrece el atributo allowOverride para indicar si deseamos que se permita la sobrescritura o no de valores de configuración determinados.

En el Código fuente 313 se ofrece un fragmento del código del fichero machine.config, en este caso se ha bloqueado la sección que define el estado de sesión para la aplicación ASP .NET cuyo directorio virtual es aplicacionWeb y se encuentra en el Sitio Web por defecto.

```
<configuration>
    <location path="Sitio Web predeterminado/aplicacionWeb" allowOverride="false">
        <system.web>
          <sessionState
            mode="InProc"
            stateConnectionString="tcpip=127.0.0.1:42424"
            sqlConnectionString="data source=127.0.0.1;user id=sa;password="
            cookieless="false"
            timeout="20"
          />
        </system.web>
     </location>
</configuration>
```

Código fuente 313

La sección <sessionState> debe seguir existiendo dentro del fichero machine.config, aquí sólo mostramos el fragmento de código que nos va a permitir bloquear la sección para una aplicación ASP .NET determinada.

Si intentamos sobrescribir este valor de configuración se producirá una excepción, como se puede ver en la Figura 184.

Figura 184

Con este apartado finalizamos el capítulo dedicado a la configuración de aplicaciones ASP .NET, en el siguiente cambiamos completamente de orientación para ocuparnos del acceso a datos desde ASP .NET mediante ADO .NET.

20

Acceso a datos con ADO .NET

Introducción

En este capítulo vamos a tratar el acceso a datos desde páginas ASP .NET haciendo uso del nuevo modelo de acceso a datos dentro de la plataforma .NET, es decir, utilizando ADO .NET.

Este capítulo no pretende ser una referencia detallada del modelo de objetos que nos ofrece ADO .NET, ni un tutorial completo de acceso a datos con ADO .NET, el objetivo no es tan ambicioso, ya que sino resultaría en un texto más, en este capítulo lo que vamos a mostrar son las tareas básicas de acceso a datos desde páginas ASP .NET utilizando ADO .NET.

Para aquel lector que desee profundizar en la tecnología ADO .NET le recomiendo el texto de mi compañero José Luis Hevia.

ADO .NET es la nueva versión del modelo de objetos ADO (ActiveX Data Objects), es decir, la estrategia que ofrece Microsoft para el acceso a datos. ADO .NET ha sido ampliado para cubrir todas las necesidades que ADO no ofrecía, ADO .NET está diseñado para trabajar con conjuntos de datos desconectados, lo que permite reducir el tráfico de red. ADO .NET utiliza XML como formato universal de transmisión de los datos.

ADO .NET posee una serie de objetos que son los mismos que aparecen en la versión anterior de ADO, como pueden ser el objeto Connection o Command, e introduce nuevos objetos tales como el objeto DataReader, DataSet o DataView.

ADO .NET se puede definir como:

- Un conjunto de interfaces, clases, estructuras y enumeraciones que permiten el acceso a los datos desde la plataforma .NET de Microsoft
- Que supone una evolución lógica del API ADO tradicional de Microsoft
- Que permite un modo de acceso desconectado a los datos que pueden provenir de múltiples fuentes de datos de diferente arquitectura de almacenamiento
- Y que soporta un completo modelo de programación y adaptación basado en el estándar XML

En una primera parte vamos a realizar una descripción de ADO .NET y más tarde veremos como utilizarlo desde nuestras páginas ASP .NET.

Comparativa de ADO /ADO .NET

Como un buen punto de partida para comprender la importancia del nuevo diseño de ADO .NET, creo que puede estar bien analizar en qué se diferencian ADO .NET del modelo ADO vigente hasta la fecha. Las diferencias existentes son muchas y van desde el mismo diseño de los interfaces de las clases hasta el nivel estructural de los componentes y en cómo manejan la información. Examinando las diferencias que existen, queda la siguiente tabla (Tabla 12).

Clases de objetos mas estructuradas	El modelo de objetos de ADO .NET es mucho más rico que el de ADO. Incorpora nuevas clases de datos y encapsula mucha más potencia de uso a la par que corrige pequeños defectos de las versiones anteriores.
Independiente del lenguaje	Aprovechando la nueva arquitectura de servicios de la plataforma .NET, ADO .NET puede ser usado como un servicio del sistema, pudiendo ser utilizado por cualquier aplicación escrita en cualquier lenguaje.
Representaciones en memoria de los datos	En ADO .NET se emplean los DataSets en lo que en ADO se emplea Recordset. No es sólo un cambio de nombre. En memoria y en rendimiento las cosas han cambiado mucho.
Número de tablas	Un Recordset de ADO sólo puede contener un único resultado (fruto de consultas complejas o no). En cambio, un DataSet puede representar un espacio de múltiples tablas simultáneamente. Esto permite obtener una representación de espejo de la base de datos a la que representa.

Navegación mejorada	En ADO sólo nos podemos mover secuencialmente fila a fila. En ADO .NET podremos navegar fila en un DataSet y consecuentemente avanzar en una fila/conjunto de filas de otro DataSet asociado a través de una colección de relaciones.
Acceso a datos Off-line (Desconectados)	En ADO .NET todos los accesos a datos se realizan en contextos desconectados de la base de datos. En ADO, es posible tener estructuras desconectadas, pero es una elección a tomar. Por defecto, ADO está pensado para contextos con conexión. Además, ADO .NET realiza todos los accesos a los servicios de datos a través del objeto DataSetCommand, lo que permite personalizar cada comando en función del rendimiento y la necesidad del programa (en ADO, el Recordset puede modificar datos a través de su API de cursores, lo que muchas veces no es óptimo).
Compartición de datos entre capas o bien entre componentes	El traspaso de un recordset Off-line en ADO es más complejo y pesado pues es necesario establecer un vínculo RPC y un proceso de COM marshalling (verificación de tipos de datos remotos). En ADO .NET para la comunicación entre componentes y capas se emplea un simple stream XML.
Tipos de datos más ricos	Debido al COM Marshalling el tipo de datos que se pueden usar vía RPC está muy limitado. En ADO .NET, gracias a los flujos XML, es posible enviar cualquier tipo de dato de manera sencilla.
Rendimiento	Tanto ADO como ADO .NET requieren de muchos recursos cuando se habla de envíos masivos de datos. El stress del sistema es proporcional al número de filas que se quieren procesar. Pero ADO .NET permite utilizar mucho más eficientemente los recursos, pues no tiene que realizar el COM Marshalling de datos, en lo que ADO sí (y eso supone un canal más de envío de datos de control que ADO .NET ahorra).

Penetración de firewalls	Por motivos de seguridad, las empresas suelen bloquear los sistemas de comunicaciones vía RPC. O bien, implementan pesados mecanismos de control de acceso y envío de la información. Lo que complica los diseños, sus rendimientos y su instalación y distribución. En ADO .NET, puesto que no se realizan llamadas al sistema, sino que se envían flujos de datos en XML, se simplifica enormemente la configuración de estos dispositivos.

Tabla 12

De esta tabla podemos sacar muy buenas conclusiones en cuanto a las mejoras introducidas en el nuevo modelo ADO .NET. Se puede resumir en un mejor mecanismo de comunicación entre procesos gracias a XML y una independencia del cliente del servidor que posibilita el funcionamiento autónomo de la aplicación (mejor tolerancia a fallos, independencia del estado de la red).

Beneficios de ADO .NET

ADO .NET ofrece una buena cantidad de mejoras respecto a modelos anteriores de ADO. Los beneficios los podremos agrupar en las siguientes categorías:

- **Interoperabilidad**

  Las aplicaciones basadas en ADO .NET recogen la ventaja de la flexibilidad y la masiva aceptación del estándar XML para el intercambio de datos. Puesto que XML es el estándar de envío de información entre capas, cualquier componente capaz de Interpretar los datos XML puede acceder a la información de ADO .NET se encuentre donde se encuentre, y procesarla. Además, puesto que la información se envía en flujos de XML, no importa la implementación empleada para enviar o recoger la información –así como la plataforma empleada-. Simplemente se exige a los componentes que reconozcan el formato XML empleado para el proceso, envío y recepción de un DataSet.

- **Mantenimiento**

  En el ciclo de vida de una aplicación los cambios poco sustanciales y modestos son permisibles. Pero cuando es necesario abordar un cambio estructural o arquitectónico del sistema, la tarea se vuelve demasiado compleja y a veces inviable. Esto es una gran desventaja de los sistemas actuales, pues muchas veces esto es una necesidad de actualización de los procesos de la propia empresa. Además, cuanto más se aumenta el proceso de la operativa de la empresa, las necesidades de proceso crecen hasta desbordar las máquinas. Es por ello que se separa la estructura de un programa en varias capas. Una de esas capas es la de datos, que es fundamental desarrollar correctamente. Gracias a los DataSets, la tarea de portar y aumentar los procesos de datos y de negocio será mas sencillo: el intercambio de información a través de XML, hace que sea más sencilla la tarea de estructurar en más capas la aplicación, lo que la hace mucho más modular y mantenible.

- **Programación**

  Los programadores pueden acceder a un API de programación estructurado, de fuerte tipificado y que además se centra en la correcta forma de presentar las cosas. Centra en la

estructura del lenguaje lo que un programador necesita para diseñar los programas sin dar muchos rodeos. Un ejemplo de código sin tipificar aparece en el Código fuente 314.

```
If CosteTotal > Table("Cliente")("Hevia").Column("CreditoDisponible") then
```

Código fuente 314

Como se puede observar, aparecen nombres de objetos genéricos del sistema que complican la lectura del código, a la par que los operadores complican también la visión de la secuencia de acceso a los datos. Podríamos interpretar lo que hace gracias a que aparecen los nombres propios de los datos que necesitamos. Veamos un ejemplo (Código fuente 315), un poco más tipificado

```
If CosteTotal> DataSet1.Cliente("Hevia").CreditoDisponible then
```

Código fuente 315

El ejemplo es exactamente igual al anterior, pero en este caso, el código se centra más en los objetos reales que en el objeto del lenguaje en sí: las palabras "Table" y "column" ya no aparecen. En su lugar vemos que aparecen los nombres de los objetos empleados de la vida real, lo que hace el código más legible. Si a esto unimos que los entornos ya son capaces de ayudarnos a escribir el código, todavía lo tenemos más sencillo, ya que podemos ver con nuestras palabras el modelo de objetos de datos que necesitamos en cada momento. Incluso a nivel de ejecución nos vemos respaldado por un sistema de control de tipos y errores que nos permitirán proporcionar una robustez "innata" que antes no se tenía sin pasar por el uso de funciones externas.

- **Rendimiento**

  Puesto que trabajamos con objetos de datos desconectados, todo el proceso se acelera, ya que no tenemos que estar comunicándonos por Marshalling con el servidor. Además, gracias al modelo de XML la conversión de tipos no es necesaria a nivel de COM. Se reduce pues el ancho de banda disponible, se independiza más el cliente del servidor y se descarga más a éste, que puede estar dedicado a otras tareas en lo que el cliente analiza sus datos.

- **Escalabilidad**

  Las aplicaciones Web tienen un número ilimitado de conexiones potenciales debido a la naturaleza de Internet. Los servidores son capaces de atender muy bien decenas y decenas de conexiones. Pero cuando hablamos de miles y millones, los servidores ya no son capaces de realizar correctamente su trabajo. Esto es debido a que por cada usuario se mantiene una memoria de proceso y conexión, un conjunto de bloqueos de recursos como puedan ser tablas, índices... y una comprobación de sus permisos. Lo que lleva su tiempo y recursos. ADO .NET favorece la escalabilidad puesto que su modelo de conexión Off-Line evita que se mantengan los recursos reservados más tiempo del considerado necesario. Y esto permite que más usuarios por unidad de tiempo puedan acceder a la aplicación sin problemas de tiempos. Además se pueden montar servicios en Cluster de alta disponibilidad que serán balanceados automáticamente por el sistema sin afectar a las conexiones ADO. Lo cual garantiza la ampliación del servicio sin representar un cambio de arquitectura de diseño.

Arquitectura de datos desconectados

ADO .NET está basado en una arquitectura desconectada de los datos. En una aplicación de datos se ha comprobado que mantener los recursos reservado mucho tiempo implica reducir el número de usuarios conectados y aumenta el proceso del sistema al mantener una política de bloqueos y transacciones. Al mismo tiempo, si la aplicación mantiene más de un objeto simultáneamente, se encuentra con el problema de tener que estar continuamente conectando con el servidor para alimentar las relaciones existentes entre ambas, subiendo y bajando información vía RPC.

Con ADO .NET se consigue estar conectado al servidor sólo estrictamente necesario para realizar la operación de carga de los datos en el DataSet. De esta manera se reducen los bloqueos y las conexiones a la mínima expresión. Se pueden soportar muchos más usuarios por unidad de tiempo y disminuyen los tiempos de respuesta, a la par que se aceleran las ejecuciones de los programas.

De siempre, el recoger información de una base de datos ha ido destinado a realizar un proceso con dicha información: mostrarla por pantalla, procesarla o enviarla a algún componente. Frecuentemente, la aplicación no necesita una única fila, sino un buen conjunto de ellas. Además, también frecuentemente, ese conjunto de filas procede no de una tabla sino de una unión de múltiples tablas (join de tablas). Una vez que estos datos son cargados, la aplicación los trata como un bloque compacto. En un modelo desconectado, es inviable el tener que conectar con la base de datos cada vez que avanzamos un registro para recoger la información asociada a ese registro (condiciones del join). Para solucionarlo, lo que se realiza es almacenar temporalmente toda la información necesaria donde sea necesario y trabajar con ella. Esto es lo que representa un DataSet en el modelo ADO .NET.

Un DataSet es una caché de registros recuperados de una base de datos que actúa como un sistema de almacenamiento virtual, y que contiene una o más tablas basadas en las tablas reales de la base de datos. Y que, además, almacena las relaciones y reglas de integridad existentes entre ellas para garantizar la estabilidad e integridad de la información de la base de datos. Muy importante es recalcar, que los DataSets son almacenes pasivos de datos, esto es, que no se ven alterados ante cambios subyacentes de la base de datos. Es necesario recargarlos siempre que queramos estar “al día” en cuanto a datos se refiere.

Una de las mayores ventajas de esta implementación, es que una vez recogido el DataSet, éste puede ser enviado (en forma de flujo XML) entre distintos componentes de la capa de negocio como si de una variable más se tratase, ahorrando así comunicaciones a través de la base de datos.

Una consecuencia lógica de este tipo de arquitecturas, es la de conseguir que los DataSets sean independientes de los orígenes de datos. Los drivers OLE-DB transformarán la consulta SQL en un cursor representado con una estructura XML, que es independiente del motor de la base de datos.

Esto nos permitirá trabajar con múltiples orígenes de datos, de distintos fabricante e incluso no-base de datos, como por ejemplo ficheros planos u hojas de cálculo, lo que representa un importante punto de compatibilidad y flexibilidad.

Si a esto unimos que disponemos de un modelo consistente de objetos (xmlDOM) que es independiente del origen de datos, las operaciones de los DataSets no se verán afectadas por dicho origen.

La persistencia es un concepto muy interesante en el mundo del desarrollo. Es un mecanismo por el cual un componente puede almacenar su estado (valores de variables, propiedades, datos...en un momento concreto del tiempo) en un soporte de almacenamiento fijo. De manera, que cuando es necesario, se puede recargar el componente tal y como quedó en una operación anterior.

En un sistema de trabajo Off-Line como el que plantea ADO .NET, la persistencia es un mecanismo fundamental. Podemos cerrar la aplicación y mantener persistentes todos los DataSets necesarios, de manera que al reiniciarla, nos encontramos los DataSets tal y como los dejamos. Ahorrando el tiempo que hubiera sido necesario para recuperar de nuevo toda esa información del servidor. Optimizando todavía más el rendimiento del sistema distribuido.

El formato que emplea ADO .NET para almacenar su estado es XML. Puesto que ya es un estándar de la industria, esta persistencia nos ofrece:

- La información podría estar accesible para cualquier componente del sistema que entienda XML.
- Es un formato de texto plano, no binario. Que lo hace compatible con cualquier componente de cualquier plataforma y recuperable en cualquier caso

DataSet

El API de ADO .NET proporciona una superclase que encapsula lo que sería la base de datos a un nivel lógico: tablas, vistas, relaciones, su integridad, etc, pero siempre con independencia del tipo de fabricante que la diseñó. Aquí se tiene el mejor concepto de datos desconectados: una copia en el cliente de la arquitectura de la base de datos basada en un esquema XML que la independiza del fabricante, proporcionando al desarrollador la libertad de trabajo independiente de la plataforma. En la Figura 185 se puede ver un esquema de un DataSet.

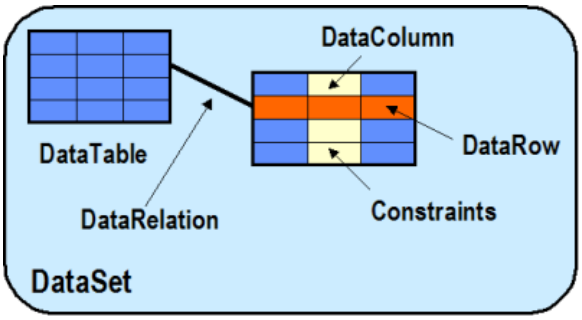


Figura 185

Esta clase se compone a su vez de subclases que representan cada una , los elementos arquitecturales de la base de datos: Tablas, las columnas, las filas, sus reglas de chequeo, sus relaciones, las vistas asociadas a la tabla, etc.

ADO .NET y XML

XML se ha convertido en la piedra angular de la informática distribuida de nuestros días. De ahí que mucho de la redefinición del API de ADO se deba a la adaptación de los objetos a un modelo de procesos que se apoya en documentos XML, no en objetos específicos de cada plataforma a partir de cursores. Esto permite que las clases de ADO .NET puedan implementar mecanismos de conversión de datos entre plataformas, lectura de datos de cualquier origen, habilitar mecanismos de persistencia en el mismo formato en el que se procesan., etc.

En esta redefinición, Microsoft ha puesto como intermediario entre un cliente y sus datos, un adaptador que transforma cada comando y cada dato en modelos de documentos XML. Tanto para consultas como para actualizaciones. Y esto es lo que posibilita la nueva filosofía de acceso a datos desconectados de ADO .NET: primero se cargan en el cliente los documentos necesarios almacenándolos en DataSet a partir de consultas a tablas, vistas, procedimientos, etc, se nos da la posibilidad de trabajar con documentos sin necesidad de estar continuamente consumiendo recursos de la red, y por último se procesarán los cambios producidos enviándolos a la base de datos, el adaptador cogerá los cambios del documento, y los replicará al servidor. En la Figura 186 se puede ver un esquema de la relación entre ADO .NET y XML.

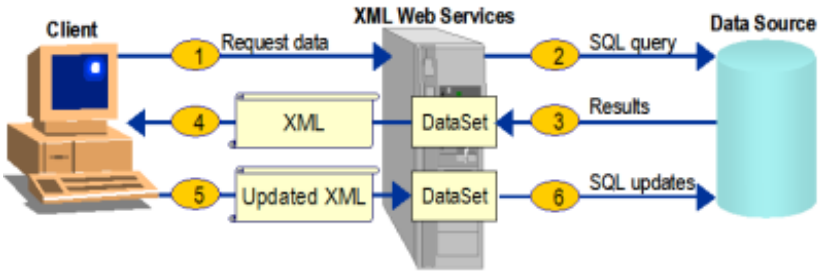


Figura 186

Una visión general de ADO .NET

Esta es la nueva versión del modelo de objetos ADO (ActiveX Data Objects), es decir, la estrategia que ofrece Microsoft para el acceso a datos dentro de la plataforma .NET.

ADO .NET. al igual que sucedía con ASP .NET, en las primeras versiones de la plataforma .NET se denominaba ADO+.

ADO .NET ha sido ampliado para cubrir todas las necesidades que ADO no ofrecía, ADO .NET está diseñado para trabajar con conjuntos de datos desconectados, lo que permite reducir el tráfico de red. ADO .NET utiliza XML como formato universal de transmisión de los datos.

ADO .NET posee una serie de objetos que son los mismos que aparecen en la versión anterior de ADO, como pueden ser el objeto Connection o Command, e introduce nuevos objetos tales como el objeto DataReader, DataSet o DataView. A continuación vamos a comentar brevemente los objetos principales que posee ADO .NET.

Los espacios con nombre que utiliza ADO .NET principalmente son System.Data y System.Data.OleDb o System.Data.SqlClient. System.Data ofrece las facilidades de codificación para el acceso y manejo de datos, y System.Data.OleDb y System.Data.SqlClient contienen los proveedores, en el primer caso los proveedores genéricos de OLE DB y en el segundo los proveedores nativos de SQL Server que ofrece la plataforma .NET.

Para los lectores que han seguido la evolución de la plataforma .NET, apunto que estos espacios con nombre se denominaban System.Data.ADO y System.Data.SQL en la beta 1 de la plataforma .NET.

El objeto Connection define como se establece la conexión con el almacén de datos, el .NET Framework ofrece dos objetos Connection: SqlConnection y OleDbConnection, que se corresponde con las dos posibilidades de proveedores que disponemos.

Otro objeto importante dentro del modelo de objetos de ADO .NET es el objeto System.Data.DataSet (conjunto de datos), este nuevo objeto representa un conjunto de datos de manera completa, pudiendo

incluir múltiples tablas junto con sus relaciones. No debemos confundir el nuevo objeto DataSet con el antiguo objeto Recordset, el objeto DataSet contiene un conjunto de tablas y las relaciones entre las mismas, sin embargo el objeto Recordset contiene únicamente una tabla. Para acceder a una tabla determinada el objeto DataSet ofrece la colección Tables.

Para poder crear e inicializar las tablas del DataSet debemos hacer uso del objeto DataAdapter, que posee las dos versiones, es decir, el objeto SqlDataAdapter para SQL Server y OleDbDataAdapter genérico de OLE DB. En la beta 1 de la plataforma .NET el objeto DataAdapter se denominaba DataSetCommand.

Al objeto DataAdapter le pasaremos por parámetro una cadena que representa la consulta que se va a ejecutar y que va a rellenar de datos el DataSet. Del objeto DataAdapter utilizaremos el método Fill(), que posee dos parámetros, el primero es una cadena que identifica el objeto DataTable (tabla) que se va a crear dentro del objeto DataSet como resultado de la ejecución de la consulta y el segundo parámetro es el objeto DataSet en el que vamos a recoger los datos.

Un DataSet puede contener diversas tablas, que se representan mediante objetos DataTable. Para mostrar el contenido de un DataSet, mediante Data Binding, por ejemplo, necesitamos el objeto DataView. Un objeto DataView nos permite obtener un subconjunto personalizado de los datos contenidos en un objeto DataTable. Cada objeto DataTable de un DataSet posee la propiedad DefaultView que devuelve la vista de los datos por defecto de la tabla. Todo esto y más lo veremos con ejemplos y detallado en los apartados correspondientes.

Otro objeto de ADO .NET es el objeto DataReader, que representa un cursor de sólo lectura y que sólo permite movernos hacia adelante (read-only/forward-only), cada vez existe un único registro en memoria, el objeto DataReader mantiene abierta la conexión con el origen de los datos hasta que es cerrado. Al igual que ocurría con otros objetos de ADO .NET, de este objeto tenemos dos versiones, que como el lector sospechará se trata de los objetos SqlDataReader y OleDbDataReader.

Las clases de ADO .NET

En el presente apartado vamos a enumerar brevemente el conjunto de clases que forman parte del API de ADO .NET.

Primero vamos a comentar los distintos NameSpaces que constituyen la tecnología ADO .NET:

- System.Data : clases genéricas de datos de ADO .NET, integra la gran mayoría de clases que habilitan el acceso a los datos de la arquitectura .NET.

- System.Data.SqlClient: clases del proveedor de datos de SQL Server, permiten el acceso a proveedores SQL Server en su versión 7.0 y superior.

- System.Data.OleDb: clases del proveedor de datos de OleDB, permiten el acceso a proveedores .NET que trabajan directamente contra controladores basados en los ActiveX de Microsoft.

- System.Data.SqlTypes: definición de los tipos de datos de SQL Server, proporciona la encapsulación en clases de todos los tipos de datos nativos de SQL Server y sus funciones de manejo de errores, ajuste y conversión de tipos, etc.

- System.Data.Common: clases base, reutilizables de ADO .NET, proporcionan la colección de clases necesarias para acceder a una fuente de datos (como por ejemplo una Base de Datos).

- System.Data.Internal: integra el conjunto de clases internas de las que se componen los proveedores de datos.

Dentro del espacio de nombres System.Data encontramos las clases compartidas que constituyen el eje central de ADO :NET, y son las siguientes:

- DataSet: almacén de datos por excelencia en ADO .NET. Representa una base de datos desconectada del proveedor de datos. Almacena tablas y sus relaciones.
- DataTable: un contenedor de datos. Estructurado como un conjunto de filas (DataRow) y columnas (DataColumn).
- DataRow: registro que almacena n valores. Representación en ADO .NET de una fila/tupla de una tabla de la base de datos.
- DataColumn: contiene la definición de una columna. Metadatos y datos asociados a su dominio.
- DataRelation: enlace entre dos o más columnas iguales de dos o mas tablas.
- Constraint: reglas de validación de las columnas de una tabla.
- DataColumnMapping: vínculo lógico existente entre una columna de un objeto del DataSet y la columna física de la tabla de la base de datos.
- DataTableMapping: vínculo lógico existente entre una tabla del DataSet y la tabla física de la base de datos.

Además de estas clases, existe otro grupo de clases, las clases específicas de un proveedor de datos. Estas clases forman parte de lo específico de un fabricante de proveedores de datos .NET. Tienen una sintaxis de la forma XxxClase donde "Xxx" es un prefijo que determina el tipo de plataforma de conexión a datos. Se definen en dos NameSpaces: System.Data.SqlClient y System.Data.OleDb.

En la Tabla 13 se ofrece una descripción de las clases que podemos encontrar en estos espacios de nombres.

Clase	**Descripción**
SqlCommand OleDbCommand	Clases que representan un comando SQL contra un sistema gestor de datos.
SqlConnection OleDbConnection	Clase que representa la etapa de conexión a un proveedor de datos. Encapsula la seguridad, pooling de conexiones, etc.
SqlCommandBuilder OleDbCommandBuilder	Generador de comandos SQL de inserción, modificación y borrado desde una consulta SQL de selección de datos.

SqlDataReader OleDbDataReader	Un lector de datos de sólo avance, conectado a la base de datos.
SqlDataAdapter OleDbDataAdapter	Clase adaptadora entre un objeto DataSet y sus operaciones físicas en la base de datos (select,insert,update y delete).
SqlParameter OleDbParameter	Define los parámetros empleados en la llamada a procedimientos almacenados.
SqlTransaction OleDbTransaction	Gestión de las transacciones a realizar en la base de datos.

Tabla 13

Para aquellos conocedores de ADO en alguna de sus versiones anteriores podemos hacer una analogía o comparación entre las antiguas clases de ADO y las nuevas de ADO .NET, en la Figura 187 se puede ver esta aproximación.

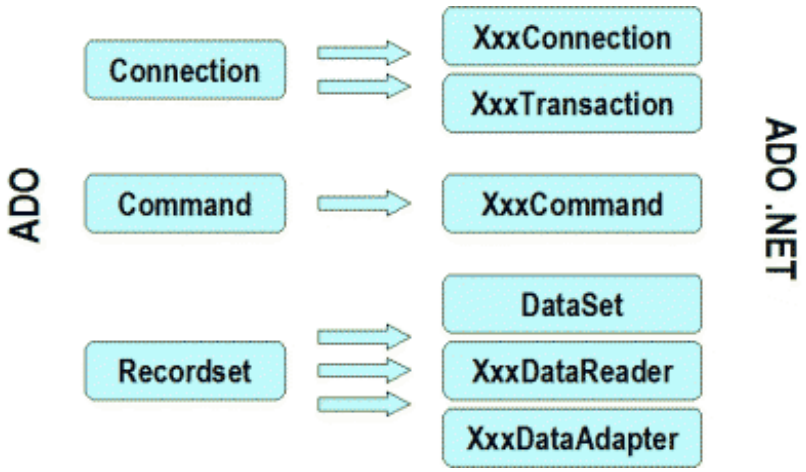


Figura 187

Hasta aquí hemos realizado una introducción a la tecnología ADO .NET, viendo su arquitectura y comentando las clases principales. En lo que resta de capítulo vamos a utilizar las distintas clases que nos ofrece ADO .NET desde páginas ASP .NET para realizar tareas comunes de acceso a datos, como pueden ser establecer una conexión, obtener un conjunto de registros, realizar operaciones con los datos, etc. Para información más detallada sobre ADO .NET el lector puede consultar el texto de mi compañero en Grupo EIDOS José Luis Hevia.

Estableciendo la conexión. Los objetos Connection

Estos objetos de ADO .NET los vamos a utilizar de forma similar al objeto Connection de ADO, es decir, para establecer una conexión con un almacén de datos (ya sea una base de datos o no), en ADO

se podía ejecutar directamente una sentencia contra el almacén de datos o bien abrir un conjunto de registro (Recordset), pero en ADO .NET esto no se va a realizar con el objeto SqlConnection o bien con el objeto OleDbConnection, según sea la situación.

Se debe recordar que existen dos implementaciones de algunos de los objetos de ADO .NET, cada uno específico del origen de datos con el que nos vamos a conectar, esto sucede con el objeto Connection que tiene dos versiones, una como proveedor de datos de SQL Server, a través de la clase System.Data.SqlClient.SqlConnection y otra como proveedor de datos OLEDB, a través de la clase Sysem.Data.OleDb.OleDbConnection.

Normalmente del objeto Connection utilizaremos los métodos Open() y Close() para abrir y cerrar conexiones, respectivamente, con el almacén de datos adecuado. Aunque tenemos el recolector de basura que gestiona de forma automática los recursos y objetos que no son utilizados, es recomendable cerrar las conexiones de forma explícita utilizando el método Close().

Las conexiones se abrirán de forma explícita utilizando el método Open(), pero también se puede hacer de forma implícita utilizando un objeto DataAdapter, esta posibilidad la veremos más adelante. Cuando lanzamos el método Open() sobre un objeto Connection (SqlConnection o OleDbConnection), se abrirá la conexión que se ha indicado en su propiedad ConnectionString, es decir, esta propiedad indicará la cadena de conexión que se va a utilizar para establecer la conexión con el almacén de datos correspondiente. El método Open() no posee parámetros.

El constructor de la clase Connection (al decir clase Connection de forma genérica nos estamos refiriendo en conjunto a las clases SqlConnection y OleDbConnection de ADO .NET) se encuentra sobrecargado, y en una de sus versiones recibe como parámetro una cadena que será la cadena de conexión que se aplique a su propiedad ConnectionString.

Si hacemos uso de la clase SqlConnection, en la cadena de conexión no podremos especificar una DSN de ODBC, ya que la conexión se va a realizar en este caso directamente con SQL Server. Y si utilizamos la clase OleDbConnection debemos especificar el proveedor OLEDB que se va a utilizar para establecer la conexión, una excepción es el proveedor OLEDB para ODBC (MSDASQL), que no puede ser utilizado, ya que el proveedor OLEDB de .NET no soporta el proveedor de ODBC, en este caso deberemos realizar la conexión utilizan el proveedor adecuado al almacén de datos. Los proveedores OLEDB que son compatibles con ADO .NET son:

- SQLOLEDB: Microsoft OLE DB Provider for SQL Server.
- MSDAORA: Microsoft OLE DB Provider for Oracle.
- Microsoft.Jet.OLEDB.4.0: OLE DB Provider for Microsoft Jet.

La sintaxis utilizada para indicar la cadena de conexión es muy similar a la utilizada en ADO, a continuación vamos a ofrecer dos ejemplos dos ejemplos de conexiones con un servidor SQL Server 2000, en un primer caso se utiliza la clase SqlConnection en segundo lugar utilizamos la clase OleDbConnection. Para más información sobre los objetos Connection de ADO .NET la sintaxis de sus cadenas de conexión remito a los lectores al texto sobre ADO .NET que ha realizado mi compañero José Luis Hevia.

En este primer ejemplo (Código fuente 316) vamos a utilizar el objeto SqlConnection para establecer y cerrar una conexión con una base de datos de SQL Server 2000 desde una página ASP .NET.

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
```

```
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<head><title>El objeto Connection</title></head>
<body>
<script runat="server">
void Conecta(Object sender, EventArgs args){
      SqlConnection conexion =new SqlConnection
                              ("server=angel;database=northwind;uid=sa;pwd=");
      try{
            conexion.Open();
            Response.Write("Se ha establecido la conexión<br>");
            conexion.Close();
            Response.Write("Se ha cerrado la conexión<br>");
      }catch(SqlException e){
            Response.Write("se ha producido una excepción: "+e);
      }
}
</script>
<form runat="server" id="formulario">
<asp:Button OnClick="Conecta" Runat="server" ID="boton" Text="Conexión"/>
</form>
</body>
</html>
```

Código fuente 316

Como se puede comprobar se ha utilizado el mecanismo de tratamiento de errores mediante bloques try/catch, para tratar las excepciones que se ocasionan al abrir o cerrar la conexión.

En la Figura 188 se puede ver el resultado de la ejecución de la página ASP .NET del ejemplo.

Figura 188

Y en el Código fuente 317 vamos a realizar la misma conexión que en el ejemplo anterior pero en este caso haciendo uso del objeto OleDbConnection.

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
<html>
<head><title>El objeto Connection</title></head>
<body>
```

```
<script runat="server">
void Conecta(Object sender, EventArgs args){
      OleDbConnection conexion =new OleDbConnection
                  ("provider=SQLOLEDB;server=angel;database=northwind;uid=sa;pwd=");
      try{
            conexion.Open();
            Response.Write("Se ha establecido la conexión<br>");
            conexion.Close();
            Response.Write("Se ha cerrado la conexión<br>");
      }catch(OleDbException e){
            Response.Write("se ha producido una excepción: "+e);
      }
}
</script>
<form runat="server" id="formulario">
<asp:Button OnClick="Conecta" Runat="server" ID="boton" Text="Conexión"/>
</form>
</body>
</html>
```

Código fuente 317

Como se puede comprobar las diferencias entre ambas páginas ASP .NET son mínimas, lo que cambia es le nombre de las clases utilizadas de ADO .NET y la cadena de conexión utilizada para establecer la conexión.

Ya hemos visto uno de lo objetos fundamentales de ADO .NET en el siguiente apartado trataremos otro objeto más, se trata de los objetos Command.

Los objetos Command

Los objetos Command de ADO .NET , que también son dos: SqlCommand y OleDbCommand, son muy similares al objeto Command existente en ADO. El objeto Command nos va a permitir ejecutar una sentencia SQL o un procedimiento almacenado sobre la fuente de datos a la que estamos accediendo.

A través de un objeto Command también podremos obtener un conjunto de resultados del almacén de datos, en este caso estos resultados se pasarán a otros objetos de ADO .NET, como puede ser un DataReader o bien un objeto DataAdapter, estos dos objetos los trataremos también más adelante.

Un objeto Command lo vamos a poder crear a partir de una conexión ya existente y va a contener una sentencia SQL para ejecutar sobre la conexión con el origen de datos.

A continuación vamos a comentar algunas de las propiedades más importantes que ofrecen los objetos SqlCommand y OleDbCommand:

- CommandText: contiene una cadena de texto que va a indicar la sentencia SQL o procedimiento almacenado que se va a ejecutar sobre el origen de los datos.

- CommandTimeout: tiempo de espera en segundos que se va a aplicar a la ejecución de un objeto Command. Su valor por defecto es de 30 segundos.

- CommandType: indica el tipo de comando que se va a ejecutar contra el almacén de datos, es decir, indica como se debe interpretar el valor de la propiedad CommadText. Puede tener los siguientes valores; StoredProcedure, para indicar que se trata de un procedimiento

almacenado; TableDirect se trata de obtener una tabla por su nombre (únicamente aplicable al objeto OleDbCommand); y Text que indica que es una sentencia SQL. EL valor por defecto es Text.

- Connection: devuelve el objeto SqlConnection o OleDbConnection utilizado para ejecutar el objeto Command correspondiente.

- Parameters: colección de parámetros que se pueden utilizar para ejecutar el objeto Command, esta colección se utiliza cuando deseamos ejecutar sentencias SQL que hacen uso de parámetros, esta propiedad devuelve un objeto de la clase SqlParameterCollection o un objeto de la clase OleDbParameterCollection. Estas colecciones contendrán objetos de la clase SqlParamter y OleDbParameter, respectivamente, para representar a cada uno de los parámetros utilizados. Estos parámetros también son utilizados para ejecutar procedimientos almacenados.

Una vez vistas algunas de las propiedades de las clases SqlCommand y OleDbCommand vamos a pasar a comentar brevemente los métodos más usuales de estas clases:

- CreateParameter: crea un parámetro para el que después podremos definir una serie de características específicas como pueden ser el tipo de dato, su valor, tamaño, etc. Devolverá un objeto de la clase SqlParameter u OleDbParameter.

- ExecuteNonQuery: ejecuta la sentencia SQL definida en la propiedad ComandText contra la conexión definida en la propiedad Connection, para un tipo de consulta que no devuelve un conjunto de registros, puede ser una sentencia Update, Delete o Insert. Este método devuelve un entero indicando el número de filas que se han visto afectadas por la ejecución del objeto Command.

- ExecuteReader: ejecuta la sentencia SQL definida en la propiedad ComandText contra la conexión definida en la propiedad Connection, para un tipo de consulta que devuelve un conjunto de registros, puede ser una sentencia Select. Este método devolverá un objeto de la clase SqlDataReader/OleDbDataReader, que nos va a permitir leer y recorrer los resultados devueltos por la ejecución del objeto Command correspondiente.

- ExecuteScalar: este otro método para ejecutar objetos Command se utiliza cuando deseamos obtener la primera columna de la primera fila del conjunto de registros, el resto de datos no se tendrán en cuenta. La utilización de este método tiene sentido cuando estamos ejecutando una sentencia SQL del tipo Select count(*). Este método devuelve un objeto de la clase Object.

- Prepare: este método construye una versión compilada del objeto Command dentro del almacén de datos.

A continuación vamos a pasar a utilizar el objeto Command de distintas formas a través de varios ejemplos desde páginas ASP .NET.

En este primer ejemplo (Código fuente 318) vamos a utilizar un objeto SqlCommand para ejecutar una sentencia SQL de tipo Insert sobre una tabla determinada, por lo que se utilizará el método ExecuteNonQuery().

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
```

```
<head><title>El objeto Command</title></head>
<body>
<script runat="server">
void Alta(Object sender, EventArgs args){
      SqlConnection conexion =new
                      SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      String sentencia="INSERT into Employees (firstname, lastname) "+
                       "VALUES('Angel','Esteban')";
      SqlCommand comando=new SqlCommand (sentencia,conexion);
      int resultado;
      try{
            conexion.Open();
            resultado=comando.ExecuteNonQuery();
            Response.Write("Se ha añadido "+resultado+" registro");
            conexion.Close();
      }catch(SqlException e){
            Response.Write("se ha producido una excepción: "+e);
      }
}
</script>
<form runat="server" id="formulario">
<asp:Button OnClick="Alta" Runat="server" ID="boton" Text="Añadir"/>
</form>
</body>
</html>
```

Código fuente 318

Como se puede ver en este ejemplo, el constructor de la clase SqlCommand presenta dos parámetros, una cadena de texto con sentencia SQL que representa y un objeto se la clase SqlConnection que representa la conexión sobre la que se va a ejecutar el comando.

Si deseamos ejecutar la misma sentencia Insert pero con la posibilidad de especificar los datos que se van a utilizar en el alta de la tabla, utilizaríamos el Código fuente 319, en el que se especifican los valores de los parámetros que se van a utilizar para ejecutar la sentencia representada por el objeto SqlCommand. Para recoger los valores de los parámetros a utilizar se va realizar a través de un Web Form que presenta dos objetos TextBox.

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<head><title>El objeto Command</title></head>
<body>
<script runat="server">
void Alta(Object sender, EventArgs args){
      SqlConnection conexion =new
                      SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      String sentencia="INSERT into Employees (firstname, lastname) "+
                       "VALUES(@nombre,@apellidos)";
      SqlCommand comando=new SqlCommand (sentencia,conexion);
      int resultado;
      try{
        conexion.Open();
        comando.Parameters.Add(new SqlParameter("@nombre",
                                                SqlDbType.NVarChar, 10));
        comando.Parameters["@nombre"].Value = nombre.Text;
        comando.Parameters.Add(new SqlParameter("@apellidos",
                                                 SqlDbType.NVarChar, 20));
        comando.Parameters["@apellidos"].Value = apellidos.Text;
        resultado=comando.ExecuteNonQuery();
```

```
        Response.Write("Se ha añadido "+resultado+" registro");
        conexion.Close();
      }catch(SqlException e){
            Response.Write("se ha producido una excepción: "+e);
      }
}
</script>
<form runat="server" id="formulario">
<asp:Label Runat="server" ID="textoNombre" text="Nombre"/>
<asp:TextBox Runat="server" ID="nombre"/><br>
<asp:Label Runat="server" ID="textoApellidos" text="Apellidos"/>
<asp:TextBox Runat="server" ID="apellidos"/><br>
<asp:Button OnClick="Alta" Runat="server" ID="boton" Text="Añadir"/>
</form>
</body>
</html>
```

Código fuente 319

En este caso se debe hacer uso de la propiedad Parameters del objeto SqlCommand, a esta colección se añaden los dos parámetros necesarios mediante el método Add(). A este método le pasamos como parámetro un objeto de la clase SqlParameter, para cada uno de estos objetos debemos indicar en su constructor el nombre del parámetro, el tipo y el tamaño del mismo. Una vez añadido el parámetro a la colección Parameters podemos asignarle el valor correspondiente, que en este caso serán los valores de los controles Web del Web Form.

En este ejemplo también se puede comprobar la sintaxis utilizada para indicar parámetro dentro de las sentencias SQL. El código de la página ASP .NET de este ejemplo se puede obtener en el siguiente enlace.

Si rescribimos el ejemplo anterior para utilizar la clase OleDbCommand el Código fuente 320 sería el resultado. Debemos fijarnos en la distinta sintaxis para la sentencia SQL .

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
<html>
<head><title>El objeto Command</title></head>
<body>
<script runat="server">
void Alta(Object sender, EventArgs args){
   OleDbConnection conexion =new OleDbConnection(
                  "Provider=SQLOLEDB;server=angel;database=northwind;uid=sa;pwd=");
   String sentencia="INSERT into Employees (firstname, lastname) VALUES(?,?)";
   OleDbCommand comando=new OleDbCommand(sentencia,conexion);
   int resultado;
   try{
       conexion.Open();
       comando.Parameters.Add(new OleDbParameter("@nombre",OleDbType.VarChar,10));
       comando.Parameters["@nombre"].Value = nombre.Text;
       comando.Parameters.Add(new OleDbParameter("@apellidos",
                                                 OleDbType.VarChar,20));
       comando.Parameters["@apellidos"].Value = apellidos.Text;
       resultado=comando.ExecuteNonQuery();
       Response.Write("Se ha añadido "+resultado+" registro");
      conexion.Close();
  }catch(OleDbException e){
      Response.Write("se ha producido una excepción: "+e);
  }
}
```

```
</script>
<form runat="server" id="formulario">
<asp:Label Runat="server" ID="textoNombre" text="Nombre"/>
<asp:TextBox Runat="server" ID="nombre"/><br>
<asp:Label Runat="server" ID="textoApellidos" text="Apellidos"/>
<asp:TextBox Runat="server" ID="apellidos"/><br>
<asp:Button OnClick="Alta" Runat="server" ID="boton" Text="Añadir"/>
</form>
</body>
</html>
```

Código fuente 320

En este nuevo ejemplo vamos a utilizar un objeto Command de ADO .NET para ejecutar otro tipo de sentencia mediante el método ExecuteScalar(). En este ejemplo (Código fuente 321) se obtiene el número de registros de una tabla mediante la instrucción count de SQL. El método ExecuteScalar() devuelve un objeto de clase Object, por lo que deberemos aplicar los mecanismos de casting necesarios para obtener el tipo de dato deseado.

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<head><title>El objeto Command</title></head>
<body>
<script runat="server">
void Cuenta(Object sender, EventArgs args){
      SqlConnection conexion =new SqlConnection(
                                    "server=angel;database=northwind;uid=sa;pwd=");
      String sentencia="SELECT COUNT(*) as numEmpleados From Employees";
      SqlCommand comando=new SqlCommand (sentencia);
      int resultado;
      try{
            conexion.Open();
            comando.Connection=conexion;
            resultado= (int)comando.ExecuteScalar();
            Texto.Text="En la tabla hay " + resultado + " registros";
            conexion.Close();
      }catch(SqlException e){
            Response.Write("se ha producido una excepción: "+e);
      }
}
</script>
<form runat="server" id="formulario">
<asp:Label Runat="server" ID="Texto" Text=""/><br>
<asp:Button OnClick="Cuenta" Runat="server" ID="boton" Text="Cuenta"/>
</form>
</body>
</html>
```

Código fuente 321

En esta página ASP .NET se ha utilizado un constructor distinto de la clase SqlCommand, ya que la conexión se la hemos asignado, una vez creado el objeto SqlCommand, a la propiedad Connection.

En la Figura 189 se puede ver un ejemplo de ejecución de esta página.

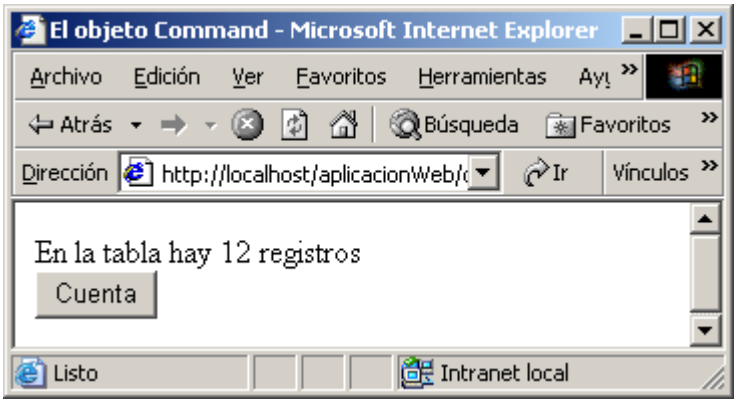


Figura 189

Nos queda ver un ejemplo de objeto Command sobre el que se lanza el método ExecuteReader(), pero lo vamos a dejar para el siguiente apartado, en el que vemos otros de los objetos principales de ADO .NET, el objeto DataReader (SqlDataReader y OleDbDataReader).

Los objetos DataReader

Los objetos SqlDataReader y OleDbDataReader ya los mencionamos en el apartado anterior, y van a ser equivalentes a los cursores de sólo lectura y movimiento hacia adelante de ADO (read only/forward-only), en este caso no se ofrece un acceso desconectado de los datos, sino que se conecta directamente almacén de datos y nos devolverá un conjunto de registros para que los podamos recorrer.

Un objeto DataReader lo vamos a obtener de la ejecución de una sentencia SQL o bien de la ejecución de un procedimiento almacenado, representados ambos por un objeto Command, como ya vimos en el apartado anterior, a partir de la llamada al método ExecuteReader().

A continuación vamos a pasar a describir las principales propiedades de las clases SqlDataReader y OleDbDataReader.

- FieldCount: devuelve el número de columnas (campos) presentes en el fila (registro) actual.
- IsClosed: devolvera´los valores true o false para indicar si el objeto DataReader está cerrado o no.
- Item: devuelve en formato nativo el valor de la columna cuyo nombre le indicamos como índice en forma de cadena de texto.

Una vez vistas las propiedades, vamos a comentar los métodos más destacables:

- Close: cierra el objeto DataReader liberando los recursos correspondientes.
- GetXXX: el objeto DataReader presenta un conjunto de métodos que nos van a permitir obtener los valores de las columnas contenidas en el mismo en forma de un tipo de datos determinado, según el método GetXXX empleado. Existen diversos métodos GetXXX atendiendo al tipo de datos de la columna, algunos ejemplos son GetBoolean(), GetInt32(), GetString(), GetChar(), etc. Como parámetro a este método le debemos indicar el número de orden de la columna que deseamos recuperar.

- NextResult: desplaza el puntero actual al siguiente conjunto de registros, cuando la sentencia es un procedimiento almacenado de SQL o una sentencia SQL que devuelve más de un conjunto de registros, no debemos confundir este método con el método MoveNext() de ADO, ya que en este caso no nos movemos al siguiente registro, sino al siguiente conjunto de registros.

- Read: desplaza el cursor actual al siguiente registro permitiendo obtener los valores del mismo a través del objeto DataReader. Este método devolverá true si existen más registros dentro del objeto DataReader, false si hemos llegado al final del conjunto de registros. La posición por defecto del objeto DataReader en el momento inicial es antes del primer registro, por lo tanto para recorrer un objeto DataReader debemos comenzar con una llamada al método Read(), y así situarnos en el primer registro.

En el Código fuente 322 se muestra una página ASP .NET que obtiene un objeto SqlDataReader a partir de la ejecución de un objeto SqlCommand, al que se ha lanzado el método ExecuteReader(). Una vez que tenemos el objeto SqlDataReader se muestra por pantalla su contenido, haciendo uso del método Read() dentro de un bucle while.

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<head><title>El objeto DataReader</title></head>
<body>
<script runat="server">
void Muestra(Object sender, EventArgs args){
      SqlConnection conexion =new SqlConnection(
                                    "server=angel;database=northwind;uid=sa;pwd=");
      String sentencia="SELECT firstname, lastname From Employees";
      SqlCommand comando=new SqlCommand (sentencia);
      SqlDataReader resultado;
      try{
            conexion.Open();
            comando.Connection=conexion;
            resultado= comando.ExecuteReader();
            while(resultado.Read()){
                  Response.Write(resultado.GetString(0)+" "+
                                    resultado.GetString(1)+"<br>");
            }
            resultado.Close();
            conexion.Close();
      }catch(SqlException e){
            Response.Write("se ha producido una excepción: "+e);
      }
}
</script>
<form runat="server" id="formulario">
<asp:Label Runat="server" ID="Texto" Text=""/><br>
<asp:Button OnClick="Muestra" Runat="server" ID="boton" Text="Muestra Datos"/>
</form>
</body>
</html>
```

Código fuente 322

En la Figura 190 se puede ver el resultado de la ejecución de la página ASP .NET del ejemplo.

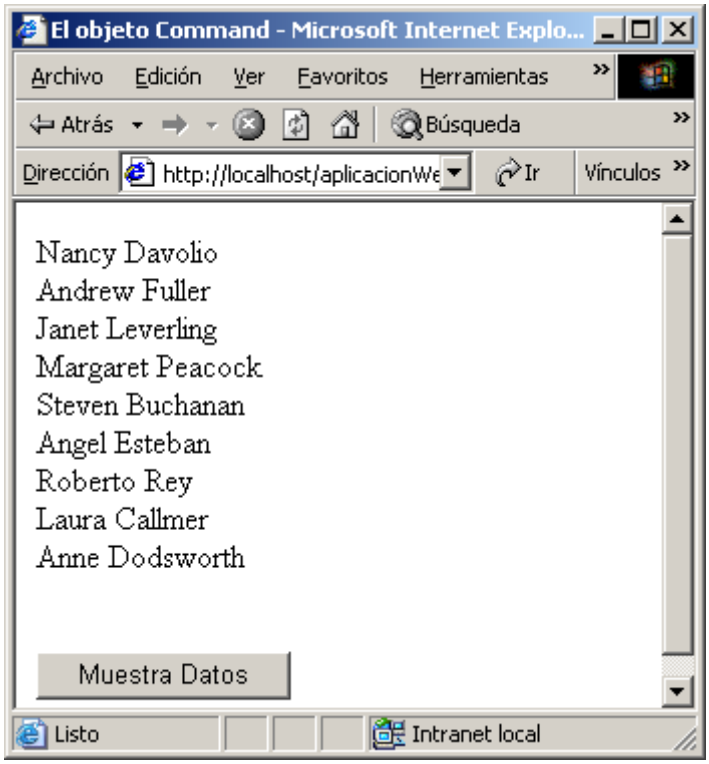


Figura 190

El recorrido del objeto DataReader lo podíamos a ver realizado de la forma que se muestra en el Código fuente 323, en este caos no se utiliza métodos GetXXX, sino que se accede directamente al objeto DataReader, en este caso estamos accediendo a su propiedad Item, ya que está funcionado como una colección.

```
while(resultado.Read()){
       Response.Write((String)resultado["firstname"]+" "+
              (String)resultado["lastname"]+"<br>");
}
```

Código fuente 323

El Código fuente 324 sería el código de la página ASP .NET del ejemplo si deseamos utilizar OLEDB en lugar del proveedor de SQL Server.

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OleDb" %>
<html>
<head><title>El objeto DataReader</title></head>
<body>
<script runat="server">
void Muestra(Object sender, EventArgs args){
   OleDbConnection conexion =new OleDbConnection(
                  "Provider=SQLOLEDB;server=angel;database=northwind;uid=sa;pwd=");
   String sentencia="SELECT firstname, lastname From Employees";
   OleDbCommand comando=new OleDbCommand (sentencia);
   OleDbDataReader resultado;
```

```
   try{
       conexion.Open();
       comando.Connection=conexion;
       resultado= comando.ExecuteReader();
       while(resultado.Read()){
              Response.Write((String)resultado["firstname"]+" "+
                   (String)resultado["lastname"]+"<br>");
       }
       resultado.Close();
       conexion.Close();
   }catch(OleDbException e){
       Response.Write("se ha producido una excepción: "+e);
   }
}
</script>
<form runat="server" id="formulario">
<asp:Label Runat="server" ID="Texto" Text=""/><br>
<asp:Button OnClick="Muestra" Runat="server" ID="boton" Text="Muestra Datos"/>
</form>
</body>
</html>
```

Código fuente 324

El objeto DataSet

Este objeto ya pertenece a los objetos comunes de ADO .NET, es decir, es la misma clase para todo tipo de proveedores .NET, no se distingue entre SqlClient y OleDb. En la introducción que hemos realizado a ADO .NET en este capítulo ya hemos realizado algunos comentarios acerca del objeto DataSet.

Básicamente un objeto DataSet va a ser similar a un objeto Recordset de ADO pero más potente y complejo, es el almacén de datos por excelencia en ADO .NET. Representa una base de datos desconectada del proveedor de datos. Almacena tablas y sus relaciones. Aquí se tiene el mejor concepto de datos desconectados: una copia en el cliente de la arquitectura de la base de datos basada en un esquema XML que la independiza del fabricante, proporcionando al desarrollador la libertad de trabajo independiente de la plataforma.

Cada tabla contenida dentro de un objeto DataSet se encuentra disponible a través de su propiedad Tables, que es una colección de objetos System.Data.DataTable. Cada objeto DataTable contiene una colección de objetos DataRow que representan las filas de la tabla. Y si seguimos con esta analogía tenemos que decir que cada objeto DataRow, es decir, cada fila, posee una colección de objetos DataColumn, que representan cada una de las columnas de la fila actual. Además también existen colecciones y objetos para representan las relaciones, claves y valores por defecto existentes dentro de un objeto DataSet.

Para cada objeto DataTable existe una propiedad llamada DefaultView que devuelve un objeto de la clase DataView, que nos ofrece una vista de los datos de la tabla para que podamos recorrer los datos, filtrarlos, ordenarlos, etc.

El objeto DataSet es el más complejo de los objetos que ofrece ADO .NET, tanto es así que podríamos dedicar un capítulo completo a su utilización, sin embargo en este apartado y en el siguiente veremos únicamente algunos sencillos ejemplos de utilización, para más información sobre el objeto DataSet remito al lector al texto de mi compañero en Grupo EIDOS José Luis Hevia.

Para poder crear e inicializar las tablas del DataSet debemos hacer uso del objeto DataAdapter, que posee las dos versiones, es decir, el objeto SqlDataAdapter para SQL Server y OleDbDataAdapter genérico de OLE DB.

Al objeto DataAdapter le pasaremos por parámetro una cadena que representa la consulta que se va a ejecutar y que va a rellenar de datos el DataSet. Del objeto DataAdapter utilizaremos el método Fill(), que posee dos parámetros, el primero es una cadena que identifica el objeto DataTable (tabla) que se va a crear dentro del objeto DataSet como resultado de la ejecución de la consulta y el segundo parámetro es el objeto DataSet en el que vamos a recoger los datos. En el siguiente apartado veremos los objetos DataAdapter, que van a funcionar como intermediarios entre el almacén de datos y el objeto DataSet que contiene una versión desconectada de los datos.

En la siguiente enumeración se encuentran los métodos más destacables que ofrece el objeto DataSet:

- Clear: elimina todos los datos almacenados en el objeto DataSet, vaciando todas las tablas contenidas en el mismo.

- AcceptChanges: confirma todos los cambios realizados en las tablas y relaciones contenidas en el objeto DataSet, o bien los últimos cambios que se han producido desde la última llamada al método AcceptChanges.

- GetChanges: devuelve un objeto DataSet que contiene todos los cambios realizados desde que se cargó con datos, o bien desde que se realizó la última llamada al método AcceptChanges.

- HasChanges: devuelve true o false para indicar si se han realizado cambios al contenido del DataSet desde que fue cargado o bien desde que se realizó la última llamada al método AcceptChanges.

- RejectChanges: abandona todos los cambios realizados en las tablas contenidas en el objeto DataSet desde que fue cargado el objeto o bien desde la última vez que se lanzó el método AcceptChanges.

- Merge: toma los contenidos de un DataSet y los mezcla con los de otro DataSet, de forma que contendrá los datos de ambos objetos DataSet.

Una vez vistos algunos de los métodos del objeto DataSet vamos a pasar a comentar algunas de sus propiedades:

- CaseSensitive: propiedad que indica si las comparaciones de texto dentro de las tablas distinguen entre mayúsculas y minúsculas. Por defecto tiene el valor false.

- DataSetName: establece o devuelve mediante una cadena de texto el nombre del objeto DataSet.

- HasErrors: devuelve un valor booleano para indicar si existen errores dentro de las tablas del DataSet.

- Relations: esta propiedad devuelve una colección de objetos DataRelation que representan todas las relaciones existentes entre las tablas del objeto DataSet.

- Tables: devuelve una colección de objetos DataTable que representan a cada una de las tablas existentes dentro del objeto DataSet.

A continuación vamos a ofrecer un sencillo ejemplo de utilización de un objeto DataSet dentro de una página ASP .NET (Código fuente 325). En este ejemplo vamos a crear un objeto DataSet que va a contener una tabla de empleados. Una vez cargado el objeto DataSet mostraremos su contenido dentro de un control DataGrid mediante el mecanismo de Data Binding.

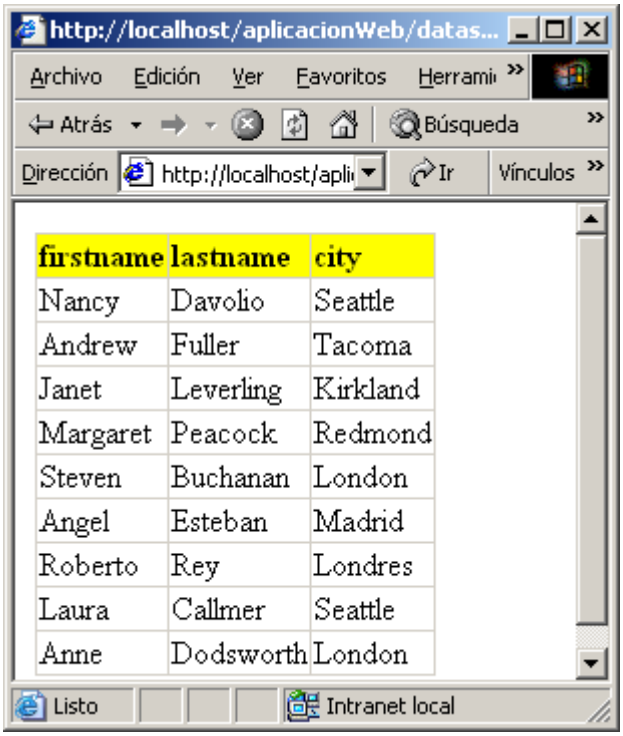
```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script runat="server">
void Page_Load(Object sender, EventArgs e) {

      SqlConnection conexion = new SqlConnection(
                              "server=angel;database=northwind;uid=sa;pwd=");
      SqlDataAdapter adapter = new SqlDataAdapter(
                  "select firstname, lastname,city from Employees", conexion);
      DataSet ds = new DataSet();
      try{
            conexion.Open();
            adapter.Fill(ds, "Empleados");
            tabla.DataSource = ds.Tables["Empleados"].DefaultView;
            tabla.DataBind();
            conexion.Close();
      }catch(SqlException ex){
            Response.Write("se ha producido una excepción: "+ex);
      }
}
</script>
<body>
<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow">
</asp:DataGrid>
</body>
</html>
```

Código fuente 325

Como se puede comprobar la tabla que se ha almacenado en el objeto DataSet la hemos llamado Empleados. Para acceder a dicha tabla no tenemos nada más que hacer la referencia correspondiente a través de la colección Tables del objeto DataSet. Para almacenar la tabla dentro del objeto DataSet hemos tenido que hacer uso de un objeto de la clase SqlDataAdapter, que veremos con detenimiento en el siguiente apartado, y que ya hemos comentado que hace la función de puente o comunicación entre el almacén de datos y el objeto DataSet.

En la Figura 191 se puede ver la ejecución de esta página ASP .NET.

Vamos a mostrar un ejemplo más de utilización del objeto DataSet antes de pasar a comentar las clases DataAdapter (Código fuente 326). En este caso el objeto DataSet va a contener tres tablas, que se han cargado dentro del objeto DataSet utilizando otros tantos objetos SqlDataAdapter. Cada una de las tablas del DataSet se van a mostrar en un control DataGrid distinto, pero además del contenido de las tablas, también se va a mostrar el contenido de la colección Tables del objeto DataSet.



firstname	lastname	city
Nancy	Davolio	Seattle
Andrew	Fuller	Tacoma
Janet	Leverling	Kirkland
Margaret	Peacock	Redmond
Steven	Buchanan	London
Angel	Esteban	Madrid
Roberto	Rey	Londres
Laura	Callmer	Seattle
Anne	Dodsworth	London

Figura 191

```
<%@ Page language="C#"%>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<html>
<script runat="server">
void Page_Load(Object sender, EventArgs e) {

	SqlConnection conexion = new SqlConnection(
					"server=angel;database=northwind;uid=sa;pwd=");
	SqlDataAdapter adapterEmpleados = new SqlDataAdapter(
			"select firstname, lastname,city from Employees", conexion);
	SqlDataAdapter adapterRegiones =
		new SqlDataAdapter("select * from Region", conexion);
	SqlDataAdapter adapterDistribuidores =
		new SqlDataAdapter("select * from Shippers", conexion);
	DataSet ds = new DataSet();
	try{
		conexion.Open();
		adapterEmpleados.Fill(ds, "Empleados");
		empleados.DataSource = ds.Tables["Empleados"].DefaultView;
		adapterRegiones.Fill(ds, "Regiones");
		regiones.DataSource = ds.Tables["Regiones"].DefaultView;
		adapterDistribuidores.Fill(ds, "Distribuidores");
		distribuidores.DataSource = ds.Tables["Distribuidores"].DefaultView;
		tablas.DataSource=ds.Tables;
		DataBind();
		conexion.Close();
	}catch(SqlException ex){
		Response.Write("se ha producido una excepción: "+ex);
	}
}
</script>

<body>
<asp:DataGrid id="tablas" runat="server" BorderWidth="1" GridLines="Both"
```

```
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="orange">
</asp:DataGrid>
<asp:DataGrid id="empleados" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow">
</asp:DataGrid>
<asp:DataGrid id="regiones" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="red">
</asp:DataGrid>
<asp:DataGrid id="distribuidores" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="green">
</asp:DataGrid>
</body>
</html>
```

Código fuente 326

En la Figura 192 se puede ver un ejemplo de ejecución de esta página ASP .NET.

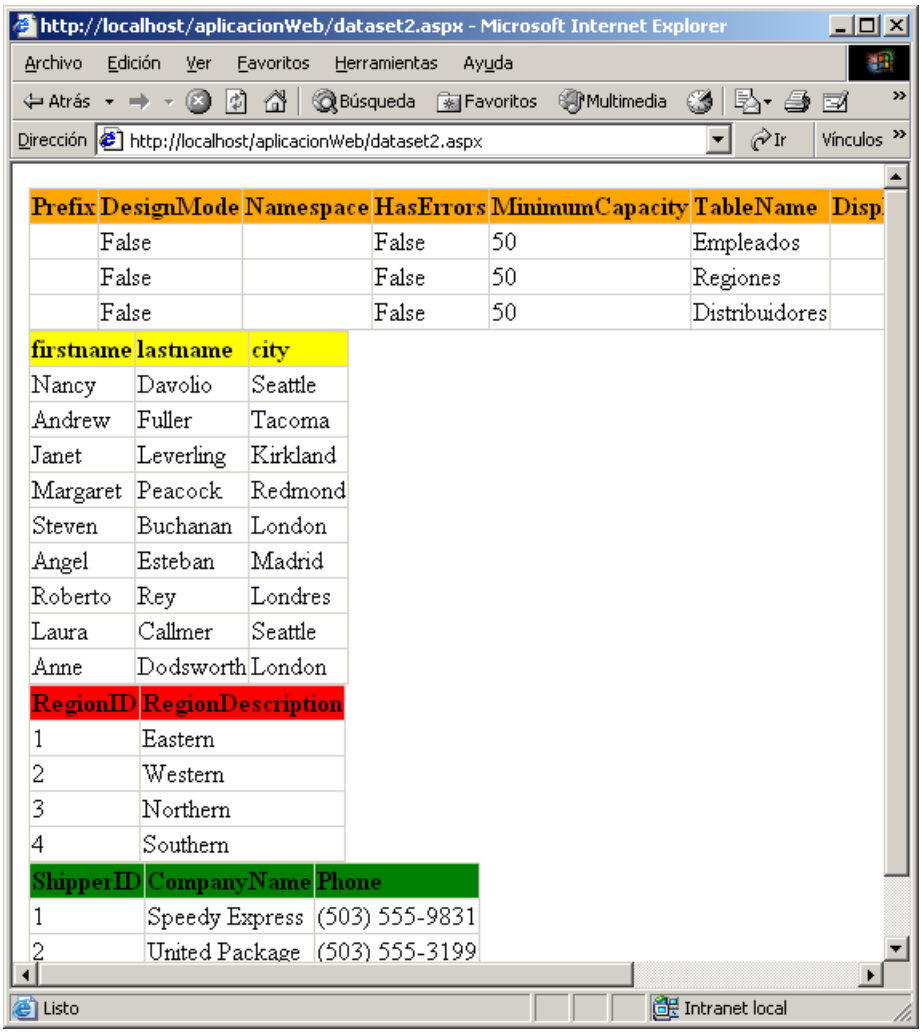


Figura 192

En el siguiente enlace se puede obtener el código de la página ASP .NET del ejemplo.

Los objetos DataAdapter

Como ya hemos comentado, los objetos DataAdapter (SqlDataAdapter y OleDbDataAdapter) van a hacer la función de puente entre el almacén de los datos y el DataSet, nos van a permitir cargar el DataSet desde el origen de los datos y después nos va a permitir actualizar los datos en el origen de datos con los del DataSet.

En los ejemplos del apartado anterior hemos utilizado los objetos DataAdapter de una forma muy sencilla, cada uno de ellos contenía una sentencia SQL, pero también pueden contener varios objetos Command.

El objeto DataAdapter va a poseer cuatro propiedades que nos van a permitir asignar una serie de objetos Command que van a realizar una operación determinada con los datos, estas propiedades son las siguientes:

- InsertCommand: objeto de la clase Command (SqlCommand o OleDbCommand), que se va a utilizar para realizar un alta.
- SelectCommand: objeto de la clase Command que se va a utilizar para ejecutar una sentencia Select de SQL.
- UpdateCommand: objeto de la clase Command que se va a utilizar para realizar una modificación de los datos.
- DeleteCommand: objeto de la clase Command que se va a utilizar para realizar una baja.

Un método destacable de las clases SqlDataAdapter/OleDbDataAdapter es el método Fill(), que ejecuta el comando de selección que se encuentra asociado a la propiedad SelectCommand, los datos obtenidos del origen de datos se cargarán en el objeto DataSet que pasamos por parámetro.

En la Figura 193 se puede ver la relación entre los objetos DataAdapter y el objeto DataSet. Esta figura ha sido extraída del tutorial rápido de ASP .NET (ASP .NET QuickStart Tutorial) que nos ofrece la instalación de Microsoft .NET Framework SDK, si lo tenemos instalado en nuestro equipo podremos acceder a este tutorial mediante nuestro navegador indicando la siguiente URL http://localhost/ quickstart/aspplus/default.aspx.

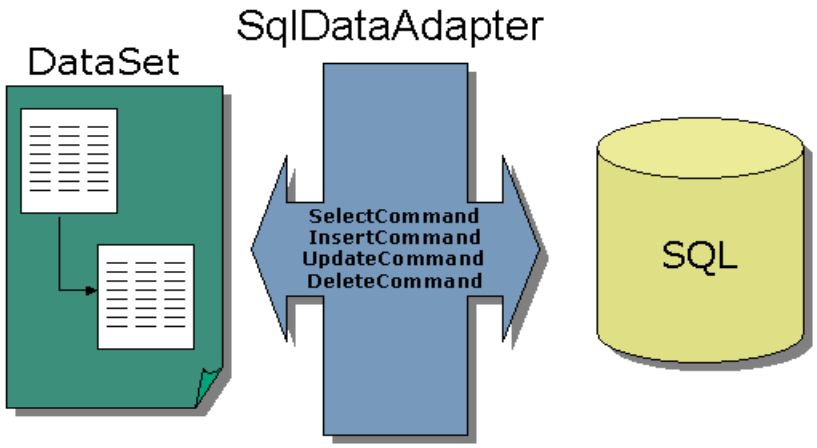


Figura 193

Recomiendo a los lectores que consulten tan interesante y completo tutorial, sobretodo porque aparecen multitud de ejemplo escritos en los lenguajes .NET: Visual Basic .NET, C# y JScript.

Para más información acerca de los objetos DataAdapter remito al lector al texto de mi compañero José Luis Hevia.

Antes de dar por finalizado el capítulo dedicado a ADO :NET, vamos a mostrar la utilización de los objetos DataAdapter mediante un sencillo ejemplo. En nuestra página ASP .NET de ejemplo vamos a utilizar un objeto SqlDataAdapter para dar de alta un registro de una tabla y también para mostrar los contenidos de la misma dentro de un objeto DataSet.

Nuestro objeto SqlDataAdapter va a contener dos objetos SqlCommand, uno que permite la inserción, que se asignará a la propiedad InsertCommand del objeto SqlDataAdapter, y otro que permite realizar una sentencia de selección sobre la tabla de la base de datos y cargar el objeto DataSet con los mismos, este objeto SqlCommand se asignará a la propiedad SelectCommand. Como ya hemos dicho anteriormente, al ejecutar el método Fill() del objeto SqlDataAdapter se ejecutará el comando de la propiedad SelectCommand.

El Código fuente 327 es el código completo de la página ASP .NET del ejemplo, este código se puede descargar aquí.

```
<%@ Page language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<html>
<head><title>El objeto DataAdapter</title></head>
<body>
<script runat="server">

SqlConnection conexion;
SqlDataAdapter adapter;
String sentenciaInsercion="INSERT into Employees (firstname, lastname) "+
                    "VALUES(@nombre,@apellidos)";
String sentenciaSeleccion="select firstname, lastname from Employees";
SqlCommand comandoInsercion;
SqlCommand comandoSeleccion;
DataSet ds;

void Page_Load(Object sender, EventArgs args){
      conexion =new SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
      adapter=new SqlDataAdapter();
      comandoInsercion=new SqlCommand(sentenciaInsercion,conexion);
      comandoSeleccion=new SqlCommand(sentenciaSeleccion,conexion);
      adapter.InsertCommand=comandoInsercion;
      adapter.SelectCommand=comandoSeleccion;
      ds=new DataSet();
      if(!Page.IsPostBack){
            MuestraDatos();
      }
}

void MuestraDatos(){
      try{
            conexion.Open();
            adapter.Fill(ds, "Empleados");
            tabla.DataSource = ds.Tables["Empleados"].DefaultView;
            tabla.DataBind();
            conexion.Close();
      }catch(SqlException ex){
            Response.Write("se ha producido una excepción: "+ex);
```

```
      }

}

void Alta(Object sender, EventArgs args){
      int resultado;
      try{
         conexion.Open();
         adapter.InsertCommand.Parameters.Add(new SqlParameter("@nombre",
                        SqlDbType.VarChar, 10));
         adapter.InsertCommand.Parameters["@nombre"].Value = nombre.Text;
         adapter.InsertCommand.Parameters.Add(new SqlParameter("@apellidos",
                                                    SqlDbType.VarChar, 20));
         adapter.InsertCommand.Parameters["@apellidos"].Value = apellidos.Text;
         resultado=adapter.InsertCommand.ExecuteNonQuery();
         Response.Write("Se ha añadido "+resultado+" registro");
         conexion.Close();
         MuestraDatos();
      }catch(SqlException e){
            Response.Write("se ha producido una excepción: "+e);
      }
}
</script>

<form runat="server" id="formulario">
<asp:Label Runat="server" ID="textoNombre" text="Nombre"/>
<asp:TextBox Runat="server" ID="nombre"/><br>
<asp:Label Runat="server" ID="textoApellidos" text="Apellidos"/>
<asp:TextBox Runat="server" ID="apellidos"/><br>
<asp:Button OnClick="Alta" Runat="server" ID="boton" Text="Añadir"/>
</form>

<asp:DataGrid id="tabla" runat="server" BorderWidth="1" GridLines="Both"
HeaderStyle-Font-Bold="True" HeaderStyle-BackColor="yellow">
</asp:DataGrid>

</body>
</html>
```

Código fuente 327

En la Figura 194 se puede apreciar la ejecución de este ejemplo del objeto SqlDataAdapter.

En los siguientes capítulos comentaremos los servicios Web (Web Services), veremos como crear y utilizar los servicios Web desde páginas ASP .NET y también veremos como utilizar ADO .NET desde los servicios Web.

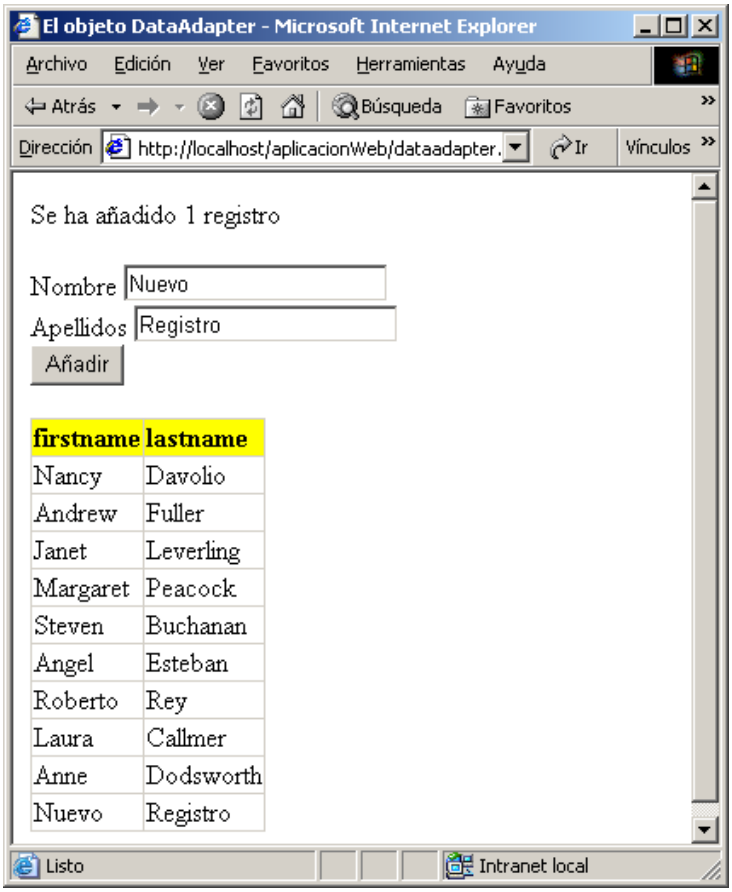

El objeto DataAdapter - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Búsqueda Favoritos
Dirección http://localhost/aplicacionWeb/dataadapter.
Ir Vínculos
Se ha añadido 1 registro
Nombre Nuevo
Apellidos Registro
Añadir
firstname lastname
Nancy Davolio
Andrew Fuller
Janet Leverling
Margaret Peacock
Steven Buchanan
Angel Esteban
Roberto Rey
Laura Callmer
Anne Dodsworth
Nuevo Registro
Listo
Intranet local

Figura 194

21

Creación de servicios Web

Introducción

En este capítulo vamos a realizar una introducción a los servicios Web de ASP .NET y veremos también como construir nuestros propios servicios Web a través de diversos ejemplos. En este capítulo trataremos todo lo relacionado con la creación de servicios Web: implementación del mismo, directivas, atributos, clases relacionadas, etc.

En el siguiente capítulo veremos como publicar los servicios Web para que puedan ser consumidos por los clientes correspondientes, en ese segundo capítulo de dedicado a los servicios Web veremos también los servicios desde el punto de vista del cliente, es decir, desde el punto de vista de utilización del servicio Web que nosotros vamos a crear.

El cliente del servicio Web antes de poder consumirlo o utilizarlo, debe localizarlo y obtener una descripción del mismo, una vez hecho esto deberá seguir una serie de pasos para el servicio Web deseado se encuentre listo para el cliente y lo pueda consumir. Todos estos pasos los veremos con detenimiento en el siguiente capítulo, ahora vamos a pasar a realizar una introducción al nuevo paradigma de desarrollo de aplicaciones que nos ofrece el .NET Framework a través de los servicios Web de ASP .NET.

Introducción a los servicios Web

La plataforma .NET posibilita la extensión del concepto de aplicaciones distribuidas mediante la construcción de servicios Web, que básicamente van a ser funciones albergadas en un sitio Web a las que se puede llamar desde otra página remota (u otro tipo de cliente). Además el .NET Framework

posibilita su fácil programación y manejo tal como veremos a lo largo de los dos capítulos dedicados a los servicios Web. Al fin y al cabo un servicio Web va a ser un tipo de software publicado en un servidor y que va a ser distribuido como un servicio.

Una de las ideas centrales en los que descansa .NET es el concepto de "Web programable". De la misma forma que un componente instalado en un servidor y gestionado mediante Transaction Server o Component Server puede dar servicio a todas las máquinas del dominio de ese servidor, la idea es que los sistemas construidos utilizando datos y servicios provean de múltiples servicios Web. Los servicios Web suministran los elementos básicos para los sistemas, la Web dota de los sistemas de acceso a ellos, y los desarrolladores funden todos esos elementos en un modo útil.

Los Servicios Web pueden ser específicos a una aplicación en concreto, sin embargo existen un buen número de servicios horizontales que la mayoría de las aplicaciones Web necesitan. Esto puede suministrarse mediante bloques de servicios Web. Un ejemplo es Microsoft Passport que pone a disposición una utilidad individualizada de recogida de firmas entre múltiples sitios Web. También hay un mercado para servicios verticales de terceras compañías que implementen alguna funcionalidad común a partir de un segmento de negocio (finanzas y manufacturación, por ejemplo).

El consumidor de un servicio Web puede ser un navegador de Internet, otro dispositivo con conexión a Internet, o una aplicación. Incluso un servicio Web puede, en sí mismo, ser consumidor de otro servicio Web (de la misma forma que un componente COM puede utilizar otro componente como parte de su implementación y funcionamiento).

Podemos decir que los Servicios Web son la extensión natural del concepto de componente que nos ofrece la plataforma .NET. Después veremos que ASP .NET ofrece soporte para construir servicios Web a través de ficheros con la extensión .ASMX. Los servicios Web se pueden considerar como el sistema de publicación de componentes del .NET Framework.

Desde un punto de vista lógico, podemos pensar en un Servicio Web como en un componente, o caja negra, que suministra algún servicio útil a los clientes, o consumidores. Lo mismo que DCOM es "COM de largo alcance", un Servicio Web puede asumirse de verdadero alcance real global. Sin embargo, a diferencia de DCOM, RMI, IIOP, o cualquier otro modelo de objetos común de uso específico, un consumidor accede a un Servicio Web usando un protocolo estándar, aceptado, y bien conocido, como es HTTP, y un formato de datos basado en XML.

Un Servicio Web puede implementarse usando una gran variedad de lenguajes. En la actualidad, C++, Jscript, C#, y VB.NET están soportados, pero probablemente habrá muchos otros en el futuro. En lo que se refiere al consumidor, el lenguaje usado por el servicio Web, e incluso el cómo ese Servicio Web ejecuta sus tareas, es intranscendente.

El punto de vista del consumidor del servicio Web es el de una interfaz que expone un numero de métodos bien definidos. Todo lo que necesita el consumidor es llamar a esos métodos usando los protocolos estándar de Internet, pasando parámetros en formato XML, y recibiendo respuestas también en formato XML.

Los servicios Web también se denominan servicios Web de ASP .NET, ya que es la tecnología ASP .NET la que soporta la base de los servicios Web. Los servicios Web de ASP .NET se basan en una arquitectura sin estado, cada vez que se produce una solicitud de un servicio se crea un nuevo objeto, y una vez ejecutada la llamada al método del servicio correspondientes este objeto es destruido. Para mantener el estado entre distintas peticiones los servicios Web pueden hacer uso de los mecanismos que ofrece ASP .NET para mantener el estado entre distintas solicitudes.

Este capítulo dedicado a los servicios Web va a tener dos partes muy bien diferenciadas, por un lado vamos a tener la parte correspondiente a la construcción de servicios Web, y a continuación vamos a

tener una segunda parte que se va a encargar de mostrar como se pueden utilizar esos servicios Web de ASP .NET desde un cliente determinado.

Arquitectura de un servicio Web

Para el mundo exterior, un Servicio Web es una aplicación que acepta peticiones de servicio, realiza algún proceso, y devuelve una Respuesta. Por dentro suceden otras cosas, un Servicio Web contiene un Auditor (Listener) que está a la espera de peticiones. Cuando se recibe una Petición, el Listener La reenvía a un componente que implementa la lógica de negocio requerida para la respuesta. El componente se puede diseñar para operar específicamente como un servicio Web o puede ser cualquier otro componente u objeto COM que el servicio quiera exponer al exterior. En este último caso, un desarrollador escribirá algún tipo de lógica que actuará como fachada del servicio Web y envíe la Petición al objeto COM mismo.

El objeto COM u otra lógica de negocio puede utilizar una base de datos u otro mecanismo de almacenamiento, al que se tiene acceso mediante una de acceso a Datos (Data Access layer). Esto recuerda a la arquitectura clásica DNA en la que estamos acostumbrados a que la capa de acceso a datos se utilice para leer información de una base de datos. Sin embargo, no hay nada que impida al servicio Web obtener sus datos de otro Servicio Web, ya sea un Servicio Web genérico de tipo componente, o uno de uso específico.

Gracias a .NET y a su modelo de desarrollo basado en servicios, se flexibiliza y enriquece el modo en el que hasta ahora se construían aplicaciones para Internet. La idea que subyace bajo esta tecnología, es la de poblar Internet con un extenso número de aplicaciones, que basadas en servicios para la web (Web Services), formen un marco de intercambio global, gracias a que dichos servicios están fundamentados en los estándares SOAP y XML, para el intercambio de información.

En este sentido, un programador puede crear Web Services para que sean utilizados por sus propias aplicaciones a modo de componentes (pero de una forma mucho más avanzada que empleando el modelo COM clásico), siguiendo una estructura de programación ya conocida. En la Figura 195 se pude ver un esquema que ilustra esta afirmación.

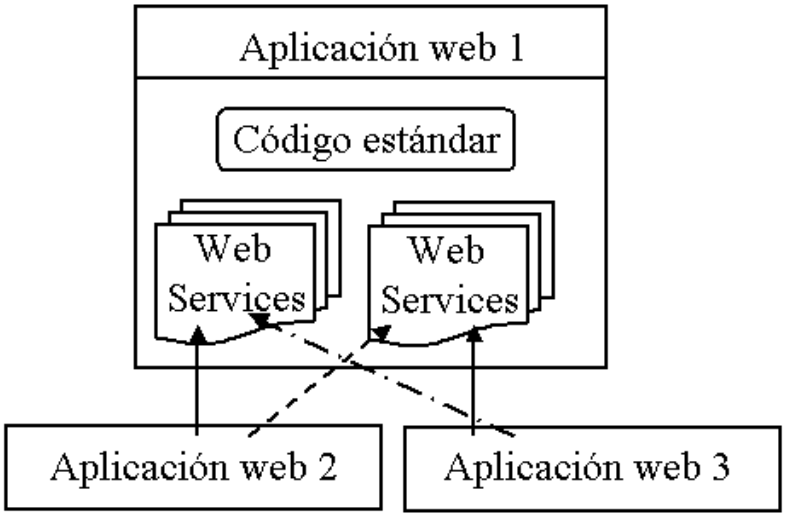


Figura 195

Sin embargo, los Web Services traen de la mano un nuevo modelo de distribución del software; el basado en el desarrollo y publicación de Web Services y en la suscripción a los mismos por parte de otras aplicaciones, potenciales usuarios de tales servicios (Figura 196).

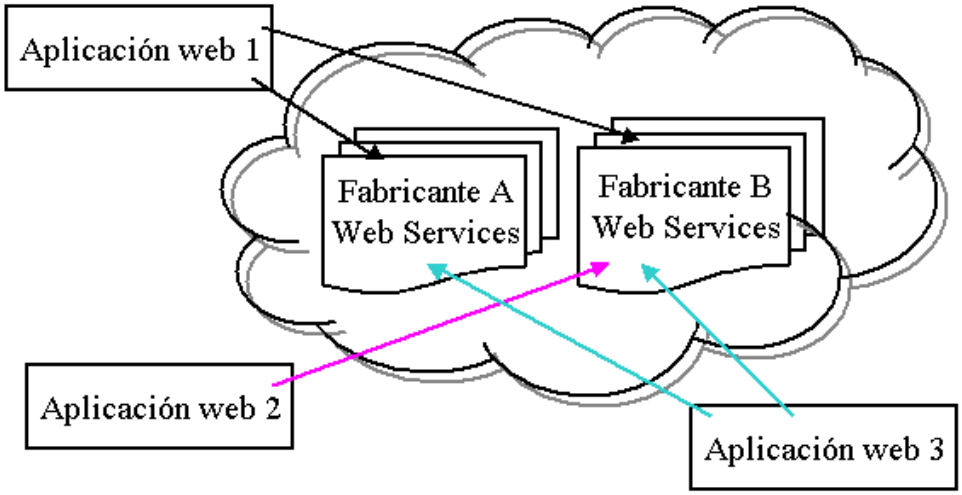


Figura 196

Los fabricantes de software, pueden de esta manera, dedicarse a la creación de servicios web y a su alquiler. Nace de esta manera, la figura del proveedor de servicios web.

Dado el esquema anterior, el programador puede construir sus aplicaciones a base de Web Services, reduciendo significativamente el tiempo y esfuerzo en el desarrollo.

Construcción de servicios Web

En este apartado vamos a tratar en detalle como se construye un servicio Web de ASP .NET a través de un sencillo ejemplo que vamos a comentar paso a paso, este ejemplo no podría ser otro que la creación del típico ejemplo “Hola Mundo” pero en este caso desde un servicio Web. Veremos en detalle la codificación de un servicio Web de ASP .NET

Este apartado está concebido desde el punto de vista del desarrollador del servicio, es decir aquel que construye y define un servicio Web determinado que una vez creado será utilizado o consumido por clientes o consumidores. La utilización de un servicio Web por parte de un cliente, así como su localización y obtención de la descripción del mismo, lo veremos en un próximo capítulo.

Como ya hemos adelantado, en este apartado vamos a mostrar como crear un sencillo servicio Web que nos devuelva el conocido mensaje “Hola Mundo”, me a parecido oportuno mostrar el código fuente de un servicio Web muy simple antes de entrar en el detalle de creación de servicios Web, para que el lector se vaya familiarizando con el código y estructura de los mismos, que como se verá no es muy diferente de cualquier clase del .NET Framework.

Para crear un servicio Web debemos hacer uso de ficheros con la extensión .ASMX, es decir, deberemos añadir a una aplicación ASP .NET un fichero ASMX, que es el mecanismo que nos ofrece ASP .NET para construir servicios Web.

El fichero .ASMX va a ser muy similar a los ficheros .ASPX, es decir, los ficheros de páginas ASP .NET, por lo que los ficheros que contienen servicios Web pueden forma parte de una aplicación ASP

.NET, de hecho es obligatorio que se encuentren en un directorio del servidor Web que sea un inicio de aplicación Web.

Lo más sencillo es crear los ficheros .ASMX dentro de un proyecto de Visual Studio .NET del tipo Aplicación Web ASP .NET que contenga ya una aplicación ASP .NET, o bien de forma separada de las aplicaciones ASP .NET dentro de un proyecto del tipo Servicio Web de ASP .NET que podrán contener todos los ficheros ASMX que van a contener la implementación de nuestros servicios Web, en la Figura 197 se puede ver la selección de este tipo de proyecto dentro de Visual Studio .NET.

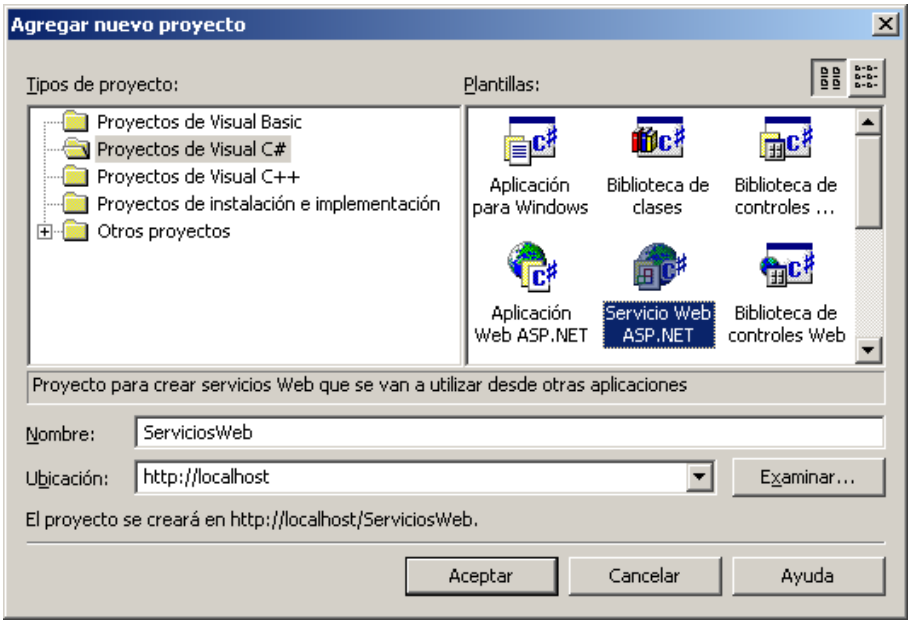


Figura 197

En nuestro caso vamos a crear los servicios Web utilizando el lenguaje C#, que ha sido el lenguaje de la plataforma .NET que hemos utilizado a lo largo de todo el texto para la creación de páginas ASP .NET y todo lo relacionado con las mismas: controles, componentes, etc.

Como ya hemos dicho nuestro servicio Web va a ser muy sencillo y únicamente va a contener un método llamado Saludo() que nos a va a devolver el mensaje “Hola Mundo” en forma de un objeto de la clase String.

El Código fuente 328 es el código fuente del servicio Web que se encuentra dentro del fichero .ASMX, a continuación vamos a comentar los aspectos más relevantes del mismo. Visual Studio .NET va a generar una serie de código fuente de forma automática, lo más recomendable es copiar este código en un fichero ASMX creado con el bloc de notas y luego añadirlo al proyecto.

```
<%@ WebService Language="C#" Class="HolaMundo" %>
using System;
using System.Web.Services;
public class HolaMundo : WebService {
     [WebMethod] public String Saludo() {
          return "Hola Mundo";
     }
}
```

Código fuente 328

Lo primero que se puede observar es la utilización de la directiva WebService, de esta forma indicamos al entorno de ejecución del .NET Framework que estamos implementando un servicio Web, y mediante las propiedades de esta directiva, que veremos en detalle en el apartado correspondiente, podemos indicar como se debe procesar y compilar el servicio Web cuando se recibe una solicitud, es similar a la directiva @Page que presentan las páginas .ASP NET.

En la directiva @WebService hemos utilizado los atributos Language, para indicar el lenguaje .NET que se va utilizar en la implementación del mismo, y Class, para indicar la clase en la que se encuentra al implementación del servicio, esta clase puede encontrarse definida en el mismo fichero ASMX (definición in-line) o bien puede estar separada dentro de un assembly (definición Code-behind),

Si deseamos realizar una definición Code-Behind de nuestro servicio Web, nuestro fichero ASMX contendría el Código fuente 329.

```
<%@ WebService Language="C#" Class="Servicios.Ejemplos.HolaMundo"%>
```

Código fuente 329

En este caso la implementación del servicio Web se encuentra en la clase Servicios.Ejemplos.HolaMundo, que deberemos compilar para generar un assembly, y para que pueda ser utilizado desde el fichero ASMX debemos registrarlo copiando el assembly al directorio BIN del proyecto en el que se encuentra el servicio Web. Esto será ya familiar para los lectores, ya que se sigue el mismo proceso que en las páginas ASP .NET, y aquí el directorio BIN tiene el mismo significado.

El código fuente de la clase sería el siguiente (Código fuente 330).

```
using System;
using System.Web.Services;

namespace Servicios.Ejemplos
{
	public class HolaMundo : WebService
	{

		[WebMethod] public String Saludo()
		{
			return "Hola Mundo";
		}

	}
}
```

Código fuente 330

Una vez comentada la directiva @WebService y vistas las dos posibilidades que tenemos de definición de servicios Web (in-line y Code-Behind), vamos a retomar el comentario del Código fuente 328.

Lo siguiente que aparece después de la directiva @WebService es la definición de la clase que va a implementar el servicio Web. Siempre deberemos importar los espacios de nombre System.Web y System.Web.Services. En este caso la clase que define el servicio Web hereda de la clase System.Web.Services.WebService, esto es opcional, pero a la hora de utilizar objetos de ASP .NET,

como pueden ser los objetos Session o Application, el acceso a los mismos es automático si heredamos de la clase WebService, en caso contrario deberemos utilizar otros mecanismos para acceder a estos objetos, esto los veremos en detalle más adelante en este mismo capítulo.

Y la última particularidad que ofrece la clase del ejemplo es que se utiliza el atributo [WebMethod], para indicar que el método que hemos definido va a ser accesible desde la Web como parte del servicio y que podrá ser utilizado por un cliente, es decir, indicamos que se puede invocar el método a través de la Web (Web callable). El atributo [WebMethod] ofrece una serie de propiedades que veremos más adelante.

Pero además de marcar, los métodos que deseamos que se ejecutan a través de la Web, con el atributo WebMethod, también es necesario declararlos como métodos públicos, haciendo uso del modificador de acceso public cuando se define el método correspondiente.

Comprobando el funcionamiento del servicio Web

Para probar el funcionamiento del servicio Web, no tememos nada más que indicar la URL del fichero ASMX dentro de nuestro navegador. Se debe tener en cuenta que el navegador no está pensado para ser un cliente de un servicio Web, sino que los vamos a utilizar únicamente con propósitos de testeo del servicio Web.

ASP .NET ofrece una plantilla de ayuda para probar los servicios Web, al recibir una solicitud de un fichero .ASMX a través del protocolo HTTP se invoca a la página ASP .NET llamada DefaultWsdlHelpGeneratos.aspx, esta página se encarga de examinar el fichero ASMX y genera la información relativa al uso del servicio Web. Recordamos al lector que esta página ASP .NET de ayuda se puede cambiar e indicar otra página a través de la sección <webservices> de los ficheros de configuración de la aplicación ASP .NET.

En nuestro caso como información se muestra el método Saludo(), tal como se puede ver en la Figura 198. Como podemos observar, la plantilla nos muestra el nombre de nuestro servicio Web, junto con el nombre del único método declarado como WebMethod en el código (Saludo()). Igualmente, nos indica que se está utilizando una especie de firma digital, llamada NameSpace, para diferenciar ese servicio de otros que pudieran pertenecer a otras aplicaciones.

Si continuamos el repaso de la plantilla por defecto, veremos también algunas recomendaciones de utilización, así como direcciones útiles de Internet, donde se encuentran las características de estandarización de los servicios Web.

Si se pulsa sobre el nombre del método Saludo() (que aparece como un enlace) pasamos a una nueva vista de esta página de ayuda para servicios Web de ASP .NET. Ahora aparece un formulario con un botón que nos permite realizar una prueba de ejecución del método Saludo() de nuestro servicio Web, el nuevo aspecto de esta página ASP .NET se puede ver en la Figura 199.

Si pulsamos sobre el botón etiquetado como Invocar se realizará la llamada al método Saludo() y en una nueva ventana aparecerá el resultado de la ejecución del mismo pero en formato XML tal como se puede observar en la Figura 200.

Llegados a este punto, tenemos que aclarar algunas cosas acerca de cómo el servicio nos ofrece la respuesta. En realidad, todo el tráfico de datos que se produce en .NET está basado íntegramente en XML. Al ser XML un estándar universal de soporte de datos en texto plano, cualquier componente, plataforma o programa es capaz de abrir su contenido y procesarlo adecuadamente. En el caso de los Servicios Web, se ha definido y aceptado un estándar de transporte, basado en XML, al que se

denomina SOAP (Simple Object Access Protocol). Funciona sobre HTTP (igual que el HTML) con lo que se consigue su fácil difusión (incluso a través de firewalls), en la red.

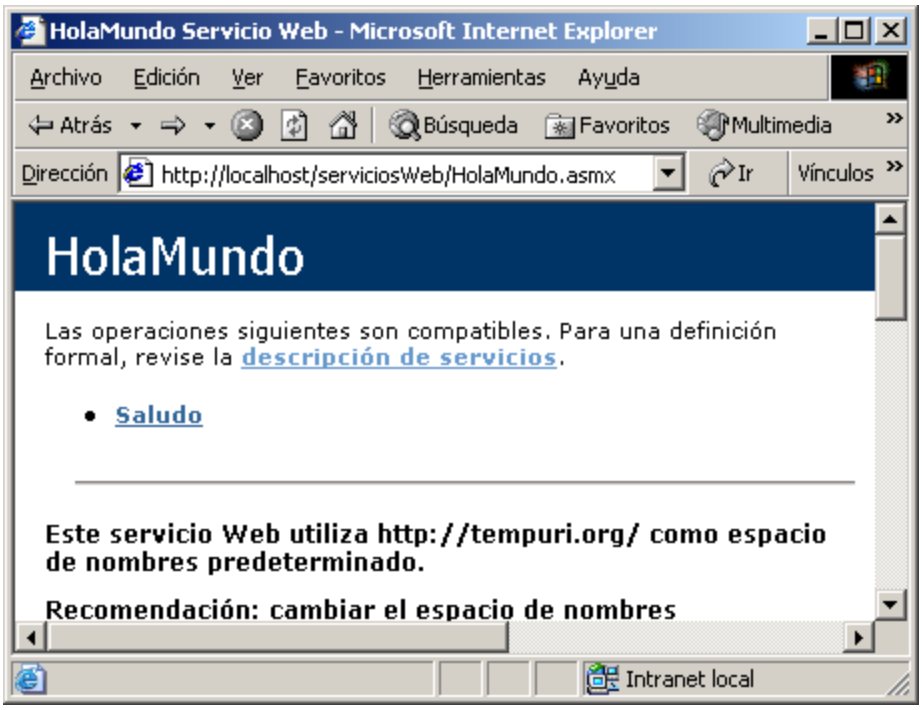


Figura 198

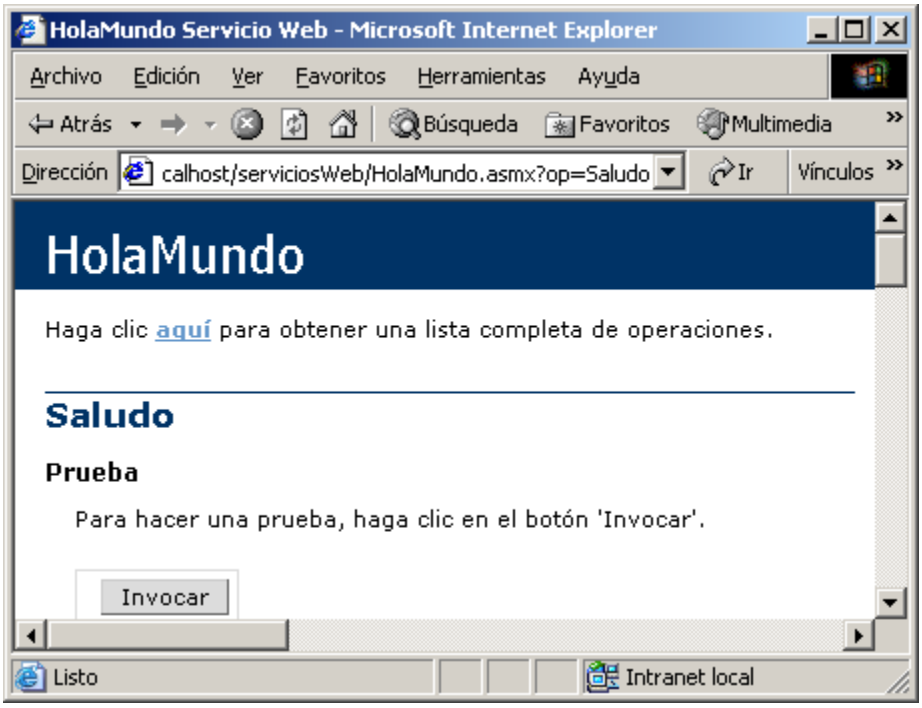


Figura 199

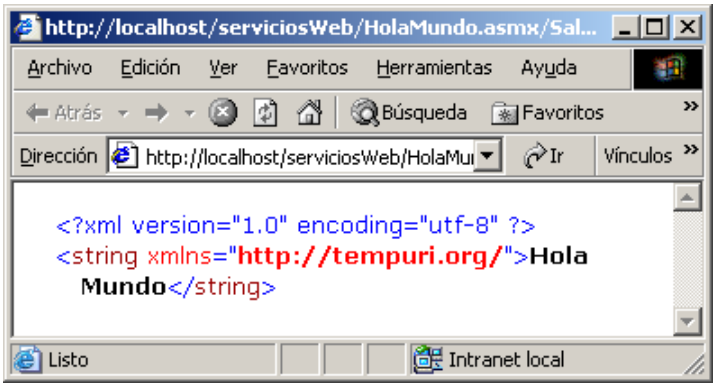


Figura 200

El mecanismo recuerda someramente al de la transmisión de datos por módem (el proceso de modulación-desmodulación). Cuando un componente necesita comunicarse con otro (por ejemplo, cuando una página Web quiere solicitar el resultado de una operación programada como Web Service), SOAP convierte la llamada en un envelope o sobre de transmisión de datos, con sintaxis de XML. Una vez emitida la petición al destinatario, éste realiza la operación contraria (obviamente SOAP deberá estar instalado en ambas máquinas), o sea, decodifica el contenido e invoca el componente solicitado, con los parámetros pedidos, si los hubiera. Cuando el componente de destino responde, vuelve a crear un SOAP Envelope de respuesta que es interpretado por el emisor. En nuestro caso, el emisor produce una página de respuesta en XML estándar como ya hemos visto en la Figura 200.

Vemos que, simplemente, se limita a devolver una "stream" o corriente de caracteres en formato XML, que el navegador es capaz de interpretar y mostrar con la plantilla predeterminada, donde vemos el resultado (Hola Mundo), como contenido de una única etiqueta string que hace las veces de nodo raíz del documento (el espacio de nombres, http://tempuri.org/, es el que se utiliza como predeterminado para este tipo de operaciones, más adelante veremos como indicar nuestro propio espacio de nombres para el servicio Web).

Si volvemos al código fuente de nuestro ejemplo podemos añadir a nuestro servicio Web un método más, en este caso se va a tratar de un método llamado SaludoNombre() que recibe como parámetro un objeto de la clase String que va a ser el nombre de la persona a la que deseamos saludar. El Código fuente 331 sería el nuevo contenido de nuestro fichero ASMX.

```
<%@ WebService Language="C#" Class="HolaMundo"%>

using System;
using System.Web.Services;

public class HolaMundo : WebService
{
	[WebMethod] public String Saludo()
	{
		return "Hola Mundo";
	}

	[WebMethod] public String SaludoNombre(String nombre)
	{
		return "Hola "+nombre;
	}
}
```

Código fuente 331

Si probamos este servicio Web de la forma que ya habíamos visto desde nuestro navegador, y si seleccionamos el método SaludoNombre(), en este caso se muestra un formulario distinto, aparece una caja de texto para indicar el valor del parámetro de este método tal como se puede apreciar en la.

En la ya veíamos que aparecía un enlace llamado "descripción de servicios", si pulsamos sobre este enlace se nos ofrece el Web Service Description Language (WSDL), que es el lenguaje utilizado en la descripción de un servicio Web. Este lenguaje es un estándar oficial de la W3C, que define una gramática XML que debe utilizarse a la hora de describir un servicio Web y permite describir todos los datos necesarios para utilizar el servicio. En otras palabras, para que un cliente utilice un servicio Web, necesita saber qué métodos (mensajes) es capaz de comprender el servicio, qué parámetros admite y qué tipo valores devuelve.

Más adelante en el texto volveremos a retomar el lenguaje de descripción de los servicios Web (WSDL) a través de distintas referencias al mismo.

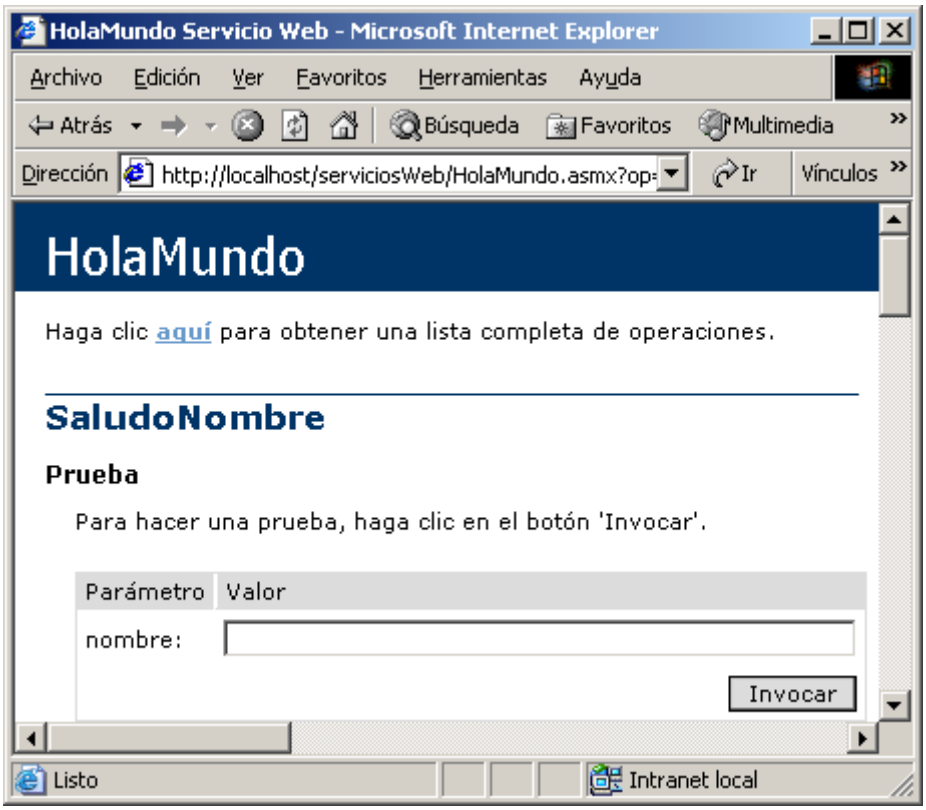


Figura 201

También podemos obtener el WSDL de un servicio Web de forma directa solicitando el fichero ASMX en el navegador pero añadiendo la cadena de consulta ?WSDL, el resultado de realizar esta petición de puede observar en la Figura 202.

Ya hemos visto como crear un servicio Web sencillo y como probarlo desde nuestro navegador, en los siguientes apartados vamos a ver algunos de los puntos que ya hemos adelantado con el ejemplo del servicio HolaMundo, como puede ser la directiva @WebService.

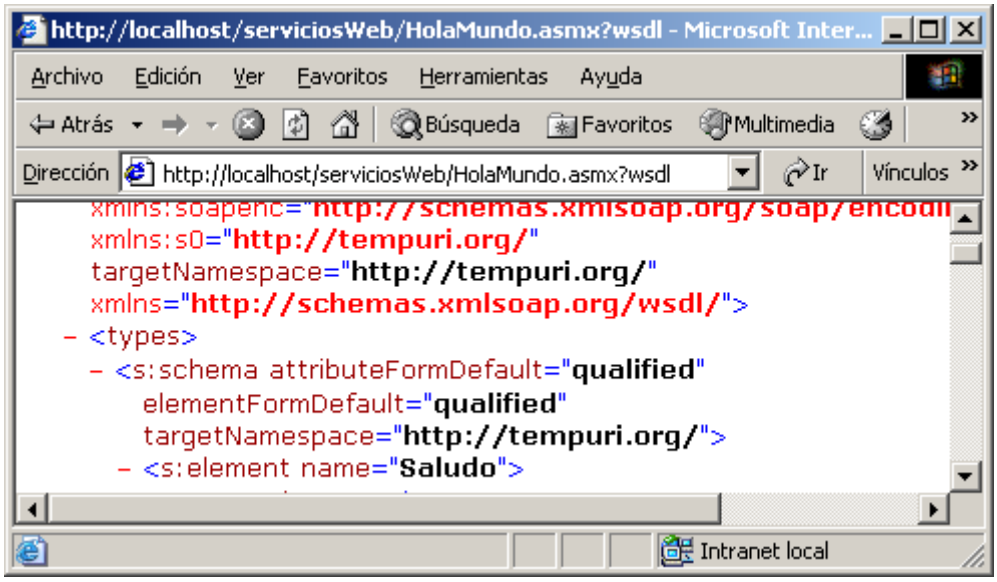


Figura 202

La directiva @WebService

Ya hemos comprobado la utilización de la directiva @WebService dentro del servicio Web que hemos construido en un aparado anterior. Esta directiva únicamente se puede utilizar en ficheros .ASMX, ya se va a utilizar para indicar la clase en la que se encuentra la implementación del servicio Web.

Esta directiva posee dos atributos:

- Class: este atributo es obligatorio y vamos a utilizarlo para indicar el nombre del la clase que implementa el servicio Web. Como ya vimos en el apartado anterior, la clase del servicio Web se puede encontrar definida en el propio fichero ASMX del servicio Web o bien de forma separada dentro de un assembly.

- Language: este otro atributo es opcional y nos permite indicar el lenguaje que se va a utilizar para compilar el servicio Web, su valor por defecto es VB, es decir, se utiliza el lenguaje Visual Basic .NET.

En el Código fuente 332 se puede ver la utilización de esta directiva, en este caso se indica el nombre de la clase completo, incluyendo su NameSpace.

```
<%@ WebService Language="C#" Class="Servicios.Ejemplos.HolaMundo"%>
```

Código fuente 332

En el caso de utilizar un assembly como implementación del servicio Web la directiva @WebService será el único contenido que presentará el fichero ASMX del servicio.

Los atributos WebService y WebMethod

Ya vimos en los distintos ejemplos de creación de un servicio Web la utilización del atributo WebMethod, este atributo se va a utilizar para indicar que una propiedad o método es accesible a través de la Web, los métodos marcados con este atributo se denominan métodos del servicio Web, ya que van a ser los únicos que van a poder ser invocados desde un cliente a través de la red. Aunque estos métodos deben declararse también como públicos para que el atributo WebMethod tenga efectividad sobre ellos y puedan ser ejecutados a través de la Web.

El atributo WebMethod se encuentra representado, dentro de la jerarquía de clases de la plataforma .NET, por la clase System.Web.Services.WebMethodAtribute.

Además de para indicar que un método determinado el servicio Web puede ser invocado a través de la Web, el atributo WebMethod también permite indicar una serie de características de los métodos del servicio. Para ello ofrece seis propiedades.

La sintaxis general que presenta el atributo WebMethod se puede observar en el Código fuente 333.

```
[WebMethod (Propiedad1=valor,...,Propiedadn=valor)]
```

Código fuente 333

A continuación vamos a describir las propiedades que presenta el atributo WebMethod:

- Description: esta propiedad nos permite indicar una breve descripción de la funcionalidad ofrecida por el método del servicio Web. El valor de esta propiedad se añade al lenguaje de descripción del servicio Web, es decir, al WSDL, también es añadida la descripción del método a la página de ayuda que se muestra cuando solicitamos el fichero .ASMX en el navegador.

- EnableSession: permite indicar si se va a utilizar el estado de sesión en el método del servicio Web o no. Por defecto tiene el valor false.

- MessageName: esta propiedad es utilizada para crear un alias de un método o propiedad del servicio Web. El uso más común de esta propiedad es para identificar de forma única un método, cuando éste posee varias versiones del mismo, es decir, se encuentra sobrecargado presentando distintos parámetros pero con el mismo nombre de método.

- TransactionOption: podemos definir mediante esta propiedad el carácter transaccional del método del servicio Web. Se debe tener en cuenta que desde el servicio Web sólo vamos a poder utilizar las transacciones que se hayan iniciado en el propio servicio, las transacciones no podrán ser iniciadas por otras aplicaciones. Esta propiedad puede tomar los siguientes valores (el valor por defecto es Required).

    - Disabled: no es posible controlar las transacciones.

    - NotSupported: el objeto no se va a ejecutar dentro del ámbito de una transacción, sin tener en cuenta de si la transacción existe o no.

    - Supported: si existe una transacción, el objeto se ejecutará dentro del contexto de la transacción, pero si no existe una transacción el objeto no iniciará una nueva.

  - Required: el objeto necesita de una transacción para su ejecución, si ya existe una transacción se ejecutará dentro de la misma, en caso contrario creará una nueva transacción.

  - RequiresNew: el objeto requiere de una nueva transacción, por lo que siempre se creará una nueva.

- CacheDuration: al igual que sucedía con las páginas ASP .NET los servicios Web de ASP .NET también soportan el mecanismo de caché de salida, recuerdo a los lectores que este mecanismo permitía almacenar el resultado de un recurso particular un tiempo determinado, evitando tener que ejecutarse en cada solicitud. El valor de esta propiedad va a ser el tiempo en minutos que se va a mantener en caché el resultado del método del servicio Web. Por defecto no encuentra activado este mecanismo, por o que le valor por defecto de esta propiedad es cero.

- BufferResponse: mediante esta propiedad vamos a indicar si se va a activar o desactivar el búfer en el resultado del método del servicio Web, si se encuentra activado el búfer, la respuesta o resultado ofrecidos por el método se devolverán únicamente cuando se encuentren completos. El valor por defecto de esta propiedad es true.

En el Código fuente 334 se ofrece un nuevo servicio Web que posee dos métodos que devuelven el cuadrado y el cubo del número que le indicamos por parámetro, en este caso se han utilizado para cada método las propiedades Description, BufferResponse y MessageName.

```
<%@ WebService Language="C#" Class="Servicios.Ejemplos.Operaciones"%>

using System;
using System.Web.Services;

namespace Servicios.Ejemplos
{
	public class Operaciones : WebService
	{

		[WebMethod (Description="devuelve el cuadrado de un entero",
		BufferResponse=false,MessageName="CuadradoEntero")]

		public double Cuadrado(int valor)
		{
			return Math.Pow(valor,2);
		}

		[WebMethod (Description="devuelve el cubo de un entero",
		BufferResponse=false,MessageName="CuboEntero")]

		public double Cubo(int valor)
		{
			return Math.Pow(valor,3);
		}

	}
}
```

Código fuente 334

Si probamos este servicio invocando el fichero .ASMX desde el navegador podremos observar que los nombres de los métodos son los alias indicados en el atributo WebMethod, debajo de cada método aparece la descripción, tal como se puede apreciar en la Figura 203.

Figura 203

Así como la utilización del atributo WebMethod resulta obligatoria si deseamos que un método sea un método del servicio Web, es decir, ejecutable a través de la Web, el atributo WebService es opcional, de hecho todavía no aparecido en ninguno de los ejemplos se servicios Web que hemos mostrado hasta ahora.

El atributo WebService nos va a permitir definir una serie de aspectos sobre la clase del servicio Web, en lugar de sobre sus método o propiedades como ocurría con el atributo WebMethod. Las propiedades que nos ofrece el atributo WebService van a tener efecto sobre la página de ayuda que se muestra del servicio Web y también sobre el WSDL.

El atributo WebService se sitúa antes de la declaración del nombre de la clase que implementa el servicio Web.

Este atributo se encuentra representado por la clase System.Web.Services.WebServiceAttribute, y va a presentar las siguiente propiedades:

- Description: esta propiedad nos va a permitir indicar una breve descripción de la funcionalidad del servicio Web. Esta descripción, al igual que sucedía con el atributo WebMethod, se añadirá a la página de ayuda del servicio Web y al WSDL.

- Namespace: esta propiedad es de gran importancia ya que es la que nos va a permitir que nuestros clientes identifiquen nuestro servicio Web para que puedan utilizarlo (consumir). XML utiliza espacios de nombres para identificar de forma única secciones dentro de un documento XML. Un servicio Web va a utilizar un documento XML en el lenguaje de descripción del servicio Web (WSDL), y dentro de ese documento deberemos indicar el espacio de nombres al que pertenece nuestro servicio Web para que se distinga del resto de espacios de nombres. Por defecto a un servicio Web se le asigna un espacio de nombres predeterminado que es http://tempuri.org, este NameSpace será válido mientras que el servicio

se encuentre en desarrollo o pruebas, pero cuando deseemos hacer público el servicio Web para que sea utilizado por nuestros clientes (servicio consumible) deberemos indicar un espacio de nombres distinto, para que nuestro servicio pueda ser distinguido del resto de servicios Web existentes. Se debe señalar que aunque el formato más común de NameSpace de XML es el de una URL, en realidad no tienen porque apuntar a ninguna dirección de Internet existente. Desde la propia página de ayuda del servicio Web se nos indica que es recomendable definir un NameSpace, esto se puede observar en la Figura 204.

- Name: en esta propiedad vamos a poder indicar el nombre del servicio Web, sino indicamos nada en esta propiedad se tomará como nombre del servicio Web el nombre de la clase que lo implementa.

Figura 204

Para ver un ejemplo de utilización del atributo WebService vamos a crear un nuevo servicio Web, en este caso se trata de un servicio que ofrece dos métodos que nos van a devolver información acerca del servidor en el que se encuentra dicho servicio Web, uno de estos métodos nos ofrecerá la hora del servidor y otro el nombre del servidor.

En este caso vamos a realizar una descripción del servicio y le vamos a dar el nombre de MiServicio Web, además vamos a definir nuestro propio espacio de nombres de XML al que va a pertenecer el servicio, todo esto se puede ver en el Código fuente 335, que también hace uso del atributo WebMethod.

```
<%@ WebService Language="C#" Class="Servicios.Ejemplos.DatosServidor"%>

using System;
using System.Web.Services;

namespace Servicios.Ejemplos
{
       [WebService (Description="Servicio Web que ofrece datos del servidor",
       Namespace="http://aesteban.com/servicios", Name="MiServicioWeb")]
       public class DatosServidor : WebService
       {

              [WebMethod (Description=
                                  "devuelve el nombre de la máquina del servidor")]
              public String NombreServidor()
```

```
            {
                  return Server.MachineName;
            }

            [WebMethod (Description="devuelve la hora del servidor")]
            public String HoraServidor()
            {
                  return Context.Timestamp.TimeOfDay.ToString();
            }

      }
}
```

Código fuente 335

Es necesario indicar un NameSpace para nuestro servicio Web ya que cada servicio Web necesita un espacio de nombres único para identificarse, para que las aplicaciones cliente puedan distinguir este servicio de otros servicios del Web. http://tempuri.org/ está disponible para servicios Web que están en desarrollo, pero los servicios Web publicados deberían utilizar un espacio de nombres permanente.

Se debe identificar nuestro servicio Web con un espacio de nombres que controle. Por ejemplo, puede utilizar el nombre de dominio de Internet de nuestra empresa como parte del espacio de nombres. Aunque muchos espacios de nombres de servicios Web parecen direcciones URL, éstos no pueden señalar a un recurso actual en el Web. (Los espacios de nombres de los servicios Web son los URI.).

Parte de comentado en estos últimos párrafos está extraído de la página de ayuda de los servicios Web, y es un texto que aparece cuando no hemos definido un espacio de nombres para nuestro servicio, pero si indicamos el espacio de nombres, como sucede en el ejemplo, no aparecerá este texto de ayuda. En la Figura 205 se puede ver el aspecto que presentaría la página de ayuda para el servicio Web del ejemplo.

En la Figura 205 también se puede apreciar la descripción del servicio Web, así como el nombre del servicio Web, todo ello ha sido extraído de los valores de las propiedades correspondientes del atributo WebService.

Figura 205

Si invocamos el método NombreServidor() desde la página de ayuda del servicio Web obtendremos un resultado similar al de la Figura 206, en este caso se puede comprobar que se ha utilizado el espacio de nombres que hemos definido para nuestro servicio.

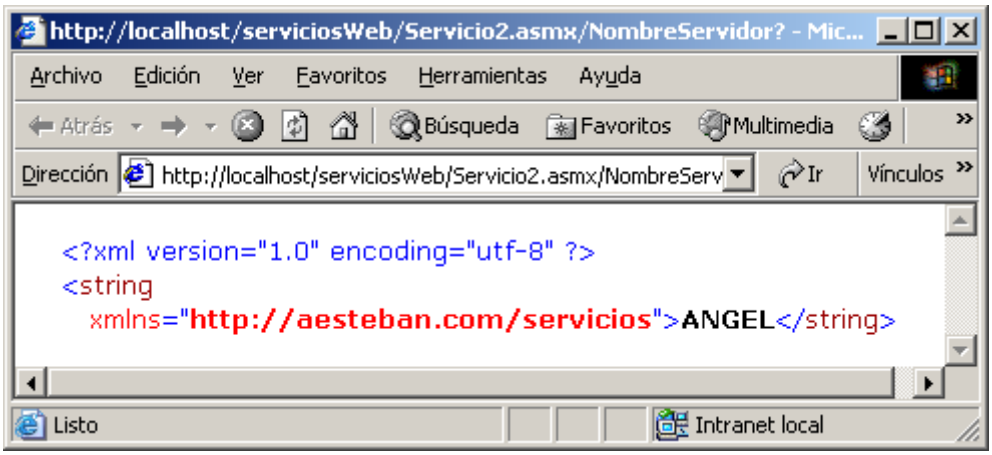


Figura 206

En el Código fuente 335 se puede observar que hemos utilizado un objeto Server para obtener a través de su propiedad MachineName el nombre de la máquina del servidor, este objeto es de la clase System.Web.HttpServerUtility, es decir, de la misma clase que la propiedad Server que presenta el objeto Page dentro de las páginas ASP .NET. En realidad se está haciendo referencia al mismo objetos desde el servicio Web a cómo lo hacíamos en las páginas ASP .NET. Lo mismo sucede con el método HoraServidor(), en el que se hace uso de la propiedad Context que es un objeto de la clase System.Web.HttpContext.

Todo esto es posible debido a que heredamos de la clase WebService y los objetos Server y Context son algunas de las propiedades de esta clase que vamos a tratar en el siguiente apartado.

La clase WebService

La clase System.Web.Services.WebService es la que representa a los servicios Web de ASP .NET de la jerarquía de clases del .NET Framework, y es la clase que hemos estado utilizando en los ejemplos de este capítulo como clase padre de nuestros servicios Web.

Si la clase que implementa un servicio Web heredad de la clase WebService, tendremos la ventaja de poder acceder directamente o indirectamente a los objetos que son propiedades de la clase Page de ASP .NET, tal como veíamos en el último ejemplo de servicio Web del apartado anterior, y que se corresponden con los antiguos objetos integrados de ASP, es decir, el objeto Session, Application, Server, Request y Response, además de a otros objetos de ASP .NET como puede ser el objeto Trace o User.

No debemos olvidar que un servicio Web puede formar parte de una aplicación ASP .NET, al igual que accedíamos a una serie de objetos a través de propiedades del objeto Page en el caso de páginas ASP .NET, en los servicios Web también podemos acceder a estos objetos. Para ello tenemos dos opciones, la más sencilla de ellas ya la estamos comentando en estos momentos, y consiste simplemente en que la clase que implementa el servicio Web debe heredar de la clase System.Web.Services.WebService. Un poco más adelante, dentro de este mismo apartado, veremos la segunda opción disponible.

Una página ASP .NET y un servicio Web de ASP .NET que residan en la misma aplicación Web van a tener acceso al mismo espacio de memoria en el que se mantiene el estado de la aplicación, de esta forma los objetos añadidos a los objetos Session, Cache o Application van a encontrarse accesibles desde las páginas ASP .NET y desde los servicios Web.

Vamos a centrarnos ahora en la clase WebService y vamos a comentar las propiedades principales que nos ofrece esta clase:

- Application: esta propiedad nos va a ofrecer una referencia a un objeto de la clase System.Web.HttpApplicationState, este objeto nos va a permitir almacenar y acceder a información que va a ser común a toda la aplicación Web, es decir, tiene la misma funcionalidad que la propiedad del mismo nombre de la clase Page y esta propiedad es equivalente al objeto integrado Application de anteriores versiones de ASP.

- Context: propiedad de la clase System.Web.HttpContext, esta propiedad ofrece información sobre la solicitud actual, esta clase va a encapsular toda la información referente a una petición HTTP individual. Esta propiedad es muy importante, ya que a través de esta propiedad también podemos tener acceso a distintos objetos de ASP .NET que no tenemos disponibles directamente como propiedades de la clase WebService, sí por ejemplo podemos obtener referencias a los siguientes objetos: Cache, Trace, Request o Response.

- Session: propiedad de la clase System.Web.SessionState.HttpSessionState, va a ser utilizada para mantener el estado de sesión, tiene el mismo significado que para las páginas ASP .NET, es decir, nos va a permitir almacenar información entre distintas solicitudes que pertenecen a una misma sesión.

- Server: ofrece una referencia a un objeto de la clase System.Web.HttpServerUtility, este objeto tiene la misma funcionalidad que el objeto Server que obtenemos a través de la clase Page dentro de una página ASP .NET, es decir, es un compendio de utilidades que permiten realizar una serie de operaciones sobre las peticiones recibidas. La utilización de esta propiedad ya la hemos visto en un ejemplo de servicio Web en el apartado anterior.

- User: propiedad que devuelve información sobre el usuario que ha realizado la petición actual, es una característica de seguridad.

Vamos a crear un servicio Web que va utilizar dos objetos que se van a almacenar en el estado de sesión y en el estado de aplicación, respectivamente, y que van a realizar la función de dos contadores. Este servicio nos va a ofrecer dos métodos que se pueden invocar a través de la Web que nos van a permitir aumentar el valor de cada uno de estos contadores.

El valor del contador almacenado en el estado de sesión únicamente se mantendrán en la sesión actual, sin embargo el segundo contador, almacenado a nivel de aplicación, se mantendrá a lo largo de toda la vida de la aplicación Web en la que se encuentra nuestro servicio Web de ASP .NET.

Evidentemente para almacenar los contadores haremos uso de los objetos Application y Session tal como lo hacíamos en las páginas ASP .NET. Estos objetos los tenemos disponibles como propiedades de a la clase WebService de la que heredamos.

A continuación se ofrece el contenido del fichero ASMX que contiene al servicio (Código fuente 336).

```
<%@ WebService Language="C#" Class="Contador" %>

using System;
```

```
using System.Web.Services;
[WebService (Description="Servicio de contadores",Name="ServicioContador",
Namespace="http;//aesteban.com/servicios")]
public class Contador : WebService {
   [ WebMethod(EnableSession=true,
   Description="Aumenta el contador a nivel de sesión") ]
   public String AumentaContadorSesion() {
        if (Session["contadorSesion"] == null) {
            Session["contadorSesion"] = 1;
        }
        else {
            Session["contadorSesion"] = ((int) Session["contadorSesion"]) + 1;
        }
        return "El valor del contador de sesión es: " +
                                  Session["contadorSesion"].ToString();
   }
   [ WebMethod(EnableSession=false,
    Description="Aumenta el contador a nivel de aplicación")]
   public String AumentaContadorAplicacion() {
        if (Application["contadorAplicacion"] == null) {
            Application["contadorAplicacion"] = 1;
        }
        else {
            Application["contadorAplicacion"] =
                                  ((int) Application["contadorAplicacion"]) + 1;
        }
        return "El valor del contador de aplicación es: " +
                    Application["contadorAplicacion"].ToString();
   }
}
```

Código fuente 336

Como se puede comprobar se utilizan también los atributos WebService y WebMethod de la misma forma que vimos en el apartado anterior. Se debe destacar que para poder utilizar el estado de sesión dentro del método AumentaContadorSesion() hemos tenido que asignar a la propiedad EnableSession del atributo WebMethod el valor true, ya que por defecto en los servicios Web no está activado el estado de sesión. En la Figura 207 se puede observar la página de ayuda que se mostrará para este servicio.

Figura 207

En la Figura 208 se puede ver el resultado de al ejecución del método AumentaContadorAplicacion().

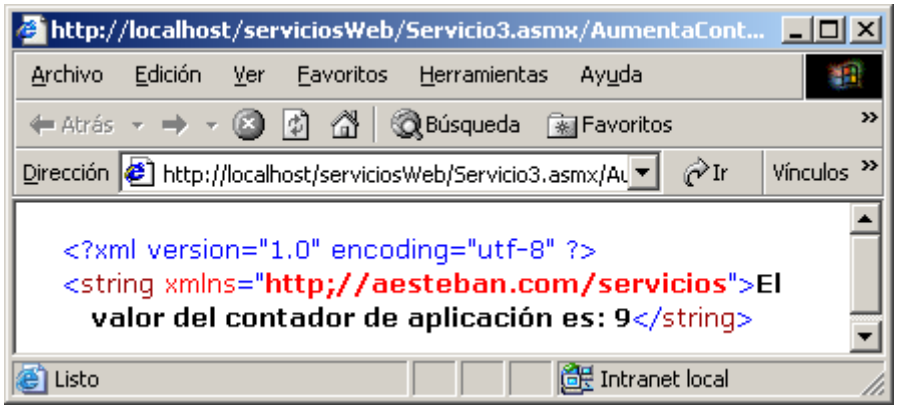


Figura 208

Como ya hemos comentado el acceso a estos objetos de ASP .NET es posible gracias a las propiedades de la clase padre WebService, pero si no heredamos de la misma, debido por ejemplo a que heredamos de otra clase (recordamos al lector que en la plataforma .NET no se soporta el mecanismo de herencia múltiple), no podremos acceder directamente a estos objetos a través de propiedades, sino que tenemos que seguir otra estrategia que se comenta a continuación.

Para tener acceso a objetos de ASP .NET desde un servicio Web que posee una clase que lo implementa que no hereda de la clase WebService tendremos que declarar dentro de los métodos correspondientes objetos de las clases deseadas, es decir, si tenemos que utilizar objetos Application o Session dentro de nuestros métodos, como sucedía en el ejemplo anterior, tendremos que declarar objetos de las clases System.Web.HttpApplicationState y System.Web.SessionState.HttpSessionState.

Una vez declarados estos objetos los debemos crear asignándoles las referencias de los objetos correspondientes de la aplicación ASP .NET actual en la que se encuentra el servicio Web. Para ello tenemos que utilizar la clase HttpContext. De esta clase utilizaremos una propiedad estática llamada Current. Esta propiedad que nos devolverá un objeto de la clase HttpContext nos ofrece acceso a toda la información de la solicitud actual, incluidos los objetos de ASP .NET, será por lo tanto a través de esta propiedad la forma de tener acceso a los objetos Application y Session.

El Código fuente 337 es el resultado de modificar el ejemplo anterior para que no herede de la clase WebService, pero pueda seguir utilizando los objetos Application y Session. Se puede observar en este código que se ha importando el espacio de nombres System.Web, este NameSpace es necesario para poder utilizar la clase HttpContext y las clases de los objetos Application y Session.

```
<%@ WebService Language="C#" Class="Contador" %>

using System;
using System.Web;
using System.Web.SessionState;
using System.Web.Services;

[WebService (Description="Servicio de contadores",Name="ServicioContador",
Namespace="http;//aesteban.com/servicios")]
public class Contador{

   [ WebMethod(EnableSession=true,
   Description="Aumenta el contador a nivel de sesión") ]
   public String AumentaContadorSesion() {
```

```
        HttpSessionState objSession;

        objSession=HttpContext.Current.Session;
        if (objSession["contadorSesion"] == null) {
            objSession["contadorSesion"] = 1;
        }
        else {
            objSession["contadorSesion"] =
                        ((int) objSession["contadorSesion"]) + 1;
        }

        return "El valor del contador de sesión es: " +
                                objSession["contadorSesion"].ToString();
    }

    [ WebMethod(EnableSession=false,
    Description="Aumenta el contador a nivel de aplicación")]
    public String AumentaContadorAplicacion() {
        HttpApplicationState objApplication;

        objApplication=HttpContext.Current.Application;
        if (objApplication["contadorAplicacion"] == null) {
            objApplication["contadorAplicacion"] = 1;
        }
        else {
            objApplication["contadorAplicacion"] =
                        ((int) objApplication["contadorAplicacion"]) + 1;
        }

        return "El valor del contador de aplicación es: " +
                    objApplication["contadorAplicacion"].ToString();
    }
}
```

Código fuente 337

En el siguiente enlace se encuentra el fichero ASMX de este servicio Web.

Para finalizar este capítulo dedicado a la construcción de servicios Web, y antes de pasar al siguiente en el que trataremos la forma de localizarlos y consumirlos por parte de los clientes, vamos a ver un ejemplo más de servicio Web, en este caso se trata de un servicio Web algo más complejo, ya que ofrece un acceso a datos utilizando ADO .NET.

Servicios Web de acceso a datos

Hasta ahora se han mostrado servicios Web de ejemplo bastante sencillo, cosa que resulta normal, ya que soy de la opinión que los nuevos conceptos se comprenden mejor a través de ejemplos sencillos. En este último apartado vamos a mostrar unos ejemplos algo más complejos.

En este apartado, en primer lugar, vamos a mostrar con un ejemplo de servicio Web que nos devuelve un el contenido de dos tablas de una base de datos utilizando para ello un objeto DataSet de ADO .NET. Para utilizar los objetos de acceso a datos ofrecidos por ADO .NET no se requiere hacer nada especial en el servicio Web, únicamente debemos importar el espacio de nombres adecuado y hacer el uso que sea necesario de los objetos, al igual que hacíamos en las páginas ASP .NET.

Nuestro servicio va a tener un único método público que se va a poder consumir por parte de un cliente, se trata del método DevuelveTablas(), que como resultado nos va a devolver un objeto DataSet

con el contenido de la tabla empleados y de la tabla de distribuidores. Para obtener el contenido de estas tablas se va a hacer uso de los objetos SqlDataAdapter.

El Código fuente 338 es el código de este servicio Web de acceso a datos.

```
<%@ WebService Language="C#" Class="Servicios.Ejemplos.Datos"%>

using System;
using System.Web.Services;
using System.Data;
using System.Data.SqlClient;

namespace Servicios.Ejemplos
{
       [WebService (Description="Servicio Web que ofrece un DataSet con dos tablas",
       Namespace="http://aesteban.com/servicios", Name="ServicioDatos")]
       public class Datos : WebService
       {
              [WebMethod (Description=
               "Devuelve el contenido de las tablas distribuidores y empleados")]
              public DataSet DevuelveTablas()
              {
                SqlConnection conexion = new
                      SqlConnection("server=angel;database=northwind;uid=sa;pwd=");
                SqlDataAdapter adapterEmpleados = new SqlDataAdapter
                      ("select firstname, lastname,city from Employees", conexion);
                SqlDataAdapter adapterDistribuidores =
                           new SqlDataAdapter("select * from Shippers", conexion);
                DataSet ds = new DataSet();
                try{
                     conexion.Open();
                     adapterEmpleados.Fill(ds, "Empleados");
                     adapterDistribuidores.Fill(ds, "Distribuidores");
                     conexion.Close();
                     return ds;
                }catch(SqlException ex){
                     Context.Response.Write("se ha producido una excepción: "+ex);
                     return null;
                }
              }
       }
}
```

Código fuente 338

Si probamos el servicio Web con nuestro navegador de la forma qu ella hemos visto en los ejemplos anteriores, al ejecutar el método DevuelveTablas() podemos comprobar que el resultado de su ejecución es más complejo que en casos anteriores, ya que este método va a devolver una estructura de datos en formato XML como es el objeto DataSet.

En el Código fuente 339 se puede ver un fragmento del resultado de la ejecución del método DevuelveTablas(), aquí se puede apreciar la definición de las tablas que contiene el objeto DataSet que devuelve este método.

```
- <DataSet xmlns="http://aesteban.com/servicios">
```

```
-<xsd:schema id="NewDataSet" targetNamespace="" xmlns="" 
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-
com:xml-msdata">
- <xsd:element name="NewDataSet" msdata:IsDataSet="true" msdata:Locale="es-ES">
      - <xsd:complexType>
       - <xsd:choice maxOccurs="unbounded">
        - <xsd:element name="Empleados">
         - <xsd:complexType>
          - <xsd:sequence>
             <xsd:element name="firstname" type="xsd:string" minOccurs="0" />
            <xsd:element name="lastname" type="xsd:string" minOccurs="0" />
            <xsd:element name="city" type="xsd:string" minOccurs="0" />
           </xsd:sequence>
          </xsd:complexType>
         </xsd:element>
        - <xsd:element name="Distribuidores">
         - <xsd:complexType>
          - <xsd:sequence>
             <xsd:element name="ShipperID" type="xsd:int" minOccurs="0" />
             <xsd:element name="CompanyName" type="xsd:string" minOccurs="0" />
             <xsd:element name="Phone" type="xsd:string" minOccurs="0" />
           </xsd:sequence>
          </xsd:complexType>
         </xsd:element>
       </xsd:choice>
     </xsd:complexType>
    </xsd:element>
 </xsd:schema>
```

Código fuente 339

En este enlace se puede obtener el código fuente del servicio Web del ejemplo.

A continuación vamos a crear otro nuevo servicio Web que permite el acceso a datos, en este caso este servicio va a poseer dos métodos. Uno de ellos, DevuelveEmpleados(), va a ser muy similar al visto en el ejemplo anterior, ya que va a devolver dentro de un objeto DataSet el contenido de la tabla de empleados. El otro método que ofrece este servicio Web, AltaEmpleado(), nos va a permitir dar de alta un nuevo empleado utilizando como nombre y apellidos los dos parámetros.

En el Código fuente 340 se ofrece el contenido del fichero ASMX correspondiente a este servicio Web de acceso a datos.

```
<%@ WebService Language="C#" Class="Servicios.Ejemplos.Datos2"%>

using System;
using System.Web.Services;
using System.Data;
using System.Data.SqlClient;

namespace Servicios.Ejemplos
{
       [WebService (Description=
       "Servicio Web que permite ver el contenido de una tabla y añadir registros",
```

```
        Namespace="http://aesteban.com/servicios", Name="ServicioDatos2")]
      public class Datos2 : WebService
      {
            [WebMethod (Description=
            "Devuelve el contenido de las tabla de empleados")]
            public DataSet DevuelveEmpleados()
            {
                  SqlConnection conexion = new SqlConnection(
                              "server=angel;database=northwind;uid=sa;pwd=");
                  SqlDataAdapter adapterEmpleados = new SqlDataAdapter(
                           "select firstname, lastname from Employees", conexion);
                  DataSet ds = new DataSet();
                  try{
                        conexion.Open();
                        adapterEmpleados.Fill(ds, "Empleados");
                        conexion.Close();
                        return ds;
                  }catch(SqlException ex){
                        Context.Response.Write
                                          ("se ha producido una excepción: "+ex);
                        return null;
                  }
            }

            [WebMethod (Description="Da de alta un nuevo empleado")]
            public String AltaEmpleado(String nombre, String apellidos)
            {
                  SqlConnection conexion = new SqlConnection(
                                    "server=angel;database=northwind;uid=sa;pwd=");
                  String sentenciaInsercion=
                        "INSERT into Employees (firstname, lastname) "+
                         "VALUES(@nombre,@apellidos)";
                  SqlDataAdapter adapter=new SqlDataAdapter();
                  SqlCommand comandoInsercion=new
                                          SqlCommand(sentenciaInsercion,conexion);
                  adapter.InsertCommand=comandoInsercion;
                  int resultado;
                  try{
                    conexion.Open();
                    adapter.InsertCommand.Parameters.Add(new
                                SqlParameter("@nombre",SqlDbType.VarChar, 10));
                    adapter.InsertCommand.Parameters["@nombre"].Value = nombre;
                    adapter.InsertCommand.Parameters.Add(new
                                SqlParameter("@apellidos",SqlDbType.VarChar, 20));
                     adapter.InsertCommand.Parameters["@apellidos"].Value =
                                                                        apellidos;
                    resultado=adapter.InsertCommand.ExecuteNonQuery();
                    conexion.Close();
                    return "Se ha añadido "+resultado+" registro";
                  }catch(SqlException e){
                    Context.Response.Write("se ha producido una excepción: "+e);
                    return null;
                  }
            }
      }
}
```

Código fuente 340

Y en el siguiente enlace se puede obtener el código fuente completo del servicio.

Como se puede comprobar en el ejemplo anterior, se han utilizado varios objetos de ADO .NET de la misma forma que lo hacíamos en las páginas ASP .NET.

Con este ejemplo de servicio Web finalizados el capítulo dedicado a la creación de servicios Web, en el siguiente capítulo trataremos como describir los servicios Web y nos ocuparemos de los mismos pero desde el punto de vista del cliente, es decir, el que va utilizar o consumir en sus aplicaciones un servicio Web ya construido.

22

Utilización de servicios Web

Introducción

En el capítulo anterior vimos como construir e implementar servicios Web, así como una introducción a los mismos, en este nuevo capítulo, también dedicado a los servicios Web, vamos a centrarnos en el punto de vista del cliente del servicio Web, ya que un servicio Web es construido para que sea consumido por alguien.

Recordamos al lector que en el capítulo anterior probábamos los servicios Web a través del navegador y gracias a una página ASP .NET de ayuda que nos ofrece el entorno de la plataforma .NET. Pero este no era el entorno de utilización que se le va a dar a los servicios Web ,sino que éstos van a ser utilizados desde una aplicación cliente como puede ser una página ASP .NET dentro de una aplicación Web.

Pero antes de que un servicio Web pueda ser utilizado por una aplicación cliente, sea del tipo que sea, es necesario que el desarrollador o fabricante del servicio describa el propio servicio y además debe registrarlo en un buscador universal de servicios para que el cliente pueda localizarlo y conocer así los servicios Web que ofrecemos.

Una vez que el cliente ha localizado un servicio y a consultado la descripción del mismo, si dicho servicio Web le interesa y desea utilizarlo, debe previamente a su uso, construir una clase proxy que va a servir de puente entre la aplicación cliente y el servicio Web, ya que debemos recordar que los resultados de los servicios Web se devolvían en formato XML, será por lo tanto la clase proxy del servicio Web la que deberá interpretar estos formatos.

La utilización de clases proxy para consumir los servicios Web en aplicaciones cliente nos van a evitar escribir nuestro propio código que empaquete parámetros, los convierta a formato apropiado XML, los

envíe utilizando las estructuras apropiadas SOAP XML, espere la respuesta y la decodifique desde el formato SOAP/XML a su propio formato de datos. Esto lleva bastante trabajo y es propenso a errores.

A lo largo del presente capítulo trataremos las distintas fases de la puesta en marcha de un servicio Web, desde que es implementado por el fabricante (esta fase corresponde a la creación del servicio que ya vimos en detalle en el capítulo anterior) hasta la utilización del servicio dentro de una aplicación cliente.

Las fases de la puesta en marcha de un servicio Web

Antes de seguir en la exposición del capítulo actual, vamos a mostrar mediante una serie de puntos las fases de las que consta la consecución de la puesta en marcha de un servicio Web para su consumición o utilización por parte de un cliente.

Se ofrece este esquema para que el lector tenga una visión general del proceso, a lo largo del capítulo iremos viendo en detalle algunos de los puntos (sobre todo los relacionados con el consumidor del servicio) que vamos a pasar a describir brevemente a continuación.

Las fases de puesta en macha o utilización de un servicio Web las podemos dividir en dos puntos de vista, desde el punto de vista del fabricante del servicio y desde el punto de vista del consumidor (cliente) del servicio Web. A continuación se comentan los distintos aspectos agrupados por fabricante del servicio Web y por consumidor del servicio Web, en este capítulo nos vamos a centrar en el consumidor del servicio Web, ya que en el capítulo anterior nos centramos en el fabricante del mismo, viendo paso a paso la construcción de diversos servicios Web de ejemplo.

A. Fabricante del servicio: es el encargado de la construcción e implementación del servicio Web.

    1. Implementación del servicio Web: esta es la fase por excelencia del fabricante del servicio y consiste en crear el servicio Web correspondiente ofreciendo una funcionalidad determinada, tal como vimos en detalle en el capítulo anterior.

    2. Publicación del servicio: supone la edición y publicación de una fichero de descripción de servicios que permita a los usuarios potenciales del mismo, conocer qué funciones hemos expuesto para su utilización pública. El lenguaje utilizado en la descripción de un servicio Web es un estándar oficial de la W3C, llamado WSDL (Web Service Description Language), que define una gramática XML que debe utilizarse a la hora de describir un servicio Web y permite describir todos los datos necesarios para utilizar el servicio. En otras palabras, para que un cliente utilice un servicio Web, necesita saber qué métodos (mensajes) es capaz de comprender el servicio, qué parámetros admite y qué tipo valores devuelve. Naturalmente, si somos los autores del servicio y conocemos su implementación, podemos escribir nosotros mismos esa descripción.

    3. Registro del servicio en un buscador universal especializado: se trata, no ya de publicar el servicio en nuestra Web, sino de registrarlo universalmente de forma que cualquiera pueda saber de su existencia. También aquí nos basaremos en un estándar: UDDI (Universal Description, Discovery and Integration). Podemos ver UDDI como una especie de buscador (como puede ser Google) para Servicios Web, que permite saber qué es lo que hay disponible pata nuestros clientes.

B. Consumidor del servicio: va a ser el usuario final o cliente del servicio Web.

1. Descubrimiento del servicio: literalmente, saber que existe, dónde está y cómo está construido.

2. Obtener la descripción del servicio: esta descripción consiste en el fichero WSDL que contiene todas las características de utilización.

3. Utilización propiamente dicha del servicio: con la descripción del servicio, podemos pasar a ponerlo en marcha. Para ello, necesitaremos un puente que nos permita conectar nuestra página con algo que sea capaz de invocar el servicio y pasarle peticiones en un formato adecuado. Esta es la parte más compleja, y por ello .NET ha hecho un esfuerzo considerable para aliviar al programador en esta tarea. Este puente se va a realizar a través de una clase especial denominada proxy.

Descripción y localización de servicios Web

Como ya hemos ido comentando en distintos puntos del capítulo anterior y del actual, la descripción del servicio Web la vamos a realizar mediante el lenguaje de descripción de servicios Web (WSDL, Web Service Definition Language). Este lenguaje es un estándar oficial de la W3C, que define una gramática XML que debe utilizarse a la hora de describir un servicio Web y permite describir todos los datos necesarios para utilizar el servicio.

En el capítulo anterior ya vimos la forma de obtener el documento XML correspondiente al WSDL de un servicio Web determinado, para ello no teníamos nada más que solicitar el fichero ASMX, que contiene el servicio Web, en el navegador pero añadiendo la cadena de consulta ?WSDL, el resultado de realizar esta petición de un servicio determinado se puede observar en la Figura 209.

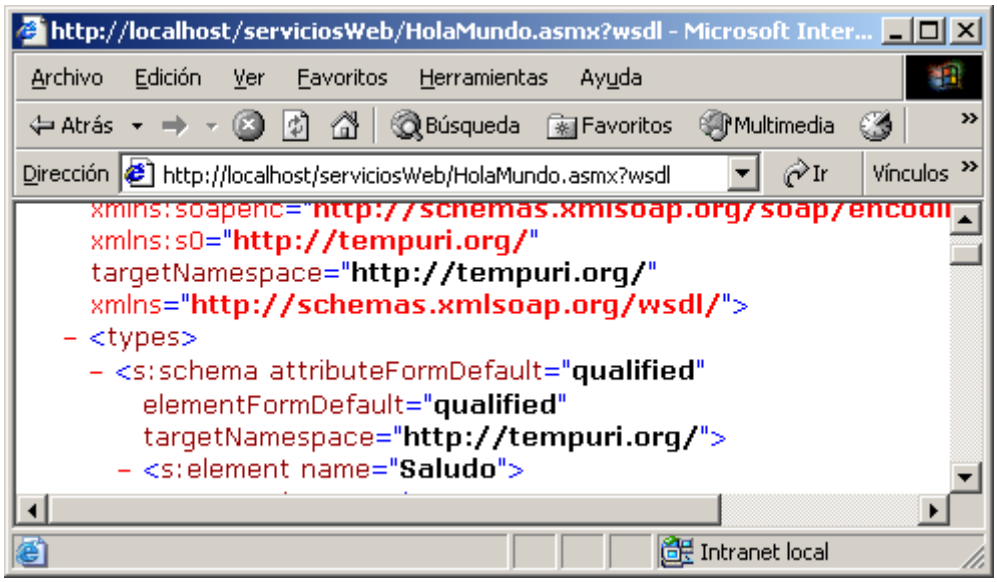


Figura 209

Este documento XML describe los siguiente:

- Cómo es utilizado el servicio Web: la forma en la que el cliente y el servidor van a enviar y recibir mensajes.

- La localización del servicio Web: por ejemplo, la URI a la que se van a enviar los mensajes para interactuar con el servicio.

- La naturaleza del intercambio de mensajes.

Realmente no tenemos porqué conocer el contenido ni entender la sintaxis empleada en el documento XML correspondiente al WSDL de un servicio Web, ya que como veremos más adelante, esta descripción es utilizada por Visual Studio .NET o bien por la herramienta WSDL.EXE (presente en el SDK del .NET Framework) para generar la clase puente (clase proxy) que va a permitir al cliente comunicarse de manera sencilla con el servicio Web correspondiente.

Una vez descrito un servicio determinado mediante el lenguaje WSDL para informar a los clientes de la funcionalidad ofrecida por el servicio así como la descripción de sus métodos, los clientes también debe ser capaces de localizar o rastrear un servicio Web, para ello se ofrecen dos protocolos: el protocolo DISCOVERY (descubrimiento) y el protocolo UDDI (Universal Description, Discovery and Integration). Vamos a comentar la forma de trabajar de cada uno de los dos protocolos.

El protocolo DISCOVERY es anterior a UDDI y es más limitado, su funcionamiento se basa en la existencia de ficheros de descubrimiento. Estos ficheros con extensión .VSDISCO son generados de forma automática por Visual Studio .NET.

Estos ficheros de descubrimiento del protocolo DISCOVERY nos van a permitir determinar que servicios Web se encuentran disponibles y que métodos contiene cada servicio implementado en un servidor determinado, ya que estos ficheros .VSDISCO (también pueden aparecer con extensión .DISCO) van a hacer referencia a los documentos XML del lenguaje WSDL de los servicios existentes en el servidor.

En un servidor Web suele aparecer en el raíz del mismo un fichero default.vsdisco, si abrimos este fichero desde el navegador Web utilizando la URL http://localhost/ default.vsdisco obtendremos el resultado de la Figura 210, que no es otra cosa que una información en formato XML que indica los distintos ficheros .VSDISCO existentes en el servidor Web. En este ejemplo concreto existen tres ficheros .VSDISCO dentro del servidor Web local.

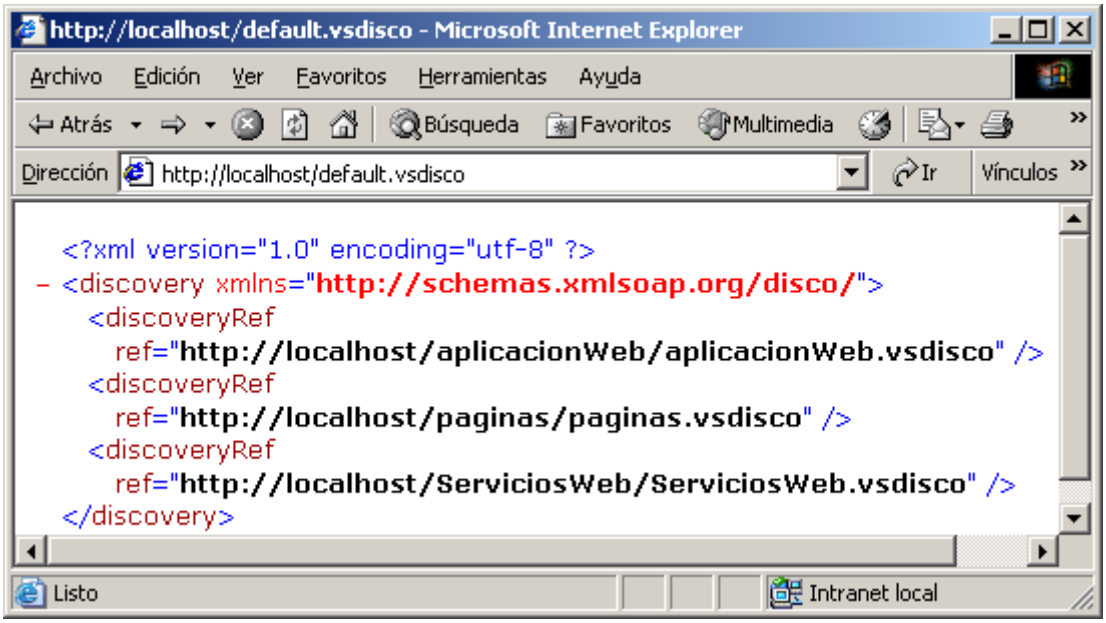


Figura 210

Si abrimos uno de estos ficheros .VSDISCO que pertenece a un proyecto de Visual Studio .NET en el que hemos implementado servicios Web, aparecerá otra información XML, pero en este caso indicando los servicios Web que existen en ese directorio del sitio Web. En la se puede ver que este caso existen ocho servicios Web disponibles.

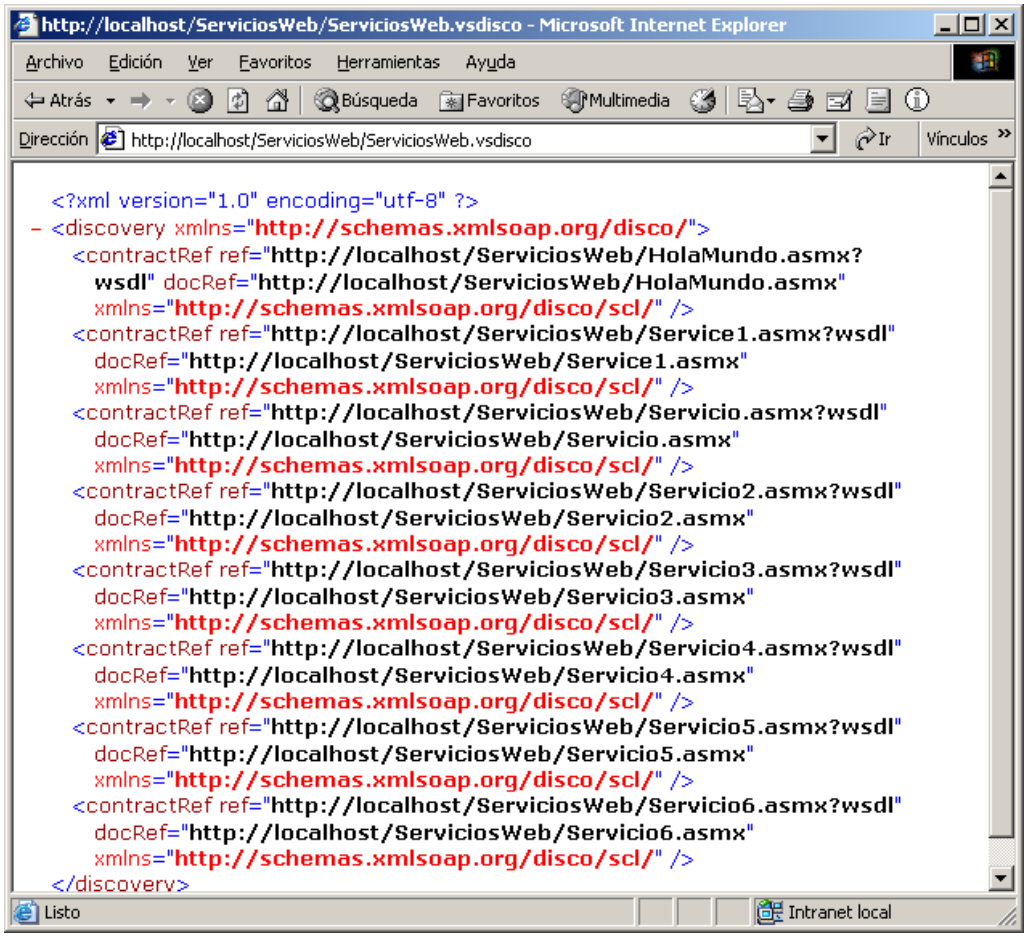


Figura 211

La propuesta que ofrece el protocolo UDDI es más amplia, ya que en el caso del protocolo DISCOVERY es necesario saber la dirección del sitio Web que contiene los servicios, pero en el caso de UDDI nos va a permitir buscar servicios en distintos sitios Web, de forma similar a como lo hacen los buscadores típicos de Internet. UDDI es un estándar de registro y descripción de servicios, y como tal, no depende de una determinada herramienta.

Así si deseamos registrar nuestro servicio Web en UDDI debemos acceder a un servidor de registro UDDI para que nuestros clientes tengan constancia de la existencia del servicio.

Actualmente UDDI es un estándar que se encuentra todavía en desarrollo, por lo que no vamos a entrar en más detalles dentro del protocolo, para más información se puede acceder a la dirección http://www.udi.org. En la Figura 212 se puede ver el servidor de registro UDDI.

Figura 212

Como ya se ha comentado UDDI nos va a permitir a las organizaciones (empresas, compañías, etc...) registrar sus servicios Web que han fabricado, así como localizar servicios de terceros. Actualmente existen tres implementaciones del protocolo UDDI, realizadas por las siguientes compañías:

- Microsoft: la implementación UDDI se encuentra disponible en la dirección http://uddi.microsoft.com.
- Ariba: http://www.ariba.com/corporate/news/uddi.cfm
- IBM: http://www-3.ibm.com/services/uddi/

En la Figura 213 podemos ver un esquema de utilización de los servidores de registro UDDI.

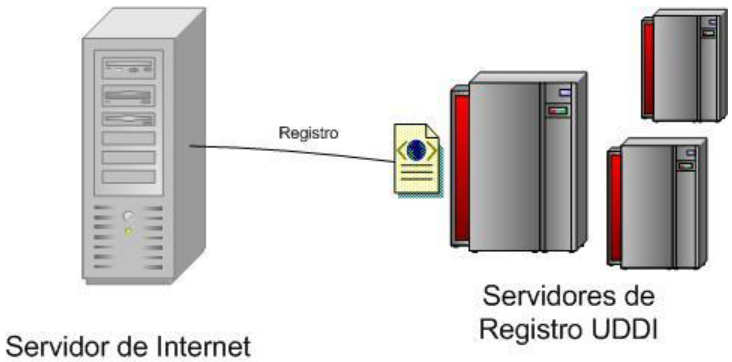


Figura 213

Una vez que el servicio Web posee su descripción y se encuentra registrado, ya sea o a través del protocolo DISCOVERY o bien a través del protocolo UDDI, se puede dar por terminada la labor del fabricante del servicio Web, ya que el mismo estaría listo para su consumición por parte de un cliente.

A continuación vamos a ver mediante un sencillo ejemplo los pasos necesarios para registrar un servicio Web dentro de un servidor UDDI.

Para poder registrar dentro de un servidor UDDI un servicio Web que hemos creado, lo que debemos hacer en primer lugar es registrarnos como usuarios de dicho servicio. A continuación se van a mostrar los distintos pasos a la hora de registrar un servicio Web dentro de un servidor UDDI, en nuestro caso hemos seleccionado el servidor UDDI de Microsoft (http://uddi.microsoft.com). La utilización de estos servidores UDDI es gratuita, tanto para los fabricantes de servicios Web que deseen registrar su servicio, tanto para los clientes de servicios Web que deseen localizar un servicio de los registrados.

Para poder utilizar el servidor UDDI de Microsoft debemos tener una cuenta de Microsoft Passport .NET, para registrarnos como usuarios del servidor UDDI debemos acudir a la opción Register. Una vez que hayamos introducido los datos de nuestra cuenta Passport .NET, consistente en dirección de correo y contraseña (Figura 214), tenemos que añadir unos datos personales. En este momento ya podremos acceder al servidor para registrar nuestros servicios Web. Este paso de registro de usuario en el servidor UDDI es común para los fabricantes de servicios Web y para los consumidores o clientes de los mismos.

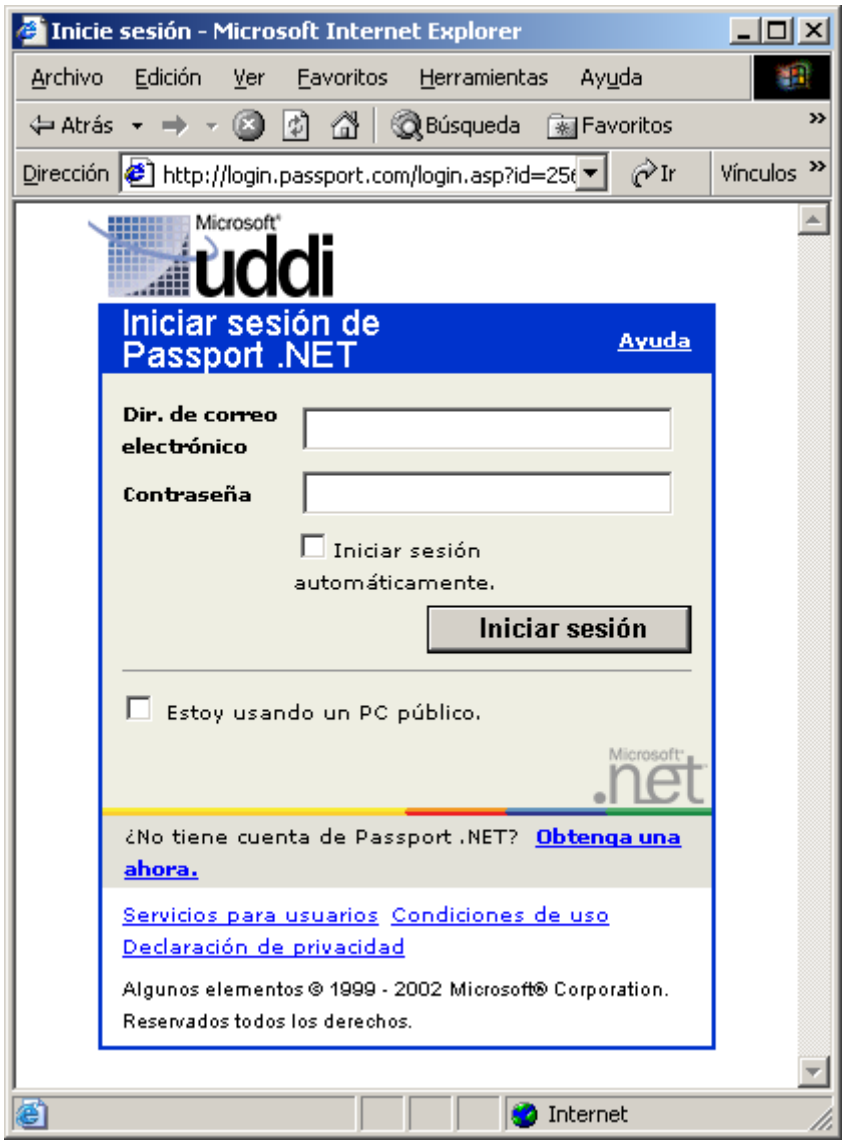


Figura 214

Para registrar un servicio Web debemos ir a la opción Administer y en esta opción debemos seleccionar la posibilidad de crear un nuevo negocio (Add a new business). A partir de aquí debemos rellenar una serie de formularios que nos piden una serie de datos sobre nuestra empresa. El más importante de los formularios y secciones que aparecen es el apartado de Services, en el que daremos una descripción del servicio Web que deseamos registrar en el servidor UDDI (Figura 215).

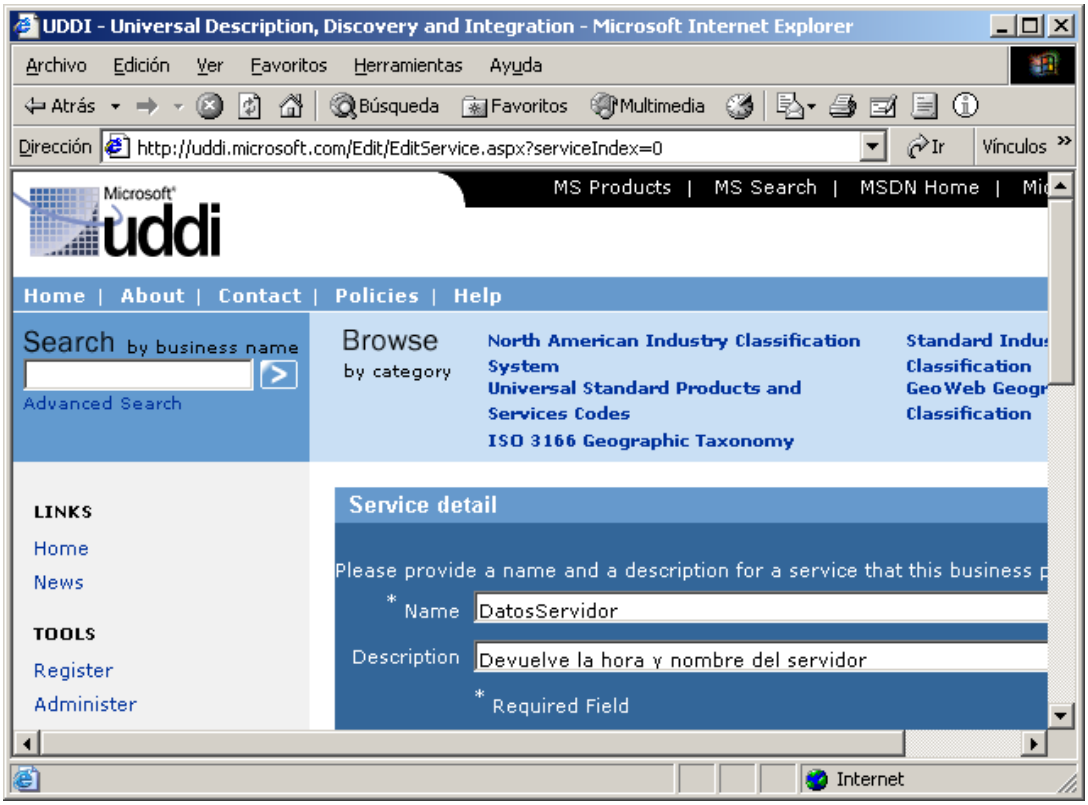


Figura 215

La sección más importante dentro de la opción Services es la que se denomina Bindings. En esta sección indicaremos la dirección en la que se encuentra nuestro servicio Web, es decir, la URL del fichero ASMX que contiene al servicio que estamos registrando. En la Figura 216 se puede ver el aspecto del formulario que nos proporciona el servidor UDDI de Microsoft para indicar la localización de nuestro servicio Web.

En este punto ya estaría registrado nuestro servicio Web. Si deseamos añadir más servicios Web no tenemos nada más que acudir a la sección Services, de nuestro negocio dado de alta dentro del servidor UDDI, y acudir a la opción correspondiente de nuevo servicio.

A continuación vamos a comentar los aspectos relacionados con la utilización de un servicio Web por parte de un cliente. Básicamente la responsabilidad del cliente es la de conocer la dirección que identifica al servicio Web y a continuación generar una clase proxy que se comunique con el servicio Web que desea utilizar.

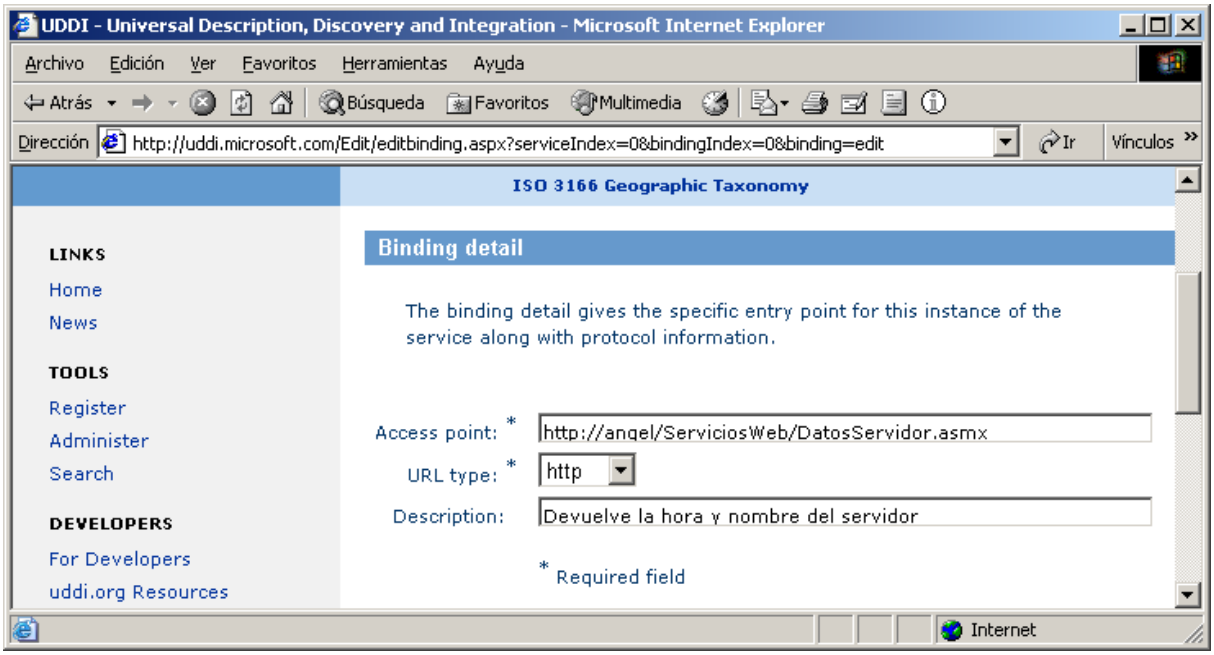


Figura 216

Ahora nos pasamos, como ya hemos dicho, al lado del cliente o consumidor del servicio Web. Vamos a seguir con el ejemplo mostrado hasta ahora mediante el servidor UDDI de Microsoft. En nuestro caso ahora deseamos localizar un servicio Web, para lo que acudiremos a la opción Search, previamente hemos tenido que registrarnos con nuestra cuenta Passport .NET.

En esta opción de búsqueda de servicios Web tenemos diversos criterios de búsqueda, en nuestro caso vamos a buscar por el nombre del negocio, que en el ejemplo anterior le dimos el nombre de DatosServidor (el mismo nombre que a la descripción del servicio). En la Figura 217 se puede ver el resultado de esta búsqueda.

Figura 217

En este punto nuestro cliente ya tiene acceso a toda la información que hemos introducido de nuestra empresa y de los servicios que ofrece. El cliente accederá ala opción Services y dentro de ella a la

opción Bindings, en la que encontrará la dirección en la que se encuentra el servicio Web. Esto se puede observar en la Figura 218.

Bindings

This details the specific access points for this service instance and allows display of additional instance specific details.

Access point	URL type	Description	Binding Key	Instance details
http://angel/ServiciosWeb/DatosServidor.asmx	http	Devuelve la hora y nombre del servidor	AC6018A4-E566-427D-AEC5-EFCCE2F847A0	No further details

Figura 218

En este momento para poder consumir el servicio, al cliente del mismo únicamente le falta generar la clase proxy, que le permitirá utilizar el servicio Web en la aplicación o aplicaciones correspondientes.

Las clases proxy

Una clase proxy creada a partir de un servicio Web nos va a permitir utilizar de forma sencilla el servicio desde un cliente cualquiera.

Debemos recordar que los resultados de los servicios Web se devolvían en formato XML, será por lo tanto la clase proxy del servicio Web la que deberá interpretar estos formatos.

La utilización de clases proxy para consumir los servicios Web en aplicaciones cliente nos van a evitar escribir nuestro propio código que empaquete parámetros, los convierta a formato apropiado XML, los envíe utilizando las estructuras apropiadas SOAP XML, espere la respuesta y la decodifique desde el formato SOAP/XML a su propio formato de datos. Esto lleva bastante trabajo y es propenso a errores.

Lo que debe hacer el cliente es generar la clase proxy, y una vez generada este clase la podrá instanciar en la aplicación correspondiente para utilizar directamente el servicio Web al que pertenece la clase proxy, lanzando los métodos necesarios de manera sencilla.

Antes de pasar a comentar como se construyen las clases proxy vamos a comentar un sencillo ejemplo de utilización de una clase proxy mediante un esquema, mediante este esquema se trata de mostrar la utilidad que ofrecen estas clases a la hora de permitir comunicarnos con servicios Web.

Un ejemplo podría ser un servicio Web de acceso a una base de datos que ha sido creado en una empresa con su central en Madrid. En el servidor Web de esta empresa tienen por lo tanto el fichero ASMX que implementa dicho servicio Web.

El cliente o consumidor de dicho servicio Web va a ser una filial de esta empresa que se encuentra en Barcelona, que va a tener una aplicación ASP .NET, alojada en el servidor Web de Barcelona, desde la que se desea utilizar el servicio Web ofrecido por la central.

El consumidor del servicio será en este caso una página ASP .NET incluida dentro de la aplicación ASP .NET de la filial. Una vez que ya conoce la dirección que identifica al servicio Web implementado en Madrid, se generará en la filial la clase proxy.

Una vez generada la clase proxy se instanciará un objeto de la misma en la página ASP .NET de la aplicación Web, de esta forma para el desarrollador de Barcelona parecerá que está accediendo a una clase local, cuando en realidad lo que esta haciendo es acceder al servicio Web remoto del servidor de Madrid.

La clase proxy y el servicio Web se comunicarán utilizando SOAP, algo que será completamente transparente para el programador de la página ASP .NET, que utilizará de forma directa los métodos de la clase proxy como si se estuvieran ejecutando directamente sobre el servicio Web.

El cliente final del servicio Web será un navegador que accederá a la página dentro de la aplicación ASP .NET, este navegador se encontrará en una localización indeterminada.

En la Figura 219 se puede ver el esquema que ilustra este ejemplo.

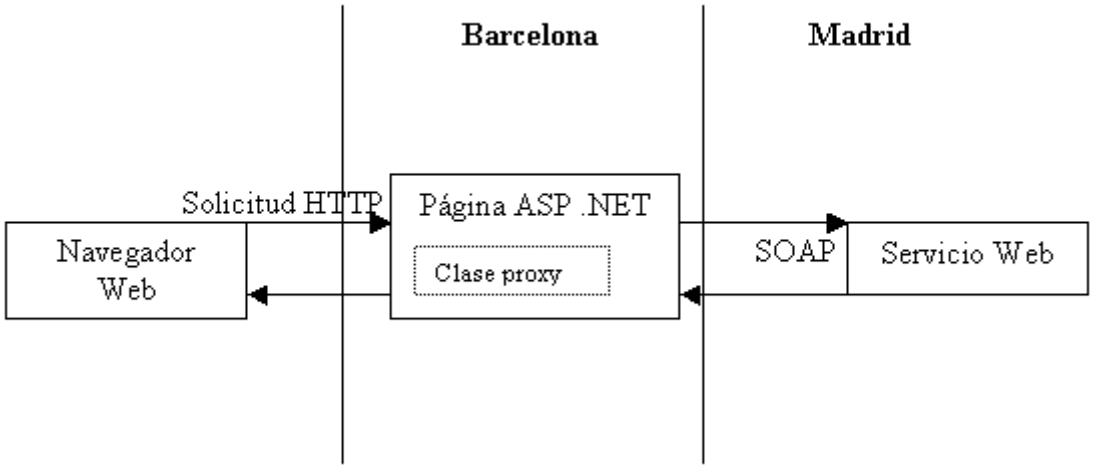


Figura 219

Existen cuatro formas distintas de crear o generar una clase proxy:

- Utilizando el entorno de desarrollo de Visual Studio .NET, a través de la opción del proyecto de añadir una referencia Web. Esta es la opción más sencilla de llevar a cabo.

- Utilizando la herramienta WSDL.EXE desde la línea de comandos, esta herramienta se encuentra disponible como parte del SDK del .NET Framework. Utilizaremos esta opción cuando no dispongamos de Visual Studio .NET.

- Utilizando alguna otra solución como puede ser la herramienta SOAP toolkit de Microsoft.

- Creando nuestra propia clase proxy, esta es la opción más complicada y menos recomendable, ya que deberemos crear una clase que tiene que interpretar el intercambio de información en forma de XML que nos envía el servicio Web.

A continuación vamos a ver dos de las distintas posibilidades que existen a la hora de generar una clase proxy de un servicio Web, en concreto veremos la creación de clases proxy con Visual Studio .NET y con al utilidad WSDL.EXE.

Creación de clases proxy con Visual Studio .NET

Para generar una clase proxy desde Visual Studio .NET y utilizar instancias de la propia clase desde una aplicación de cualquier tipo deberemos acudir a la opción del proyecto de Agregar referencia Web. Esta forma de trabajar con referencias nos puede recordar a los proyectos de Visual Basic 6, que también nos permitían añadir referencias, que en aquel caso hacían referencia a componentes.

En el caso de Visual Studio .NET lo que vamos va a permitir el entorno de desarrollo es incluir una referencia en el proyecto que apunte al fichero ASMX en el que se encuentra implementado un servicio Web determinado.

Para agregar una referencia Web no tenemos nada más que en el explorador de soluciones de Visual Studio .NET pulsar con el botón derecho sobre el proyecto que va a hacer las veces de aplicación cliente o consumidora del servicio Web. En el menú contextual que aparece seleccionamos la opción Agregar referencia Web, tal como se puede apreciar en la Figura 220. El tipo de aplicación consumidora puede ser cualquiera, puede ser tanto una aplicación ASP . NET como una aplicación Windows.

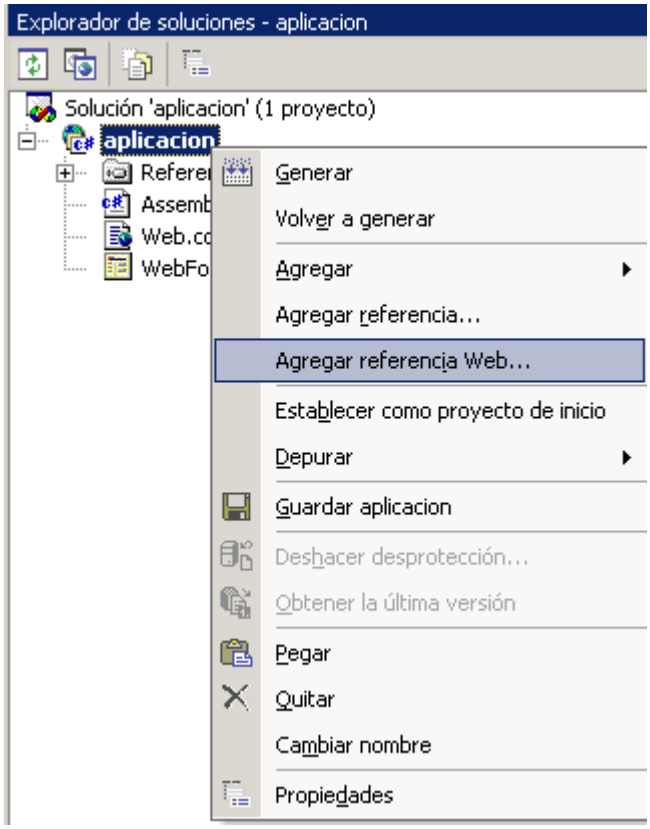


Figura 220

Una vez seleccionada la opción de Agregar referencia Web aparece un diálogo como el de la Figura 221, este diálogo nos permite indicar la dirección en la que se encuentra el servicio Web, es decir, la ruta al fichero ASMX.

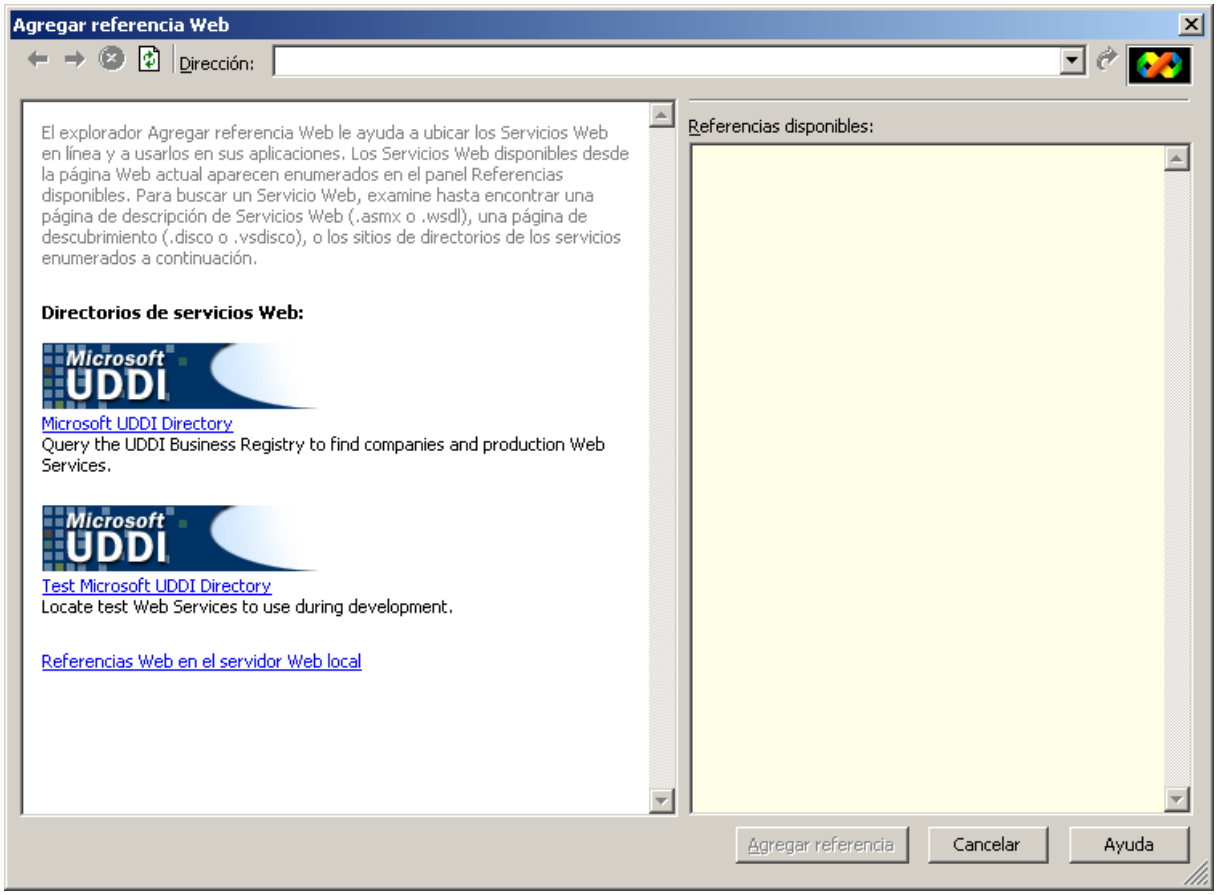


Figura 221

En este diálogo tenemos dos posibilidades, podemos pulsar sobre el enlace que nos llevará a la implementación UDDI de Microsoft para localizar allí el servicio Web que deseamos utilizar en nuestra aplicación (la forma de utilizar el servidor de registro UDDI de Microsoft ya la hemos visto en apartados anteriores), o bien podemos indicar nosotros mismos la dirección que va a identificar al servicio Web que deseamos consumir.

En nuestro caso vamos a indicar la dirección del mismo (http://localhost/serviciosWeb/ServicioImpuestos.asmx). En este ejemplo se trata de un servicio Web que se encuentra en el servidor Web local y el fichero en el que está implementado es el fichero ServicioImpuestos.asmx.

Este servicio es un servicio Web muy sencillo que nos ofrece un método que nos devuelve el importe del impuesto del IVA sobre la cantidad especificada, utilizando también el tipo de IVA que le pasemos por parámetro.

El Código fuente 341 es el código completo de este servicio.

```
<%@ WebService Language="C#" Class="Servicios.Ejemplos.Impuestos"%>

using System;
using System.Web.Services;

namespace Servicios.Ejemplos
{
	[WebService (Description="Servicio Web que calcula impuestos",
```

```
        Namespace="http://aesteban.com/servicios", Name="ServicioImpuestos")]
        public class Impuestos : WebService
        {

                [WebMethod (Description="Devuelve el importe del IVA")]
                public double CalculaIVA(int importe, int tipoIVA)
                {
                        return (importe * tipoIVA) / 100;
                }


        }
}
```

Código fuente 341

En la ventana de diálogo de agregar referencia Web, una vez que hemos indicado la dirección del servicio, mostrará un aspecto similar al de la Figura 222. Como se puede comprobar ofrece la página de ayuda del servicio Web y también dos enlaces que nos van a permitir ver el documento WSDL asociado al servicio (enlace Ver contrato) y la documentación asociada.

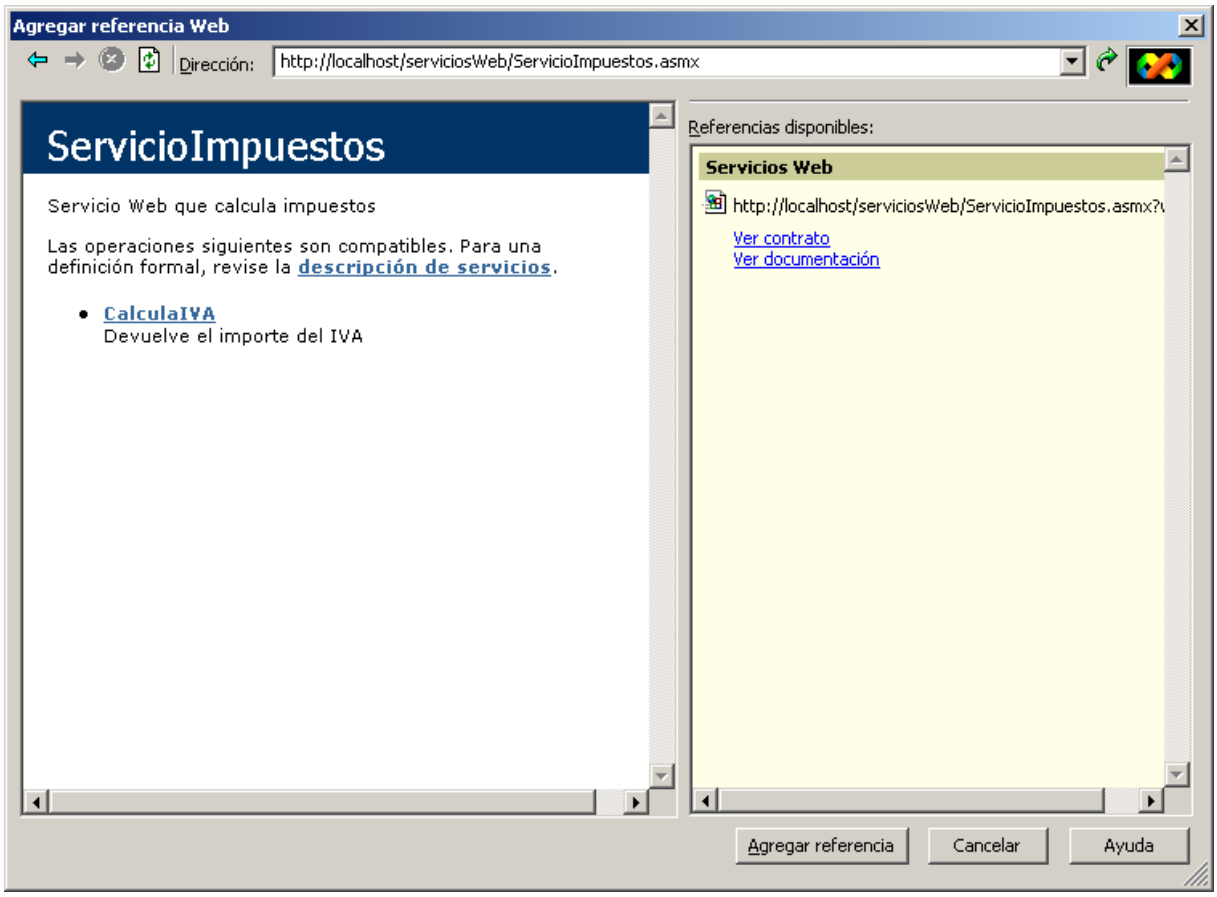


Figura 222

En este momento ya se encuentra activado el botón de Agregar referencia, por lo que ya podemos pulsarlo. Una vez pulsado este botón Visual Studio .NET crea una nueva sección dentro del explorador de soluciones llamada Referencias Web que contendrá todas las referencias Web del proyecto actual.

Internamente Visual Studio .NET ha interpretado el documento WSDL del servicio Web y a partir del mismo ha creado una clase proxy que representa a dicho servicio. El espacio de nombres de la clase proxy es, por defecto, el nombre del servidor en el que se encuentra el servicio Web, en este caso localhost, por lo que el nombre completo de la clase proxy generada es aplicacion.localhost. ServicioImpuestos. También se ha añadido una copia local del documento WSDL, si pulsamos para abrir este elemento aparece la clase proxy que se ha generado de forma automática, cuyo código fuente se puede ver en el Código fuente 342. El código de la clase proxy lo podemos modificar si lo deseamos, aunque no es recomendable hacerlo, ya que puede dejar de funcionar la comunicación con el servicio Web correspondiente. El nombre de esta clase, será por defecto, el nombre indicado en la propiedad Name del atributo [WebService]. El lenguaje utilizado en la clase proxy es por defecto el indicado en el atributo Language de la directiva @WebService, en este caso el fichero de clase resultante será ServicioImpuestos.cs.

```
//------------------------------------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.0.2914.16
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------


namespace aplicacion.localhost {
    using System.Diagnostics;
    using System.Xml.Serialization;
    using System;
    using System.Web.Services.Protocols;
    using System.Web.Services;


    [System.Web.Services.WebServiceBindingAttribute(Name="ServicioImpuestosSoap",
Namespace="http://aesteban.com/servicios")]
    public class ServicioImpuestos :
System.Web.Services.Protocols.SoapHttpClientProtocol {

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public ServicioImpuestos() {
            this.Url = "http://localhost/serviciosWeb/ServicioImpuestos.asmx";
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://aesteban.com/ser
vicios/CalculaIVA", RequestNamespace="http://aesteban.com/servicios",
ResponseNamespace="http://aesteban.com/servicios",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
        public System.Double CalculaIVA(int importe, int tipoIVA) {
            object[] results = this.Invoke("CalculaIVA", new object[] {
                        importe,
                        tipoIVA});
            return ((System.Double)(results[0]));
        }


        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public System.IAsyncResult BeginCalculaIVA(int importe, int tipoIVA,
System.AsyncCallback callback, object asyncState) {
            return this.BeginInvoke("CalculaIVA", new object[] {
                        importe,
```

```
                        tipoIVA}, callback, asyncState);
        }

        [System.Diagnostics.DebuggerStepThroughAttribute()]
        public System.Double EndCalculaIVA(System.IAsyncResult asyncResult) {
            object[] results = this.EndInvoke(asyncResult);
            return ((System.Double)(results[0]));
        }
    }
}
```

Código fuente 342

La Figura 223 muestra el explorador de soluciones con todos los elementos que ha generado de forma automática Visual Studio .NET.

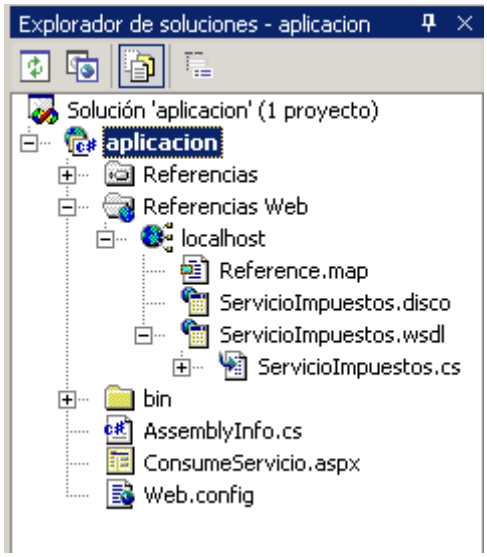


Figura 223

Y si acudimos a la vista de clases obtendremos una ventana similar a la Figura 224.

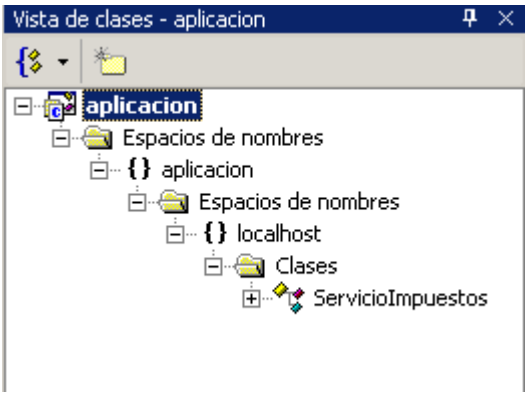


Figura 224

Ya está generada de forma sencilla la clase proxy, ahora lo que vamos a hacer es crear una instancia de dicha clase dentro de una página ASP .NET. En dicha página vamos a utilizar la clase proxy para

invocar al método CalculaIVA() del servicio Web, y le vamos a pasar por parámetro los valores del formulario Web de la página. El Código fuente 343 es el código de esta página, como se puede apreciar hemos tenido de importar el espacio de nombres en el que se encuentra la clase proxy.

```
<%@ Page language="c#"%>
<%@ Import NameSpace="aplicacion.localhost"%>
<html><head>
<script runat="server">
      void Pulsado(Object sender, EventArgs args){
            ServicioImpuestos proxy=new ServicioImpuestos();
            resultado.Text="Resultado: "+proxy.CalculaIVA(
                int.Parse(importe.Text),int.Parse(tipo.Text)).ToString();
      }
</script>
<title>Cosumidor</title>
</head>
<body>
<form id="WebForm" method="post" runat="server">
Importe<asp:TextBox ID="importe" runat="server"/><br>
Tipo<asp:TextBox ID="tipo" runat="server"/><br>
<asp:Button id="boton" runat="server" Text="Calcula IVA"
onclick="Pulsado"></asp:Button><br>
<asp:Label id="resultado" runat="server"/>
</form>
</body></html>
```

Código fuente 343

Antes de ejecutar la página ASP .NET que hace uso de la clase proxy debemos generar el proyecto para que se compile dicha clase.

En la Figura 225 podemos ver un ejemplo de ejecución de la página ASP .NET que consume el servicio Web.

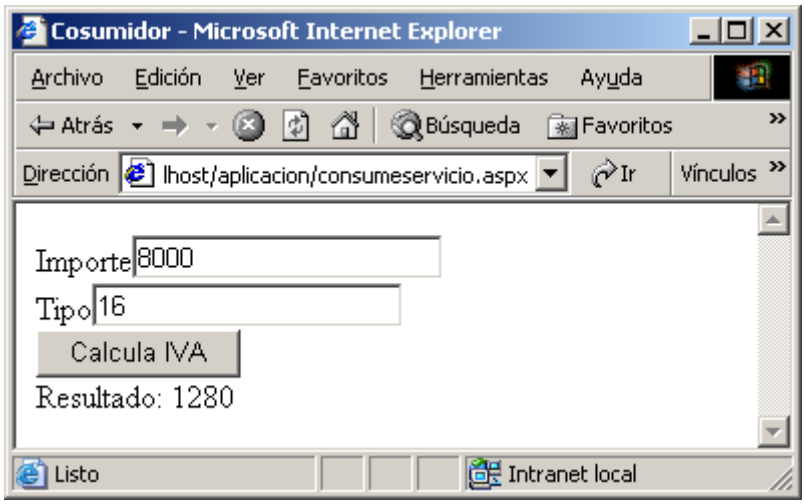


Figura 225

Tanto en lo que al código fuente se refiere como a la ejecución, la utilización de la clase proxy para comunicarnos con el servicio Web es completamente transparente.

De la misma forma que hemos consumido un servicio Web desde una aplicación ASP .NET a través de una clase proxy, también lo podemos hacer pero desde una aplicación Windows, no tenemos nada más que abrir o crear la aplicación Windows correspondiente y seleccionar la opción de Agregar referencia Web. El aspecto que mostraría el explorador de soluciones para la aplicación Windows se puede observar en la Figura 226.

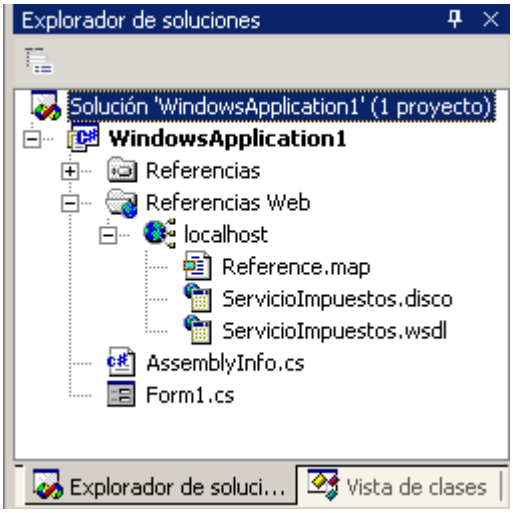


Figura 226

El Código fuente 344 es el código fuente del formulario Windows que consume el servicio Web que calcula el IVA, gran parte del código que aparece es el que genera de forma automática Visual Studio .NET, el código que hemos añadido para utilizar el servicio a través de la clase proxy es el que aparece resaltado en negrita.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using WindowsApplication1.localhost;

namespace WindowsApplication1
{
	/// <summary>
	/// Summary description for Form1.
	/// </summary>
	public class Form1 : System.Windows.Forms.Form
	{
		private System.Windows.Forms.TextBox importe;
		private System.Windows.Forms.TextBox tipo;
		private System.Windows.Forms.Button boton;
		private System.Windows.Forms.Label resultado;
		/// <summary>
		/// Required designer variable.
		/// </summary>
		private System.ComponentModel.Container components = null;

		public Form1()
		{
			//
			// Required for Windows Form Designer support
```

```
                //
                InitializeComponent();

                //
                // TODO: Add any constructor code after InitializeComponent call
                //
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
                if( disposing )
                {
                        if (components != null)
                        {
                                components.Dispose();
                        }
                }
                base.Dispose( disposing );
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
                this.importe = new System.Windows.Forms.TextBox();
                this.tipo = new System.Windows.Forms.TextBox();
                this.resultado = new System.Windows.Forms.Label();
                this.boton = new System.Windows.Forms.Button();
                this.SuspendLayout();
                //
                // importe
                //
                this.importe.Location = new System.Drawing.Point(32, 40);
                this.importe.Name = "importe";
                this.importe.Size = new System.Drawing.Size(104, 20);
                this.importe.TabIndex = 0;
                this.importe.Text = "";
                //
                // tipo
                //
                this.tipo.Location = new System.Drawing.Point(32, 72);
                this.tipo.Name = "tipo";
                this.tipo.Size = new System.Drawing.Size(104, 20);
                this.tipo.TabIndex = 1;
                this.tipo.Text = "";
                //
                // resultado
                //
                this.resultado.Location = new System.Drawing.Point(32, 144);
                this.resultado.Name = "resultado";
                this.resultado.Size = new System.Drawing.Size(144, 24);
                this.resultado.TabIndex = 3;
                //
                // boton
                //
                this.boton.Location = new System.Drawing.Point(32, 104);
                this.boton.Name = "boton";
                this.boton.Size = new System.Drawing.Size(80, 24);
                this.boton.TabIndex = 2;
                this.boton.Text = "Calcula IVA";
                this.boton.Click += new System.EventHandler(this.button1_Click);
```

```
                        //
                        // Form1
                        //
                        this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                        this.ClientSize = new System.Drawing.Size(292, 273);
                        this.Controls.AddRange(new System.Windows.Forms.Control[] {
                                    this.resultado,this.boton,this.tipo,this.importe});
                        this.Name = "Form1";
                        this.Text = "Aplicación Windows";
                        this.ResumeLayout(false);

                }
                #endregion

                /// <summary>
                /// The main entry point for the application.
                /// </summary>
                [STAThread]
                static void Main()
                {
                        Application.Run(new Form1());
                }

                private void button1_Click(object sender, System.EventArgs e)
                {
                        ServicioImpuestos proxy=new ServicioImpuestos();
                        resultado.Text="Resultado: "+proxy.CalculaIVA(
                                int.Parse(importe.Text),int.Parse(tipo.Text)).ToString();

                }

        }
}
```

Código fuente 344

Como se puede observar lo único que hemos hecho es importar el espacio de nombres en el que se encuentra la clase proxy y programar el evento de pulsación del botón para que instancie un objeto de la clase proxy para que le pase como parámetros los valores indicados en el formulario.

En la Figura 227 se puede ver en ejecución esta sencilla aplicación Windows, el formulario que aparece tiene la misma funcionalidad que el formulario Web que aparecía en la página ASP .NET que también hacía uso de este mismo servicio.

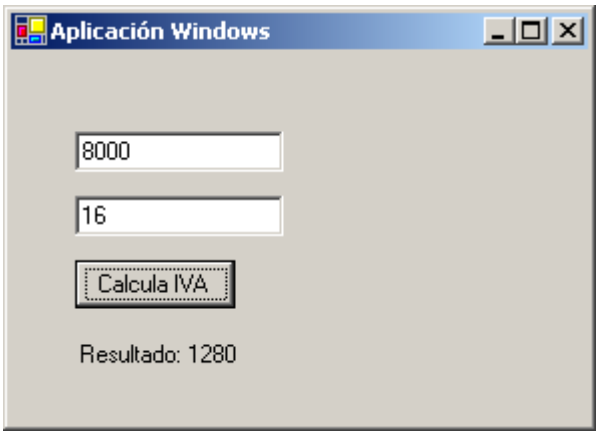


Figura 227

En el siguiente apartado vamos a ver como crear clases proxy mediante la utilidad WSDL.EXE que nos la ofrece el SDK .NET Framework.

Creación de clases proxy con la utilidad WSDL.EXE

Se utilizará a través de la línea de comandos la utilidad WSDL.EXE, ofrecida por el SDK .NET Framework, para generar clases proxy cuando no dispongamos del entorno de desarrollo de Visual Studio .NET. La utilidad WSDL.EXE se encuentra instalada en la ruta C:\Archivos de programa\Microsoft.NET\FrameworkSDK\Bin.

En primer lugar vamos a mostrar un nuevo servicio Web que deseamos consumir, en este caso desde una aplicación ASP .NET. Este nuevo servicio va a ser un servicio que ofrece un método llamado Calcular() que realiza cuatro operaciones básicas (suma, resta, multiplicación y división) sobre lo dos dígitos que le pasemos como parámetro. Este método también recibe como parámetro el tipo de operación que deseamos utilizar para realizar el cálculo. A continuación podemos ver el contenido del fichero ASMX que implementa este servicio (Código fuente 345).

```
<%@ WebService Language="C#" Class="Servicios.Ejemplos.Calculadora"%>

using System;
using System.Web.Services;

namespace Servicios.Ejemplos
{
   [WebService (Description="Servicio Web que hace cálculos sencillos",
   Namespace="http://aesteban.com/servicios", Name="ServicioCalculadoraBasica")]
   public class Calculadora : WebService{

            [WebMethod (
                 Description="Realiza una operación básica con dos dígitos")]
            public String Calcular(String Operacion, float Par1, float Par2){
                  Object resultado=null;
                  switch (Operacion)
                  {
                        case "+":
                        resultado = (Par1 + Par2);
                        break;
                        case "-":
                        resultado = (Par1 - Par2);
                        break;
                        case "*":
                        resultado = (Par1 * Par2);
                        break;
                        case "/":
                        resultado = (Par1 / Par2);
                        break;
                  }
                  if(resultado == null)
                        resultado="Operación no definida";
                  return resultado.ToString();
            }
      }
}
```

Código fuente 345

Para generar la clase proxy que nos permitirá comunicarnos con el servicio Web descrito, tenemos que pasarle como argumento a la utilidad WSDL.EXE la dirección que identifica al servicio Web, al igual que lo hacíamos desde el diálogo de Agregar referencia Web de Visual Studio .NET. Pero en este caso debemos incluir la cadena ?WSDL al final de la dirección, ya que la clase proxy se va a generar a partir del documento WSDL que describe al servicio Web.

Así para generar la clase proxy escribiríamos en la línea de comandos la siguiente instrucción (Código fuente 346).

```
wsdl http://localhost/serviciosWeb/ServicioCalculadora.asmx?WSDL
```

Código fuente 346

En la Figura 228 se puede ver una ventana de la línea de comandos desde la que hemos utilizado la herramienta WSDL.EXE para generar la clase proxy.

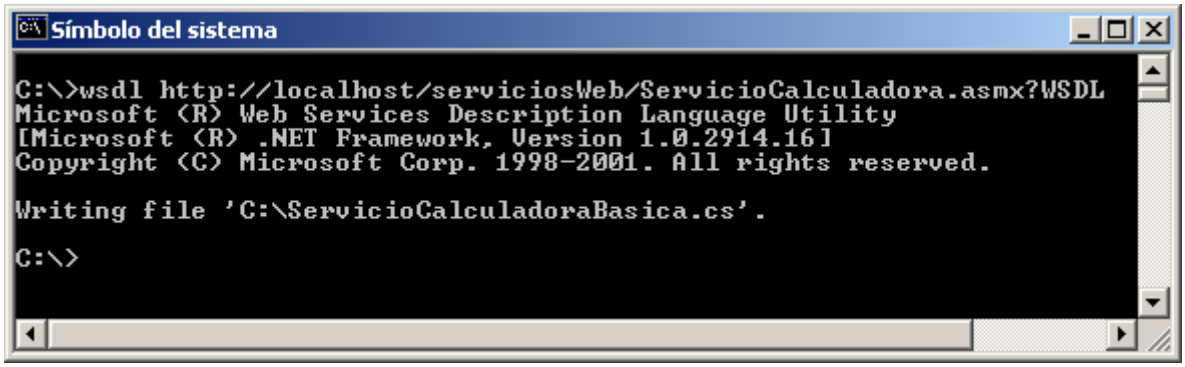


Figura 228

Como resultado la utilidad WSDL.EXE nos devuelve la clase proxy correspondiente. El nombre de esta clase, será por defecto, el nombre indicado en la propiedad Name del atributo [WebService], en este caso el fichero de clase resultante será ServicioCalculadoraBasica.cs, cuyo código fuente se puede ver a continuación (Código fuente 347). El lenguaje utilizado por defecto en la clase proxy es C#.

```
//------------------------------------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.0.2914.16
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------

//
// This source code was auto-generated by wsdl, Version=1.0.2914.16.
//
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
```

```
using System.Web.Services;


[System.Web.Services.WebServiceBindingAttribute(Name="ServicioCalculadoraBasicaSoap
", Namespace="http://aesteban.com/servicios")]
public class ServicioCalculadoraBasica :
System.Web.Services.Protocols.SoapHttpClientProtocol {

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public ServicioCalculadoraBasica() {
        this.Url = "http://localhost/serviciosWeb/ServicioCalculadora.asmx";
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://aesteban.com/ser
vicios/Calcular", RequestNamespace="http://aesteban.com/servicios",
ResponseNamespace="http://aesteban.com/servicios",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public string Calcular(string Operacion, System.Single Par1, System.Single
Par2) {
        object[] results = this.Invoke("Calcular", new object[] {
                    Operacion,
                    Par1,
                    Par2});
        return ((string)(results[0]));
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public System.IAsyncResult BeginCalcular(string Operacion, System.Single Par1,
System.Single Par2, System.AsyncCallback callback, object asyncState) {
        return this.BeginInvoke("Calcular", new object[] {
                    Operacion,
                    Par1,
                    Par2}, callback, asyncState);
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public string EndCalcular(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return ((string)(results[0]));
    }
}
```

Código fuente 347

Al utilizar la herramienta WSDL.EXE ofrecida por el SDK del .NET Framework, no se crea ningún espacio de nombres para la clase proxy, a no ser que lo especifiquemos a la hora de generar la clase proxy utilizando el parámetro /namespace: de la utilidad WSDL.EXE.

El siguiente paso, para poder hacer uso de la clase proxy en una aplicación, es compilar dicha clase mediante el compilador de C#, en nuestro caso. La utilización del compilador CSC ya la vimos en capítulos anteriores, a continuación se muestra la llamada que debemos hacer a dicho compilador (Código fuente 348).

```
csc /t:library /out:ServicioCalculadora.dll ServicioCalculadoraBasica.cs
```

Código fuente 348

Una vez que tenemos el assembly de la clase proxy ya podemos utilizarlo en la aplicación cliente del servicio Web, para ello debemos copiarlo al directorio BIN de la aplicación ASP .NET correspondiente.

El Código fuente 349 se corresponde con una página ASP .NET que hace uso de la clase proxy para comunicarse con el servicio Web de calculadora. El código es muy sencillo, ofrece un formulario Web para indicar todos los parámetros necesarios del método Calcular().

```
<%@ Page language="c#"%>
<html><head>
<script runat="server">
void Pulsado(Object sender, EventArgs args){
      ServicioCalculadoraBasica proxy=new ServicioCalculadoraBasica();
      resultado.Text="Resultado: "+
          proxy.Calcular(operacion.SelectedItem.Value,int.Parse(num1.Text),
                                                              int.Parse(num2.Text));
      }
</script>
<title>Cosumidor</title>
</head>

<body>
<form id="formulario" method="post" runat="server">
<asp:TextBox ID="num1" runat="server"/>
<asp:DropDownList ID="operacion" Runat="server">
      <asp:ListItem Text="+"  Value="+" Runat="server"/>
      <asp:ListItem Text="-" Value="-" Runat="server"/>
      <asp:ListItem Text="*" Value="*" Runat="server"/>
      <asp:ListItem Text="/" Value="/" Runat="server"/>
</asp:DropDownList>
<asp:TextBox ID="num2" runat="server"/><br>
<asp:Button id="boton" runat="server" Text="Calcula" onclick="Pulsado"/><br>
<asp:Label id="resultado" runat="server"/>
</form>
</body></html>
```

Código fuente 349

En la Figura 229 se puede ver un ejemplo de ejecución de esta página ASP .NET.

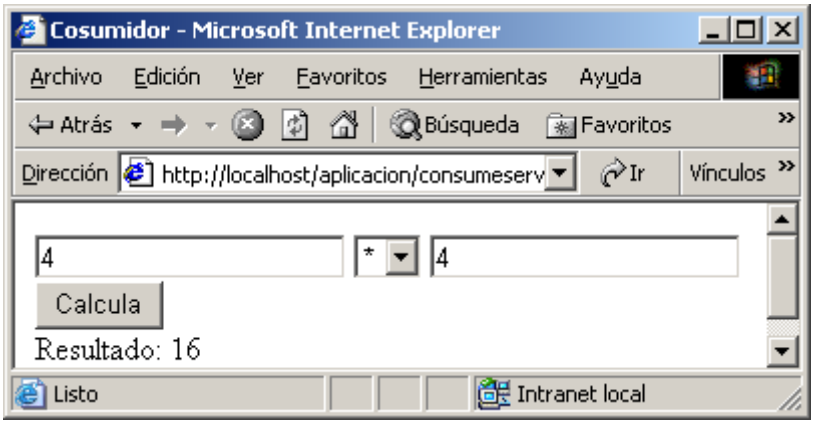


Figura 229

La herramienta WSDL.EXE la hemos utilizado de la manera más sencilla, sin utilizar ninguno de sus parámetros, a continuación pasamos a describir los más relevantes:

- <url>: este es el único parámetro obligatorio de la utilidad WSDL.EXE, y es el que hemos utilizado anteriormente para generar la clase proxy. La url puede ser a un documento WSDL (como ha sido nuestro caso), a un XDS Schema o bien a un documento DISCO.

- /language: en este parámetro (abreviatura /l:) podemos indicar el lenguaje con el que se va a generar la clase proxy, y así sobrescribir el lenguaje por defecto del servicio Web. La herramienta WSDL.EXE admite tres lenguajes C# (/language:CS), Visual Basic .NET (/language:VB) y JScript (/language:JS).

- /namespace: en este parámetro (abreviatura /n:) indicaremos el espacio de nombres en el que se va a crear la clase proxy, por defecto no se utiliza ningún espacio de nombres.

- /out: aquí indicaremos el nombre del fichero de código fuente que se generará para la clase proxy. El nombre del fichero será por defecto, el nombre indicado en el atributo Name de la directiva @WebService, seguido de la extensión del lenguaje a utilizar.

- /protocol: parámetro que nos permite indicar el protocolo que se va a utilizar para la comunicación entre la clase proxy y el servicio Web, por defecto el protocolo utilizado es SOAP, aunque también se permiten los valores HTTPGet y HTTPPost.

Con esta descripción de la herramienta WSDL.EXE finalizamos la parte del texto dedicada a los servicios Web. En el siguiente capítulo, que será el último, revisaremos el entorno de desarrollo ofrecido por Visual Studio .NET, veremos de forma general las distintas posibilidades que nos ofrece este entorno de desarrollo en .NET a la hora de desarrollar aplicaciones ASP .NET.

23

Visual Studio .NET

Introducción

El disponer en la actualidad de un entorno de desarrollo (Integrated Development Environment o IDE) para una herramienta de programación, puede parecer algo natural o incluso elemental para el propio lenguaje. Ello hace que en ocasiones no se conceda la importancia que realmente tiene a este aspecto del desarrollo.

Las cosas no han sido siempre tan fáciles en este sentido, ya que en tiempos no muy lejanos, los programadores debían de realizar de una forma digamos artesanal, todos los pasos en la creación de una aplicación.

Cuando utilizamos el término artesanal, nos referimos a que el programador debía escribir el código fuente en un editor y diseñar el interfaz con un programa generador de pantallas. Después ejecutaba otro programa que contenía el compilador, sobre el código fuente, para obtener los módulos compilados. Finalmente, debía de enlazar los módulos compilados con las librerías del lenguaje y terceras librerías de utilidades si era necesario, para obtener el ejecutable final de la aplicación. Todo se hacía a base de pasos independientes.

Tal dispersión de elementos a la hora de desarrollar resultaba incómoda en muchas ocasiones, por lo que los fabricantes de lenguajes de programación, comenzaron a incluir en sus productos, además del propio compilador, enlazador y librerías, una aplicación que permitía al programador realizar todas las fases del desarrollo: los editores de código, diseñadores visuales, compilador, etc., estaban incluidos en el mismo entorno a modo de escritorio de trabajo o taller de programación, se trataba de los primeros IDE.

El largo camino hacia la convergencia

En lo que respecta a los lenguajes de Microsoft, los programadores disponemos desde hace ya tiempo del IDE de Visual Studio.

Los diseñadores del entorno de programación de Microsoft, sobre todo desde su versión 5 han tenido un objetivo principal: hacer que el IDE de cada uno de los lenguajes sea lo más similar posible al resto, de forma que el desarrollo con varios lenguajes no suponga un cambio traumático en el entorno de trabajo.

Esto quiere decir que, por ejemplo, un programador que debe desarrollar aplicaciones tanto en Visual Basic como en Visual C++, cada vez que abra el entorno de trabajo de alguna de estas herramientas, va a encontrar, salvo las particularidades impuestas por el lenguaje, un IDE casi igual, lo que evita un entrenamiento por separado para cada lenguaje y una mayor productividad, al acceder a los aspectos comunes de igual manera.

A pesar de todas las similitudes, y hasta la versión 6, cada lenguaje seguía teniendo su propio IDE.

Visual Studio .NET, el primer paso de la total integración

Con la llegada de la tecnología .NET, el panorama ha cambiado sustancialmente, ya que al estar todos los lenguajes bajo el abrigo de un entorno de ejecución común, se ha podido desarrollar también un IDE común.

Ya no debemos elegir en primer lugar el lenguaje y abrir su IDE particular. Todo lo contrario, ahora debemos iniciar el IDE de Visual Studio .NET y después, elegir el lenguaje con el que vamos a trabajar. Esto materializa la idea de disponer de un IDE único para diversos de lenguajes. Este concepto es además extensible, ya que al ser .NET Framework una plataforma multilenguaje, los lenguajes desarrollados por terceros fabricantes también podrán engrosar la lista de los disponibles a través del IDE.

En este capítulo vamos a realizar una descripción general de la herramienta Visual Studio .NET pero desde el punto de vista del desarrollo de aplicaciones ASP .NET. En muchos casos veremos procesos que ya hemos explicado puntualmente a lo largo del texto, lo que pretendemos en este capítulo es agrupar todos ellos.

La página de inicio

Nada más iniciar VS.NET, se muestra la página de inicio del IDE (Figura 230).

Desde esta página podemos realizar una primera configuración del entorno, ya que si hacemos clic en el vínculo Mi perfil, situado en la columna izquierda, accederemos a una pantalla en la que podemos establecer una configuración adaptada al lenguaje con el que vamos a programar.

Como puede comprobar el lector, podemos configurar el perfil general para adaptar a nuestra comodidad la totalidad del IDE, o bien hacerlo sólo sobre ciertos elementos como el teclado, diseño de ventana, etc.

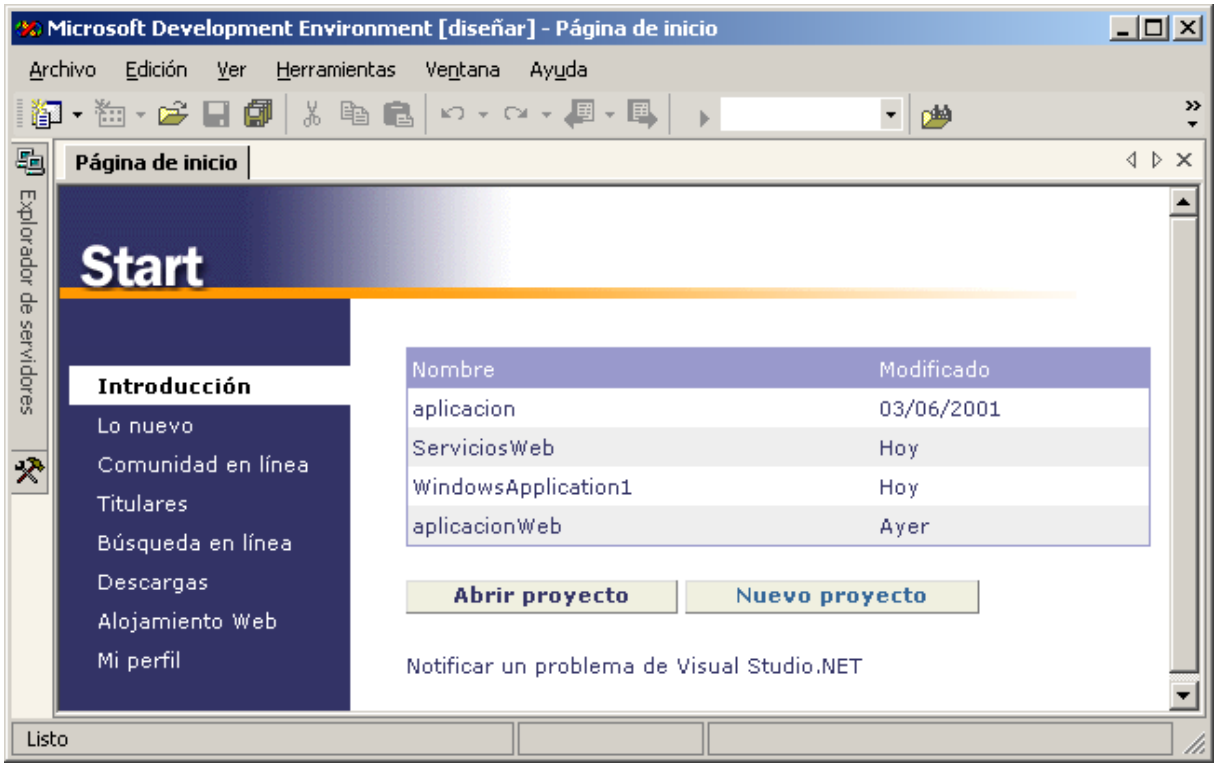


Figura 230

Establezca el lector el perfil que prefiera. Si seleccionamos el perfil Programador de Visual InterDev, que es el más próximo al desarrollador de aplicaciones ASP, aparecerá una página como la de la Figura 231.

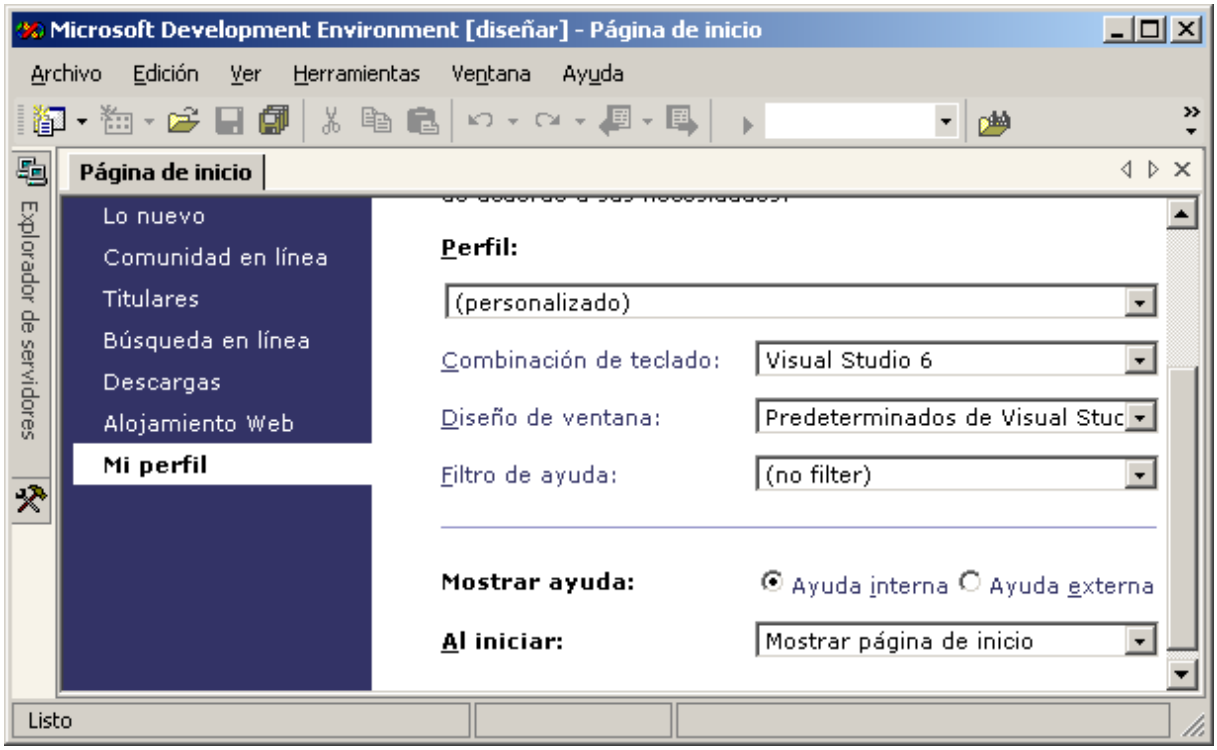


Figura 231

Si por cualquier motivo, cerramos la página de inicio, podemos volver a visualizarla utilizando alguna de las siguientes formas:

- Opción de menú Ayuda + Mostrar página de inicio o bien con Ver + Explorador Web + Inicio.
- Opción de menú Ver + Explorador Web + Inicio.

Tecleando la dirección vs:/default.htm, en el campo Dirección URL, de la barra de herramientas Web.

Creación de una aplicación ASP .NET

Cuando se inicia la ejecución de Visual Studio .NET debemos seleccionar la creación de un nuevo proyecto, esto se puede hacer con la opción de menú Archivo|Nuevo|Proyecto. Aparecerá entonces una ventana similar a la de la Figura 232. En esta ventana que debemos seleccionar el tipo de aplicación que vamos a desarrollar. En nuestro caso seleccionaremos la aplicación de tipo ASP .NET Web Application, utilizando el lenguaje C#.

También debemos indicar el nombre del directorio que va a contener la aplicación y el servidor en el que se va a desarrollar. En nuestro caso como se puede ver en la Figura 232, la aplicación se llama aplicacionASPNET y se va a crear en la URL http://localhost/aplicacionASPNET, localhost va a ser el nombre genérico con el que se hace referencia a la máquina local, es decir, a nuestro propio equipo.

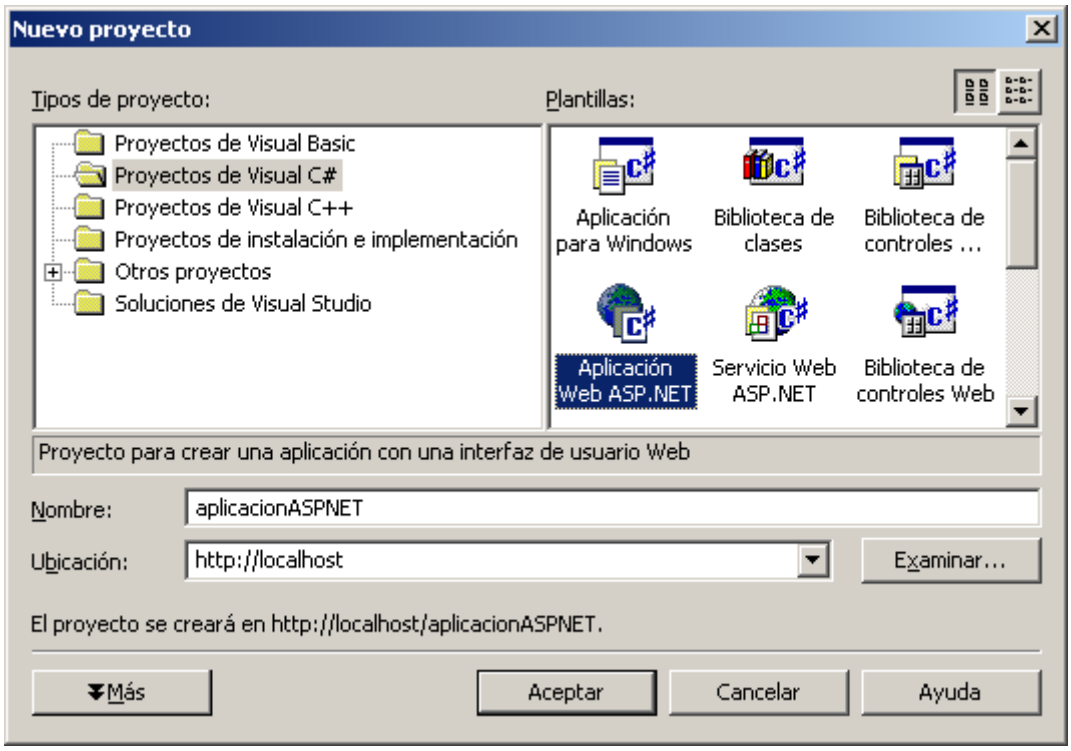


Figura 232

Cuando pulsamos el botón etiquetado como Aceptar Visual Studio .NET procede a crear la aplicación ASP .NET en el lugar y con el nombre indicados. De forma automática se crean una serie de ficheros en la aplicación ASP .NET, que se pueden observar en la Figura 37, esta figura muestra el explorador de la solución de Visual Studio .NET.

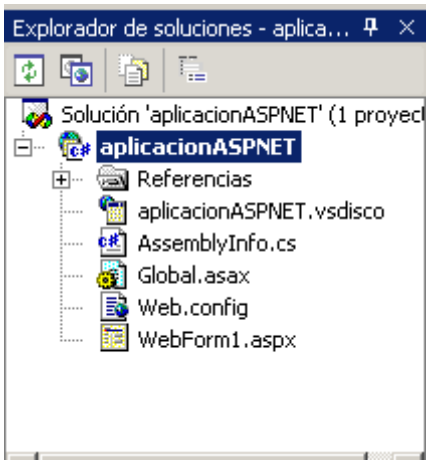


Figura 233

Los ficheros creados son:

- El fichero especial GLOBAL.ASAX, con funciones similares la fichero GLOBAL.ASA de las versiones anteriores de ASP, es decir, la de definir los eventos propios de la aplicación

- El fichero WEB.CONFIG que va a contener la configuración de la aplicación ASP .NET.

- Una página ASP .NET llamada WEBFORM1.ASPX, que va a ser una página que contiene un Web Form y que podremos utilizar en nuestra aplicación ASP .NET de la forma que deseemos.

- Respecto al fichero ASSEMBLYINFO.CS, contiene información adicional del ensamblado, fundamentalmente en forma de atributos, para el entorno de ejecución.

- El fichero .VSDISCO es un fichero de descubrimiento del protocolo DISCOVERY. Nos va a permitir determinar que servicios Web se encuentran disponibles y que métodos contiene cada servicio implementado en un servidor determinado, ya que estos ficheros .VSDISCO van a hacer referencia a los documentos XML del lenguaje WSDL de los servicios existentes en el servidor.

Esta aplicación ASP .NET (basada en el lenguaje C#) que hemos creado nos va a servir a lo largo del capítulo para ir mostrando distintos aspectos de Visual Studio .NET.

Al crear un proyecto del tipo aplicación Web ASP .NET se creará en el servidor Web correspondiente una nueva carpeta que contendrá todos los ficheros que vayamos agregando al proyecto dentro de VS .NET. Este directorio será un directorio de inicio de aplicación de Internet Information Server. En la Figura 234 se puede ver el aspecto que tendría la carpeta que se correspondería con el proyecto creado en el ejemplo anterior, en este caso aparecen diversas aplicaciones Web, tanto aplicaciones ASP como aplicaciones ASP .NET.

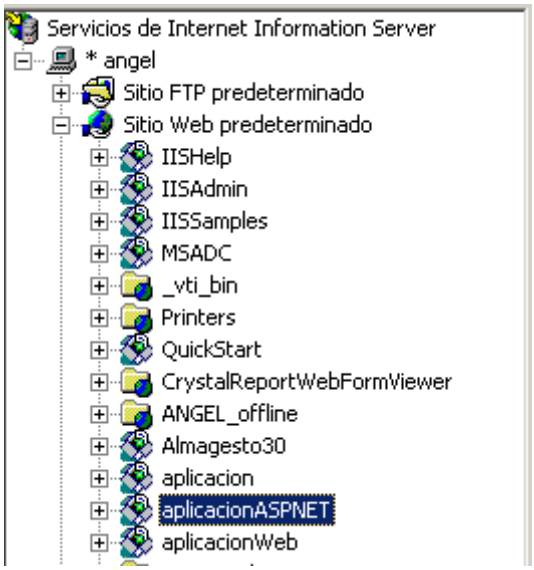


Figura 234

Una vez que se encuentre creado en el servidor Web de Microsoft, IIS 5.0, el directorio de inicio de la aplicación, tenemos la opción de crear proyectos dentro de VS .NET que apunten a dicha aplicación ASP .NET, es decir, creamos un proyecto a partir de una aplicación ASP .NET existente en el servidor Web. Para ello debemos seguir los siguientes pasos:

- Crear una solución en blanco desde la opción de menú Archivo|Nuevo|Solución en blanco (Figura 235).

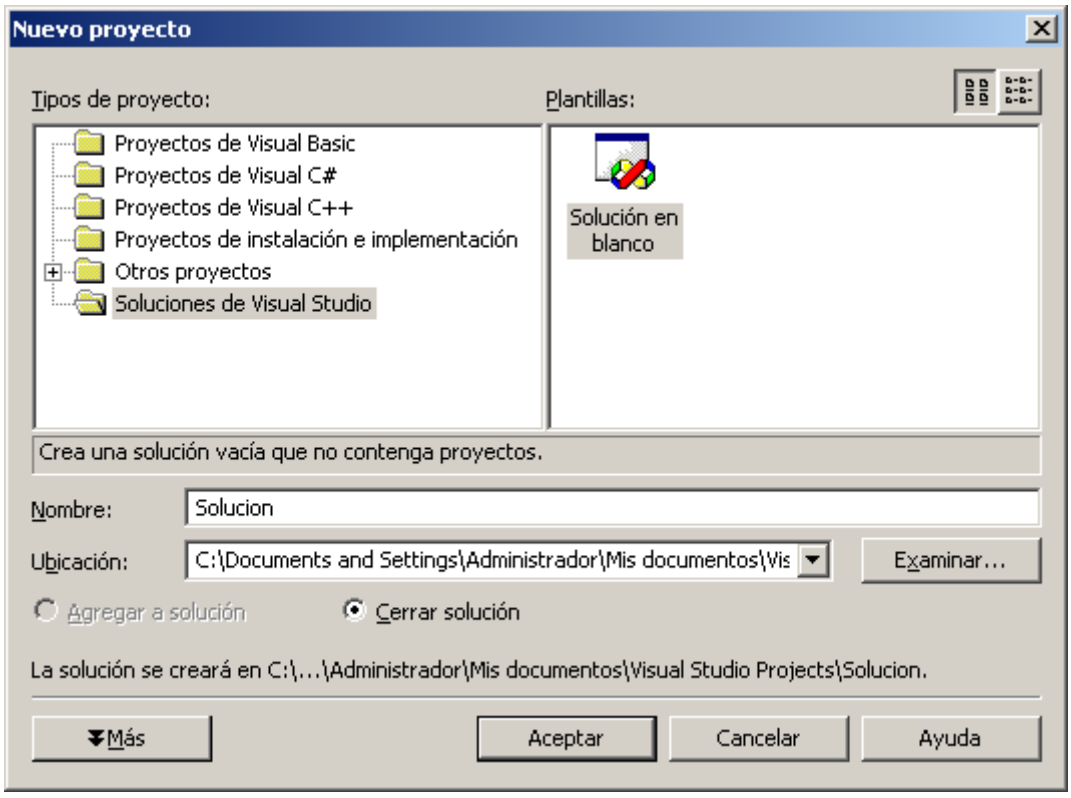


Figura 235

- A continuación seleccionamos la opción de menú Archivo|Agregar proyecto|Proyecto existente del Web. En la ventana que aparece (Figura 236) debemos indicar la ruta al directorio que contiene la aplicación ASP .NET a la que nos queremos conectar.

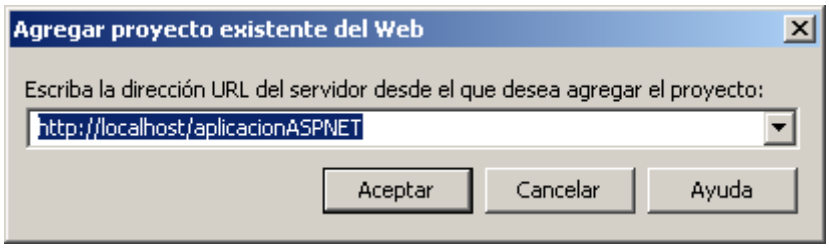


Figura 236

- Al pulsar el botón Aceptar aparecerá un nuevo diálogo (Figura 237) que nos permite seleccionar el fichero del proyecto.

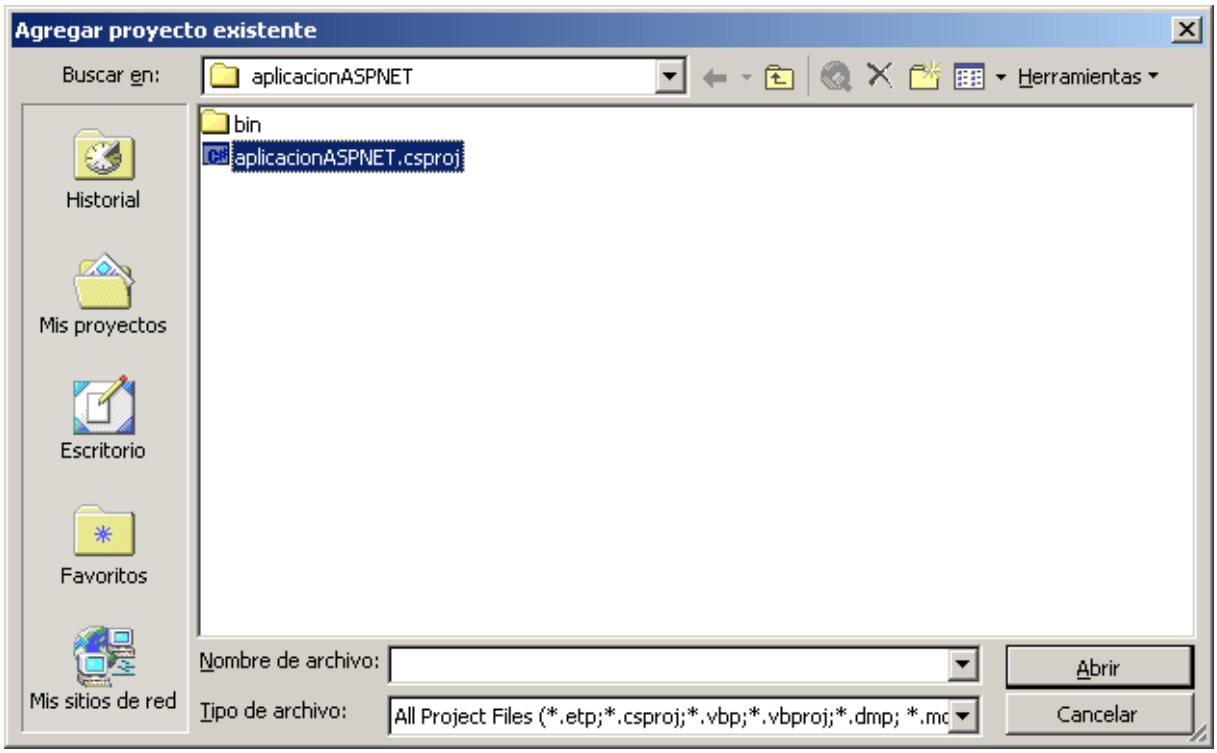


Figura 237

- Si seleccionamos el fichero con extensión .CSPROJ se agregará el proyecto a nuestra solución, por lo que ya tendremos acceso a la aplicación ASP .NET correspondiente.

Ventana principal de trabajo

De forma predominante y ocupando la mayor parte del IDE, encontramos la ventana o zona principal de trabajo. Esta ventana principal contiene todos los editores de código y diseñadores que vayamos abriendo, organizados en base a unas fichas o pestañas, que nos permiten trasladarnos de uno a otro cómodamente.

Si abrimos la página ASP .NET que se incluye de forma inicial en el proyecto (WebForm1.aspx), o añadimos una nueva página ASP .NET mediante la opción de menú, Proyecto+Agregar formulario Web, podremos ver esta ventana principal de VS .NET.

En la parte inferior de esta ventana tenemos dos pestañas, una de ellas nos permite acceder a la vista de diseño de la página ASP .NET y la otra nos permite acceder a la vista de código HTML de la página ASP .NET.

En este punto se deben realizar una serie de puntualizaciones. Podemos elegir entre dos modos de trabajo con VS .NET a la hora de desarrollar aplicaciones ASP .NET. Por un lado podemos utilizar la forma de trabajo que nos ofrece VS .NET que se encuentra basada en el mecanismo de clases Code-Behind, más adelante veremos como ofrece este mecanismo VS .NET en un apartado concreto.

Pero por otro lado podemos decidir utilizar nuestra propia forma de trabajo, en la que incluiremos código C# (código de servidor) en el propio fichero .ASPX de la página ASP .NET, y cuando lo creamos necesario utilizaremos el mecanismo de clases Code-Behind u otro que sea adecuado para separar la presentación de la lógica de la aplicación, como puede ser la utilización de componentes .NET. Recomiendo al lector que siga esta última opción a la hora de trabajar con VS .NET.

Se ha realizado este inciso debido a que según la opción que tomemos las distintas vistas dentro de la ventana principal tendrán distinto significado. Vamos a comentarlas a continuación.

En la vista de diseño, cuyo significado es común para las dos opciones de trabajo que comentábamos anteriormente, vamos a poder diseñar el formulario Web de una página ASP .NET. Nos va a permitir arrastrar y soltar controles ASP .NET desde el cuadro de herramientas hasta el área de la pantalla de diseño del formulario, de esta forma podremos crear el interfaz de usuario de nuestra página ASP .NET.

En la Figura 238 se puede ver una vista de diseño de una página ASP .NET en la que se han añadido diversos controles ASP .NET para crear un formulario Web. Más adelante volveremos a tratar esta vista de diseño, comentando también el cuadro de herramientas.

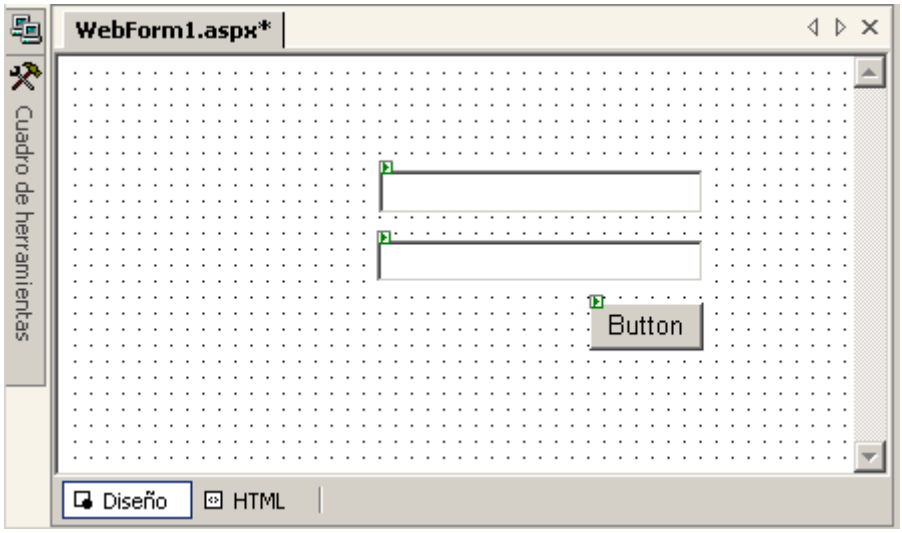


Figura 238

La vista de HTML si que tiene distintas consideraciones según la opción que tomemos de las comentadas anteriormente. Algo que resulta común a las dos opciones es que en esta vista de código HTML (que como veremos realmente no es HTML) vamos a poder observar el código que se ha ido

generando al ir añadiendo los distintos controles ASP .NET en la vista de diseño. En el Código fuente 350 se puede observar el código de la página ASP .NET que le corresponde el formulario anterior.

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs" AutoEventWireup="false"
Inherits="aplicacionASPNET.WebForm1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
    <meta name="CODE_LANGUAGE" Content="C#">
    <meta name=vs_defaultClientScript content="JavaScript (ECMAScript)">
    <meta name=vs_targetSchema
content="http://schemas.microsoft.com/intellisense/ie5">
  </HEAD>
  <body MS_POSITIONING="GridLayout">

    <form id="Form1" method="post" runat="server">
<asp:Button id=Button1 style="Z-INDEX: 101; LEFT: 265px; POSITION: absolute; TOP:
123px" runat="server" Text="Button"></asp:Button>
<asp:TextBox id=TextBox1 style="Z-INDEX: 102; LEFT: 159px; POSITION: absolute; TOP:
91px" runat="server" Width="163px" Height="21px"></asp:TextBox>
<asp:TextBox id=TextBox2 style="Z-INDEX: 103; LEFT: 160px; POSITION: absolute; TOP:
56px" runat="server" Width="162px" Height="22px"></asp:TextBox>

    </form>

  </body>
</HTML>
```

Código fuente 350

En el caso de seguir el mecanismo de clases Code-Behind propuesto por VS .NET en la vista HTML únicamente veremos el código necesario para crear el interfaz de usuario, es decir, la presentación de la aplicación, ya que la lógica de la aplicación se encontrará en una clase Code-Behind asociada a la página ASP .NET.

Por defecto VS .NET crea un fichero de código fuente de la clase Code-Behind, cuyo nombre será el nombre completo de la página ASP .NET con extensión ASPX incluida y añadiendo la extensión del lenguaje que estemos utilizando, en este caso CS por el lenguaje C#. Así la página WebForm1.aspx tiene asociada el fichero de clase WebForm1.aspx.cs. Este fichero contendrá toda la lógica de la página ASP .NET.

Para ver el código fuente de la clase Code-Behind, podemos pulsar con el botón derecho del ratón sobre cualquier lugar de la vista de diseño de la página ASP .NET y seleccionar la opción de ver código. El Código fuente 351 es el código de la clase Code-Behind asociada la página ASP .NET del ejemplo.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
```

```
namespace aplicacionASPNET
{
	/// <summary>
	/// Summary description for WebForm1.
	/// </summary>
	public class WebForm1 : System.Web.UI.Page
	{
		protected System.Web.UI.WebControls.TextBox TextBox1;
		protected System.Web.UI.WebControls.TextBox TextBox2;
		protected System.Web.UI.WebControls.Button Button1;

		public WebForm1()
		{
			Page.Init += new System.EventHandler(Page_Init);
		}

		private void Page_Load(object sender, System.EventArgs e)
		{
			// Put user code to initialize the page here
		}

		private void Page_Init(object sender, EventArgs e)
		{
		//
		// CODEGEN: This call is required by the ASP.NET Web Form Designer.
		//
			InitializeComponent();
		}

		#region Web Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
			this.Load += new System.EventHandler(this.Page_Load);

		}
		#endregion
	}
}
```

Código fuente 351

Al seleccionar Ver código (no Ver código fuente HTML) aparecerá en la zona superior de la ventana principal de trabajo otra pestaña que nos permitirá ir pasando del diseño del formulario al código fuente de la clase Code-Behind. Tal como se puede ver en la Figura 239.

Si tomamos la decisión de no utilizar la clase Code-Behind que nos ofrece por defecto VS .NET, en la vista de código HTML también podremos observar el código C# que se añada para la lógica de la página ASP .NET.

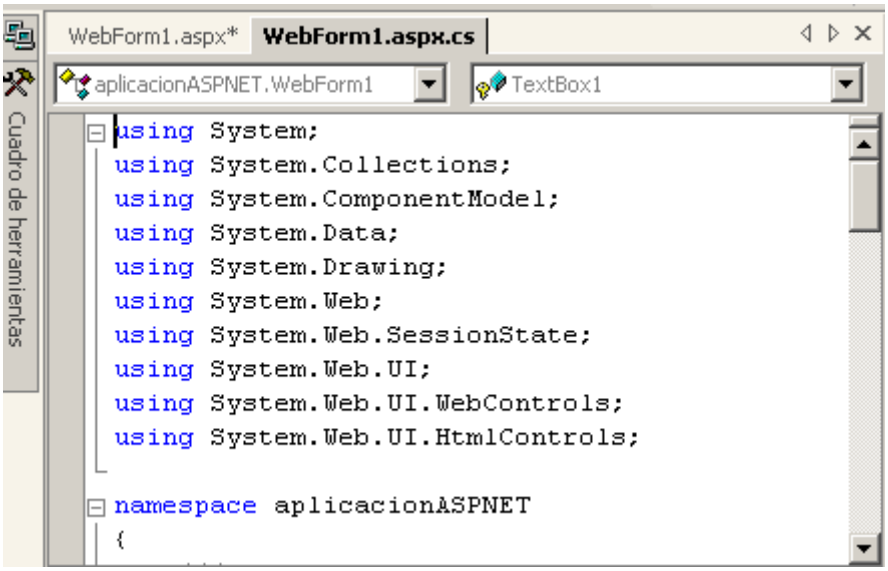


Figura 239

En resumen, para una página ASP .NET tenemos distintas vistas de la misma:

- Vista de diseño: nos permite diseñar el formulario Web existente en la página.
- Vista de código HTML: podremos ver el código que se genera al ir añadiendo controles ASP .NET a la página, así como el código C# de servidor que añadamos nosotros.
- Vista de código: tenemos acceso al código fuente de la clase Code-Behind asociada a la página ASP .NET.

El Explorador de soluciones

Al desarrollar una aplicación ASP .NET, los elementos que contiene: páginas ASP .NET, clases, referencias, servicios Web, etc., se organizan dentro de un proyecto.

También es posible tener varios proyectos abiertos simultáneamente en la misma sesión de trabajo del IDE. Dichos proyectos se organizan dentro de lo que en VS.NET se denomina una solución.

Una solución puede contener proyectos desarrollados en los diferentes lenguajes de la plataforma .NET, y el medio más cómodo para manejarlos es a través de la ventana Explorador de soluciones. La Figura 240 muestra el aspecto típico de esta ventana con una solución que contiene un proyecto, en el que a su vez hay contenido una página ASP .NET.

La carpeta Referencias contiene las referencias que están establecidas dentro del proyecto, hacia los diferentes espacios de nombre que pueden ser necesarios a la hora de escribir el código. Al expandir esta carpeta tendremos referenciados entre otros: System, System.Web, System.Data, System.Web.Services, etc.

Al crear un nuevo proyecto desde VS .NET, dichas referencias son establecidas automáticamente por el IDE, facilitando el trabajo del programador que no necesita preocuparse por los espacios de nombres esenciales necesita para su aplicación.

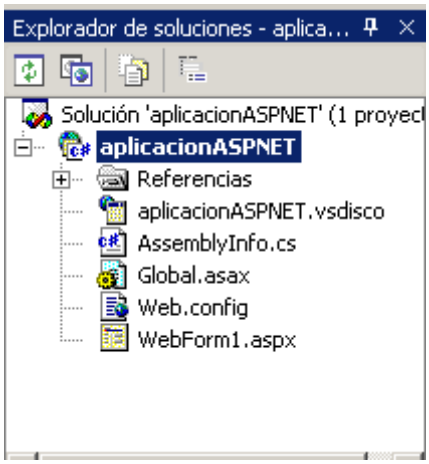


Figura 240

Las referencias establecidas por el IDE varían en función del estilo de proyecto elegido: aplicación de Windows, aplicación Web de ASP .NET, biblioteca de clases, Servicio Web de ASP .NET, etc. . El programador puede, naturalmente, establecer referencias adicionales en función de las necesidades del programa.

En la Figura 241 se ofrece un explorador de soluciones de una solución más compleja que contiene cinco proyectos distintos: una aplicación Web de ASP .NET, dos bibliotecas o librerías de clases en C#, un proyecto de servicios Web de ASP .NET y una aplicación Windows.

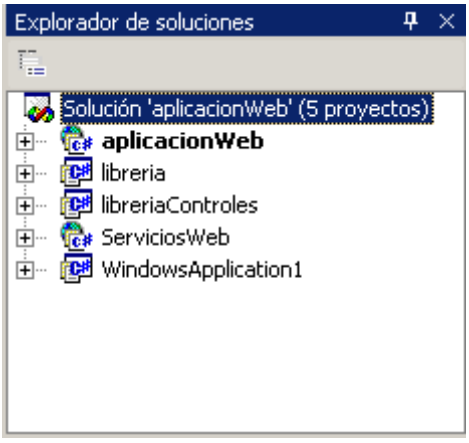


Figura 241

Agregar nuevos elementos a un proyecto

Una vez creado un nuevo proyecto, necesitaremos con toda probabilidad añadir páginas ASP .NET, clases, etc., adicionales. Para ello, seleccionaremos alguna de las opciones del menú Proyecto, que comienzan por Agregar <NombreOpción>, y que nos permiten agregar un nuevo elemento al proyecto (Figura 242).

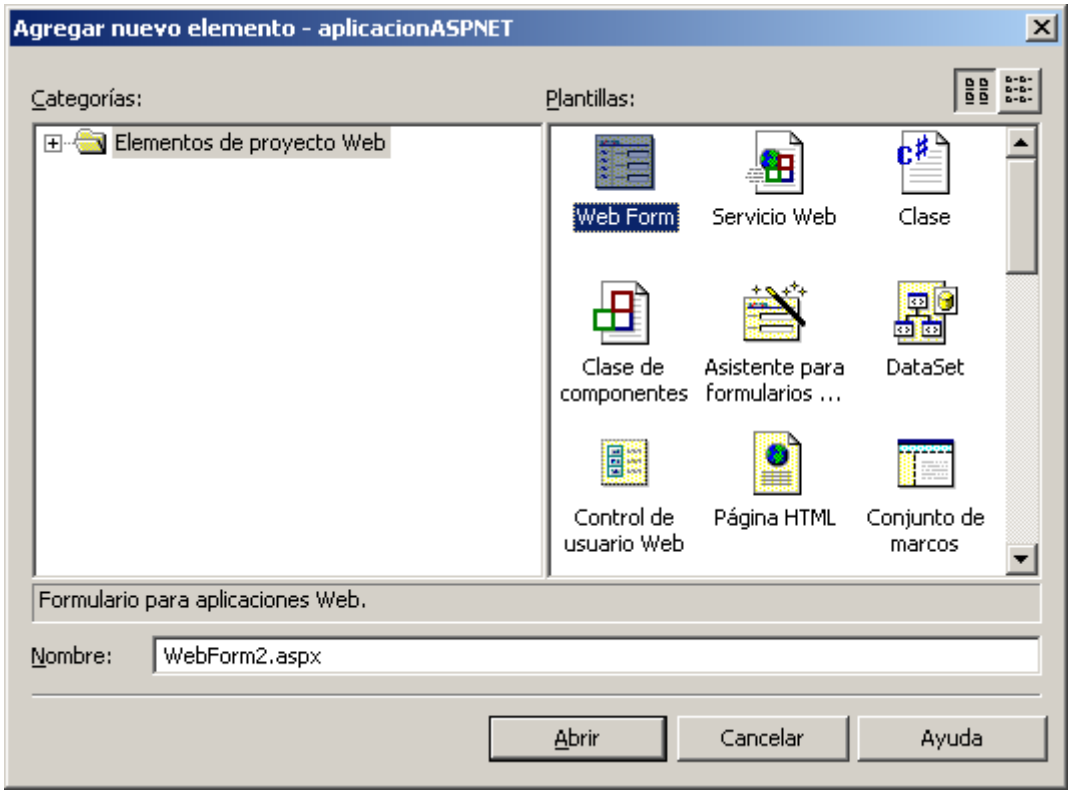


Figura 242

Estas opciones hacen uso común de la caja de diálogo Agregar nuevo elemento. Lo que sucede, es que si elegimos añadir por ejemplo un formulario, la caja de diálogo se abre posicionándonos ya en la plantilla correspondiente al elemento que queremos añadir, ello supone un atajo y nos ahorra el paso de selección del elemento. Pero podemos cambiar libremente dicho elemento, seleccionando uno diferente en el panel derecho de este diálogo. La Figura 243 muestra un proyecto en el que además de la página ASP .NET por defecto, se ha agregado al proyecto una clase en C#, un control de usuario y un servicio Web.

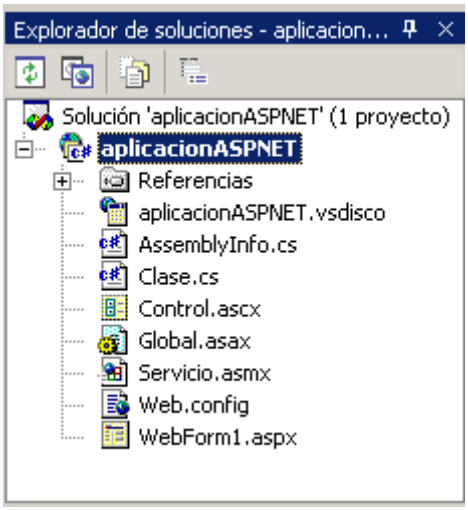


Figura 243

El menú contextual

Tanto si nos encontramos en la ventana del explorador de soluciones como en cualquier otra, podemos acceder de un modo rápido a múltiples opciones de los elementos situados en la ventana, haciendo clic derecho sobre un elemento, de modo que se abrirá el menú contextual correspondiente, en el que podremos elegir operaciones relacionadas con el elemento seleccionado. La Figura 244 muestra el menú contextual de un proyecto de una aplicación ASP .NET.

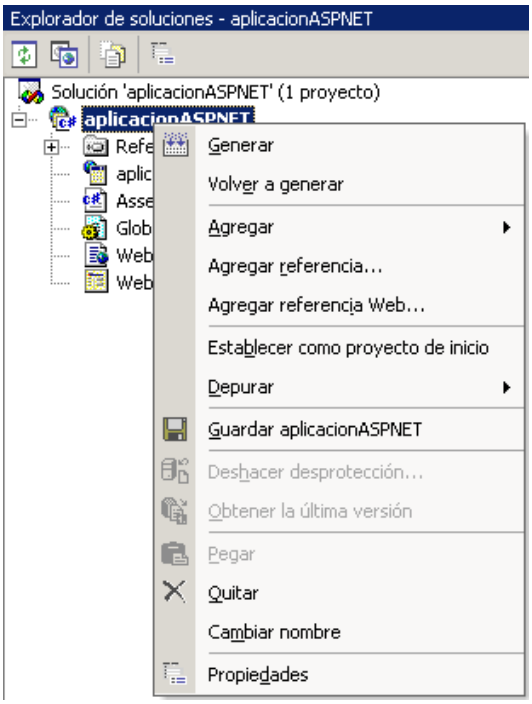


Figura 244

Propiedades del proyecto

Si necesitamos variar los parámetros de configuración del proyecto, deberemos abrir la ventana de propiedades del proyecto, haciendo clic sobre el mismo y seleccionando la opción de menú Proyecto + Propiedades (Figura 245).

Esta ventana de propiedades está organizada en dos carpetas: Propiedades comunes y Propiedades de configuración, situadas en la parte izquierda, y que contienen cada una, los diversos apartados en los que se organizan las propiedades. Al hacer clic en cada apartado, la parte derecha cambiará mostrando las propiedades relacionadas. De esta forma podemos configurar aspectos tales como el tipo de aplicación resultante, el punto de entrada, nombre de ensamblado, espacios de nombres importados, generación de nombres para ensamblados compartidos, ruta de generación de ficheros del proyecto, etc.

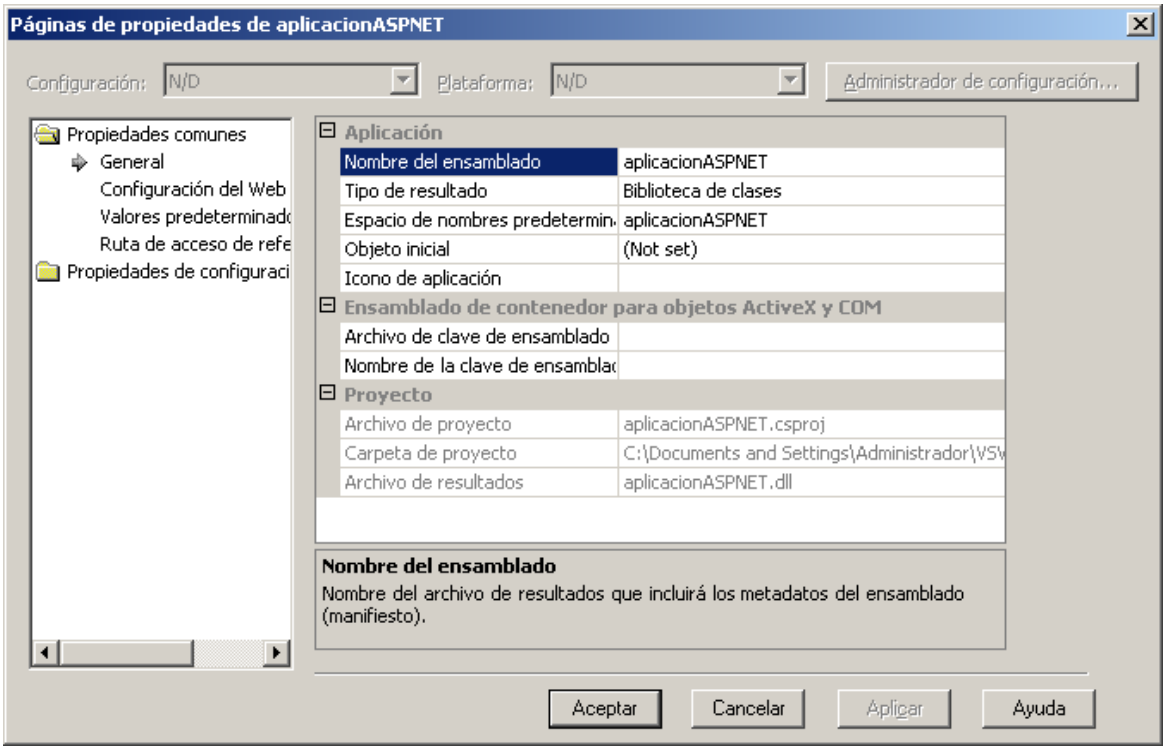


Figura 245

Mostrar la pantalla completa

Dada la gran cantidad de elementos existentes en el entorno de desarrollo, y puesto que en ocasiones no es necesario disponer de todos ellos simultáneamente, la opción de menú Ver + Pantalla completa, amplia el espacio ocupado por el elemento con el que estemos trabajando en ese momento (por ejemplo, el editor de código), ocultando otros componentes del IDE, con lo cual, ganamos espacio de trabajo.

El diseñador de formularios Web

Hemos querido destacar esta característica de VS .NET asignándole un apartado dedicado íntegramente a la misma, debido a la gran novedad que supone para los programadores de Visual InterDev el tener integrados el diseño de formularios Web para páginas ASP .NET y el código de servidor que presenta dicha página.

Para añadir los distintos tipos de controles disponibles debemos arrastrarlos desde el cuadro de herramientas. En el cuadro de herramientas los distintos tipos de controles se encuentran agrupados, tal como se puede ver en la Figura 246.

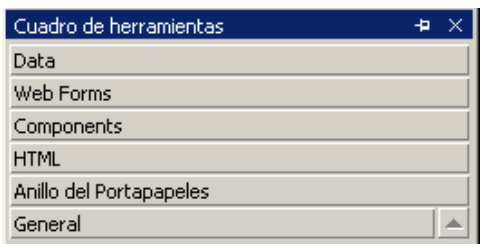


Figura 246

Los controles dentro de una ficha se visualizan por defecto con el icono del control y su nombre, en el llamado modo Vista de lista. No obstante, podemos hacer que los controles se muestren en vista de iconos, al estilo de VB6, haciendo clic derecho sobre el cuadro de herramientas y seleccionando la opción de menú Vista de lista, que se encontrará marcada, desmarcándola de esa forma, y quedando esta ventana con un aspecto parecido al de la Figura 247.

Figura 247

Aunque esta vista pueda ser inicialmente de mayor agrado para los programadores de VB6, creemos que la vista de lista, al incluir el nombre del control es más práctica de manejar, por lo que vamos a dejar de nuevo esta ventana con dicha vista.

Podemos ordenar los controles de una ficha por su nombre, haciendo clic derecho sobre esta ventana y eligiendo la opción de menú Ordenar elementos alfabéticamente.

El orden alfabético puede tener el inconveniente de situar algunos controles de uso frecuente (por ejemplo TextBox), al final de la lista. Para remediar este problema, podemos cambiar la posición de los controles dentro de la lista de dos maneras: haciendo clic sobre el control y arrastrándolo hasta la posición que deseemos; o bien, haciendo clic derecho sobre la lista de controles y eligiendo las opciones Subir o Bajar. En este último modo podemos lograr una ordenación mixta, en parte alfabética, pero con nuestro controles de mayor uso en posiciones preferentes. La Figura 248 muestra un ejemplo de este orden combinado.

Los distintos grupos de elementos que aparecen por defecto en el cuadro de herramientas son los siguientes

- Data: presenta una serie de objetos de ADO .NET para el acceso a datos desde nuestras páginas ASP .NET.

- Web Forms: estos serán los controles que más utilizaremos, ya que se corresponden con los controles Web de ASP .NET, los controles de validación, los de lista y los ricos o controles avanzados.

- Anillo del Portapapeles: permite el intercambio de código fuente entre diversos elementos del proyecto, a través de un acceso visual al Portapapeles del sistema.

- HTML: aparecen los distintos controles del lenguaje HTML que podemos añadir a la página ASP .NET, no se deben confundir con los controles HTML de ASP .NET.

- Components: distintos componentes avanzados.

- General: controles generales, en nuestro caso esta ficha se encuentra vacía.

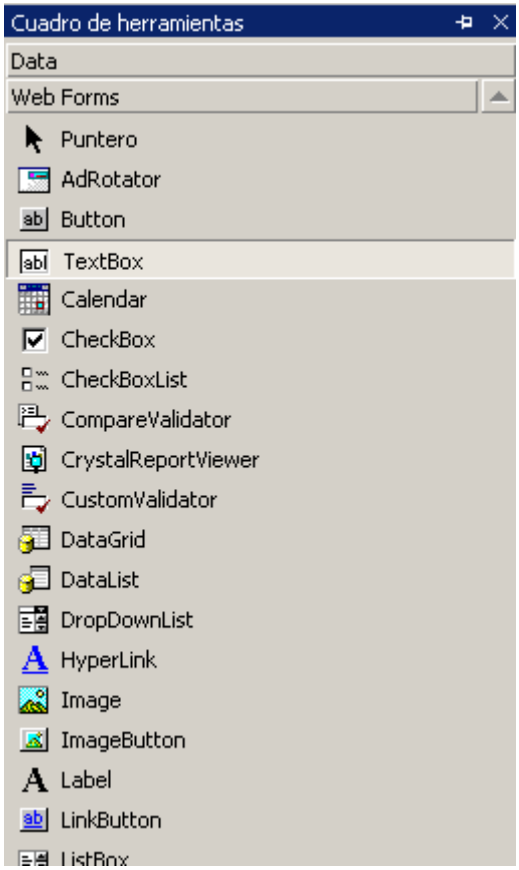


Figura 248

Las barras de herramientas

El IDE de VS.NET dispone de un gran número de barras de herramientas, de las cuales sólo se muestran por defecto la estándar. El programador puede posteriormente, visualizar barras adicionales y crear sus propias barras de herramientas personalizadas.

Para mostrar alguna de las barras de herramientas definidas en el IDE, podemos hacerlo de las siguientes formas:

- Clic derecho sobre una de las barras actualmente visible, lo que mostrará un menú contextual con todas las barras existentes, en el que podremos elegir una (Figura 249).

- Opción de menú Herramientas + Personalizar, que mostrará la ventana Personalizar, en la que dentro de la ficha Barras de herramientas, podremos visualizar y ocultar barras, marcando y desmarcando respectivamente la casilla asociada a dicha barra (Figura 250).

Marcando por ejemplo, la barra Editor de texto, se visualizará esta barra, situándose debajo de la estándar (Figura 251).

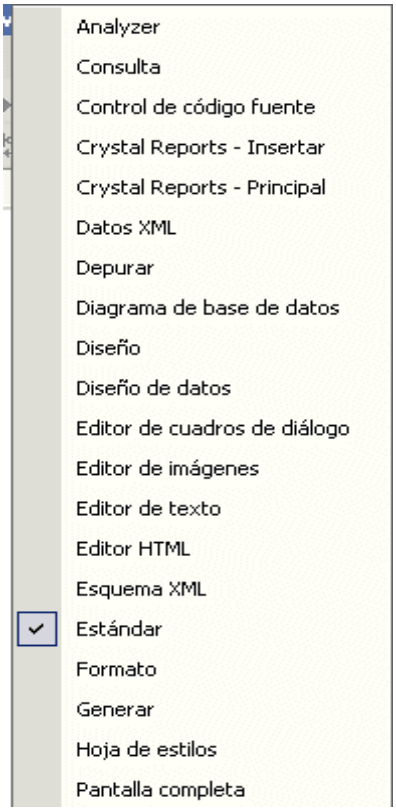


Figura 249

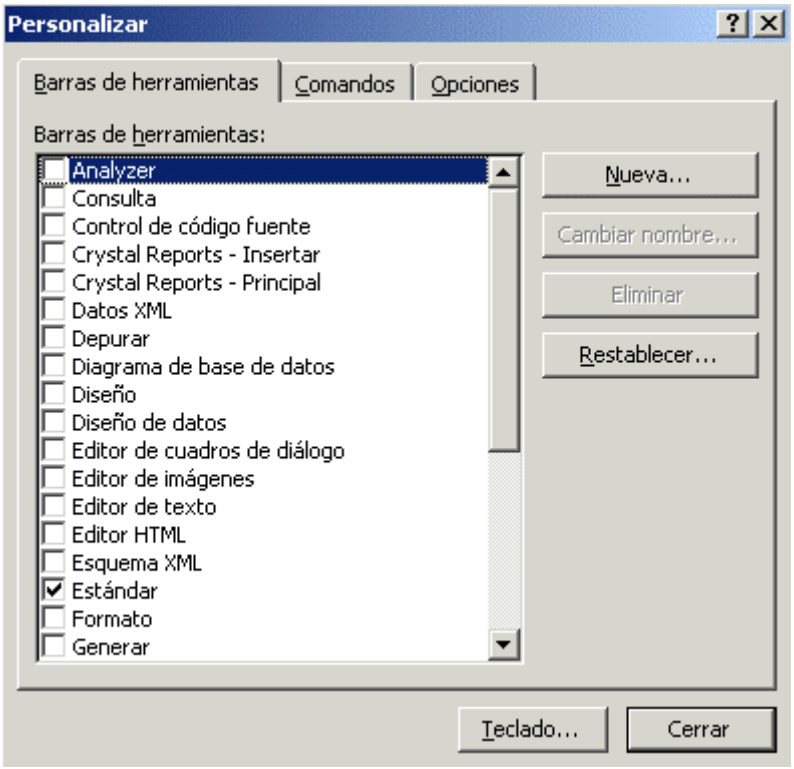


Figura 250

Figura 251

El Examinador de objetos

La ventana del examinador de objetos la abrimos con la opción de menú Ver + Otras ventanas + Examinador de objetos. Una vez abierta, se sitúa como una ficha más de la ventana principal del IDE, organizada en tres paneles principales.

El panel izquierdo muestra la organización de espacios de nombres, clases, etc. El panel derecho visualiza los miembros de la clase actualmente seleccionada. Finalmente, el panel inferior muestra la declaración del miembro seleccionado en el panel derecho (Figura 252).

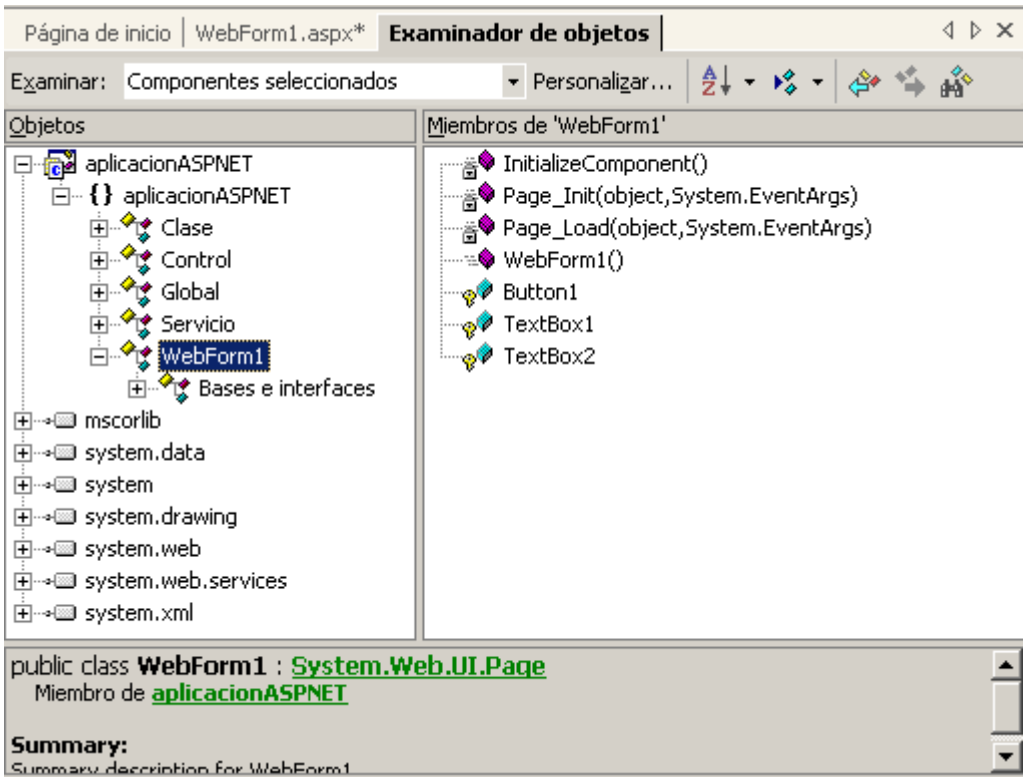


Figura 252

La diferencia respecto a la vista de clases, reside en que con el examinador de objetos podemos buscar información sobre cualquiera de las clases que componen la plataforma .NET Framework, pulsando sobre el último botón de la barra de herramientas: Buscar símbolo (Figura 253).

Figura 253

Para buscar un símbolo dentro de las clases, se muestra una caja de diálogo en la que introducimos el nombre del símbolo. Al pulsar el botón Buscar, se abre la ventana Resultados, situada habitualmente en la parte inferior del IDE, con la lista de símbolos coincidentes encontrados. Al hacer doble clic sobre alguno de los símbolos encontrados, se actualiza la información del examinador de objetos, mostrando la clase y símbolo seleccionado. Como ejemplo, en la Figura 254, hemos buscado por el símbolo Button.

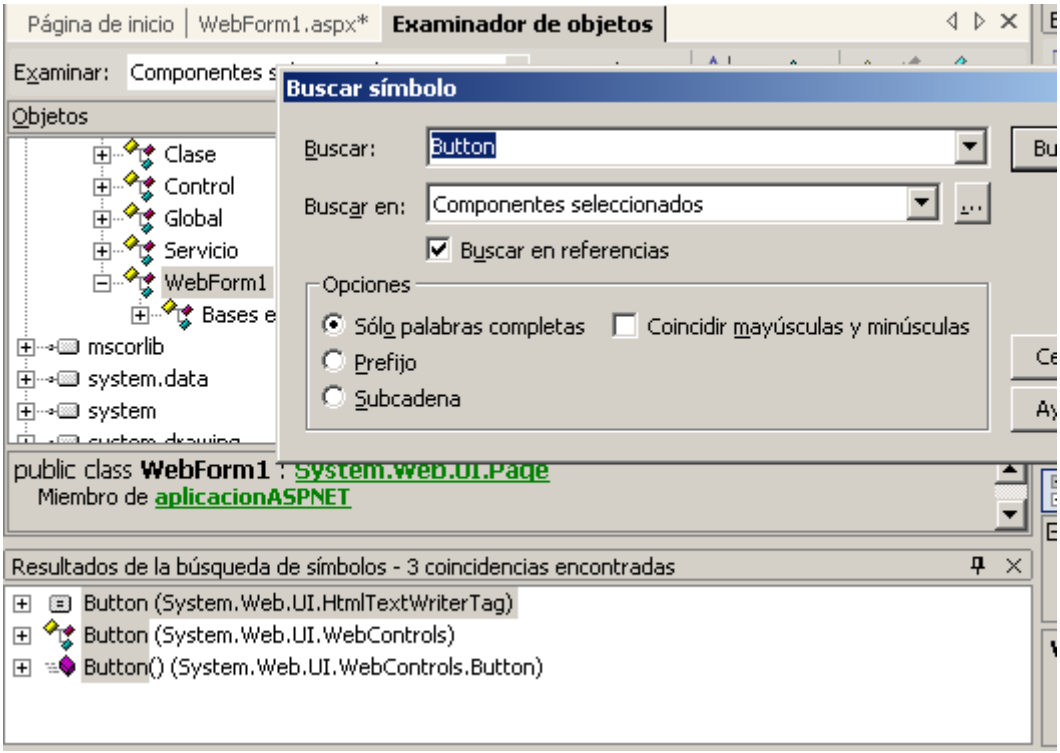


Figura 254

Clases Code-Behind y Visual Studio .NET

En el capítulo correspondiente vimos la utilización de clases Code-Behind, en este apartado vamos a comentar como se utiliza desde VS .NET este mecanismo de programación de páginas ASP .NET.

El propio entorno de desarrollo Visual Studio .NET hace uso de forma automática de clases Code-Behind para los Web Forms que se crean dentro de un proyecto determinado. Para comprobarlo de primera mano vamos a realzar el siguiente proceso dentro de un proyecto de Visual Studio .NET:

- Se añade un nuevo Web Form al proyecto, pulsado con el botón derecho del ratón sobre el nombre del proyecto seleccionamos la opción Agregar|Agregar formulario Web.

- Añadimos desde el cuadro de herramientas un control Web Label y un control Button en la vista de diseño.

- Se hace doble clic sobre el control Button, de forma automática VS .NET nos muestra el fichero .CS que contiene la clase Code-Behind asociada a la página ASP .NET actual. En este momento podemos escribir una línea de código dentro del método en el que nos encontramos posicionados (este método debería ser Button1_Clic), en esta línea de código lo único que hacemos es asignar a la propiedad Text del objeto Label un valor.

- Para poder probar la página ASP .NET con su clase Code-Behind asociada debemos generar el proyecto, para ello acudimos a la opción de menú Generar. En Visual Studio .NET las clases Code-Behind se compilan dentro de un assembly que se localizará en el directorio bin de la aplicación ASP .NET actual.

Por lo tanto si estamos utilizando VS .NET de la forma antes indicada, para utilizar las clases Code-Behind debemos generar el proyecto. Esto es así debido a que el código que genera de forma automática VS .NET en la directiva Page de la página no incluye el atributo Src, para indicar el lugar en el que se encuentra el fichero de la clase Code-Behind, por lo tanto se buscará en el directorio bin compilado ya como un assembly.

En el Código fuente 352 se ofrece el código que genera de forma automática VS .NET para la clase Code-Behind. En el código aparece destacado en negrita lo más interesante del mismo.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace aplicacionWeb
{
	/// <summary>
	/// Summary description for WebForm3.
	/// </summary>
	public class WebForm3 : System.Web.UI.Page
	{
		protected System.Web.UI.WebControls.Label Label1;
		protected System.Web.UI.WebControls.Button Button1;

		public WebForm3()
		{
			Page.Init += new System.EventHandler(Page_Init);
		}

		private void Page_Load(object sender, System.EventArgs e)
		{
			// Put user code to initialize the page here
		}

		private void Page_Init(object sender, EventArgs e)
		{
		//
		// CODEGEN: This call is required by the ASP.NET Web Form Designer.
		//
			InitializeComponent();
		}

		#region Web Form Designer generated code
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
			this.Button1.Click +=
				new System.EventHandler(this.Button1_Click);
```

```
                    this.Load += new System.EventHandler(this.Page_Load);

            }
            #endregion

            private void Button1_Click(object sender, System.EventArgs e)
            {
                  Label1.Text="Prueba";
            }
      }
}
```

Código fuente 352

Y en el Código fuente 353 se ofrece el código que se genera para la página ASP .NET, al igual que en el código anterior se destaca lo más relevante.

```
<%@ Page language="c#" Codebehind="WebForm3.aspx.cs" AutoEventWireup="false"
Inherits="aplicacionWeb.WebForm3" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
<meta content="Microsoft Visual Studio 7.0" name=GENERATOR>
<meta content=C# name=CODE_LANGUAGE>
<meta content="JavaScript (ECMAScript)" name=vs_defaultClientScript>
<meta content=http://schemas.microsoft.com/intellisense/ie5 name=vs_targetSchema>
  </HEAD>
<body MS_POSITIONING="GridLayout">
<form id=WebForm3 method=post runat="server">
<asp:Label id=Label1 style="Z-INDEX: 101; LEFT: 363px; POSITION: absolute; TOP:
95px" runat="server">Label</asp:Label>
<asp:Button id=Button1 style="Z-INDEX: 102; LEFT: 561px; POSITION: absolute; TOP:
55px" runat="server" Text="Button"></asp:Button></FORM>

  </body>
</HTML>
```

Código fuente 353

Utilizando componentes en VS .NET

Si utilizamos el entorno de desarrollo Visual Studio .NET, y nuestra clase del componente se encuentra en el mismo proyecto que la aplicación ASP .NET actual, podemos seleccionar la opción de generar el proyecto, de esta forma se generará el componente también dentro de un assembly para todo el proyecto. Este assembly contendrá todas las clases de la aplicación ASP .NET que representa al proyecto actual, organizadas según los espacios de nombre correspondientes. El assembly se localizará en el directorio BIN y presentará el nombre del proyecto.

Para tener una visión más clara del assembly del proyecto de Visual Studio .NET, y de las clases y espacios de nombre que contiene, podemos seleccionar la opción de mostrar la utilidad Vista de clases, para ello seleccionamos la opción Ver|Vista de clases, y aparecerá un árbol que muestra la jerarquía de espacios de nombres y clases contenidas en los mismos. En la Figura 255 se puede ver la estructura de espacios de nombres que presenta al clase del componente que acabamos de crear.

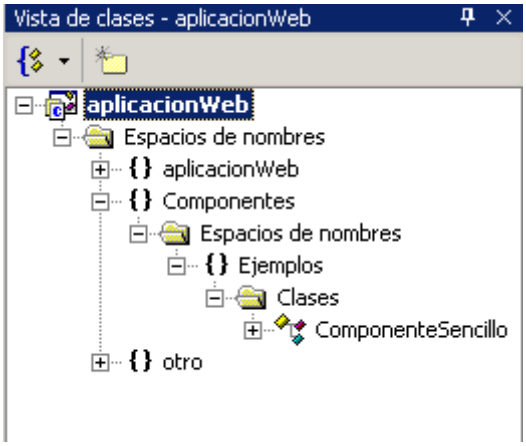


Figura 255

Desde Visual Studio .NET también tenemos la posibilidad de añadir a la solución actual un nuevo proyecto, en este caso sería un proyecto de biblioteca de clases que va a contener la clase de nuestro componente. De esta forma podemos generar de forma independiente el assembly que va a contener nuestro componente .NET. El fichero DLL generado en este caso es el assembly que deberemos copiar al directorio BIN de la aplicación ASP .NET en la que deseamos hacer uso del mismo.

En la Figura 256se puede observar el aspecto que mostraría en este segundo escenario la Vista de clases.

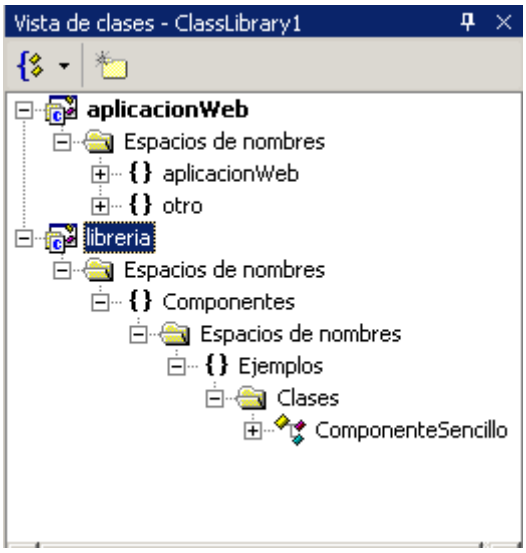


Figura 256

Las páginas ASP .NET utilizarán el componente de la misma forma, sin tener en cuenta si el componente .NET se encuentra en el assembly general del proyecto que representa a la aplicación ASP .NET actual o si se encuentra en un assembly separado, en cualquier caso el assembly resultante debe encontrase en el directorio BIN de la aplicación ASP .NET .

Copiar el assembly que contiene a nuestro componente .NET al directorio BIN de la aplicación ASP .NET es el mecanismo común para registrar un componente en una aplicación ASP .NET, esto es una de las mejoras que presentan los componentes .NET respecto a los componentes COM+, además si

queremos modificar el fichero DLL y reemplazarlo por otro no tenemos nada más que sobrescribir, sin necesidad de cerrar ningún tipo de proceso o descargar el sitio Web de memoria.

Depurando con Visual Studio .NET

En el entorno de desarrollo Visual Studio .NET, la depuración es una parte del propio entorno, es decir, se encuentra integrada con le herramienta de desarrollo.

Para depurar desde Visual Studio .NET únicamente debemos establecer como página de inicio la página ASP .NET que deseamos depurar. Esta página debe asignar el valor True al atributo Debug de la directiva @Page, esto es algo común para cualquier depurador.

En la vista HTML del editor de Visual Studio .NET estableceremos los puntos de ruptura necesarios, una vez hecho esto acudimos a la opción de menú Depurar|Iniciar, y de forma automática se cargará la página ASP .NET indicada como página de inicio y se iniciará la depuración de la misma.

El aspecto y forma de trabajar que ofrece el depurador de Visual Studio .NET es muy similar a la ofrecida por el depurador CLR, pero además permite la depuración remota, es decir, permite depurar páginas ASP .NET que se encuentran en servidores Web remotos, y además permite modificar el código fuente de las páginas mientras éstas se están ejecutando.

En la Figura 257 se puede ver Visual Studio .NET en pleno proceso de depuración.

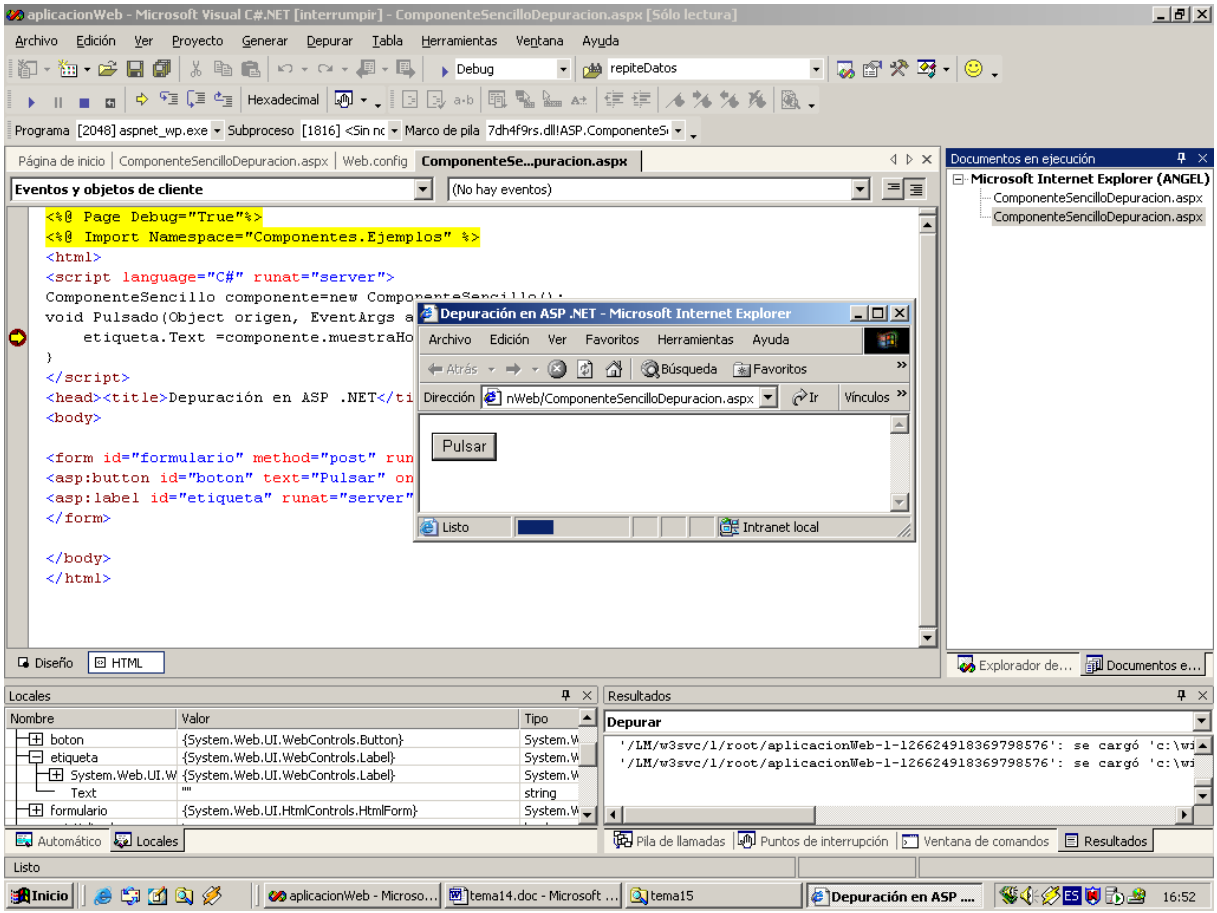


Figura 257

Creación de clases proxy con Visual Studio .NET

Para generar una clase proxy, que se comunique con un servicio Web determinado, desde Visual Studio .NET y utilizar instancias de la propia clase desde una aplicación de cualquier tipo deberemos acudir a la opción del proyecto de Agregar referencia Web. Esta forma de trabajar con referencias nos puede recordar a los proyectos de Visual Basic 6, que también nos permitían añadir referencias, que en aquel caso hacían referencia a componentes.

En el caso de Visual Studio .NET lo que vamos va a permitir el entorno de desarrollo es incluir una referencia en el proyecto que apunte al fichero ASMX en el que se encuentra implementado un servicio Web determinado.

Para agregar una referencia Web no tenemos nada más que en el explorador de soluciones de Visual Studio .NET pulsar con el botón derecho sobre el proyecto que va a hacer las veces de aplicación cliente o consumidora del servicio Web. En el menú contextual que aparece seleccionamos la opción Agregar referencia Web, tal como se puede apreciar en la Figura 258. El tipo de aplicación consumidora puede ser cualquiera, puede ser tanto una aplicación ASP . NET como una aplicación Windows.

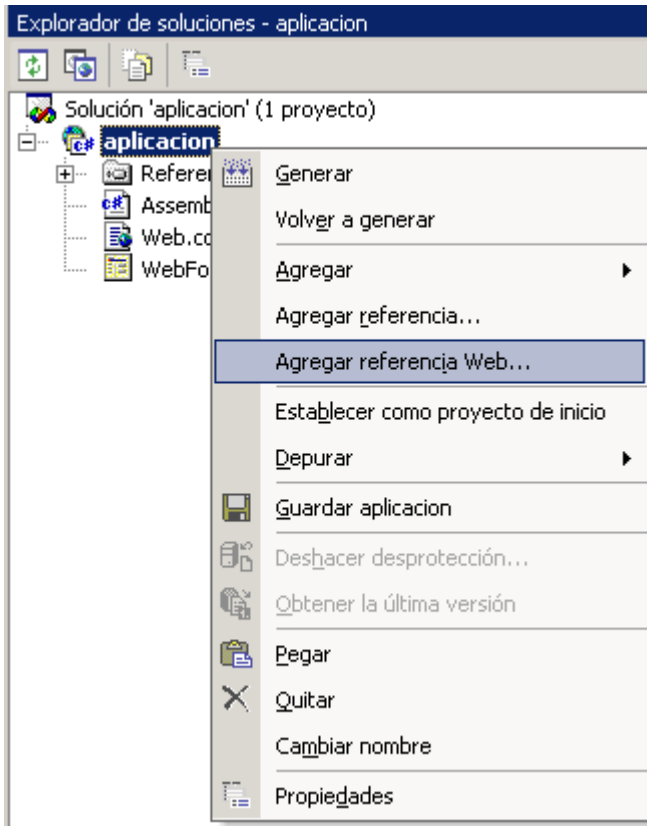


Figura 258

Una vez seleccionada la opción de Agregar referencia Web aparece un diálogo como el de la Figura 259, este diálogo nos permite indicar la dirección en la que se encuentra el servicio Web, es decir, la ruta al fichero ASMX.

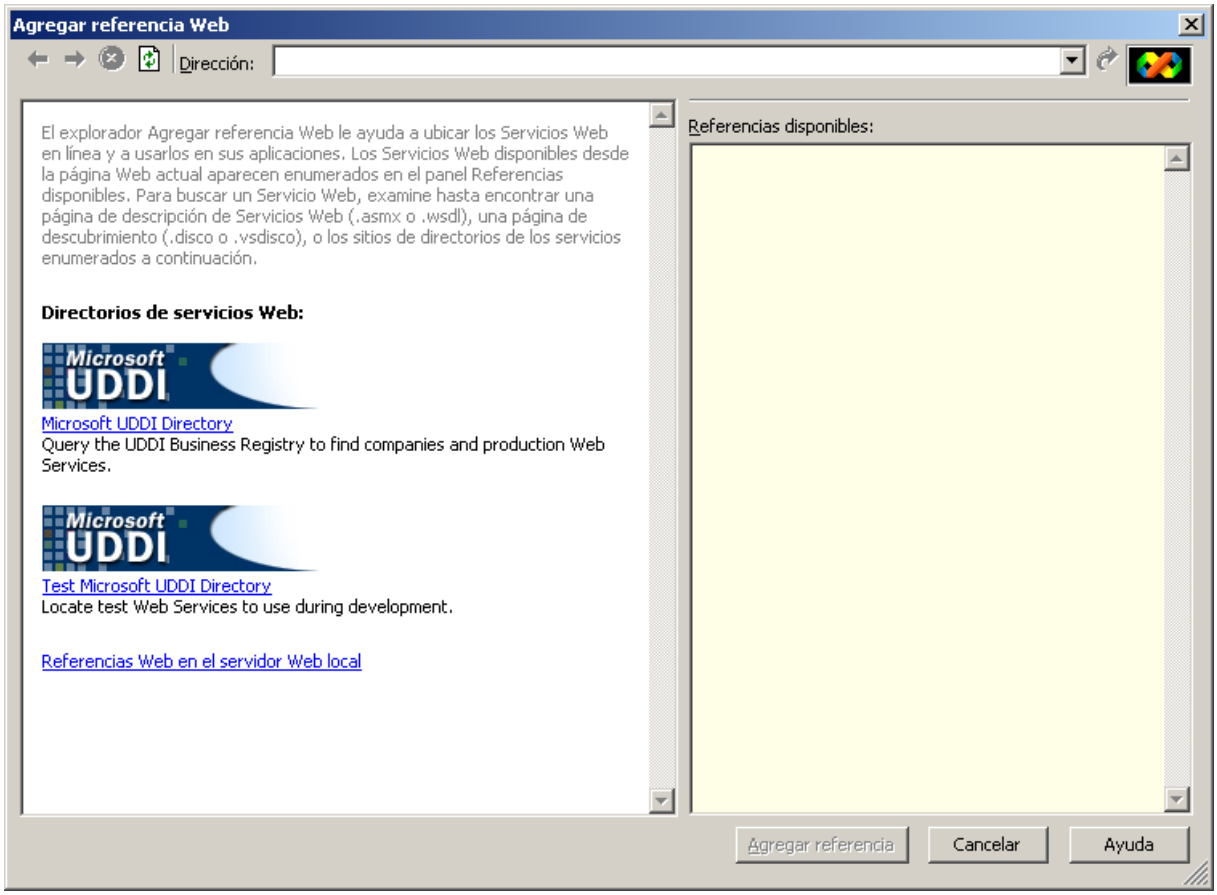


Figura 259

En este diálogo tenemos dos posibilidades, podemos pulsar sobre el enlace que nos llevará a la implementación UDDI de Microsoft para localizar allí el servicio Web que deseamos utilizar en nuestra aplicación o bien podemos indicar nosotros mismos la dirección que va a identificar al servicio Web que deseamos consumir.

En nuestro caso vamos a indicar la dirección del mismo (http://localhost/serviciosWeb/ServicioImpuestos.asmx). En este ejemplo se trata de un servicio Web que se encuentra en el servidor Web local y el fichero en el que está implementado es el fichero ServicioImpuestos.asmx.

En la ventana de diálogo de agregar referencia Web, una vez que hemos indicado la dirección del servicio, mostrará un aspecto similar al de la Figura 260. Como se puede comprobar ofrece la página de ayuda del servicio Web y también dos enlaces que nos van a permitir ver el documento WSDL asociado al servicio (enlace Ver contrato) y la documentación asociada.

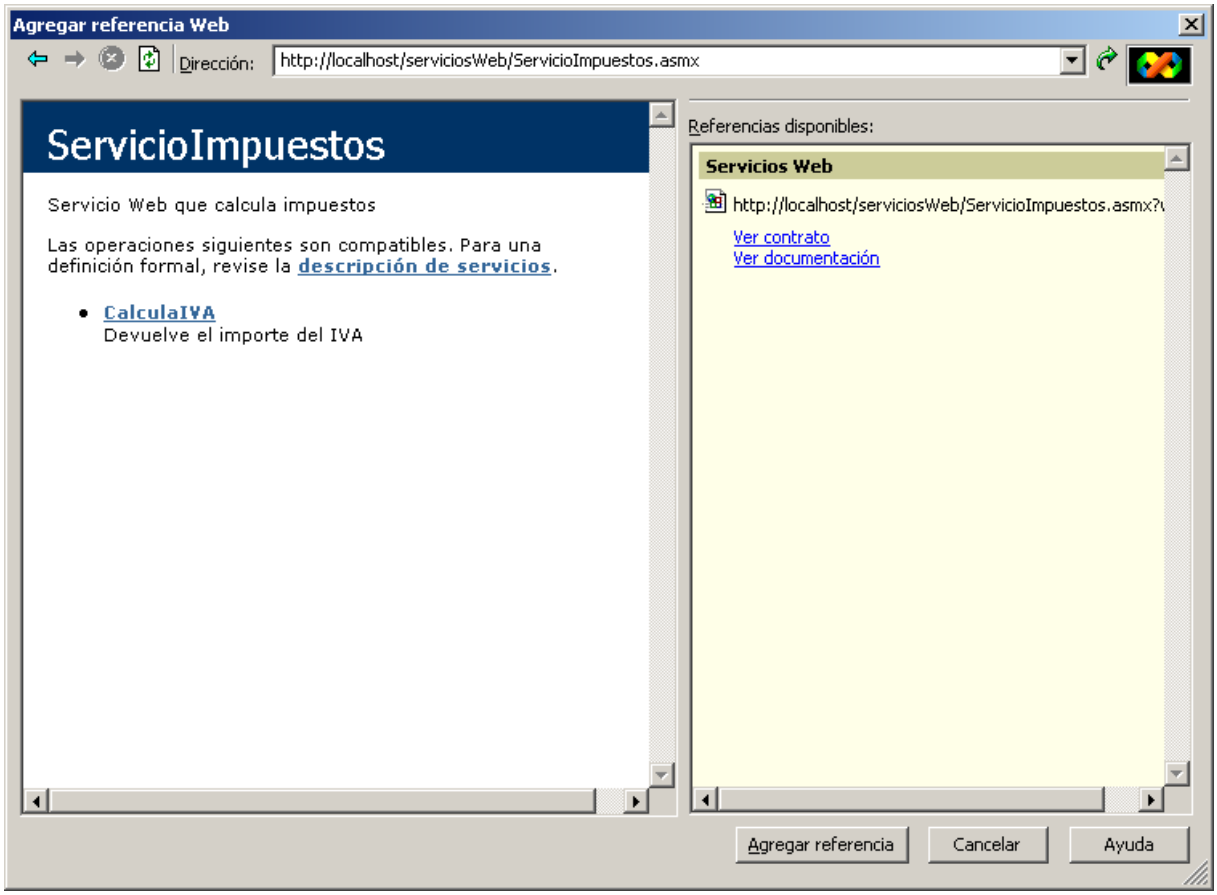


Figura 260

En este momento ya se encuentra activado el botón de Agregar referencia, por lo que ya podemos pulsarlo. Una vez pulsado este botón Visual Studio .NET crea una nueva sección dentro del explorador de soluciones llamada Referencias Web que contendrá todas las referencias Web del proyecto actual.

Internamente Visual Studio .NET ha interpretado el documento WSDL del servicio Web y a partir del mismo ha creado una clase proxy que representa a dicho servicio. También se ha añadido una copia local del documento WSDL, si pulsamos para abrir este elemento aparece la clase proxy que se ha generado de forma automática.

La Figura 261 muestra el explorador de soluciones con todos los elementos que ha generado de forma automática Visual Studio .NET.

Y si acudimos a la vista de clases obtendremos una ventana similar a la Figura 262.

Ya está generada de forma sencilla la clase proxy. Antes de ejecutar la página ASP .NET que hace uso de la clase proxy debemos generar el proyecto para que se compile dicha clase.

Con este apartado finalizamos este capítulo dedicado a la herramienta de desarrollo Visual Studio .NET. Hemos tratado únicamente algunos de los aspectos de este entorno de desarrollo, invito al lector a que investigue y acceda a la ayuda de VS .NET para que siga profundizando en el uso de esta completa herramienta.

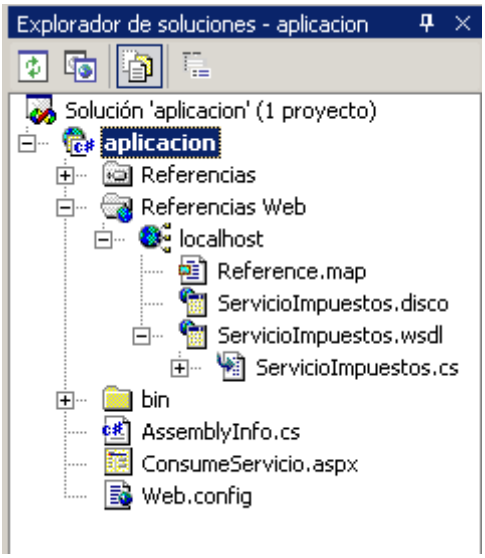


Figura 261

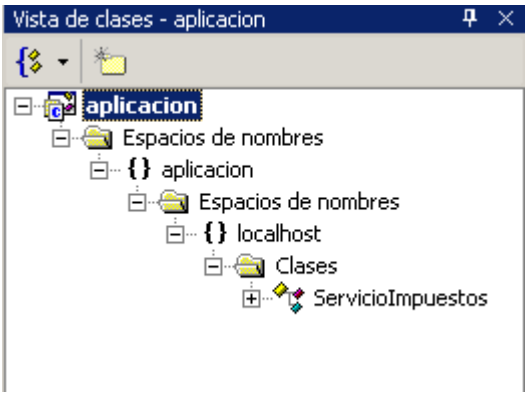


Figura 262

El sistema de ayuda

Al igual que ha sucedido con multitud de elementos del IDE, la ayuda o MSDN Library (Microsoft Development Network Library) en esta versión de Visual Studio ha sido ampliamente mejorada. La aplicación utilizada para navegar por la documentación de la ayuda es Microsoft Document Explorer, que permite al programador una enorme flexibilidad a la hora de realizar consultas sobre la ayuda disponible de los diferentes productos de desarrollo que componen la plataforma .NET.

La integración ahora del sistema de ayuda con el IDE es total, siguiendo como es natural, con la tónica establecida en versiones anteriores de proporcionar ayuda contextual siempre que sea posible. Si por ejemplo, nos encontramos diseñando un formulario y hacemos clic sobre un control TextBox, al pulsar [F1] se abrirá una nueva pestaña en la ventana principal del IDE que iniciará la ayuda, y nos posicionará en un documento relativo a dicho control (Figura 263).

La documentación de ayuda está creada en formato HTML, lo cual nos permite, gracias a su elevado grado de integración, guardar los documentos que visitemos dentro del apartado Favoritos de Internet Explorer, y consultarlos igualmente utilizando este navegador de Internet.

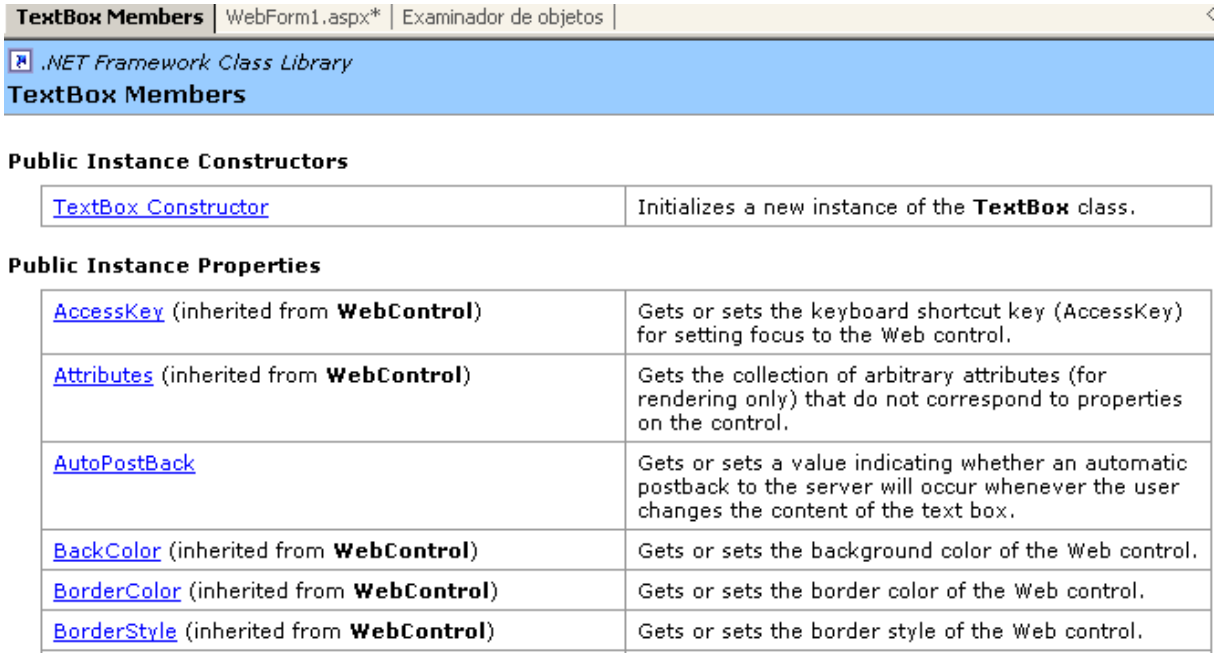


Figura 263

Ayuda dinámica

Esta ventana muestra enlaces a los temas de ayuda del entorno .NET. Podemos acceder a ella mediante la opción de menú Ayuda + Ayuda dinámica. Tal y como su propio nombre indica, los enlaces que muestra esta ventana son activos y van actualizándose dependiendo del elemento de código u objeto del diseñador en el que estemos posicionados. La Figura 264 muestra un ejemplo de esta ventana cuando estamos posicionados en un objeto Button.

Al hacer clic sobre alguno de los enlaces de esta ventana, la ficha del IDE que muestra la ayuda se actualizará igualmente con el tema seleccionado.

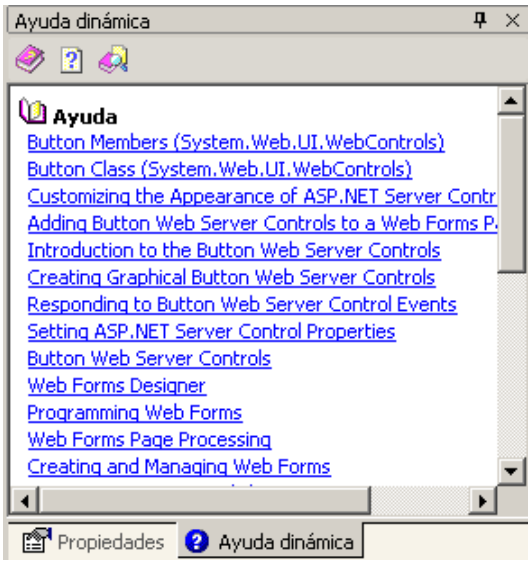


Figura 264

Contenido

Esta ventana muestra la documentación al completo de la plataforma .NET Framework organizada en áreas temáticas. Podemos abrirla de las siguientes formas:

- Haciendo clic en el primer botón de la barra de herramientas de la ventana *Ayuda dinámica* (icono con forma de libro).
- Menú Ayuda + Contenido.

La ventana mostrada tendrá el aspecto de la Figura 265.

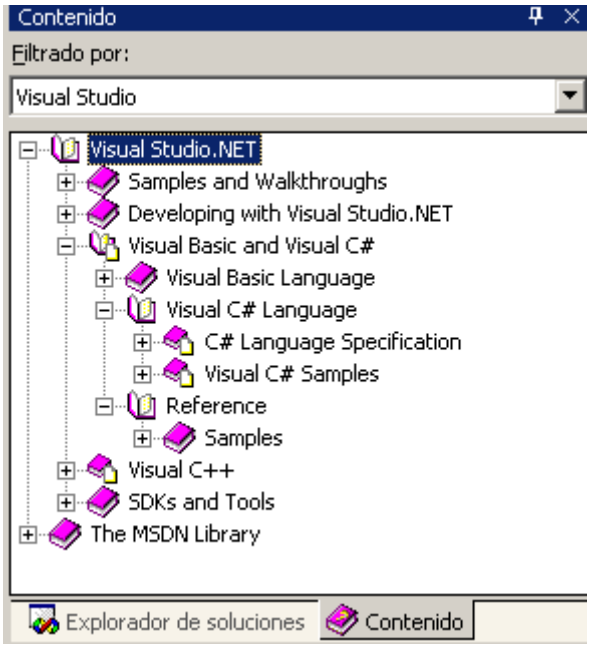


Figura 265

El manejo de la ayuda con esta ventana se basa en expandir o cerrar los libros que muestra. Al hacer clic en uno de los iconos en forma de documento, se mostrará su contenido en la ventana de ayuda que tengamos abierta en el IDE.

Índice

Esta ventana nos permite realizar una búsqueda dinámica de un elemento dentro de la ayuda. Podemos acceder a ella de las siguientes formas:

- Haciendo clic en el segundo botón de la barra de herramientas de la ventana *Ayuda dinámica* (icono con forma de interrogación).
- Menú Ayuda + Índice.

Según tecleamos un valor en el campo Buscar de esta ventana, se realizará una búsqueda dentro del MSDN, del valor más parecido a lo que hasta ese momento hemos tecleado. Podemos adicionalmente, seleccionar en la lista desplegable Filtrado por, un área para acotar la búsqueda (Figura 266).

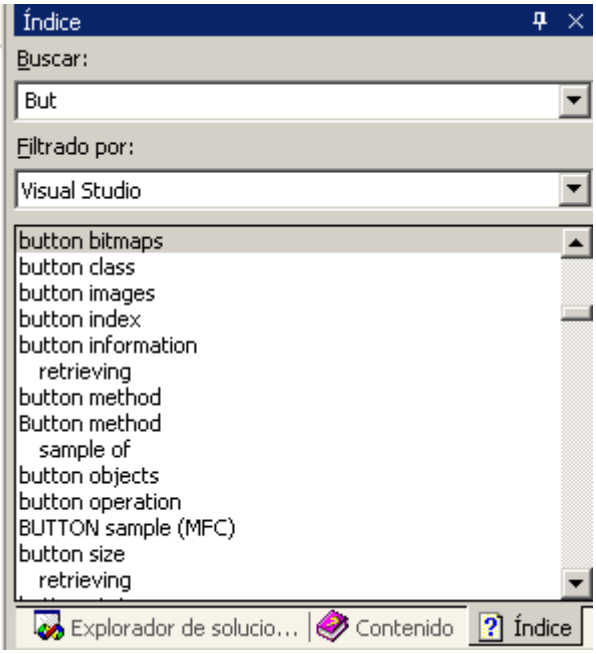


Figura 266

Una vez que localicemos en la lista de elementos, el tema deseado, haremos clic sobre él y se mostrará su contenido en la ventana principal del IDE.

Ayuda externa

En el caso de que el lector esté más acostumbrado a trabajar con la ayuda como un elemento aparte del entorno de desarrollo, puede configurarla para que sea mostrada de forma externa al IDE de VS.NET.

Para ello debemos situarnos en la página de inicio del IDE y hacer clic en el vínculo Mi perfil. A continuación, en el apartado Mostrar ayuda, haremos clic en la opción Ayuda externa. Los cambios quedarán grabados, pero no se harán efectivos hasta la siguiente ocasión en que iniciemos el IDE.

A partir de ese momento, cuando invoquemos la ayuda en cualquiera de las maneras anteriormente descritas, se abrirá una nueva ventana perteneciente a la colección combinada de Visual Studio .NET, conteniendo el tema de ayuda seleccionado (Figura 267).

La ventaja de usar la ayuda de forma externa reside en que al ejecutarse en su propia ventana, disponemos de más espacio para visualizar cada uno de los temas seleccionados.

Los mecanismos de búsqueda dentro de la ayuda están igualmente disponibles a través de las fichas desplegables situadas en el lateral, o el menú de esta ventana.

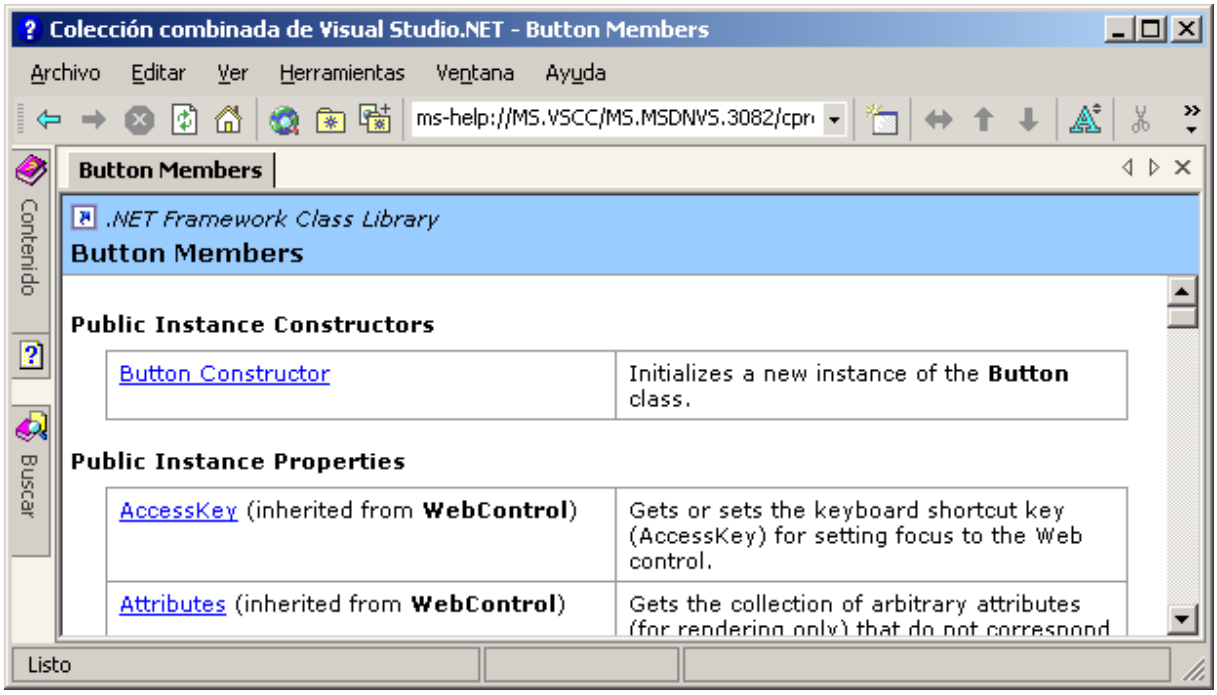


Figura 267

Otros modos de acceso a la ayuda

La ruta de menú del sistema operativo Inicio + Programas + Microsoft .NET Framework SDK, nos lleva a un conjunto de opciones que contienen toda la documentación sobre la plataforma disponible en forma de ayuda, ejemplos, artículos, etc.