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

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

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

La mejor forma de aprender a programar desde cero

★★★★★ 10

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

Sitios web

Páginas maestras - Temas y skins
Navegación por el sitio

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Acceso a datos

Controles de datos de origen
DataBinding - Enlace a datos

ISBN 978-987-1347-43-8
00010
9 789871 347438

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

Páginas maestras

La mayoría de las aplicaciones Web tiene secciones que se repiten en todas las páginas. Veamos cómo facilitarnos este trabajo.

En las primeras versiones de ASP.NET, para tener esos contenidos comunes, debíamos colocaros a mano en todas las páginas, en general, utilizando controles de usuario. Obviamente, esta solución no era muy práctica, ya que teníamos que agregarlos en cada nueva página, y si hacíamos algún cambio radical en la estructura común de éstas, lo más probable era que tuviéramos que modificar todos los web forms de la aplicación. Master Pages es la opción que incluye ASP.NET 2.0 para facilitarnos el trabajo y evitar la repetición de datos entre las distintas páginas.

¿Qué son las páginas maestras?

La versión 2.0 de ASP.NET introdujo el concepto de página maestra, o *master page* en inglés. Se trata de plantillas en las que podemos definir el contenido común a todas las páginas, de manera sencilla y centralizada. Luego, cada página que utilice una master page heredará el contenido de ella y podrá agregar sus propios elementos visuales. De este modo, sólo necesitamos definir una única vez la estructura general de las páginas (el *layout*) y, luego, concentrarnos en el contenido particular de cada formulario. Además, cualquier cambio que hagamos a la página maestra llegará a las demás automáticamente.
Desde el punto de vista sintáctico, una página maestra es un archivo con extensión .master, que funciona de manera muy similar a los .aspx. Sin embargo, el texto del archivo master no comienza con la directiva @Page, sino con una directiva @Master. Además, la página maestra debe contener al menos un control de tipo ContentPlaceHolder (veremos este control en detalle más adelante). Por lo demás, master page es igual a un web form: posee su archivo de código subyacente (code behind), debe contener un tag Form con el atributo runat="Server", etc.

Creación de páginas maestras

Si bien podemos crear una página maestra sin utilizar Visual Studio, veremos cómo hacerlo con el asistente que nos provee el IDE, para simplificar el trabajo. Para hacerlo con Visual Studio, utilizamos la opción Agregar/Nuevo elemento, del menú Sitio Web, y seleccionamos Página Principal.

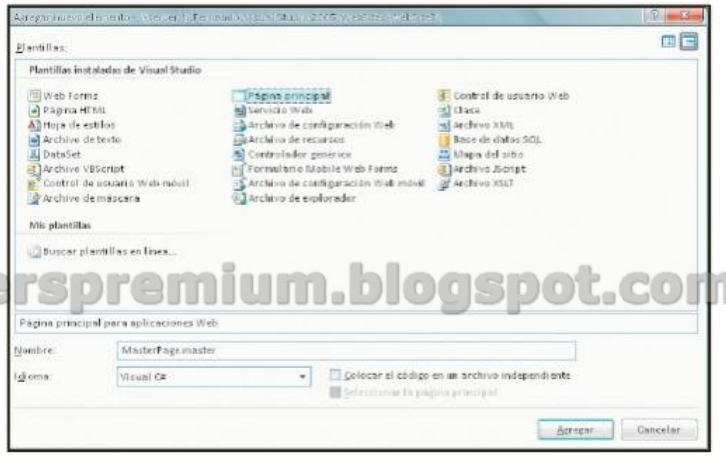


FIGURA 019 | Visual Studio incluye una plantilla para crear páginas maestras.

Al aceptar, Visual Studio creará el archivo .master y lo agregará al sitio que estamos desarrollando. Si exploramos un poco el archivo de code behind (.master.cs o .master.vb, según el lenguaje con el que trabajemos), veremos que la clase no hereda de System.Web.UI.Page sino de System.Web.UI.MasterPage. La clase MasterPage no tiene tantos atributos y métodos como Page, lo cual tiene sentido, porque en el code behind de la página maestra, no deberíamos colocar mucha lógica, sólo aquella relacionada con la manipulación del contenido común.
Si miramos el código de la master page, veremos que comienza con la directiva @Page:

```
<%@ Master Language="C#" CodeFile="Pagina
Maestra. master.cs" Inherits="PaginaMaestra" %>
```

Como podemos notar, al igual que los web forms, tiene una referencia a su archivo de code behind y a la clase de la que heredará el código generado al momento de compilar.
Un poco más abajo en el mismo archivo, vemos que el código es casi idéntico al de un web form, a excepción del control ContentPlaceHolder, que es el lugar donde colocaremos el contenido específico de cada página. Una página maestra debe tener por lo menos un ContentPlaceHolder, y hasta todos los que precisemos. Como el ContentPlaceHolder es el espacio para colocar los controles de la página, necesariamente debe estar dentro del tag form con el atributo runat="Server". Todo el código HTML y los controles web que coloquemos alrededor del ContentPlaceHolder serán repetidos en cada web form que utilice la página maestra.

Si pasamos a la vista de diseño, veremos que el ContentPlaceHolder se dibuja como un rectángulo con su ID a modo de título, y fuera de él podremos arrastrar controles.

Un ejemplo simple

Para ver las páginas maestras en funcionamiento, desarrollaremos un simple ejemplo. Siguiendo con la master page que creamos al principio, vamos a agregar un título sobre el ContentPlaceHolder y un pie de página debajo. El código de la página maestra nos quedará así:

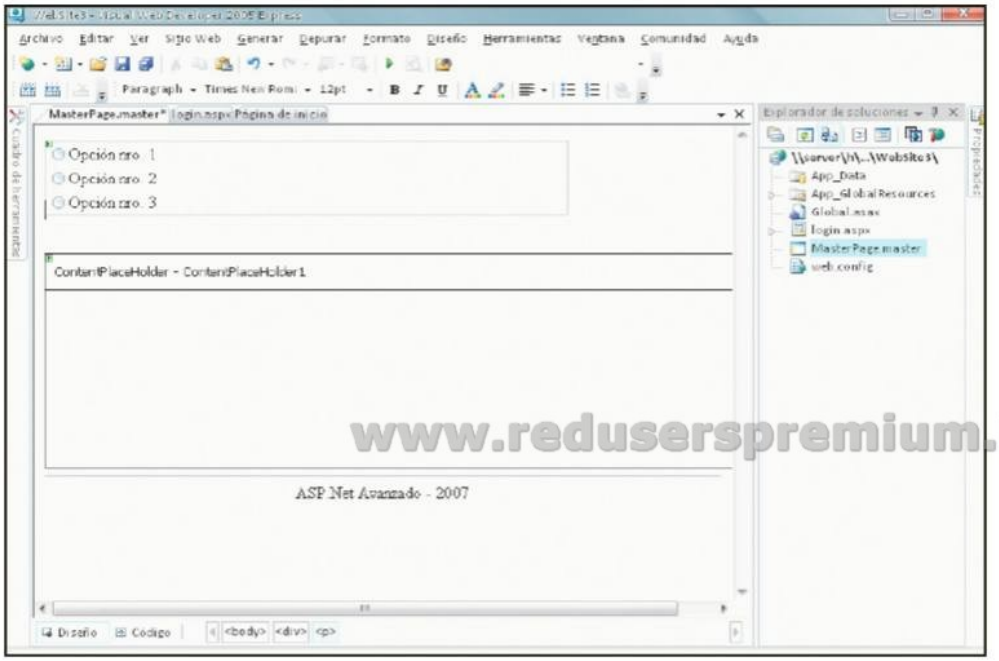


FIGURA 020 | En el diseñador de la master page podemos colocar también controles fuera de ContentPlaceHolder.

EN UN MISMO PROYECTO O SITIO WEB, PODEMOS TENER TODAS LAS PÁGINAS MAESTRAS QUE NECESITEMOS; INCLUSO, ES POSIBLE CAMBIAR LA PÁGINA MAESTRA DE UN FORMULARIO DURANTE LA EJECUCIÓN DE LA APLICACIÓN.

```
<%@ Master Language="C#" AutoEventWireup
="true" CodeFile="PaginaMaestra.master.cs"
Inherits="PaginaMaestra" %>

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

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>Aprendiendo a usar páginas
        maestras</h1>
        <asp:contentplaceholder id="Content
        PlaceHolder1" runat="server">
        </asp:contentplaceholder>
        <hr size="1" noshade />
        <center>ASP.Net Avanzado - 2007</center>
    </div>
    </form>
</body>
</html>
```

Ya tenemos creada una página maestra; ahora debemos hacer que nuestros formularios la utilicen. Para no complicarnos modificando un formulario ya hecho, vamos a crear uno nuevo, diciendo que queremos usar la master page. Utilizamos otra vez la opción Add New Item, del menú Web Site, y seleccionamos el template WebForm, pero prestando atención al checkbox que está en la parte inferior de la ventana, con el texto Select Master Page.

Al hacerlo, se abrirá una segunda ventana que no hemos visto aún. En ella aparece la lista de todas las páginas maestras que tenemos en el proyecto, y se nos da la posibilidad de seleccionar la que queremos utilizar para el nuevo formulario. En nuestro caso, como tenemos una sola, la seleccionamos directamente.

Cuando hayamos aceptado, el IDE creará el web form junto con su archivo code behind. Si miramos el código del nuevo formulario, veremos que no se parece en nada a los que

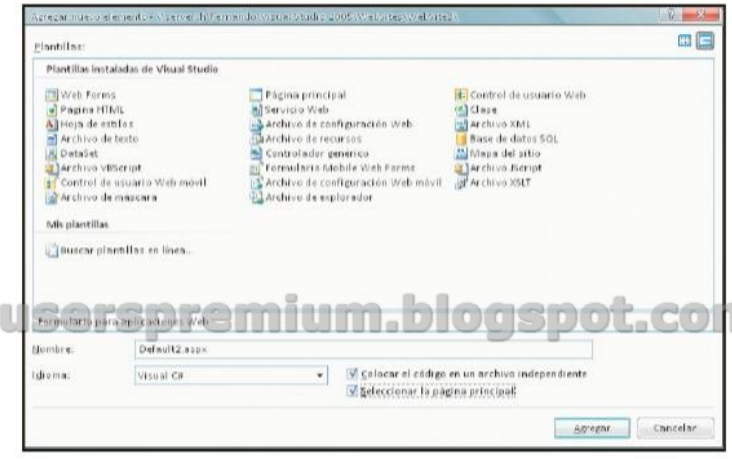


FIGURA 021 | Al crear un nuevo formulario, podemos indicar que queremos seleccionar la master page.

VISUAL STUDIO 2005 NO SOPORTA EL DISEÑO VISUAL DE MASTER PAGES ANIDADAS. TENDREMOS QUE ESPERAR A VISUAL STUDIO 2008 PARA PODER EDITAR VISUALMENTE LAS PÁGINAS MAESTRAS QUE USAN OTRA PÁGINA MAESTRA.

ya conocemos. El código que Visual Studio creó es el siguiente:

```
<%@ Page Language="C#" MasterPageFile=
"~/PaginaMaestra.master" AutoEventWireup
="true" CodeFile="UsoMasterPage.aspx.cs"
Inherits= "UsoMasterPage" Title="Untitled
Page" %>

<asp:Content ID="Content1" ContentPlace
HolderID= "ContentPlaceHolder1" Runat=
"Server">
</asp:Content>
```

Analicemos un poco esta porción de código. Comienza con la directiva @Page como cualquier otro formulario web, pero posee un atributo que no conocíamos: MasterPageFile. Como ya se imaginarán, éste le indica a ASP.NET cuál es la página maestra que se va a utilizar. Los demás atributos de la directiva @Page son los mismos que podemos encontrar en cualquier otro web form. Luego, hay un control del lado del servidor llamado Content. Desde el punto de vista del diseño, éste funciona como un panel donde colocaremos el contenido de la página en la que estamos trabajando. Si observamos los atributos del control Content, veremos que uno de ellos es ContentPlaceHolderID. Este atributo hace referencia al control ContentPlaceHolder de la página maestra. ¿Cómo funciona

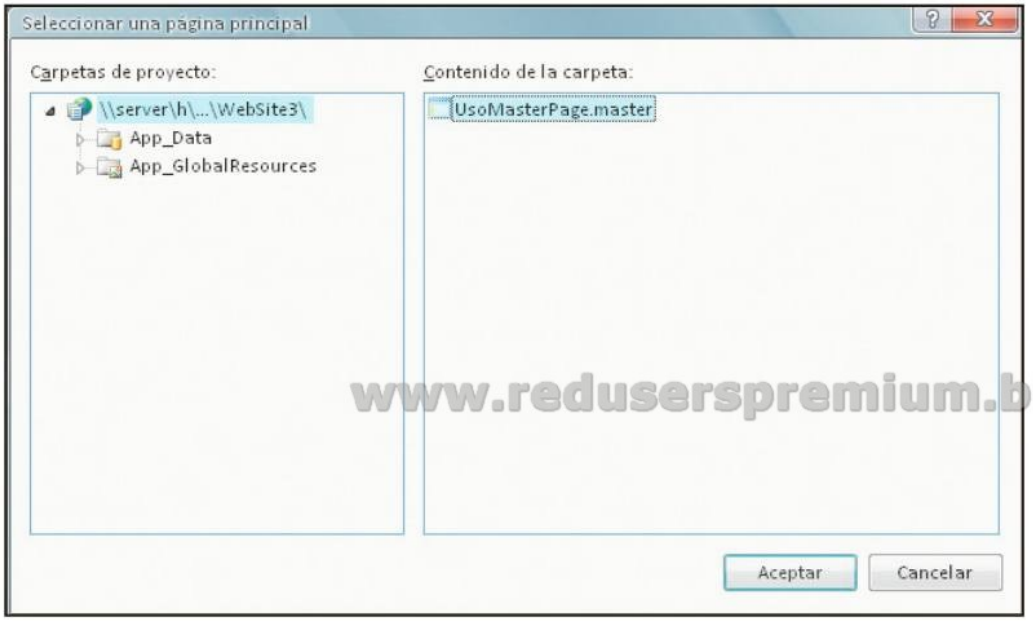


FIGURA 022 | En esta ventana podemos seleccionar la master page que vamos a usar en el nuevo formulario.

esto? Bien, como adelantamos al principio, cuando la página sea renderizada, todo lo que esté dentro de Content será colocado en el lugar del ContentPlaceHolder de la página maestra que tenga el ID indicado.
Siguiendo con el ejemplo, coloquemos un poco de HTML y algún web control dentro del control Content:

```
<asp:Content ID="Content1" ContentPlaceHolder
ID= "ContentPlaceHolder1" Runat="Server">
Este es el contenido propio de la página <br />
<asp:Button runat="server" ID="boton"
Text="Botón de Prueba" />
</asp:Content>
```

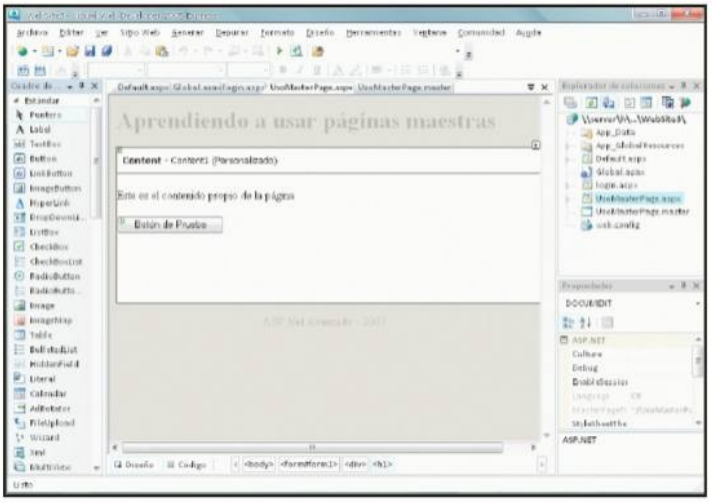
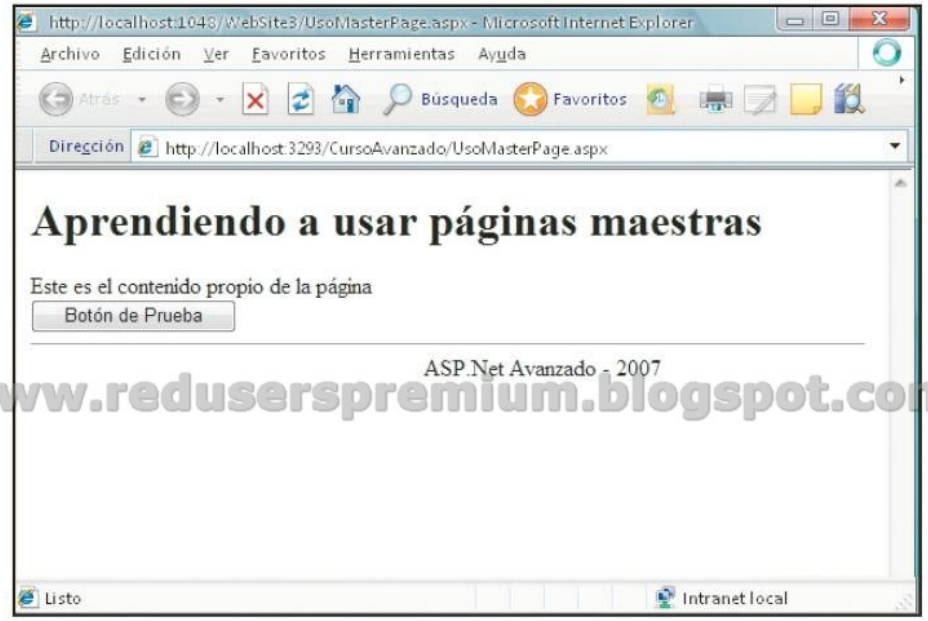


FIGURA 023 | Visual Studio nos muestra el contenido de la página maestra junto con el de la que estamos creando.

Ahora pasemos a la vista de diseño y veremos algo interesante. Aparecerá el contenido que creamos en la master page y el contenido que acabamos de generar, pero con la salvedad de que el de la página maestra aparece como deshabilitado y no podemos modificarlo. Ésta es una característica de Visual Studio que nos permite previsualizar cómo quedará la página completa cuando sea renderizada (Figura 23).

Vamos a seleccionar el nuevo formulario como página de inicio (haciendo clic derecho sobre el archivo en el explorador de soluciones y luego eligiendo Establecer como página de inicio) y a ejecutar la aplicación. Cuando la página termine de ejecutarse y llegue al navegador, veremos que tiene un título en la parte superior, un texto y un botón en la parte central, y un texto separado con una línea en la parte inferior.

FIGURA 024 | Cuando ejecutamos la aplicación, el contenido de la página maestra se combina con el del formulario.



Temas y skins

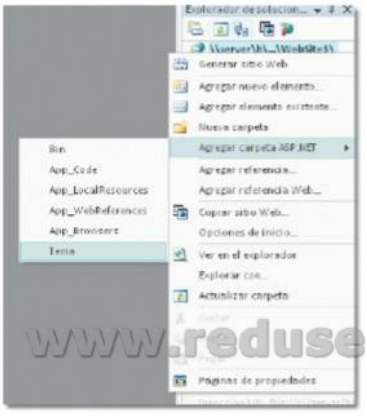
Veremos cómo aprovechar todo el potencial de ASP.NET 2.0 para lograr aplicaciones con una buena experiencia de usuario.

Temas

Los temas se encargan de agrupar skins, hojas de estilo en cascada (css), imágenes y otros recursos, ubicados en subcarpetas de una carpeta especial del sitio llamada /App_Themes. Los temas permiten obtener distintas configuraciones visuales para un mismo sitio, que pueden intercambiarse cuando la aplicación ya está lista, sin modificar nada en los formularios (incluso, podemos cambiar el tema dinámicamente por código). Para crear un tema, sólo debemos hacer clic con el mouse sobre la carpeta de proyectos, en la ventana Explorador de soluciones, y del menú emergente que se despliega, elegir la opción Agregar carpeta ASP.NET/Tema y darle el nombre que queramos, por ejemplo, Principal. Si aún no tenemos creada la carpeta App_Themes, Visual Studio nos lo advertirá y pedirá confirmación para generarla. Al aceptar, será creada, y dentro de ella habrá una carpeta con el nombre que le hayamos dado al tema. Una vez que generamos los temas, le indicamos a ASP.NET cuál de ellos utilizar. Para hacerlo, tenemos dos alternativas: indicarlo en cada formulario, mediante un

Guía Visual 001 | Agregar un nuevo tema

1) Desplegamos el menú contextual del mouse sobre la carpeta del proyecto ubicada en el Explorador de Soluciones. Seleccionamos del menú la opción Agregar carpeta ASP.NET, y del submenú que se despliega, Tema.



2) Siempre es conveniente darle un nombre distintivo al tema, para poder identificarlo mejor si es que nuestra aplicación va a disponer de varios temas distintos.

atributo de le directiva @Page; o en el archivo web.config para todas las páginas. A esta altura del curso, el lector ya sabrá que indicar el nombre del tema en cada página nos limita mucho a la hora de cambiarlo, porque deberemos hacerlo en todos los formularios. La mejor opción, entonces, es señalarlo en el archivo de configuración. Para establecer el nombre del tema que queremos usar en las páginas, debemos utilizar el atributo **theme** del elemento **pages**, del archivo web.config:

```
<system.web>

    <pages theme="Principal">
    </pages>

</system.web>
```

Creación de temas

Para entender mejor cómo funcionan los temas y dónde radica su fortaleza, hagamos un pequeño ejercicio. Tal como dijimos anteriormente, creamos dos temas, llamados Principal y Blanco, y dentro de cada uno generamos una carpeta denominada Imágenes, en la que colocaremos una cualquiera, en lo posible, con dimensiones iguales, aunque no es importante para el ejemplo. Lo que sí es importante para ver los temas en acción es que las imágenes sean diferentes. El explorador de soluciones se verá como se muestra en la Figura 26. Luego, definimos una skin para cada uno de los temas (ya veremos luego con mayor detalle qué son las skins y cómo las definimos), y en cada una colocamos el siguiente texto:

```
<asp:Image runat="server" ImageUrl="Imagenes/
Encabezado.gif" SkinID="Encabezado" />
```

Aquí, Imágenes/Encabezado.gif hace referencia a la imagen que agregamos dentro de la carpeta correspondiente. Notemos que no estamos yendo a la raíz del sitio, sino que partimos de la carpeta del tema; esto es muy importante y es, en parte, lo que nos ayudará a crear temas totalmente independientes unos de otros.

> Es posible intercambiar los temas cuando la aplicación ya está lista, sin modificar nada en los formularios.

Uso de temas

Ahora que ya tenemos nuestros temas creados, vamos a agregar un nuevo formulario Web al sitio y a colocar una imagen en él:

```
<asp:Image SkinID="Encabezado" runat="server" />
```

Notemos, en particular, el atributo SkinID, con el valor Encabezado. Si volvemos a mirar el código de ambas skins, veremos que dicho valor claramente hace referencia al que definimos dentro de las pieles. Sólo resta configurar el tema por usar en el archivo web.config y ejecutar la aplicación. Si todo sale bien,

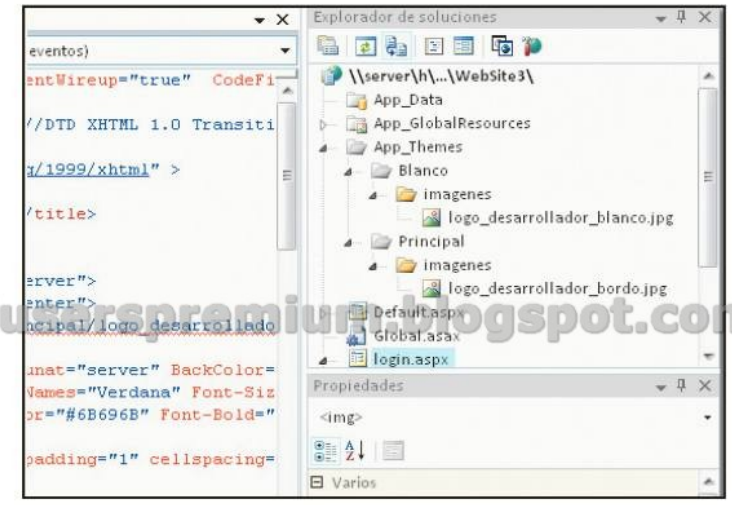


FIGURA 025 | Nuestro sitio con dos temas y las respectivas imágenes.

veremos una de las imágenes que agregamos al tema. Ahora, sin cerrar el navegador de Internet, cambienos Principal por Blanco, en el archivo de configuración, y refrescamos la página. Veremos que se muestra la otra imagen, con lo cual comprobamos que, efectivamente, estamos utilizando otro tema para la aplicación. Como podemos notar, los temas nos permiten crear diferentes estilos visuales y, luego, utilizar el que queramos. Incluso, podemos cambiar de tema mediante código y, de ese modo, permitir que los usuarios elijan el que más les guste.

Skins

Las skins (pieles) son simples archivos con extensión .skin que contienen la definición de propiedades para controles ASP.NET del lado del servidor. El código es similar al que se escribe en los archivos .aspx. Por ejemplo, veamos la siguiente porción de una skin:

```
<asp:GridView runat="server"
  BackColor="LightGoldenrodYellow"
  BorderColor="Tan"
  BorderWidth="1px"
  CellPadding="2"
  ForeColor="Black"
  GridLines="None">
  <FooterStyle BackColor="Tan" />
  <SelectedRowStyle BackColor=
  "DarkSlateBlue" ForeColor="GhostWhite" />
  <PagerStyle BackColor="PaleGoldenrod"
  ForeColor="DarkSlateBlue"
  HorizontalAlign="Center" />
  <HeaderStyle BackColor="Tan"
  Font-Bold="True" />
  <AlternatingRowStyle
  BackColor="PaleGoldenrod" />
</asp:GridView>
```

En el ejemplo, estamos definiendo un conjunto de propiedades para una grilla (GridView). Como vemos, el código es casi igual

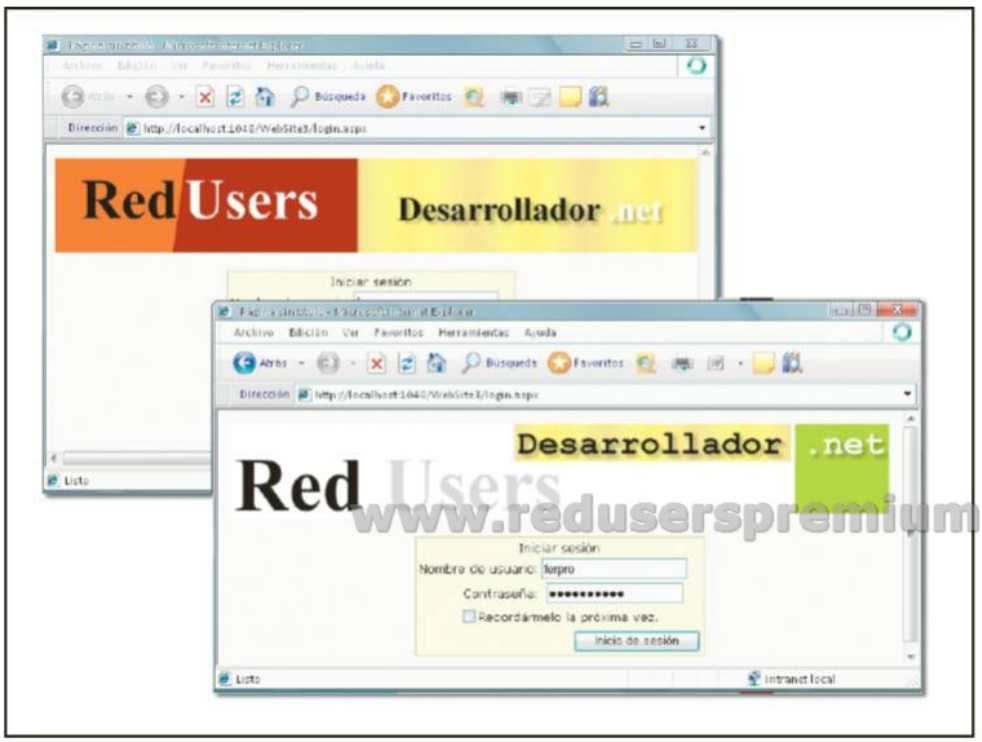


FIGURA 026 | La misma aplicación corriendo con dos temas diferentes.

al que usamos cuando colocamos una grilla en una página, a excepción de que aquí no tenemos el atributo ID. En una skin podemos definir las propiedades de más de un control. Del mismo modo, en un mismo tema podemos tener más de un archivo de skin.
Cuando tenemos una skin en nuestro sitio o aplicación Web y agregamos un nuevo control, ASP.NET asociará automáticamente el control con las propiedades definidas en la skin para ese tipo de control, y nos ahorraremos volver a definir todas las propiedades por cada control. Siguiendo con el ejemplo, si agregamos el control GridView a una página, ésta automáticamente tendrá la apariencia definida mediante las configuraciones del archivo .skin. Por medio de las skins, logramos que todos los controles del mismo tipo tengan igual apariencia y propiedades. Esto puede hacer pensar que las skins reemplazan a las hojas de estilos en cascada (CSS), pero no es así, sino que ambos elementos se complementan. Veamos un poco cómo es este tema. En las hojas de estilo, definimos atributos visuales para los elementos HTML (recordemos que los web controls terminan generando código HTML), y para que un control se vea con el estilo definido en la hoja CSS, debemos relacionar la clase con él, en general, mediante una propiedad llamada **CssClass**.
Por otro lado, las pieles permiten establecer propiedades que son particulares de los controles Web (como grillas o calendarios) y que no necesariamente son visuales; por ejemplo, podemos definir la apariencia del paginador de una grilla, algo que no podemos hacer con CSS. Veamos cómo combinar los estilos en cascada, suponiendo que tenemos un archivo de estilos llamado styles.css, con las siguientes clases:

```
.grilla
{
  background-color: #EFEFEF;
  color: Black;
}
```

```
.grilla_fila_alternativa
{
  background-color: White;
}
```

Por otro lado, en el archivo .skin, podemos tener definidas las propiedades para los objetos de tipo GridView, de esta manera:

```
<asp:GridView runat="server" BorderColor="Tan"
BorderWidth="1px" CellPadding="2" CssClass=
"grilla" GridLines="None">
    <HeaderStyle BackColor="Tan"
    Font-Bold="True" />
    <AlternatingRowStyle CssClass="grilla_fila_
    alternativa" />
</asp:GridView>
```

Como podemos observar, estamos utilizando las clases de la hoja de estilos para definir atributos visuales en la skin, que, luego, serán heredados por todos los controles de tipo GridView.
En un tema, podemos tener más de una skin definida para un mismo tipo de control. En estos casos, podemos identificar cada una con un ID, mediante el atributo SkinID. Podemos asociar un control con una skin en particular, relacionando la propiedad SkinID de ésta con la del control.

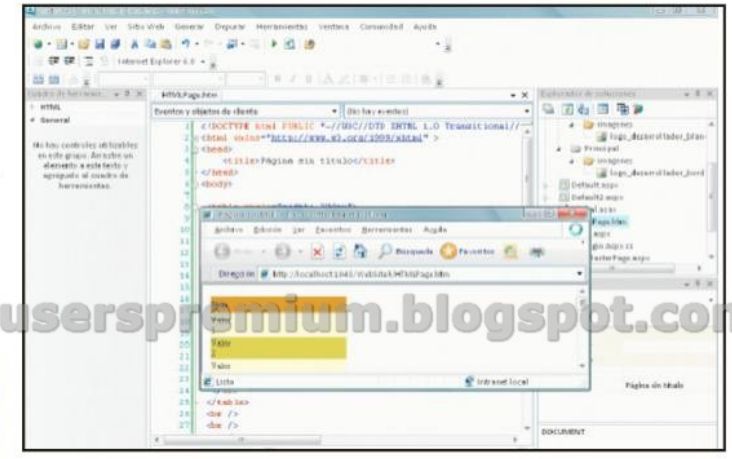


FIGURA 027 | Teniendo una skin, al agregar un nuevo control, éste recibe automáticamente todas las propiedades definidas en ella.

Navegación por el sitio

Los controles de navegación determinarán la experiencia del usuario al recorrer nuestra aplicación Web.

Consideraciones

Una de las principales características de todo sitio de Internet –ya sea de un portal, una pequeña página institucional o un completo sistema corporativo– es la navegación o flujo de interacción entre el usuario y las distintas secciones/pantallas/módulos que lo componen. Como desarrolladores, nuestro principal objetivo será dar al usuario todas las alternativas posibles para que éste llegue a destino, con el menor número posible de clics, pero sin sobrecargar nuestra pantalla con vínculos o menús innecesarios.

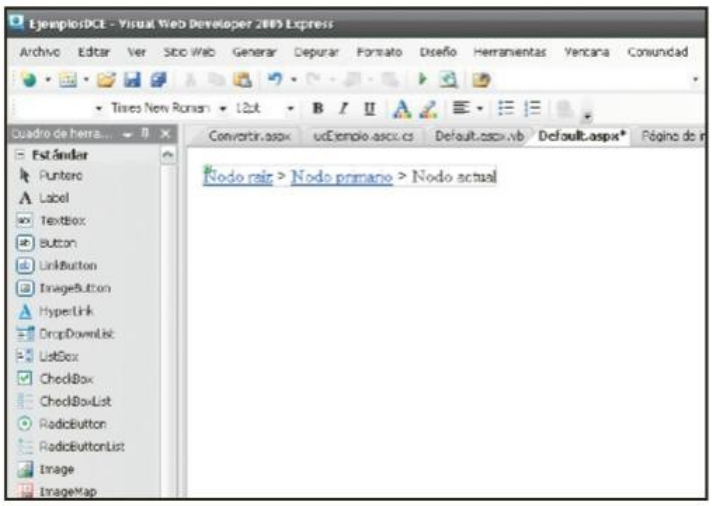


FIGURA 028 | En este caso vemos un control SiteMapPath, con su configuración por defecto.

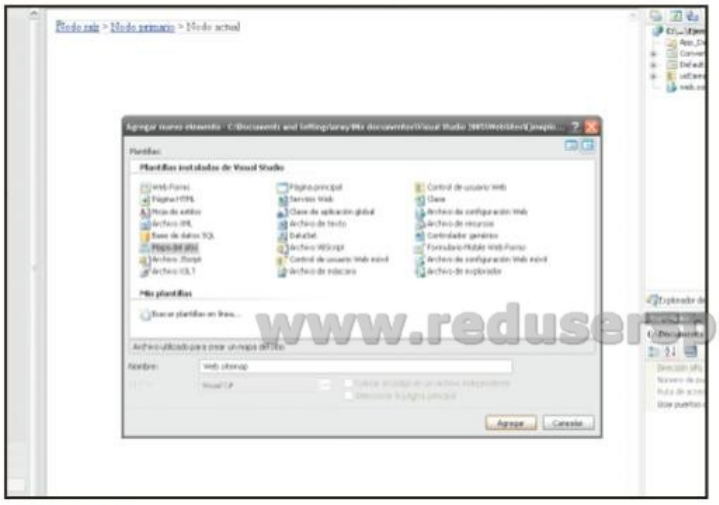


FIGURA 029 | En el asistente seleccionamos la opción Mapa del sitio y hacemos clic en Agregar.

El control SiteMapPath

Este control soluciona de manera automática los problemas relacionados con la navegación en niveles, y permite al usuario saber exactamente dónde se encuentra parado dentro del sitio y en qué "nivel" o profundidad está situado.

Características

Para esto, SiteMapPath nos presenta un pequeño cuadro dinámico donde se mostrará, según corresponda al nivel, el título de la página actual y todas sus "progenitoras", separadas mediante una imagen o símbolo personalizable, como vemos en la Figura 29.

Bendito XML

Al tratarse de una estructura de tipo árbol, el uso de un archivo XML para la configuración del SiteMapPath es la opción más práctica. Este archivo, llamado por lo general Web.sitemap, puede incorporarse a mano, desde la pantalla Agregar/Nuevo Elemento, seleccionada del menú contextual en la ventana Explorador de Soluciones, como observamos en la Figura 30.

Una vez hecho esto, se mostrará automáticamente el archivo en el editor de código, para completarlo con la información necesaria. El siguiente fragmento de código representa el mapa de un sitio básico, con sólo dos niveles de profundidad:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.
com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="default.aspx" title=""
  description="Sitio de Prueba Sitemap">
    <siteMapNode url="pagina1.aspx"
    title=""description="Pagina de
    Prueba 1" />
    <siteMapNode url="pagina2.aspx"
    title=""description="Pagina de
    Prueba 2" />
  </siteMapNode>
</siteMap>
```

El enlace al control se realiza de manera automática, pero debemos comprobar que los nombres de cada página en el archivo Web.sitemap estén correctamente escritos. Debemos verificar minúsculas y mayúsculas, para ver en funcionamiento el control SiteMapPath.

Utilizando menús

La implementación de un menú para nuestro sitio podía ser, en muchos casos, una tarea compleja y difícil de mantener. En versiones anteriores, los llamados menús debían ser programados a mano en cada página o, en su defecto, dispuestos en forma de tabla dentro de un conjunto de marcos o FrameSet. Con ASP.NET 2.0 y el control Menú, podemos crear un menú dinámico altamente configurable, en tres sencillos pasos, que desarrollaremos a continuación.

El control SiteMapPath posee enlace automático con el archivo Web.sitemap.

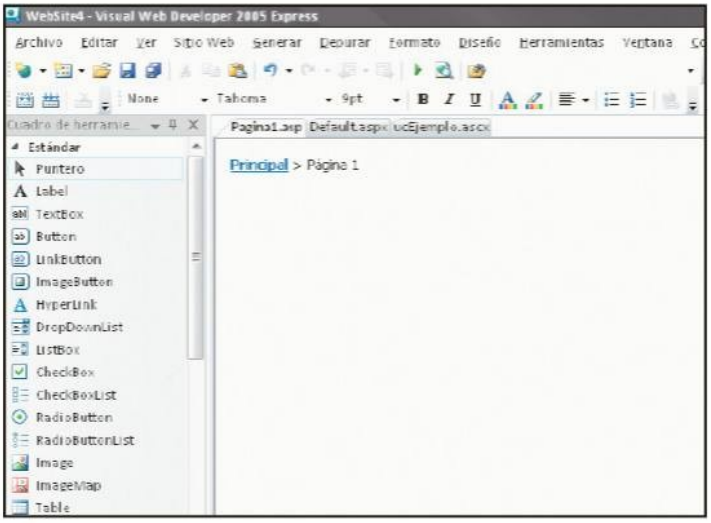


FIGURA 030 | Posicionados en la Página 1, nuestro SiteMapPath nos muestra el camino recorrido, con un link a la página principal.

Introducción del Menú en el formulario

Para utilizar el control, arrastramos a nuestro formulario un objeto Menú, de la solapa Exploración (en el cuadro de herramientas).

ⓘ Dirección de SiteMapPath

La propiedad PathDirection del control SiteMapPath nos permite configurar desde qué perspectiva se mostrarán los vínculos en él. Por defecto, esta propiedad viene establecida en RootToCurrent, es decir que desde el nodo padre del árbol se llega hasta la página actual. En contrapartida, la opción CurrentToRoot mostrará los vínculos de manera inversa.

Todo el JavaScript necesario para el funcionamiento dinámico del menú es generado automáticamente en tiempo de ejecución.

Al hacerlo, se desplegarán las tareas relacionadas con el control Menú, donde podremos configurar el origen de datos (puede ser una base de datos o un archivo XML), el tipo de vista dinámica o estática, y añadir/quitar elementos.

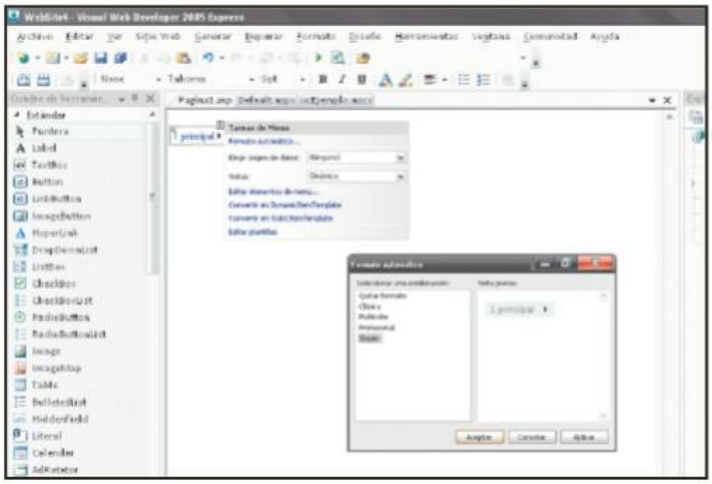


FIGURA 031 | El formato automático es una alternativa para brindar estilo a nuestro menú.

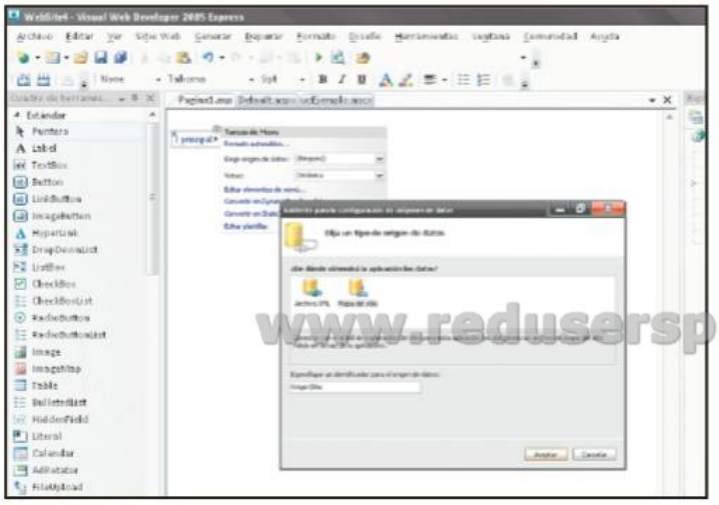


FIGURA 032 | Configuramos un nuevo origen de datos, llamado OrigenSitio. Estará basado en el archivo Web.sitemap.

El origen de datos

Al seleccionar el elemento Nuevo origen de datos, desde el menú contextual Elegir origen de datos, se presenta un asistente de configuración en donde podemos optar por un archivo XML cualquiera; es decir, un archivo propio o el mismo Web.sitemap utilizado para nuestro control SiteMapPath. Al crear el origen de datos a partir de nuestro Web.sitemap, el control Menú se mostrará automáticamente en el nodo padre de dicho documento XML. De la misma manera en que sucedió con SiteMapPath, el correcto funcionamiento del control Menú dependerá de la correlatividad del archivo Web.sitemap con la realidad.

Enlazando a XML

En el ejemplo utilizamos un objeto de tipo SiteMapDataSource para enlazar nuestro menú con el archivo Web.sitemap. Pero como ya mencionamos, es posible emplear cualquier archivo con la estructura correcta y en formato XML para hacerlo.
Si seleccionamos esta opción del asistente, será necesario especificar el archivo XML que contiene los datos para enlazar.

Treeview

El control Treeview es una conjunción de capacidades del SiteMapPath y el menú. Nos permite mostrar un hipervínculo a cada una de las páginas en nuestra aplicación Web y, a la vez, presentar de manera jerárquica en cuál de ellas estamos actualmente posicionados.

Al igual que el menú, Treeview posee comportamiento dinámico en JavaScript, completamente autogenerado y personalizable en todos sus aspectos.

DataBinding

El control Treeview soporta las mismas fuentes de datos que el menú, que, como ya mencionamos, son un archivo XML propio o la representación del mapa del sitio: Web.sitemap.

Consideraciones finales

Todos los controles aquí mencionados son complementos ideales para trabajar de manera óptima con Master Pages. Si realizamos una buena combinación de páginas maestras y controles de navegación, tal como lo fuimos desarrollando en este capítulo, lograremos solucionar los problemas más frecuentes de exploración del sitio y mejorar además notablemente la experiencia de navegación del usuario.

Algunos estilos prediseñados del control Treeview nos recuerdan al sitio de MSDN, a la lista de correos de Outlook o al mismo mensajero instantáneo, MSN Messenger.

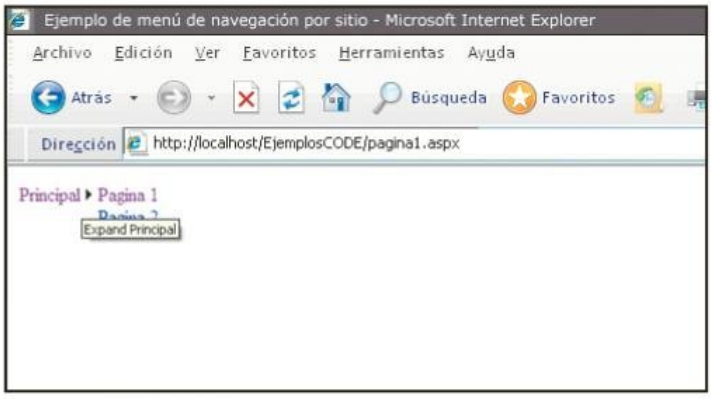
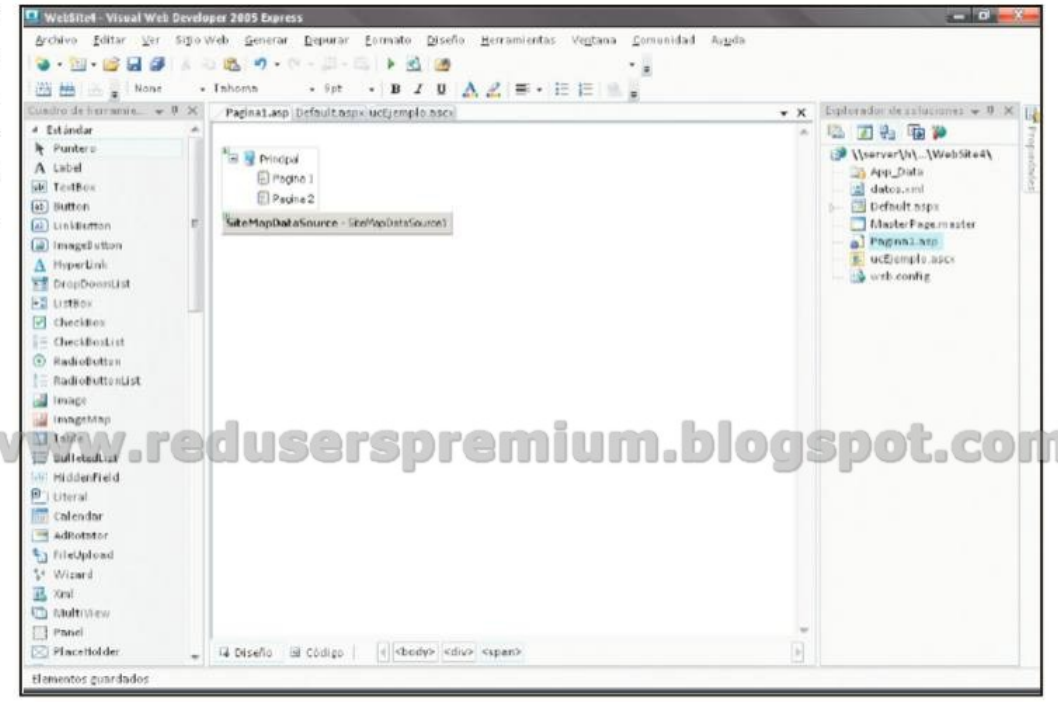


FIGURA 033 | El menú dinámico se despliega en el evento MouseOver, mediante JavaScript.

FIGURA 034 | Treeview configurado con el archivo Sitemap. Cabe destacar la variedad de formatos rápidos disponibles.



Acceso a datos

Veamos qué herramientas nos brinda ASP.NET para mostrar la información de una base en una página Web.

Data binding

Podemos traducir data binding como enlace a datos, para hacer referencia a la capacidad que tienen ciertos controles de ASP.NET de asociar alguna de sus propiedades con un origen de datos, como un campo de una tabla de un DataSet o un objeto de la capa de negocios. El enlace a datos nos permite simplificar gran parte del trabajo a la hora de mostrar información proveniente de una base en las páginas del sitio Web. Por ejemplo, en ASP 3.0 o ASP "tradicional", si queríamos mostrar una lista desplegable con valores tomados de una tabla de la base de datos, debíamos recorrer la tabla y, por cada registro, generar los tags **option** para renderizar el contenido de la manera correcta. Mediante data binding, simplemente indicamos cuál es la tabla, y ASP.NET se encarga de recorrerla y de generar los tags HTML por nosotros, lo cual representa una ventaja más que considerable.

Cómo funciona el DataBinding

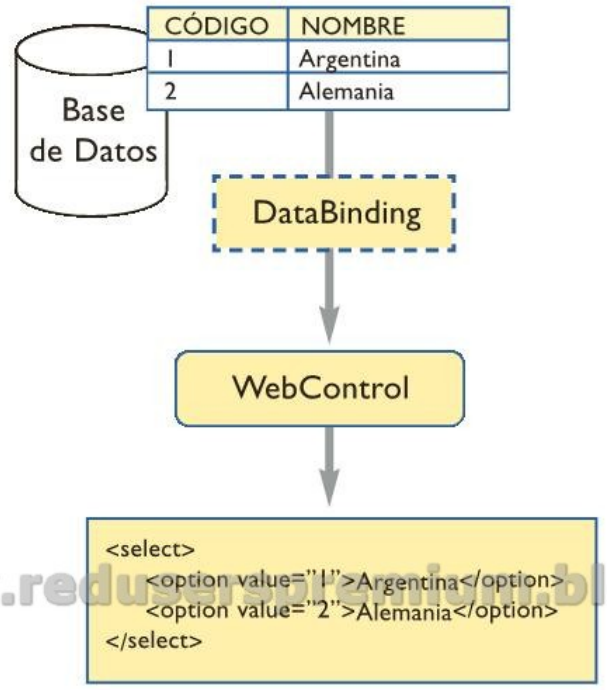


FIGURA 035 | El data binding nos permite enlazar registros de una base de datos con controles Web, lo cual facilita la generación de código HTML que refleje la información de la base.

ASP.NET 2 INCORPORÓ LA CAPACIDAD DE HACER ENLACE A DATOS BIDIRECCIONAL. ANTES MOSTRÁBAMOS DATOS DESDE LOS VALORES DE LOS CAMPOS, PERO AHORA PODEMOS ACTUALIZAR LA INFORMACIÓN DE LA BASE DE DATOS DESDE LAS PROPIEDADES DE LOS CONTROLES.

Data binding sencillo

Visual Studio permite hacer el enlace a datos tanto por código como de manera visual utilizando un asistente. A los efectos de entender mejor el funcionamiento, veremos primero cómo hacer data binding mediante código, ya que los asistentes suelen ocultarnos un poco el código que hay detrás. Luego veremos algunos controles de ASP.NET 2.0 que permiten hacer el enlace visualmente, pero de una manera mucho más ordenada y clara.

Sin importar el tipo de control que vamos a enlazar a datos, el proceso es siempre el mismo, y consta de dos pasos bien definidos: primero, establecer la propiedad DataSource (origen de datos) del control y, luego, llamar al método DataBind, también del control. La invocación a este método es fundamental porque es el encargado de concretar el enlace entre los datos y el control. Hagamos un breve ejercicio para empezar a ver data binding en acción. Vamos a crear un nuevo proyecto desde Visual Web Developer Studio 2005, y le colocaremos un control de tipo DropdownList:

```
<asp:DropDownList runat="server"
ID="clientes"></asp:DropDownList>
```

Para simplificar, vamos a asumir que tenemos un método llamado ObtenerClientes, que devuelve un DataTable con la lista de clientes tomada de la base de datos (para practicar, el lector puede hacer todos los pasos, desde crear la base, hasta llenar el DataTable mediante ADO.NET). Asumamos también que la tabla tiene la estructura que se observa en la Figura 37.

DropDownList

CÓDIGO	NOMBRE
1	Juan Perez
2	Gomez & Gomez S.A.
3	Kiosco "Novedades"
...	...

FIGURA 036 | Estructura de la tabla que vamos a usar para enlazar al DropDownList.

El control DropDownList renderiza un tag HTML llamado "<select>" junto con los correspondientes tags option. Éstos tienen dos atributos fundamentales: uno es el texto por mostrar en la lista, y el otro es un valor, que no se muestra pero que nos permitirá identificar unívocamente cada uno de los elementos que la integran.

Por ejemplo, el siguiente código HTML produce una lista desplegable de tres elementos:

```
<select>
   <option value="1">Bariloche</option>
   <option value="2">Alpachiri</option>
   <option value="3">Bahía Blanca</option>
</select>
```

Cuando se abra la página que contiene esta porción de HTML en el navegador, veremos una lista desplegable (de tipo Combo Box) con los nombres de las tres ciudades. El valor que le hemos dado a cada uno de los atributos value no aparece en la página, pero podremos usarlo en el código de nuestra aplicación, como veremos luego.
Volviendo al enlace de datos, para que el DropDownList renderice el código HTML correctamente a partir de un origen de datos (nuestra tabla de clientes), debemos indicarle qué campo de la tabla usar para el texto que se mostrará, y qué campos deberá utilizar para el atributo value.

Veamos cómo queda el código y, después, analizaremos cada línea en particular:

```
//C#
DataTable tabla = ObtenerClientes();
clientes.DataSource = tabla;
clientes.DataValueField = "Codigo";
clientes.DataTextField = "Nombre";
clientes.DataBind();

'VB.Net
Dim tabla as DataTable = ObtenerClientes()
```

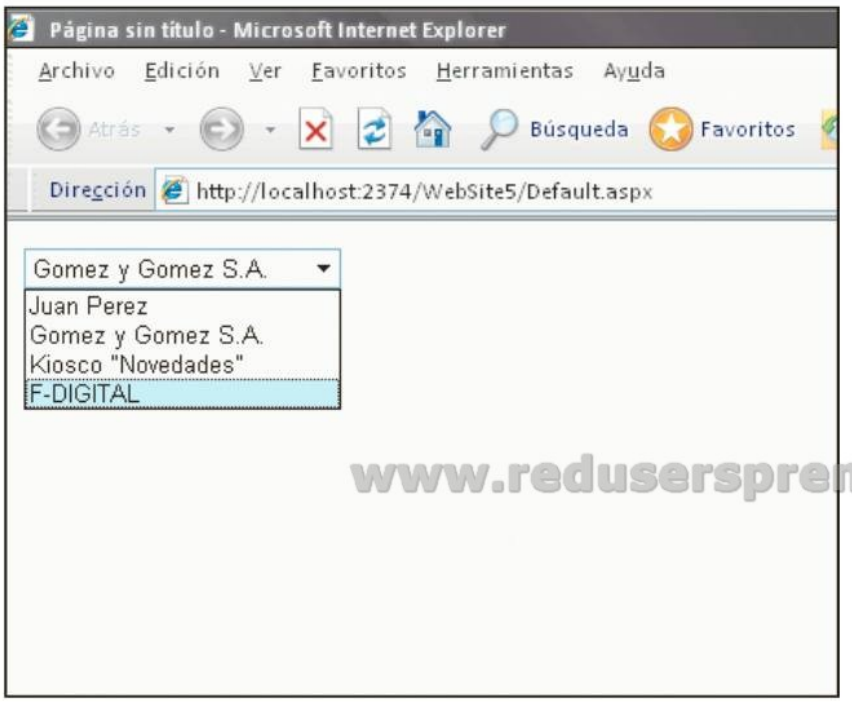


FIGURA 037 | Nuestra aplicación con data binding en ejecución.

```
clientes.DataValueField = "Codigo"
clientes.DataTextField = "Nombre"
clientes.DataBind()
```

La primera línea, como ya mencionamos, invoca al método ObtenerClientes, para conseguir los registros de la tabla de clientes de la base. La segunda línea le indica al control que utilice el objeto tabla (de tipo DataTable) para llenar la lista. Luego, mediante la propiedad DataValueField, se le indica al control que tome los valores del campo Codigo para asignar los atributos value de los tags option que genere para cada registro. La siguiente línea señala al control que utilice el campo Nombre para tomar los valores que debe emplear como texto de cada elemento de la lista. Por último, la llamada al método DataBind se encarga de tomar los datos de la tabla y de asegurar que se generen los tags HTML correspondientes. Si ejecutamos la aplicación y navegamos a la página que acabamos de crear, veremos la lista desplegable con los valores de la tabla.

Controles de origen de datos

ASP.NET 2.0 incorporó un conjunto de controles que funcionan como origen de datos y permiten enlazarlos luego a los controles, todo de manera visual y declarativa. La única restricción es que el control al que enlazaremos los datos debe soportar la propiedad DataSouceID. Además, poseen la enorme ventaja de encapsular mucha lógica para hacer por nosotros operaciones como paginado, ordenamiento y edición de datos directamente sobre la base (actualizaciones, inserciones y eliminaciones). Por lo tanto, representan una mejora sustancial con respecto a las versiones anteriores de ASP.NET, en las que debíamos hacer todas esas operaciones a mano, capturando eventos del lado del servidor. Podemos encontrar estos nuevos controles en la sección Datos del cuadro de controles de Visual Studio.

> Al usar DataBinding, debemos tener siempre presente el ciclo de vida de la página y la propiedad IsPostback, para no hacer enlaces innecesarios cuando los datos ya se han cargado.

Veamos a continuación cuáles son y cómo se utilizan los controles de origen de datos.

El control SqlDataSource

Permite crear un origen de datos para enlazar los controles a partir de una base de datos relacional, que no necesariamente debe ser SQL Server, ya que se puede conectar a través de cualquier provider de ADO.NET (como OleDb y ODBC).
Como ya mencionamos, este control evita tener que escribir el código de acceso a datos en el archivo code behind de la página,

ⓘ No sólo de DataSets vive el hombre

Es importante tener en cuenta que no necesariamente debemos enlazar los controles con un objeto DataTable, sino que también podemos hacerlo con un DataSet (indicando, además, el nombre de la tabla) e, incluso, con listas de objetos u objetos simples; por ejemplo, podríamos enlazar con una instancia de la clase Cliente.

Para no repetir las consultas, podemos crear procedimientos almacenados en la base y configurar SqlDataSource para que los invoque.

dado que mediante propiedades, podemos especificar las consultas SQL (o los stored procedures) necesarias para realizar toda la manipulación de los datos, tanto su recuperación como su actualización.

Al igual que los demás controles de origen de datos, SqlDataSource posee un asistente que nos ayudará a definir los valores de las propiedades imprescindibles para su funcionamiento. Luego, podremos utilizar otras propiedades para trabajar sobre detalles más finos de su comportamiento.

Veamos, entonces, los pasos que debemos seguir para configurar apropiadamente el SqlDataSource. Lo arrastramos desde la barra de controles a una página cualquiera (donde queramos utilizarlo). Si estamos en modo diseño, Visual Studio nos mostrará el típico rectángulo gris, representando un control que no será visible en tiempo de ejecución, y al pasar el mouse sobre él, aparecerá una pequeña flecha azul a la derecha. Este icono es el acceso directo a un menú contextual que, en el caso del SqlDataSource, tiene una sola opción: Configurar Origen de Datos. Haciendo clic en ella, pasamos directamente al asistente.

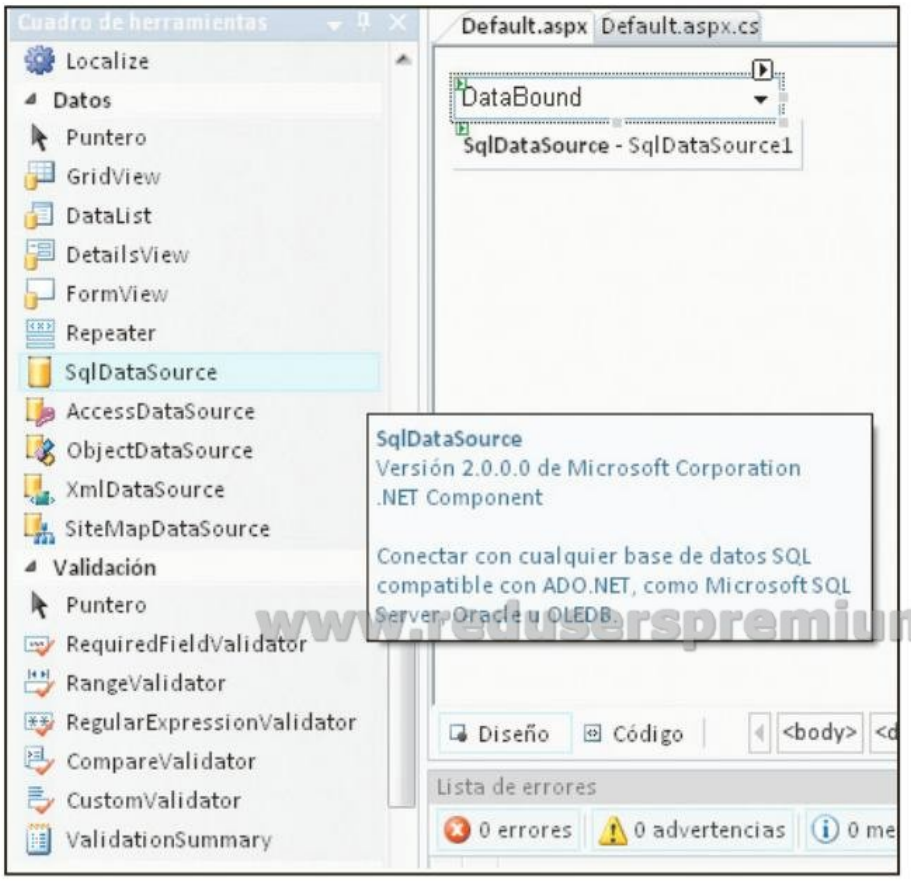


FIGURA 038 | La sección Data de la barra de controles contiene los nuevos controles de origen de datos.

El primer dato que nos solicita el asistente es la cadena de conexión a la base. En este punto tenemos dos opciones: seleccionar una de las que ya estén creadas en el proyecto (configuradas en el archivo web.config) o crear una nueva, como haremos en este caso. Haciendo clic en el botón Nueva conexión, se abre un pequeño cuadro de diálogo donde podemos seleccionar el tipo de proveedor; elegimos Microsoft SQL Server y aceptamos. A continuación, se despliega otro cuadro de diálogo en el que deberemos seleccionar el servidor y la base de datos a la cual queremos conectarnos. Una vez que seleccionamos la base de datos y hacemos clic en Siguiente, el asistente nos pregunta si deseamos guardar la cadena de conexión en el archivo de configuración. Es recomendable hacerlo, ya que así quedará guardada y podremos utilizar siempre la misma.

En el siguiente paso, tenemos la opción de especificar una consulta SQL o procedimiento almacenado, o de seleccionar de qué tabla y de qué campos queremos traer los datos. En el segundo caso, el asistente genera automáticamente una consulta SQL según las columnas que indiquemos. Aquí podemos seleccionar la opción que más nos guste o nos resulte más conveniente, pero siempre recordemos que es aconsejable emplear procedimientos almacenados. Para este ejemplo, elegimos uno que trae una lista de productos (ProductID y ProductName) de la base de datos Northwind. Si optamos por el procedimiento almacenado o la consulta SQL, tendremos la posibilidad, también, de escribir las consultas para la actualización de los datos, pero por el momento no vamos a hacer nada de eso, sólo los traeremos desde la base. Esto es todo lo que necesitamos para configurar el control SqlDataSource

Si ahora consultamos el código HTML de la página, veremos algo así:

```
<asp:SqlDataSource ID="SqlDataSource1"
runat="server"

   ConnectionString="<%$ ConnectionStrings:
   CursoConnectionString %>"
      ProviderName="System.Data.SqlClient"
      SelectCommand="spRecuperarPaises"
      SelectCommandType="StoredProcedure">

</asp:SqlDataSource>
```

Aquí se observa con claridad que el asistente les dio valores a las propiedades de acuerdo con las selecciones que hicimos. Particularmente interesan las propiedades ConnectionString y SelectCommand. Si analizamos el código anterior, notaremos que el valor de la propiedad Connection String contiene unos caracteres especiales: "<%$". Éstos le indican a ASP.NET que

¿Es profesional usar el asistente?

Muchos sostienen que los asistentes son para ayudar a los principiantes a lograr una aplicación funcional en pocos minutos, y que los profesionales deben escribir código. Esto es cierto en muchos casos, pero con los controles de Data Source, esa afirmación no tiene validez, porque el asistente no genera código, sino que presenta una interfaz amigable para darles valores a las propiedades imprescindibles para el funcionamiento del control. Además, por más que no lo usemos, podemos lograr el mismo cometido sin escribir código, sólo configurando propiedades.

LAS PROPIEDADES MÁS IMPORTANTES DEL CONTROL SQLDATASOURCE SON CONNECTIONSTRING Y SELECTCOMMAND. SI USAMOS CONSULTAS SIMPLES, PODEMOS OMITIR EL VALOR DE LA PROPIEDAD SELECTCOMMANDTYPE, YA QUE EL CONTROL LO INFERIRÁ AUTOMÁTICAMENTE.

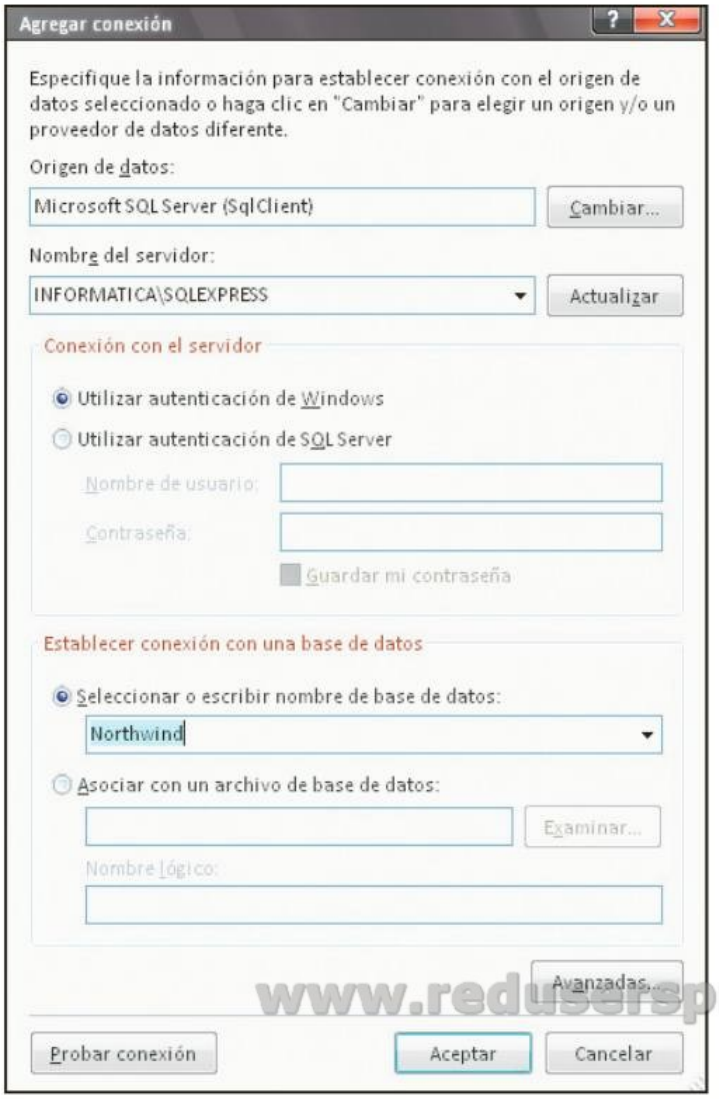


FIGURA 039 | El asistente del SqlDataSource nos permite seleccionar el servidor y la base de datos, y crea la cadena de conexión por nosotros.

lo que sigue no es directamente el valor de la propiedad, sino el lugar de donde debe tomar el valor; en este caso, de la cadena de conexión llamada CursoConnection String, dentro de la sección Connection Strings del archivo de configuración.

Ya tenemos creado nuestro origen de datos, ahora sólo resta enlazarlo a un control para verlo en acción. Arrastramos a la página un control DropDownList desde la barra de controles. Al igual que sucede con el SqlDataSource, podemos desplegar un menú contextual donde figura la opción Seleccionar Origen de datos. Al hacerlo, se abre un cuadro de diálogo con tres listas desplegables. En la primera están los controles de origen de datos que hay en la página; en nuestro caso, sólo uno, así que lo elegimos. Las siguientes dos listas nos permiten seleccionar el campo para mostrar en la lista y para tomar como valor de los elementos (tal como vimos en la sección anterior). Seleccionamos los campos que correspondan y ejecutamos la aplicación.

Si todo salió bien, la página contendrá una lista desplegable con los registros que devolvió la consulta o procedimiento almacenado que le definimos a SqlDataSource. Lo interesante de todo esto es que no hemos escrito una sola línea de código en el archivo de code behind.

Parametrizar las consultas

Para ser sinceros, pocas veces en nuestro trabajo de desarrollo tendremos tareas tan sencillas como enlazar un control a todos los datos de una tabla, haciendo una simple consulta SQL. Por lo general, deberemos traer un dato en particular, dependiendo de algún valor ingresado por el usuario o que se encuentre en otra tabla que ya se está mostrando en la página. En estos casos, el ejemplo que vimos anteriormente no resulta de gran utilidad, de modo que debemos tener alguna forma de parametrizar la consulta o la llamada al procedimiento almacenado para traer sólo los datos que sean necesarios.

Por suerte, los controles de origen de datos permiten tomar datos de distintos lugares y usarlos para modificar la consulta en función de esos valores. Es posible utilizar un parámetro del Query String, un valor almacenado en la sesión, una cookie o cualquiera de los controles del formulario.

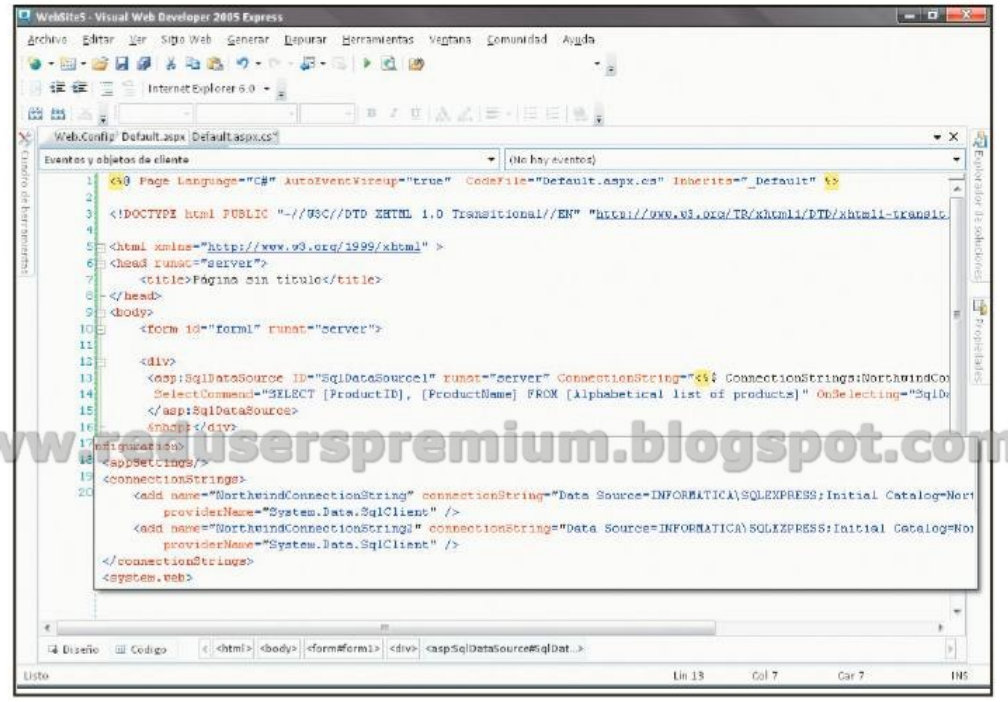
Hagamos un ejercicio para entender el funcionamiento de la parametrización de consultas con SqlDataSource. Partimos del esquema de tablas de la Figura 42, con una tabla de Provincias y otra de Localidades, donde cada localidad tiene una referencia a la provincia en la que se encuentra.

Lo primero que vamos a hacer es colocar un control SqlDataSource para recuperar la

ⓘ Parámetros en consultas a SQL Server

La forma de especificar los parámetros en SQL Server es mediante el prefijo "@" en el nombre de las variables usadas en la consulta. Por ejemplo, si escribimos "SELECT * FROM Usuarios WHERE Nombre = @nombreUsuario", estamos introduciendo un parámetro llamado @nombreUsuario. Esto nos permitirá dar distintos valores al parámetro para obtener sólo los datos necesarios.

FIGURA 040 | Mediante código de escape podemos asociar la propiedad ConnectionString del SqlDataSource con una de las cadenas de conexión configuradas en el archivo web.config.



POCAS VECES TENDREMOS TAREAS TAN SENCILLAS COMO ENLAZAR UN CONTROL A LOS DATOS DE UNA TABLA CON UNA CONSULTA SQL. POR LO GENERAL, DEBEREMOS TRAER UN DATO, DEPENDIENDO DE ALGÚN VALOR INGRESADO POR EL USUARIO O QUE SE ENCUENTRE EN OTRA TABLA.

lista de todas las provincias, y asociarlas a un DropDownList para que el usuario seleccione. En este caso, los pasos son los mismos que vimos en la sección anterior, usando el campo ID como DataValueField, y el campo nombre como DataTextField.
Si el lector no logra asociar el DropDownList a la tabla de provincias, recomendamos volver a leer los párrafos anteriores y seguir el ejemplo paso a paso para incorporar los conceptos.
El segundo paso consiste en configurar un control SqlDataSource para recuperar los registros de la tabla Localidades, pero sólo los que corresponden a la provincia seleccionada en el DropDownList. Arrastramos un nuevo control SqlDataSource a la página y abrimos el asistente. Si hasta ahora hicimos todo bien, cuando configuremos el SqlDataSource de Provincias, tendremos ya creado un objeto Connection String en el archivo de configuración, así que directamente lo seleccionamos de la lista que nos muestra el asistente. En el siguiente paso, de la lista de tablas disponibles seleccionamos Localidades. En la parte inferior de la ventana, aparecerán listados todos los campos que contiene; elegimos sólo ID y Nombre.
En la misma ventana del asistente, sobre el borde derecho, hay tres botones: WHERE, ORDER BY y Avanzado. El primero nos permite generar la condición lógica de la cláusula WHERE de la instrucción SQL; el segundo se utiliza para determinar el ordenamiento de los campos; y el último lo veremos más adelante. Haciendo clic en WHERE, accedemos a otra ventana donde podemos especificar filtros sobre los campos de la tabla (Figura 43).

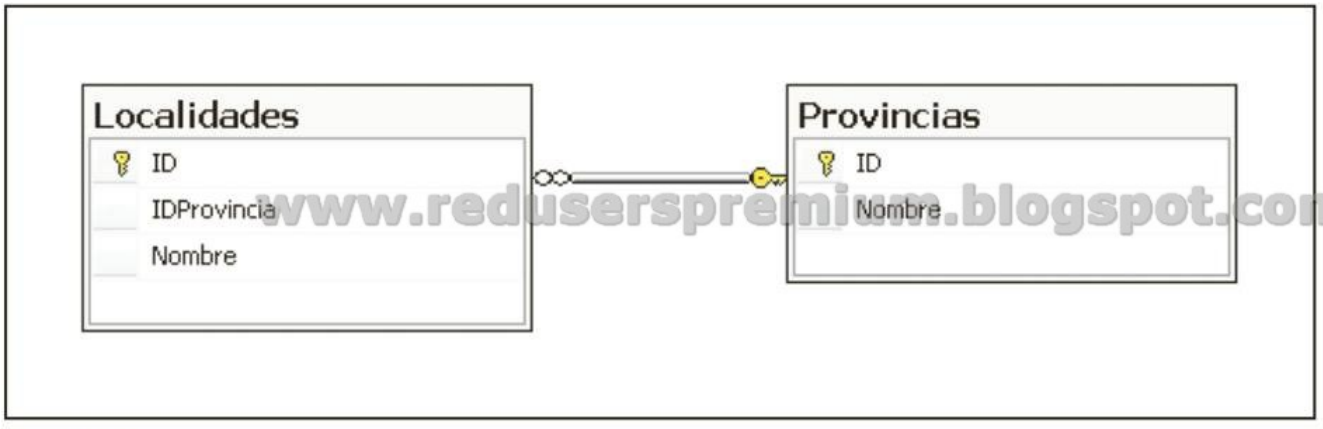


FIGURA 041 | Para este ejemplo, cada localidad tiene una referencia a la provincia en la que se encuentra.

EL ASISTENTE DE CONFIGURACIÓN DEL CONTROL SQLDATASOURCE DETECTA EL TIPO DE LAS COLUMNAS Y SÓLO NOS MUESTRA LOS OPERADORES RELACIONALES VÁLIDOS PARA ESE TIPO DE DATOS, CON LO CUAL SE MINIMIZA EL RIESGO DE ERROR.

Como se observa en la figura, esta ventana nos permite seleccionar en primera instancia la columna sobre la que queremos hacer el filtro (en nuestro caso, el IDProvincia, para relacionarlo con la provincia seleccionada); luego, permite elegir el operador relacional por aplicar, nosotros usaremos el "igual". Un detalle interesante es que el asistente sólo nos muestra los operadores válidos para el tipo de datos de la columna que hemos seleccionado, lo que reduce la posibilidad de errores de tipos. La siguiente lista nos ofrece elegir el origen del valor que se tomará para el filtro, entre uno de los siguientes: Ninguno, Control, Cookie, Formulario, Perfil del usuario, Query String o Session. Como tenemos el valor del ID de provincia en el SelectedValue del control DropDownList, seleccionamos Control. Cuando elegimos el origen del valor, en el cuadro Propiedades del parámetro, se nos permite seleccionar el control del cual queremos tomarlo. Indicamos cmbProvincias (o el nombre que le hayamos dado) y, automáticamente, el asistente asignará el valor cmbProvincias.SelectedValue como

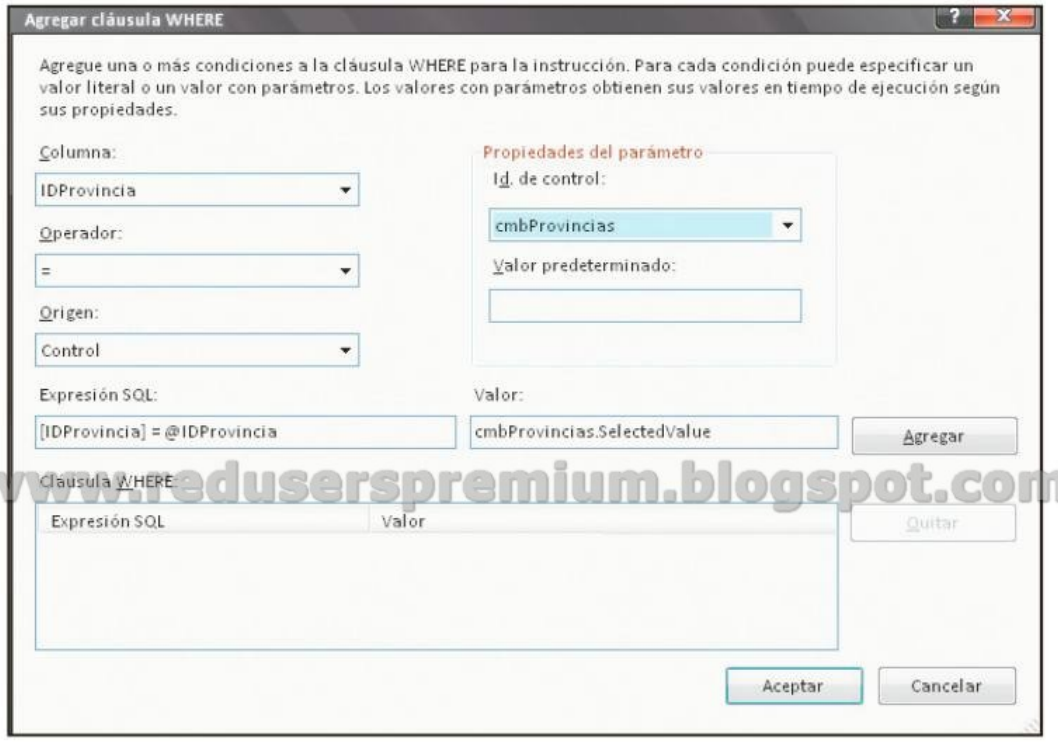


FIGURA 042 | El asistente nos permite configurar los filtros de la consulta, en función de controles u otros valores de la página.

Cuando la página se renderice en el browser, veremos dos listas: la primera, con provincias, y la segunda, con localidades. Al cambiar la provincia, veremos cómo la página es enviada al servidor y, al regresar, tendremos las localidades correspondientes.

valor del parámetro. Por último, hacemos clic en el botón Agregar para confirmar el filtro por parámetro y finalizamos el asistente.
Sólo nos resta agregar un nuevo control DropDownList a la página y asociarlo con el segundo SqlDataSource. Para que la página se actualice automáticamente cuando el usuario seleccione una provincia, al primero de los controles DropDownList que hemos colocado le asignamos el valor True en la propiedad Autopostback.

¡Listo! Ahora ejecutamos la aplicación para ver que todo funcione como esperamos. Cuando la página se renderice en el browser, veremos dos listas: la primera, con provincias, y la segunda, con localidades. Al cambiar la provincia seleccionada, veremos cómo la página es enviada al servidor y, al regresar, tendremos las localidades correspondientes en la segunda lista.
Y lo mejor de todo es que esto lo pudimos hacer ¡sin una línea de código!

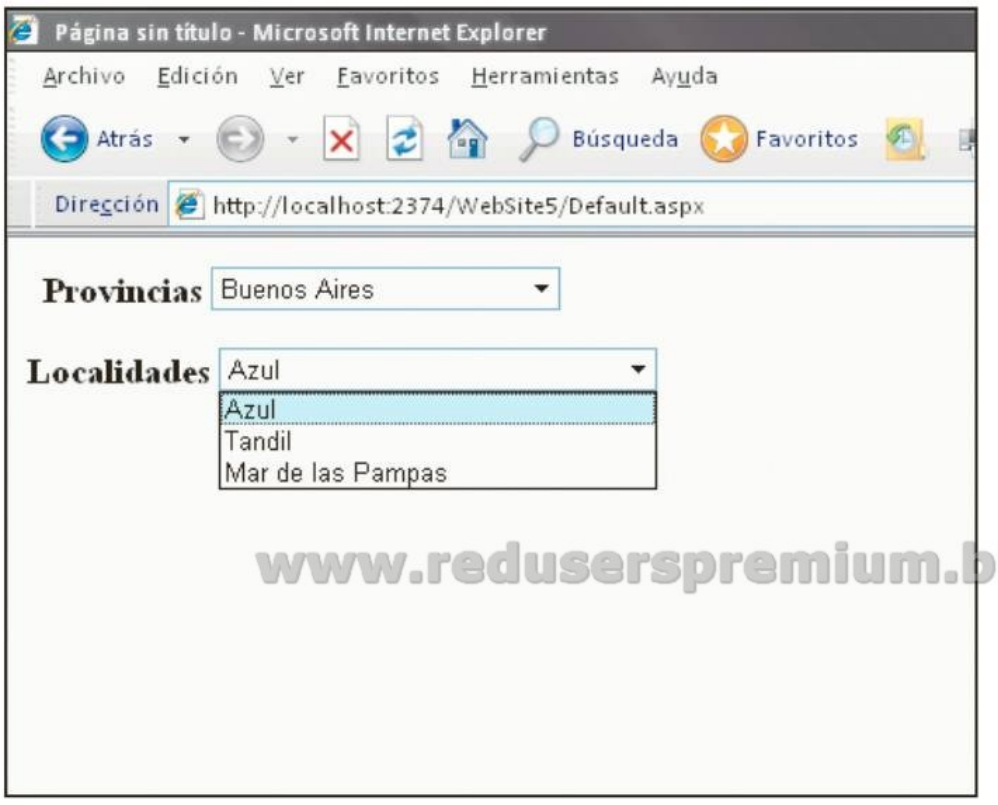


FIGURA 043 | Al ejecutar la aplicación, veremos cómo los datos de la segunda lista se filtran en función del valor seleccionado en la primera.

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

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

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

La mejor forma de aprender a programar desde cero

★★★★★ 10

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

Sitios web

Páginas maestras - Temas y skins
Navegación por el sitio

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Acceso a datos

Controles de datos de origen
DataBinding - Enlace a datos

ISBN 978-987-1347-43-8
00010
9 789871 347438