Argentina $8,90 (recargo al interior $0,20)) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net 7

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

Controles simples y de acción

Etiquetas - Botones - Hipervínculos

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Controles de lista

Listas desplegables - Botones de radio
Listas de verificación

ISBN 978-987-1347-43-8
9 789871 347438 00007

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

HTML y XHTML

HTML es el acrónimo de *HyperText Markup Language* (lenguaje de marcas hipertextuales). El hipertexto es un paradigma o modelo de interfaz de usuario que permite saltar de un documento a otro cuando el usuario lo solicita. HTML fue creado por Tim Bernes-Lee (www.w3.org/People/Berners-Lee) con la idea de constituir un sistema para publicar documentos sobre el protocolo TCP/IP. Él se basó en un lenguaje de hipervínculos y SGML (un estándar de marcación generalizado). Por lo tanto, HTML es una aplicación de SGML que fue estandarizada por el comité de estándares internacional en la ISO 8879. Actualmente, el estándar de HTML es mantenido por la W3C (www.w3.org). La versión más difundida es la 4.01, cuya especificación se encuentra en www.w3.org/TR/html401.

El hipertexto es un paradigma o modelo de interfaz de usuario que permite saltar de un documento a otro cuando el usuario lo solicita.

Cuando decimos que HTML está basado en marcas, nos referimos a los elementos que lo constituyen. Estas marcas, también conocidos como tags, se utilizan para definir la interfaz de usuario.

Tabla 1 | Etiquetas que definen una página HTML

Composición de HTML	
<html>	Define el inicio del HTML, lo que le indica al navegador que el texto que viene debe ser interpretado como tal.
<head>	Define la cabecera del documento HTML. Contiene información sobre el documento que no se muestra directamente al usuario, como el título de la ventana de su navegador.
<title>	Define el título dado a la página que estamos visualizando. Suele definirse dentro de la etiqueta **<head>**.
<link>	Vincula el sitio a una hoja de estilo, para que su contenido mantenga un estándar en cuanto a fuentes, colores, vínculos, formato, etc.
<body>	Define el contenido principal o cuerpo del documento. Dentro de esta etiqueta se definen muchas otras, que conforman la página Web en sí.
<h1><h2><h3> <h4><h5><h6>	Encabezados o títulos del documento con diferente relevancia dentro de él.
<tr>	Fila de una tabla.
<td>	Celdas de datos de una tabla.
<a>	Hipervínculo o enlace a una página dentro o fuera del sitio Web en el cual estamos.
	Área de la página.
<img>	Imagen. Requiere el atributo <src>, que indica dónde está alojada para poder mostrarla.
<b>	Define el texto a continuación en negrita.
<i>	Define el texto a continuación en cursiva.
<u>	Define el texto a continuación subrayado.

La mayoría de las etiquetas debe cerrarse para delimitar el contenido afectado a ella, de la misma manera en que se la abre, pero incluyendo una barra **</etiqueta>**.

EL CÓDIGO REALIZADO EN EL LENGUAJE XHTML ESTÁ ORIENTADO A OPERAR CON DISTINTOS TIPOS DE AGENTES, NO SÓLO CON NAVEGADORES O BROWSERS.

Por ejemplo, utilizamos el tag a (<a>) para indicar un hipervínculo:

```
Los videos <a
href="VideosMasVisitados.html">
mas visitados</a> estan desde ayer...
```

El contenido dentro del tag (el texto "mas visitados") se muestra subrayado para indicar al usuario que se trata de un enlace. El usuario aprende rápidamente que, haciendo clic sobre él, indica al navegador que se dirija hacia la dirección que representa. Un desarrollador manipula el contenido de las páginas a través de la modificación de los tags o de sus propiedades mediante código del lado del servidor. También es posible modificar el contenido de la página desde el cliente, manipulando la página con JavaScript, por ejemplo. El hecho es que un desarrollador debe conocer en profundidad HTML para trabajar en aplicaciones Web.

Lenguaje HTML enriquecido

Podemos pensar en XHTML como una extensión de HTML 4. XHTML define un tipo de documento basado en XML para presentar el contenido de una página HTML.
Una página hecha en XHTML es similar a una hecha en HTML, aunque por dentro, son distintas. XHTML es XML; por lo tanto, puede ser validado utilizando cualquier herramienta estándar de éste.
Básicamente, XHTML facilita la creación de tags personalizados. En XML es muy fácil agregar tags y atributos. Por eso, ya que XTHML es XML, es más sencillo agregar tags propios o atributos a tags existentes en XHTML.
XHTML está orientado a operar con distintos tipos de agentes, no sólo con navegadores o browsers. Esto agrega el factor de extensibilidad, para hacer frente a una Internet a la que accedemos con toda clase de dispositivos: PCs, PDAs, celulares, Tablet PCs, consolas de juegos modernas con acceso a Internet, etc.
La especificación XHTML puede encontrarse en www.w3.org/TR/xhtml1. El borrador de la nueva versión XHTML 2 está en www.w3.org/TR/xhtml2.

Lenguajes de cliente

Como mencionamos anteriormente, existen algunos lenguajes que se ejecutan dentro del contexto de un navegador; éstos son interpretados en el cliente y no son convertidos a código binario en ningún momento.
El código de un programa escrito con algún lenguaje de script que se ejecuta del lado del cliente suele estar embebido dentro de un documento del tipo HTML.
Los programas escritos en lenguajes de cliente, por lo general, interactúan con el documento HTML de manera dinámica, con lo cual mejoran la experiencia del usuario.

EL CÓDIGO DE UN PROGRAMA ESCRITO CON ALGÚN LENGUAJE DE SCRIPT QUE SE EJECUTA DEL LADO DEL CLIENTE SUELE ESTAR EMBEBIDO DENTRO DE UN DOCUMENTO HTML.

El siguiente código muestra cómo escribir una función en JavaScript dentro de una página Web. Para hacerlo, utilizamos el tag <script>:

```
<script language="Javascript">
        alert("Hola");
</script>
```

El siguiente código muestra una página Web con un saludo personalizado:

```
<html>
<head>
    <title>Untitled Page</title>
     <script language="Javascript">
        function BotonPresionado(){
            var nombre = document.getElement
            ById("nombre").value;
            alert("Hola " + nombre);
        }
    </script>
</head>
<body>

    Ingrese su nombre y presione el botón
    Saludar
    <input type="text" id="nombre" />
    <input type="button" onclick="Boton
    Presionado()" value="Saludar" />

</body>
</html>
```

El botón especificado mediante el tag input type="button" tiene un evento llamado **onclick**, que es invocado cada vez que se lo presiona. Podemos escribir código personalizado para dicho evento y asignarlo como en el ejemplo:

```
<input type="button" onclick="Boton
Presionado()" value="Saludar" />
```

De esa manera, enlazamos el código con la interfaz escribiendo código para los eventos de ésta. Es posible almacenar el código de script en un archivo separado y asignarlo a la página utilizando el tag.
El siguiente fragmento de código muestra una página con un botón que llama al método LlamarOOJS, definido fuera del documento HTML y vinculado mediante el uso del tag script:

```
<html>
<head>
    <script src="script1.js"></script>
</head>

<body>
    <input type="button" onclick="Llamar
    OOJS();" value="Presione el boton" />
</body>
</html>
```


El archivo script1.js contiene lo siguiente:

UNA PÁGINA HECHA EN XHTML ES SIMILAR A UNA HECHA EN HTML, AUNQUE POR DENTRO, SON DISTINTAS. XHTML ES XML.

```
/* Implementacion de persona en javascript */

function Persona(){
    this.Nombre = "";
    this.Apellido = "";
    this.Edad = 0;
    this.ToString = ToString;
    /* metodo ToString() */
    function ToString(){
        return this.Nombre + ", " +
        this.Apellido + ", " + this.Edad;
    }
}
```

```
/* Llamada */

function LlamarOOJS(){
    var p = new Persona();
    p.Nombre = "Gabriel";
    p.Apellido = "Bulfon";
    p.Edad = 37;
    alert(p.ToString());
}
```

Notemos que en JavaScript es posible encapsular funcionalidad de manera similar a como se hace en una clase.

Esquema básico de aplicaciones Web

FIGURA 004 | Aquí podemos observar cómo funciona en forma básica una aplicación Web.

Introducción a ASP.NET

Es el momento de conocer el lenguaje más destacado en el desarrollo de aplicaciones y sitios de Internet.

ASP.NET es un entorno de desarrollo de aplicaciones Web basado en el concepto de formularios Web (*Web Form*), páginas que contienen tags especiales, cuya extensión es .ASPX. Cada formulario Web contiene código asociado, dentro del cual se gestiona la lógica de la página, se manejan los controles y objetos de todo tipo y se modifica la página en función de la interacción del usuario.

Estos formularios representan la interfaz de usuario del sistema; son lo que el usuario ve a través de su navegador.

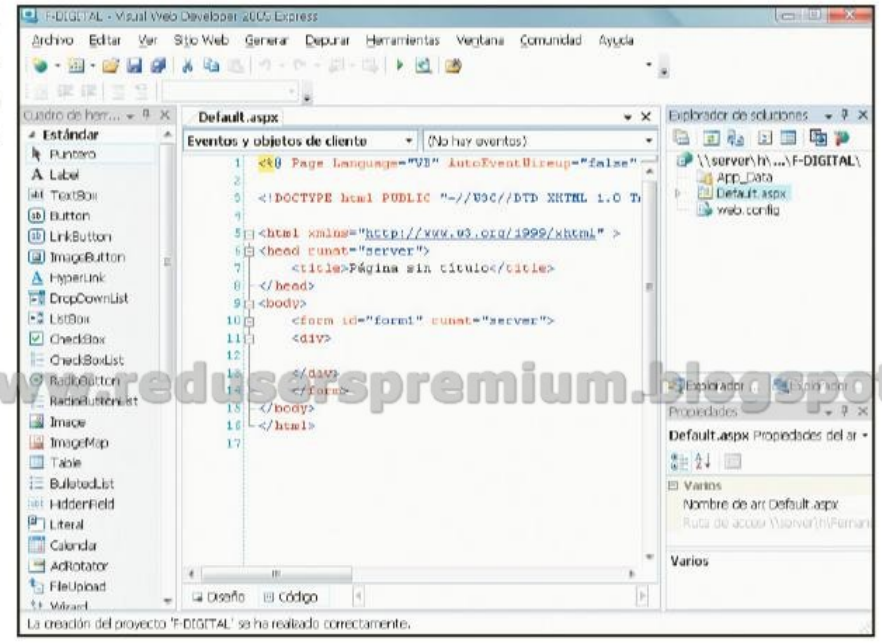
Como dijimos anteriormente, los formularios Web contienen código asociado. Ése no es código que está dentro de la página, mezclado con el HTML; en ciertas ocasiones, está dentro del archivo .ASPX, y en otras, dentro de archivos específicos de código. De esta manera, la interfaz de usuario queda perfectamente separada de la lógica de la aplicación.

> Un formulario Web ASP.NET permite separar la interfaz de usuario, de la lógica de la aplicación.

Cuando un usuario carga una página Web con extensión .ASPX en su navegador, éste la solicita a un servidor Web, que pasa el pedido a un proceso encargado de analizar la página. Cada tag propio de ASP.NET es analizado, y para cada uno se crea una clase, que se une con el código para formar la aplicación. Todo esto es procesado para armar la página de salida, que, finalmente, el servidor entrega al usuario.

Cuando hablamos de ASP.NET, nos referimos a esta tecnología que nos permite crear

FIGURA 005 | Visual Web Developer, la herramienta ideal para desarrollo ASP.NET.



Es posible desarrollar aplicaciones que se renderizan o dibujan de acuerdo con el navegador sobre el que se mostrarán.

aplicaciones Web de la misma manera en que creamos aplicaciones de escritorio.
ASP.NET es la parte Web, que incluye la creación de formularios, servicios, manejadores de HTTP, controles para formulario, controles personalizados, y más.
Internet Information Server es el servidor Web de Microsoft. Además de ser un servidor HTTP, también es un servidor FTP, NNTP y HTTPS.

Problemas que resuelve

La homogeneidad del modelo nos posiciona como desarrolladores multiplaforma, ya que podemos dar soluciones completas más allá de las fronteras de una aplicación Web.
La separación entre la interfaz de usuario (UI) y la lógica de la página está muy bien lograda en los web forms. Éstos son una parte fundamental en el proceso de desarrollo. Dicha separación permite que dos personas trabajen sobre la misma página; por ejemplo, un diseñador podría estar modificando el UI directamente sobre el formulario Web, mientras que el desarrollador agrega lógica en los eventos correspondientes.
El modelo mejorado en ASP.NET 2.0 permite que varias personas trabajen sobre una misma clase distribuida en múltiples archivos. Esto supone una ventaja importante para grupos de desarrollo.
También es posible crear aplicaciones que se renderizan o dibujan de acuerdo con el navegador sobre el que se mostrarán.
La posibilidad de utilizar múltiples lenguajes de desarrollo en el mismo proyecto implica la rotura de una barrera existente. Un desarrollador en VB.NET puede heredar clases generadas en C# sin ningún problema.
El modelo de aplicación de ASP.NET protege el código del desarrollador al permitir el despliegue del código compilado en un servidor. De esta manera, se protege la autoría del código, porque el usuario final que ve la página en su navegador sólo verá una página estática y no, el código fuente que lleva la original.

Tabla 2 | Ventajas de ASP sobre HTML

ASP.NET vs. HTML		
Funciones	**ASP.NET**	**HTML**
Creación de páginas con vínculos, imágenes y textos	X	X
Incorporación de animaciones y videos	X	X
Motor de búsqueda en los textos	X	
Acceso a base de datos	X	
Contenido dinámico actualizable periódicamente	X	
Conocimientos de usuario básicos sobre actualización del sitio Web	X	
Desarrollo de aplicaciones Web	X	
Seguridad en la protección del contenido del sitio	X	

Una aplicación Web ASP.NET, al igual que cualquier aplicación .NET, se ejecuta sobre un entorno conocido como CLR por sus siglas en inglés (*Common Language Runtime*). Dentro del CLR existe un compilador JIT (*Just In Time*) que interpreta el código IL (*Intermediate Language*) y produce código máquina. Este proceso se realiza de forma dinámica a medida que se usa la aplicación. Una ventaja del modelo interpretado radica en la posibilidad de cambiar porciones de código sin modificar toda la aplicación. Otra ventaja es la opción de intercambiar la capa de ejecución para correr el programa sobre otra plataforma.

La interpretación del modelo de ejecución no se realiza cada vez que un usuario pide un recurso. Es decir, al pedir una página, CLR compila el recurso y lo entrega. La página queda compilada para la próxima petición, y salvo que algo cambie en ella, se servirá compilada desde ese momento. Este modelo acelera la respuesta de la aplicación, ya que el código se ejecuta totalmente compilado una vez que los recursos han sido pedidos.
Esto puede notarse al solicitar por primera vez una página de la aplicación. Las peticiones siguientes no necesitan la intervención del compilador de CLR para ser servidas.
El modelo de ejecución y desarrollo ha unido los beneficios de un modelo interpretado y uno compilado. El resultado proporciona más ventajas a los desarrolladores de aplicaciones.

Controles

Dentro del modelo de desarrollo de ASP.NET es posible encapsular funcionalidad en los controles. Éste es un mecanismo de reutilización de código mejorado. Hay dos tipos de controles que se pueden crear: de usuario (*Users Controls*) y Web personalizados (*Web Custom Controls*).
Los primeros contienen HTML y código, al igual que una página ASPX. Se emplean como un mecanismo de reutilización de código, separando las porciones de páginas que suelen repetirse en una aplicación Web y armando componentes que se colocan dentro de ellas. Podemos hacer un control de usuario que contenga todo el HTML y la lógica del encabezado (*header*). Luego, en cada página, colocamos el control realizado. De esa manera, el encabezado se encuentra sólo en un lugar de la aplicación y es sencillo efectuar modificaciones. En el siguiente código podemos ver la muestra de un encabezado simple encapsulado dentro de un control de usuario:

> Las aplicaciones .NET se ejecutan sobre el CLR compilándose en modo JIT (Just In Time).

```
<%@ Control Language="C#" AutoEvent
 Wireup="true" CodeFile="Header.ascx.cs"
 Inherits="Header" %>
<div id="header">
    <h1>Mi Sitio web</h1>
</div>
```

Notemos que, salvo la primera línea, el resto es HTML puro.

! Requerimientos

Para desarrollar aplicaciones ASP.NET y servicios Web necesitamos tener instalado el entorno de aplicaciones Web gratuito llamado Visual Web Developer Express, disponible en http://msdn.microsoft.com/vstudio/express/vwd. Si queremos desarrollar librerías de clases propias, precisamos contar, además, con alguno de los entornos para los lenguajes determinados.

La Figura 006 muestra una misma página Web desarrollada en Dreamweaver MX y Visual Web Developer Express 2005.
Los controles personalizados son componentes compilados que se ejecutan en el servidor, encapsulando la interfaz de usuario y la funcionalidad dentro de paquetes reutilizables. Es posible realizar controles personalizados derivando controles de la librería de controles de .NET.

Librería de clases

Una librería de clases es un conjunto de clases utilizadas para un fin determinado, en general, empaquetado de manera que facilita su distribución. Las librerías que se distribuyen con .NET son extensas y tienen cientos de clases agrupadas en múltiples ensamblados, cada uno de los cuales se empaqueta en una DLL. La librería de .NET se conoce como *.Net Class Library*.
Quienes desarrollan con la tecnología .NET disponen en este framework de clases para acceso a datos, manejo de archivos XML, control de interfaces, manejo de recursos de la máquina, seguridad de la aplicación, criptografía, controles, etc. El uso del framework implica una ventaja al momento de desarrollar aplicaciones Web.

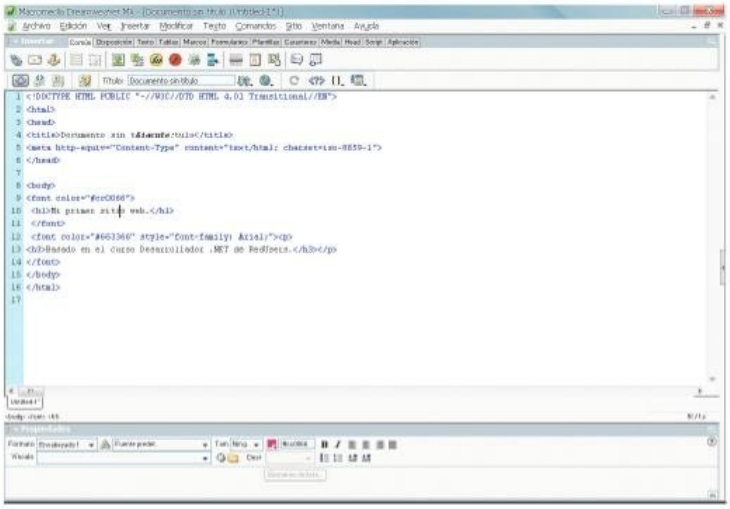
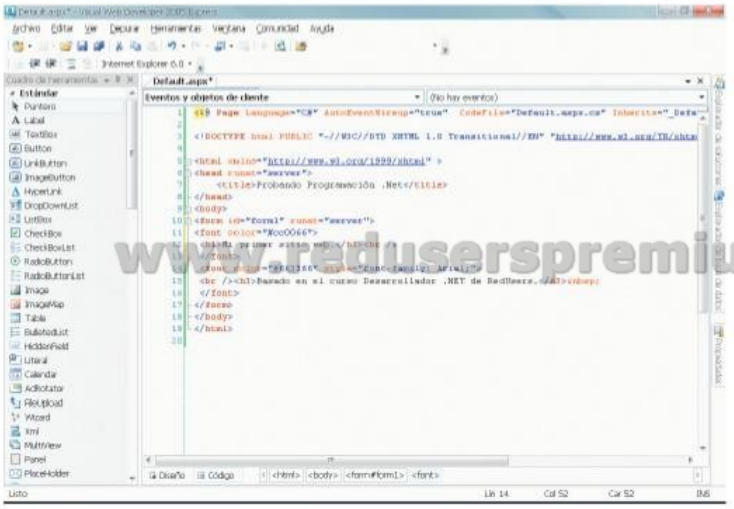


FIGURA 006 | Comparación de una misma página desarrollada en distintas plataformas.

Página desarrollada en Dreamweaver MX



Página desarrollada en Visual Web Developer

Formularios Web

Una aplicación Web muestra la información al usuario a través de un browser u otro dispositivo. Los clientes solicitan a la aplicación páginas Web, que implementan la lógica utilizando código que se ejecuta del lado del servidor. Éste realiza las funcionalidades propias del proceso y genera HTML de forma dinámica. Entonces, una página Web en ASP.NET está formada por dos partes: la visual, que se implementa en un archivo con extensión .ASPX; y la lógica, que se puede implementar dentro de un script en la misma página o en un archivo de código separado.
La página ASPX contiene una directiva que la vincula al archivo donde reside el código. El siguiente fragmento de código muestra el atributo **CodeBehind** utilizado en las primeras versiones de ASP.NET. Se puede ver que la segunda línea utiliza el atributo **CodeFile**, disponible a partir de la versión 2.0:

```
<%@ Page Language="C#" CodeBehind="Default.
 aspx.cs" Inherits="WApp1._Default" %>

<%@ Page Language="C#" CodeFile=
"Test.aspx.cs" Inherits="Test" %>
```

Este modelo de trabajo es similar al que se usó durante años para desarrollar aplicaciones Windows. La parte visual está formada por etiquetas HTML y etiquetas de controles del lado del servidor (*Server Controls Tags*). La parte lógica puede estar dentro de un bloque de script en la misma página o en un archivo de código separado. En este último caso, el archivo que contiene el código se conoce como *codebehind file*.
La primera vez que una página es solicitada en el servidor o la primera vez que es solicitada luego de ser modificada, el proceso analiza la página HTML y los controles, y crea una clase para ella. Esta clase es compilada y procesada para producir el código HTML de salida.

> Una página Web en ASP.NET está formada por dos partes: la visual y la lógica, que se implementa dentro de un script.

Conceptos de HTML Forms

La acción dentro de una página Web ocurre entre las etiquetas **<form>** y **</form>**; esto es lo que se conoce como HTML Form: un grupo relacionado de elementos de interfaz. Entre ambas etiquetas ponemos todas las de HTML y las de controles del servidor que interactuarán con el usuario:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <!– Aqui se coloca el contenido de
        la pagina –>
    </div>
    </form>
</body>
</html>
```

Dentro de un HTML Form colocamos otros controles. El siguiente fragmento de código muestra un campo de edición y un botón para enviar los datos al servidor:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
```

UN HTML FORM ES UN GRUPO RELACIONADO DE ELEMENTOS DE INTERFAZ. ENTRE LAS ETIQUETAS <FORM> Y </FORM> PONEMOS TODAS LAS ETIQUETAS DE HTML Y DE CONTROLES DEL SERVIDOR QUE INTERACTUARÁN CON EL USUARIO.

```
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <!– Aqui se coloca el contenido de
        la pagina –>
        Nombre <asp:TextBox ID="Usuario"
        runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat=
        "server" Text="Enviar" />
    </div>
    </form>
</body>
</html>
```

Cuando se presiona el botón, el formulario es enviado por el método **POST** o **GET** al servidor. En él, las clases relacionadas con los controles de página son instanciadas y cargadas con los valores que ingresó el usuario. La página pasa por distintos momentos, y permite que en cada uno de ellos se escriba código asociado. Así, es posible escribir código antes de que la página se cargue, para cargar los controles de página con datos de una base.

El mecanismo de suscripción a los "momentos" referidos anteriormente es un mecanismo de eventos. Cada vez que la página cambia de

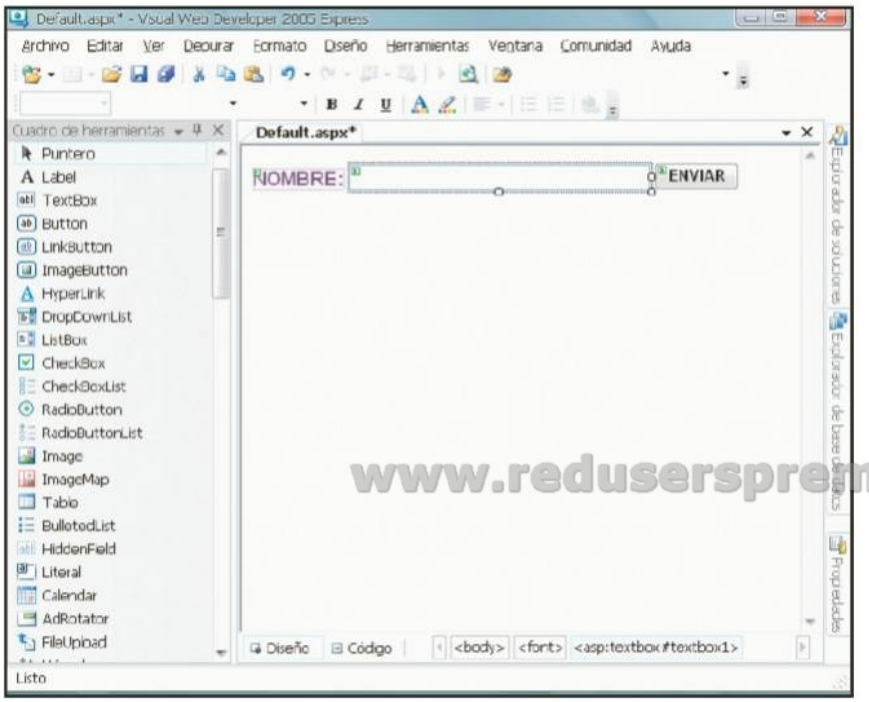


FIGURA 007 | Aquí vemos la apariencia de un simple formulario que permite enviar el dato introducido a través de los métodos POST o GET.

estado, dispara un evento asociado. Podemos escribir manejadores para cada uno de ellos.

Eventos

Los eventos son un mecanismo utilizado para indicar que hubo algún cambio en el estado interno de un objeto. En ocasiones, necesitamos realizar tareas cuando ocurren modificaciones en los objetos. Para no tener que observar o consultar permanentemente el estado de un objeto, recurrimos a este mecanismo de suscripción a los cambios. Así, podemos escribir código y asociarlo al evento. Cuando el evento se produzca, nuestro código será invocado.
El siguiente código muestra una página Web ASP.NET que contiene un control **TextBox**, un **Label** y un **Button**; este último, con un manejador para el evento **OnClick**:

```
<%@ Page Language="C#" AutoEventWireup=
"true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
```

FIGURA 008 | Aquí vemos el saludo según el nombre ingresado en el textbox.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Pagina ejemplo: Saludo</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>Ingrese su nombre y presione el
        boton:</h1>
        <asp:TextBox ID="TextBox1" runat=
        "server">Tipee aqui</asp:TextBox>
        <asp:Button ID="Button1" runat=
        "server" Text="Saludar"
            OnClick="Button1_Click" />
        <asp:Label ID="Label1" runat=
```

Tabla 3 | Funciones de cada uno de los ciclos de vida

Estado de la página	Qué ocurre
Solicitud de página	Analiza si la página debe ser revisada o no. Si la versión de caché puede ser entregada, se envía.
Inicio	Setea Request y **Response**. Se decide si se trata de un **Postback** o de un nuevo **Request**.
Inicialización de página	Se inicializan los controles y se les asigna el ID de la página.
Carga (**Load**)	Si se trata de un Postback, se cargan los controles utilizando la información de **ViewState**.
Validación	Se llama al método **Validate** de todos los controles de la página.
Manejador de eventos de **Postback**	Si se trata de un Postback, se procesan todos los manejadores de eventos de la página.
Rendering	Una vez que el estado de cada control se grabó en **ViewState**, se llama al método **Render** de cada control de la página.
Unload	Durante esta etapa se liberan recursos, como **Request** y Response.

```
        "server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

El archivo de código **Default.aspx.cs** contiene el manejador para el evento:

```
protected void Button1_Click(object sender,
EventArgs e)
{
Label1.Text = "Hola " + TextBox1.Text;
}
```

Al probar la página presionando el botón, se disparará el evento **OnClick**, y todos los manejadores de eventos asociados serán avisados. En este caso, nuestro manejador de eventos modificará el control **Label1** para que muestre el saludo.

Ciclo de vida

El ciclo de vida ocurre cuando una página Web ASP.NET es solicitada y, entonces, pasa por una serie de estados o pasos de procesamiento. La Tabla 3 muestra qué ocurre en cada uno de ellos.

En cada uno de los estados mencionados, la página dispara eventos. Podemos escribir código asociado a ellos para actuar en consecuencia. Los eventos más importantes están listados en la Tabla 4.

Render: Es el método que se encarga de escribir el HTML de salida para cada control. Si escribimos nuestros propios controles personalizados, debemos sobrescribir el método **Render**, cuya salida serán las etiquetas correspondientes al control.

Tabla 4 | Función de cada evento disparado por una página

Evento	Descripción
PreInit	Consultando la propiedad **IsPostBack**, podemos determinar si es la primera vez que se carga la página.
Init	Se dispara luego de que todos los controles se inicializaron. En este evento podemos leer o inicializar propiedades de los controles.
InitComplete	Todos los controles ya fueron inicializados.
PreLoad	Momento previo a la carga de la página. Se carga **ViewState** para la página y todos los controles.
Load	Se llama al método **OnLoad** para cada uno de los controles de la página. Este evento es el más utilizado para setear propiedades de los controles.
Control Events	En este momento se llama a todos los manejadores de eventos de los controles, como el evento **Click** de un botón.
LoadComplete	Se utiliza para operaciones que requieren que todos los controles de la página estén cargados.
PreRender	Se llama al evento **DataBind** para cada control de enlace de datos (*Data Bound Controls*). Este evento ocurre para cada control.
SaveStateComplete	Este evento se dispara cuando **ViewState** ya fue grabado. Aquí ya no se pueden hacer cambios en la página.
Unload	Se puede utilizar **Unload** de la página para eliminar recursos que ya no se utilizan; por ejemplo, para cerrar conexiones a base de datos.

Controles Web

A continuación veremos cómo emplear los controles provistos en ASP.NET en nuestras aplicaciones Web.

¿Qué es un control Web?

Cuando hablamos de controles Web, nos estamos refiriendo a componentes que se ejecutan en el servidor. La mayoría de los controles Web se utiliza para encapsular porciones de interfaz de usuario. Otro de los modos de usarlos en forma declarativa es escribiendo el siguiente código en la vista de código dentro de la página ASPX:

```
<asp:TextBox ID="TextBox1" runat=
"server">Tipee aquí</asp:TextBox>
```

Podemos utilizar los controles Web arrastrándolos desde la barra de herramientas hacia el formulario Web en la vista de diseño, y modificando las propiedades desde la ventana correspondiente.

Para utilizar estos controles desde el código, tenemos que especificar dos atributos: **runat="server"** e **ID**. Este último se utiliza para darle un nombre único al control. No podemos repetir ese nombre dentro de una misma página. Los controles nos indican que algo ha ocurrido por medio de la generación de eventos. Éstos son métodos que podemos utilizar para escribir código en respuesta a los cambios de un control.

Cuando un control necesita notificar algún evento, lo hace emitiendo un **postback**, una nueva solicitud a la misma página. Entre cada uno de ellos, los controles mantienen su estado, definido por el valor de sus propiedades. **ViewState** es un mecanismo automático que mantiene el valor de los controles entre cada postback.

> Los controles Web nos ayudan a desarrollar aplicaciones Web en forma sencilla.

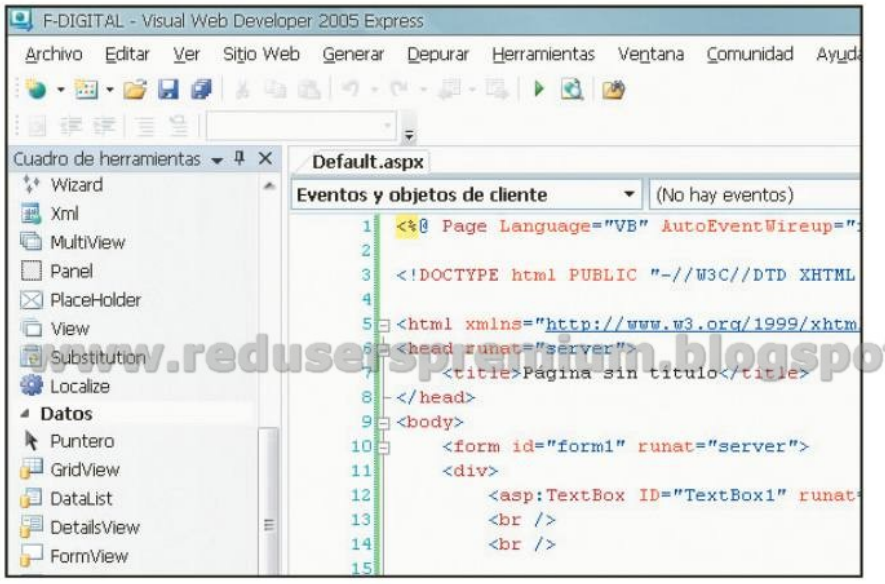


FIGURA 009 | Como podemos observar, el cuadro de herramientas es similar al de Visual Basic .NET y Visual C#.

CUANDO UN CONTROL NECESITA NOTIFICAR ALGÚN EVENTO, LO HACE EMITIENDO UN POSTBACK. VIEWSTATE ES UN MECANISMO QUE MANTIENE EL VALOR DE LOS CONTROLES.

Tipos de controles Web

Clasificaremos los controles Web en dos categorías:

- Controles de servidor HTML
- Controles de servidor Web

Los controles de servidor HTML son etiquetas HTML con el agregado de dos atributos: **ID** y **runat="server"**. Por lo tanto, podemos convertir cualquier elemento de HTML en un control Web agregando esos atributos.

El motivo por el cual podemos querer convertir una etiqueta HTML en un control de servidor HTML es que, de esa manera, podemos cambiar sus propiedades programáticamente. Así, podremos acceder a un control de servidor HTML desde el código fuente, como lo muestra el siguiente fragmento de código:

```
<form id="form1" runat="server">
    <div>
        <h1>Soy una etiqueta H1</h1>
        <h1 id="ControlH1" runat=
        "server"></h1>
    </div>
</form>
```

En este caso vemos dos etiquetas **H1**; la segunda es un control de servidor HTML. Como dijimos antes, podemos acceder programáticamente desde el servidor a las propiedades del control de servidor **HTML H1**. El siguiente fragmento de código accede al control

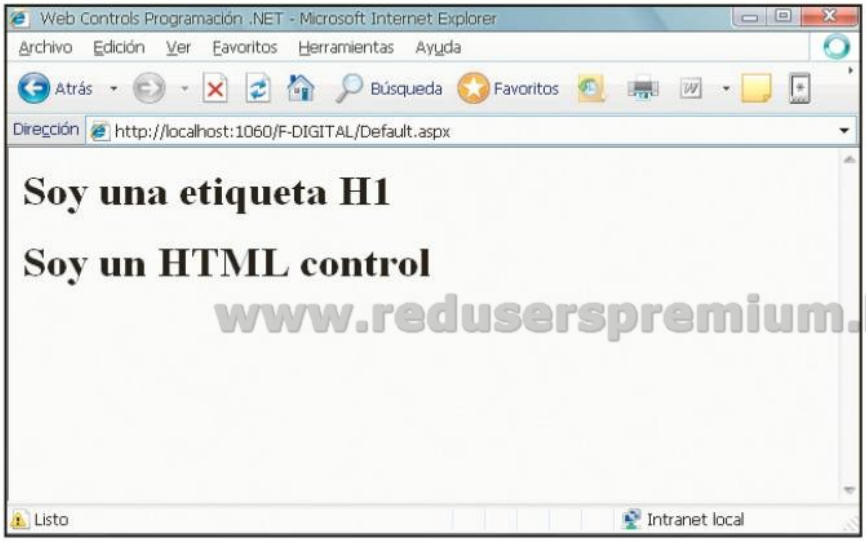


FIGURA 010 | El control de servidor HTML H1 es modificado desde el servidor.

ES IMPORTANTE SABER QUE UN CONTROL DE SERVIDOR WEB VARÍA SU SALIDA DE ACUERDO CON EL BROWSER UTILIZADO, Y DEBEREMOS CUIDAR SU USO PARA EVITAR QUE FALLE.

de servidor HTML H1 del lado del servidor dentro del evento de página **Page_Load()**:

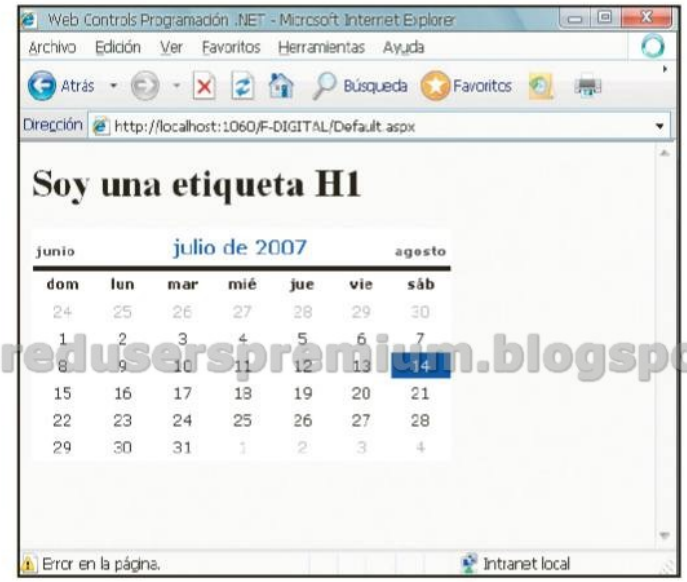
```
protected void Page_Load(object sender,
EventArgs e)
{
  this.ControlH1.InnerHtml = "Soy un HTML
  control";
}
```

Los controles de servidor Web son controles Web con más funcionalidades y características que los HTML. En este tipo encontramos controles más complejos, como calendarios o árboles. Este grupo no está limitado al mapeo de etiquetas HTML, y cuando se renderizan, lo hacen generalmente utilizando varias etiquetas de HTML. Por ejemplo, el control de servidor Web Calendar se renderiza como una etiqueta HTML de tabla con estilos y otras etiquetas HTML dentro.

Un control de servidor Web tiene la característica de detectar automáticamente el tipo y la versión del browser utilizado. Esta posibilidad es explotada para renderizarse de la mejor manera posible.

Existen también controles de servidor Web que permiten escribir el HTML de salida. Éstos manejan el concepto conocido como plantillas o templates.

FIGURA 011 | Web Calendar es parte de los controles incorporados en la versión 2.0 de ASP.NET.



El siguiente fragmento de código muestra el empleo del control de servidor Web Repeater, que utiliza template:

```
<form id="form1" runat="server">
    <div>
        <asp:Repeater ID="Repeater1"
        runat="server">
            <HeaderTemplate>
                <h1>Listado de libros</h1>
            </HeaderTemplate>
            <ItemTemplate>
                <%# Eval("Nombre") %>
                <br />
            </ItemTemplate>

            <FooterTemplate>
                <hr />
                Consultas a
                <a href="mailto:consultas@
                mimail.com">consultas
                @mimail.com</a>
            </FooterTemplate>
        </asp:Repeater>
    </div>
</form>
```

Observando el código, podemos ver que la etiqueta **<asp:Repeater />** contiene otras más. En el ejemplo, **<HeaderTemplate />**, **<ItemTemplate />** y **<FooterTemplate />** están dentro de **<asp:Repeater/>**.
Este control de servidor Web nos permite definir el HTML de salida dentro de las etiquetas. Al "enlazar" una fuente de datos con el control mediante la propiedad **DataSource**, el control repite el contenido de la etiqueta **<ItemTemplate />** para cada elemento de la fuente de datos. De esta manera, estamos estableciendo una plantilla para cada uno de los registros de la fuente. Por eso, estos controles se conocen con el nombre de controles de plantilla o *template controls*.

Para ver el funcionamiento del control de servidor **Web Repeater**, vamos a escribir el siguiente código dentro del evento **Page_Load()**:

```
public partial class _Default :
System.Web.UI.Page
{
    protected void Page_Load(object sender,
    EventArgs e)
    {
        // declara una coleccion de objetos
        libro
        System.Collections.Generic.List
        <Libro> libros =
            new System.Collections.
            Generic.List<Libro>();

        // agregar algunos
        libros.Add(new Libro("Usando
        XHTML"));
        libros.Add(new Libro("ASP.NET
        PRO"));
        libros.Add(new Libro("Windows
        Vista"));

        // Enlazar el control template
        Repeater
        this.Repeater1.DataSource = libros;
        DataBind();
    }

    class Libro {
        protected string _nombre;
        public string Nombre {
            get { return _nombre; }
            set { _nombre = value; }
        }

        public Libro(string nombre) {
        this.Nombre = nombre; }
    }
}
```

EXISTEN TAMBIÉN CONTROLES DE SERVIDOR WEB QUE PERMITEN ESCRIBIR EL HTML DE SALIDA. ÉSTOS MANEJAN EL CONCEPTO DE PLANTILLAS O TEMPLATES.

La expresión **<%# Eval("Nombre") %>** se utiliza para indicar en qué lugar deseamos colocar la propiedad Nombre del objeto actual. Hay que tener presente lo siguiente: el control de servidor Web **Repeater** es enlazado con una fuente de datos representada aquí por la colección **libros**, que contiene tres instancias de la clase **Libro**. Cada una tendrá una propiedad llamada **Nombre**. El control de servidor Repeater repetirá tres veces el elemento <**ItemTemplate /**>, y en cada repetición reemplazará la expresión anterior por el valor de la propiedad **Nombre** correspondiente.

Tipos de controles de servidor Web

Los controles de servidor Web permiten trabajar de una manera más flexible que los HTML, ya que contienen una interfaz más rica, definida por las propiedades y los métodos públicos que presentan.
En su forma declarativa –es decir, en su representación en la vista de HTML–, todos los controles de servidor Web utilizan el prefijo **asp**.

Control Label

El control de servidor **Label** brinda una forma de usar texto en una página Web. Se utiliza siempre que necesitemos cambiar un texto de una página desde el servidor en tiempo de ejecución.

```
<asp:Label ID="Label1" runat="server"
Text="Hola"></asp:Label>
```

Si se quiere utilizar texto estático, hay que recurrir a HTML. Una alternativa es el uso del control del servidor **Literal**.
El valor de las propiedades del control puede modificarse en tiempo de diseño o ejecución; es decir, cuando utilizamos Visual Studio o cuando la aplicación se ejecuta, respectivamente. El control de servidor Label contiene las propiedades **AccessKey** y **AssociatedControlID**, las cuales, juntas, se utilizan para configurar un **active caption** (combinación de teclas para acceder a un control de formulario).
Se puede utilizar el control Label dentro de controles de Listas (Repeater, DataList, GridView, DetailsView y FormView) para mostrar información de una base de datos.

Seguridad

La propiedad **Text** de un **Label** puede emplearse con cualquier *string*, incluyendo texto que contenga etiquetas HTML. Si el control contiene estas etiquetas, intentará interpretarlas. Por ejemplo, si asignamos a la propiedad **Text** de un control **Label** el valor **"Hola"**, el control renderizará el texto *Hola* en cursiva.

SETEAR LA PROPIEDAD TEXT UTILIZANDO EL MÉTODO HTMLENCODE CONVIERTE CUALQUIER MARKUP EN SU REPRESENTACIÓN DE TEXTO.

Como principal regla de seguridad, debemos evitar asignar directamente la propiedad Text de un control Label si desconocemos o no estamos seguros de la fuente de los datos. De lo contrario, nos expondremos a un ataque por medio de la inyección de código.
Si no estamos seguros respecto de la fuente de informacion, debemos codificar el string **string encode**. Codificar el string convierte los elementos de HTML utilizando caracteres reservados, para que éstos sean mostrados en vez de ejecutados.

Control Literal

Es similar a **Label**, pero la diferencia fundamental radica en que no soporta estilos, *themes* o *skins*.

```
<asp:literal ID="Literal1" runat="server"
Text="Hola"></asp:literal>
```

Este control contiene una propiedad llamada **Mode**, que se utiliza para especificar de qué manera manejar el HTML. Tenemos tres posibilidades, según se observa en la Tabla 5.

Control TextBox

Representa un control para el ingreso de datos en un formulario. Contiene una propiedad llamada **TextMode**, cuyos posibles valores modifican la salida para renderizar un control de ingreso de texto, un control de ingreso de password o un control de ingreso de múltiples líneas.

Control Button

Utilizamos este control para colocar un botón **submit** o de comando, que envía los datos del formulario al servidor.
El evento **OnClick** del botón permite declarar un manejador de evento que se dispara en el momento en que el usuario lo presiona. Por lo tanto, si queremos programar alguna acción

Tabla 5 | Control Literal

Mode	Descripción
Transform	El HTML es acomodado según el navegador de Internet. Se utiliza cuando se renderiza sobre dispositivos móviles que emplean otro protocolo en vez de HTML
PassThrough	Cualquier HTML se renderiza como es.
Encode	Se codifica el HTML utilizando **HtmlEncode**. Por ejemplo, **<b>** es renderizado como **<b&gt**. Esta opción es útil si se desea mostrar HTML en vez de utilizarlo.

La tabla muestra los posibles valores de la propiedad Mode de un Control Literal.

que se ejecute cuando el usuario presiona el botón, ese evento es el lugar adecuado.

```
<asp:Button ID="Button1" runat="server"
Text="Button" OnClick="Button1_Click" />
```

En el archivo de código asociado debemos completar el manejador de evento para el evento **OnClick**, de la siguiente manera:

```
protected void Button1_Click(object sender,
EventArgs e)
{
      // escribir el codigo aquí
}
```

Así, podemos llevar la acción desde la interacción del usuario al servidor donde controlamos el acceso a los datos o la lógica de la aplicación.

Control HyperLink

Se utiliza para crear un link a otra página. Su representación más sencilla renderiza una etiqueta **<a/>** en la página.

Control LinkButton

Tiene la misma apariencia que **HyperLink**, pero con la funcionalidad del control **Button**. Es decir, si cliqueamos sobre el control, se producirá un evento en el servidor que nos permitirá escribir código asociado a él.

El control **LinkButton** se renderiza como una etiqueta **<a/>**, que emite un **Postback** a la página de la siguiente manera:

```
<a id="A1" href="javascript:__doPostBack
('LinkButton1','')">LinkButton</a>
```

Control ImageButton

Este control se emplea para colocar una imagen como un botón.
Al renderizarse, genera la etiqueta HTML **<input type="image" .../>**:

```
<input type="image" name="ImageButton1"
id="Image1" src="MiFoto.jpg"
style="border-width:0px;" />
```

Control DropDownList

Representa un control que permite al usuario seleccionar elementos de una lista. Tiene propiedades para manejar la apariencia de salida mediante el uso de los atributos **BorderColor**, **BorderStyle** y **BorderWidth**, entre otros.
Cada elemento dentro del control es un control del tipo **ListItem**.

```
// Declara una lista de seleccion para dos
valores posibles.
// Cada valor estara asociado a un valor
representado por el
// atributo Value
<asp:DropDownList ID="DropDownList1"
runat="server">
      <asp:ListItem Text="Azul"
      Value="1"></asp:ListItem>
      <asp:ListItem Text="Verde" Value="2"
      Enabled="true"></asp:ListItem>
</asp:DropDownList>
```

Tabla 6 | Valores de la propiedad TextMode

TextBoxMode.SingleLine	<input type="text" value="Ingrese un valor..." />
TextBoxMode.Password	<input type="password" value="Ingrese su contraseña..." />
TextBoxMode.MultiLine	<textarea name="Comentarios" rows="2" cols="20" id="Textarea1"></textarea>

La tabla muestra los posibles valores de TextMode y la salida asociada a cada uno de ellos.

LOS CONTROLES DE SERVIDOR WEB PERMITEN TRABAJAR DE UNA MANERA MÁS FLEXIBLE QUE LOS HTML, YA QUE CONTIENEN UNA INTERFAZ MÁS RICA, DEFINIDA POR PROPIEDADES Y MÉTODOS.

El control **DropDownList** soporta *data binding* (una forma de conectar controles de interfaz con objetos de la aplicación). Gracias a esto, podemos conectarlo para que se cargue en función de una fuente de datos o una lista de elementos.
Para saber qué elemento seleccionó el usuario, consultamos la propiedad **SelectedIndex** o **SelectedValue**.

Control ListBox

Representa un control de selección de elementos con opción de selección múltiple. La propiedad **SelectedMode** controla el comportamiento del control. El valor **SelectionMode = Multiple** indica que se puede seleccionar más de un elemento en la lista.

```
<asp:ListBox ID="ListBox1"
runat="server"></asp:ListBox>
```

Cada elemento dentro del control es un control del tipo **ListItem**:

```
<asp:ListBox ID="ListBox1" runat="server"
AppendDataBoundItems="True">
          <asp:ListItem Text="Telefono fijo"
          Value="1"></asp:ListItem>
```

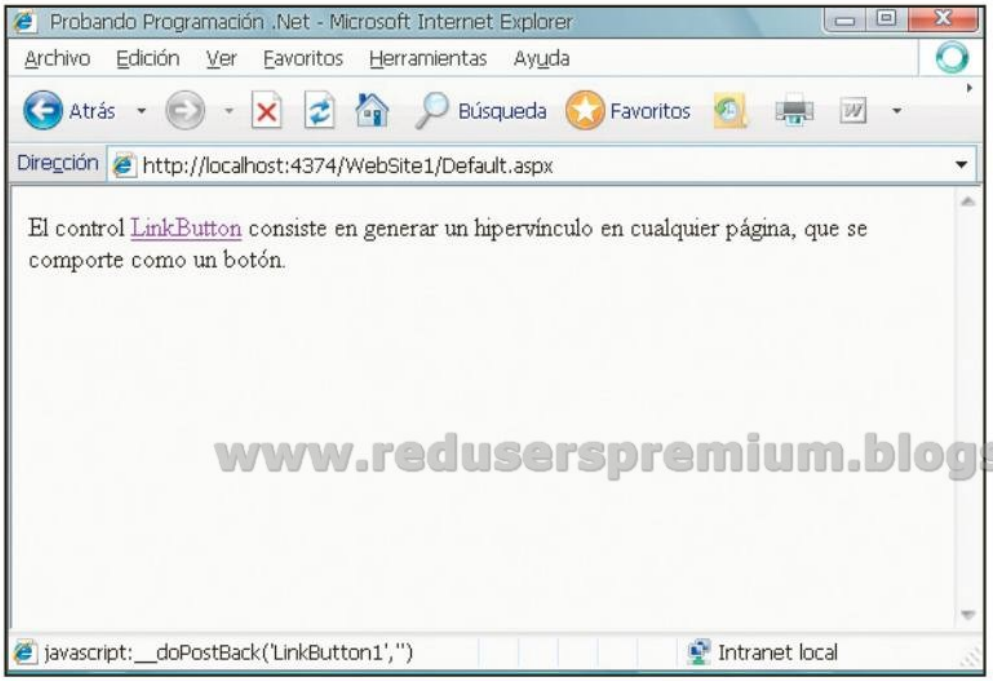


FIGURA 012 | El control LinkButton se ve como un hipervínculo, pero se usa como un botón.

```
        <asp:ListItem Text="Telefono movil"
        Value="2"></asp:ListItem>
</asp:ListBox>
```

Si necesitamos examinar la lista de objetos que contiene el control, podemos hacerlo a través de la propiedad **Items**, que contiene una lista de controles **ListItem**.
Para obtener todos los elementos seleccionados por el usuario, recorremos la colección **Items** y examinamos cada **ListItem** consultando la propiedad **SelectedValue**.
Al igual que el **DropDownList**, el control **ListBox** soporta *data binding*.
Si no utilizamos la opción de selección múltiple antes descripta, podemos consultar la propiedad **SelectedIndex** o **SelectedValue** para obtener la selección del usuario.

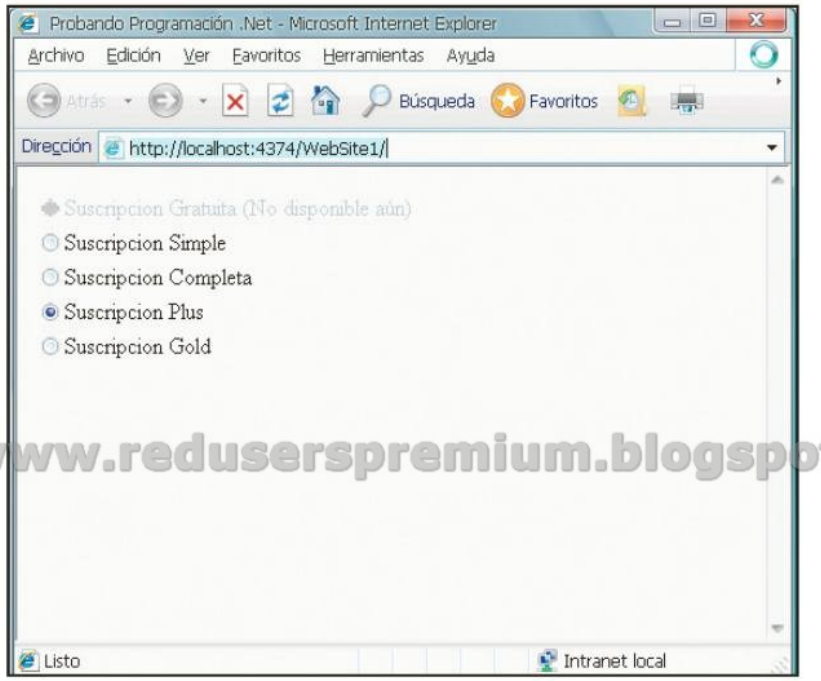
Control RadioButtonList

Representa un control que encapsula un grupo de botones de radio (correspondiente a la etiqueta **<input type="Radio" />** de HTML).
Ya que el control **RadioButtonList** soporta el enlace dinámico con una fuente de datos, es posible armar rápidamente un grupo de botones de radio de selección simple a partir de una fuente:

```
<asp:radiobuttonlist ID="Radiobuttonlist1"
runat="server">
      <asp:ListItem Text="Suscripcion Gratuita
      (No disponible aún)"
            Value="5" Enabled="false" />
            <asp:ListItem Text="Suscripcion
            Simple" Value="1" />
            <asp:ListItem Text="Suscripcion
            Completa" Value="2" />
            <asp:ListItem Text="Suscripcion
            Plus" Value="3" Selected="true" />
            <asp:ListItem Text="Suscripcion
            Gold" Value="4" />
        </asp:radiobuttonlist>
```

Un **RadioButtonList** es un contenedor de elementos **ListItem**. La colección de elementos está disponible a través de la propiedad **Items** del control. Para saber qué elemento fue

FIGURA 013 | La opción Enabled = "false" muestra el elemento desactivado.



Si necesitamos examinar la lista de objetos que contiene el control, podemos hacerlo a través de la propiedad Items.

seleccionado, consultamos la propiedad **SelectedItem**, que devuelve el elemento **ListItem** elegido. Luego, por medio de **Text** y **Value** de **ListItem** podemos obtener la descripción y el valor de la opción seleccionada.
Es posible modificar la forma en que se renderiza el control a través de las propiedades **RepeatLayout** (Table | Flow) y **RepeatDirection** (Horizontal | Vertical).

Los controles CheckBox y CheckBoxList

El control **CheckBox** permite al usuario seleccionar una condición entre dos valores posibles: verdadero o falso (true | false). Muestra una lista de controles **CheckBox** que puede enlazarse o vincularse a una fuente de datos. Cada uno de los elementos dentro de un CheckBoxList está representado por un elemento **ListItem**.

```
<asp:CheckBoxList ID="CheckBoxList1"
runat="server">
      <asp:ListItem Selected="True" Text=
      "Casado" Value="Casado"></asp:ListItem>
      <asp:ListItem Text="Casado"
      Value="Soltero"></asp:ListItem>
</asp:CheckBoxList>
```

Este control tiene una propiedad llamada **Checked**, cuyo valor es **true** si el control está seleccionado; y **false** en caso contrario. En caso de estar seleccionado, se renderiza "marcado".

El control CheckBoxList tiene una propiedad **Items**, que es una colección de elementos **ListItem**, y que puede ser recorrida para evaluar el valor de cada uno de ellos.

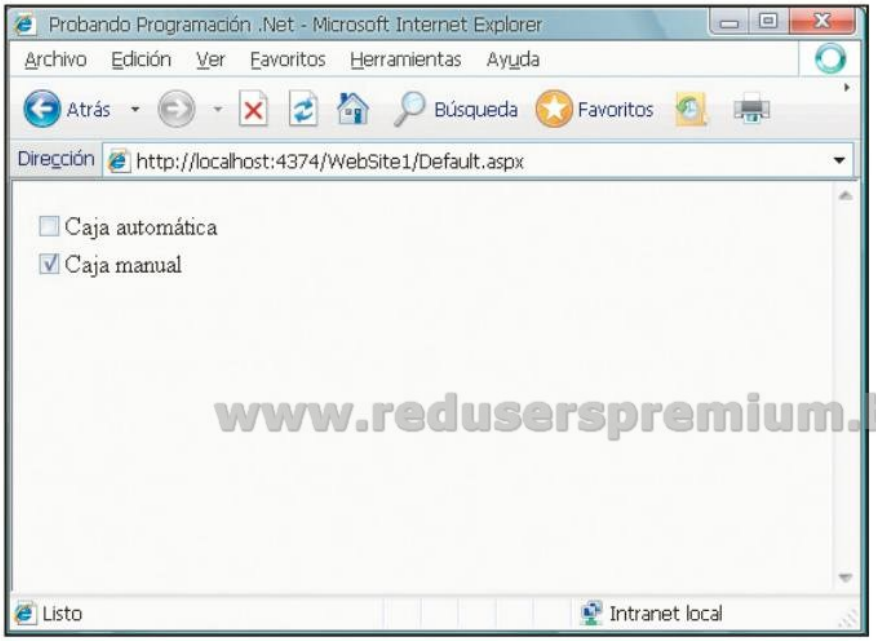


FIGURA 014 | La propiedad Checked en true renderiza el CheckBox marcado.

Controles de lista

En esta sección analizaremos controles que se utilizan para presentar al usuario opciones de selección.

ListItem

Cada opción dentro de un control de lista está modelada como un control ListItem.
Como los controles de lista presentan múltiples opciones de selección, cada uno tiene una colección de controles ListItem.
El control ListItem representa un elemento de un control de lista. Contiene una propiedad **text** para especificar el texto que se va a mostrar y una propiedad **value** para determinar el valor relacionado con el elemento.
En forma declarativa, podemos indicar el texto por mostrar y el valor relacionado, de la siguiente manera:

```
<asp:ListItem Text="Texto a mostrar"
Value="1" />
```

El control ListItem contiene la propiedad **selected**, que si está en **true**, redibuja el control en su forma seleccionada resaltando la opción respecto al resto, de acuerdo con el tipo de control que lo contenga. Es posible desactivar un control ListItem estableciendo la propiedad **enabled = false**. De esta manera, aparecerá junto al resto de los controles de selección, pero no podrá ser seleccionado.
Una colección de controles ListItem se utiliza dentro de los controles de selección. Esta colección puede ser enlazada contra una fuente de datos (lo que nosotros conocemos como base de datos). Más adelante veremos este tema; por ahora debemos recordar que la posibilidad de enlazar cualquiera de los siguientes controles contra una fuente de datos constituye una ventaja muy importante.

A continuación, analizaremos en detalle cada control de lista, y las operaciones básicas de llenado y selección.

Control DropDownList

Presenta una lista de selección simple. Es utilizado cuando se quiere presentar al usuario la posibilidad de elegir una opción de una lista posible de valores, cada uno de los cuales es especificado mediante un control ListItem.
El siguiente fragmento de código muestra cómo declararlo:

```
<asp:DropDownList ID="DropDownList1"
runat="server">
        <asp:ListItem Text="Mazzinger >"
        Value="1" />
        <asp:ListItem Text="Robotech"
        Value="2" />
        <asp:ListItem Text="Star Wars"
        Value="3" />
        <asp:ListItem Text="Transformers"
        Value="4" />
</asp:DropDownList>
```

Los controles de lista permiten a los usuarios seleccionar datos, como el país en el que se encuentran u otras opciones que indiquemos en el desarrollo.

FIGURA 015 | En esta imagen vemos cómo la lista es presentada utilizando el valor de la variable Text.

Veremos que ninguna de las opciones está seleccionada de antemano.

Cuando el usuario selecciona una opción, el control dispara un evento llamado **Selected IndexChanged**. La opción por defecto de este control no hace que la página se vuelva a cargar cuando se dispara el evento; sin embargo, es posible modificar el valor de la propiedad **AutoPostBack = true** a fin de obligar a la página a cargarse cuando una opción de la lista es seleccionada.

§ Uso de la propiedad SelectedItem

Cuando utilizamos la propiedad SelectedItem, obtendremos como respuesta el objeto seleccionado (que es del tipo ListItem).
Y como vimos anteriormente, un objeto ListItem contiene una propiedad text y otra value; por lo tanto, es posible acceder al valor seleccionado o al texto correspondiente a la selección mediante estas propiedades.

El siguiente fragmento muestra el mismo control, con el agregado de un manejador para el evento **SelectedIndexChanged**. La propiedad **AutoPostBack** se ha puesto en **true**:

```
<asp:DropDownList ID="DropDownList1"
runat="server"
AutoPostBack="true" OnSelectedIndexChanged=
"SelectedIndexChanged">
    <asp:ListItem Text="Mazzinger Z"
    Value="1" />
    <asp:ListItem Text="Robotech" Value="2" />
    <asp:ListItem Text="Star Wars" Value="3" />
    <asp:ListItem Text="Transformers"
    Value="4" />
</asp:DropDownList>
```

El manejador para el evento tiene la siguiente forma:

```
protected void SelectedIndexChanged(object
sender, EventArgs e) {

        string valorSeleccionado =
        DropDownList1.SelectedValue;

        // hacer algo
}
```

Consultando la propiedad **SelectedValue**, es posible obtener el valor asociado a la opción seleccionada.

```
protected void SelectedIndexChanged(object
sender, EventArgs e){
      ListItem li = DropDownList1.
      SelectedItem;
      string valor = li.Value;
      string texto = li.Text;

        // hacer algo
}
```

Argentina $8,90 (recargo al interior $0,20)) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net 7

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

Controles simples y de acción

Etiquetas - Botones - Hipervínculos

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Controles de lista

Listas desplegables - Botones de radio
Listas de verificación

ISBN 978-987-1347-43-8
9 789871 347438 00007