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

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

8

Controles ricos

Campos de Texto - Calendario
Casillas de Verificación

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Controles de usuario

Creación - Instalación - Personalización

ISBN 978-987-1347-43-8
00008
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

Control ListBox

Este control modela una lista de valores de selección, con la diferencia fundamental de que permite la selección múltiple. La propiedad **SelectionMode** controla este aspecto, y da la posibilidad de realizar selección simple o múltiple. Si utilizamos este control con la propiedad **SelectionMode = Single**, la opción seleccionada puede obtenerse de la misma manera que el control DropDownList. En cambio, si **SelectionMode = Multiple**, debemos consultar la colección de opciones a través de la colección **Ítems**, de la siguiente manera:

```
<asp:ListBox ID="ListBox1" runat="server"
Height="99px" SelectionMode="Multiple"
    Width="226px" AutoPostBack="True">
    <asp:ListItem>Some Girls</asp:ListItem>
    <asp:ListItem>Tatto You</asp:ListItem>
    <asp:ListItem Selected="True">Still
    Alive</asp:ListItem>
    <asp:ListItem>Flashpoint</asp:ListItem>
    <asp:ListItem>steel wheels</asp:ListItem>
    <asp:ListItem Selected="True">Bridges To
    Babylon</asp:ListItem>
    <asp:ListItem>a bigger bang</asp:ListItem>
</asp:ListBox> </p>
```

En el código anterior se agregó un botón para producir el **PostBack** y ejecutar el manejador de evento para **SelectedIndexChanged**.

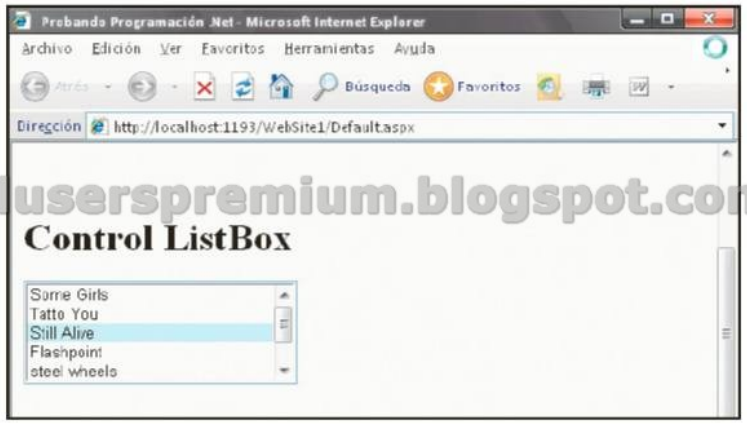
```
protected void ListBox1_SelectedIndexChanged
(object sender, EventArgs e)
{
        foreach (ListItem li in
        ListBox1.Items)
        {
        if (li.Selected)
            {
              // hacer algo
            }
  }
}
```

Si **AutoPostBack** se encuentra en **true**, cada vez que el usuario seleccione una opción (independientemente de estar en modo Single o Multiple), se producirá un PostBack que cargará la página y se ejecutará el evento **SelectedIndexChanged**. Por lo general, la opción **AutoPostBack** se desactiva para la selección múltiple.

Control CheckBoxList

Modela un grupo de controles CheckBox como un **ListItem**, de manera tal que el código necesario para obtener los valores seleccionados es similar al del control ListBox. Existe la posibilidad de controlar la forma en que las opciones son presentadas utilizando las propiedades **RepeatLayout** y **RepeatDirection**.

FIGURA 016 | Podemos utilizar este control para seleccionar más de un elemento a la vez.



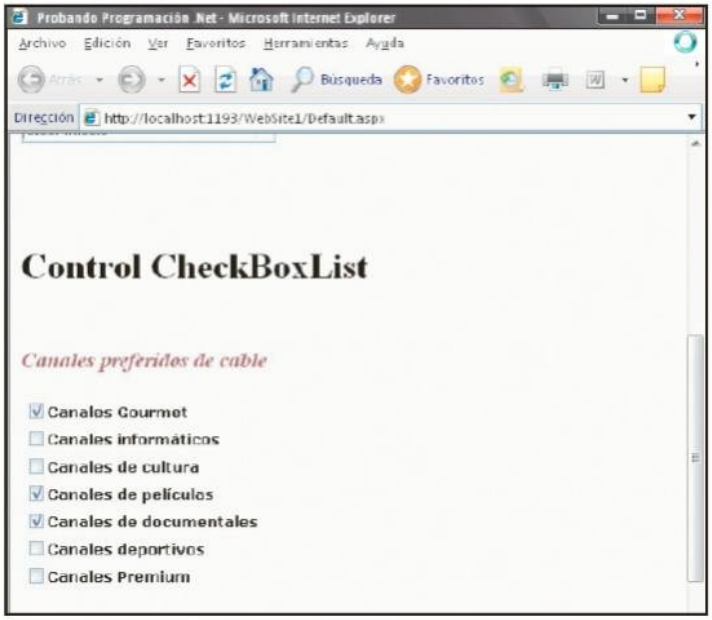


FIGURA 017 | Es posible preseleccionar valores mediante la propiedad Selected.

Los valores válidos para **RepeatLayout** están modelados en la enumeración **RepeatLayout**. Si se necesita mayor control en la presentación de las opciones, es posible utilizar controles CheckBox en vez de un control CheckBoxList.
El siguiente fragmento de código muestra cómo presentar tres opciones utilizando controles CheckBox y un CheckBoxList. Siempre tendremos mayor control sobre los aspectos de presentación si utilizamos los primeros. Sin embargo, el uso de CheckBoxList presenta un modelo para usar programáticamente que es más consistente para este tipo de situaciones, ya que todas las opciones están agrupadas en una lista.

```
<asp:CheckBox ID="CheckBox1" runat="server"
Text="Antes de 6 meses" />
<asp:CheckBox ID="CheckBox2" runat="server"
Text="Luego de 6 meses" />
<asp:CheckBox ID="CheckBox3" runat="server"
Text="Mas de un año" />
<br />
<asp:CheckBoxList ID="CheckBoxList1"
runat="server"
      RepeatDirection="horizontal">
      <asp:ListItem Text="Antes de 6 meses"
      Value="1" />
      <asp:ListItem Text="Luego de 6 meses"
      Value="2" />
      <asp:ListItem Text="Mas de 1 año"
      Value="3" />
</asp:CheckBoxList>
```

Para obtener los valores seleccionados por el usuario, deberemos escribir:

```
// opciones seleccionadas en el primer grupo
if (CheckBox1.Checked)
{
      // hacer algo
}
if (CheckBox2.Checked)
{
      // hacer algo
}
if (CheckBox3.Checked)
{
      // hacer algo
```

Tabla 7 | Forma en que se representan las opciones

Enumeración RepeatLayout	
RepeatLayout.Table	Utiliza una tabla para estructurar las opciones.
RepeatLayout.Flow	Las opciones son presentadas sin ninguna estructura.
Enumeración RepeatDirection	
RepeatDirection.Vertical	Muestra la lista en forma vertical.
RepeatDirection.Horizontal	Muestra la lista en forma horizontal.

```
}
// opciones seleccionadas en el CheckBoxList
foreach(ListItem li in CheckBoxList1.Items){
        if(li.Selected){
                // hacer algo
        }
}
```

El código es consistente con todos los modelos de listas utilizados hasta ahora.

Control RadioButtonList

Se encarga de modelar una lista de controles de radio. La ventaja de utilizarlo sobre un grupo de botones de radio radica en el modelo consistente empleado para acceder a las opciones seleccionadas. Para determinar la opción, utilizamos la propiedad **SelectedItem** del tipo ListItem. Al igual que con CheckBoxList, es posible controlar la manera en que se presentan las opciones utilizando **RepeatLayout** y **Repeat Direction**. Por defecto, este control no cumple los estándares de la WCAG. Deberíamos tratar de utilizarlo con la propiedad **RepeatLayout.Flow** y aplicar estilos para manejar el aspecto visual.

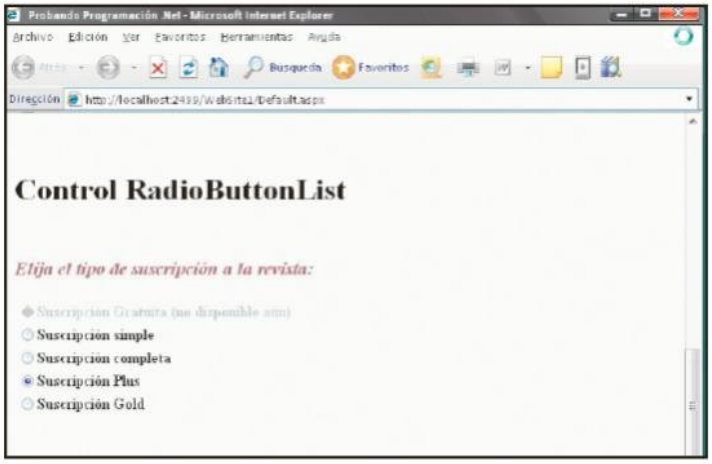


FIGURA 018 | Podemos desactivar programáticamente algunas opciones presentadas.

Tabla 8 | Propiedades BulletStyle y ListDisplayMode

Control BulletedList	
Propiedad BulletStyle	
Estilo	**Descripción**
NotSet	Sin viñetas
Numbered	Números
LowerAlpha	Letras minúsculas
UpperAlpha	Letras mayúsculas
LowerRoman	Números romanos en minúscula
UpperRoman	Números romanos en mayúscula
Disc	Círculo pintado
Circle	Círculo vacío
Square	Cuadrado pintado
CustomImage	Una imagen personalizada
Enumeración BulletedListDisplayMode	
HyperLink	Cada elemento es renderizado como una etiqueta <a/>. El valor de la propiedad **Value** del ListItem se utiliza como URL.
LinkButton	Cada elemento es renderizado como un LinkButton.
Text	Cada elemento es renderizado como una etiqueta .

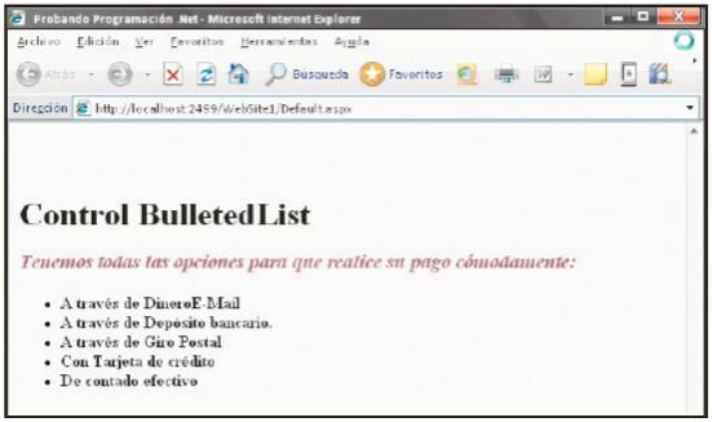


FIGURA 019 | El control BulletedList enumera distintas opciones.

Control BulletedList

Modela una lista de elementos con viñetas. La viñeta es especificada mediante la propiedad **BulletStyle**, manejada por la enumeración llamada de la misma manera. Cuando configuramos la propiedad **BulletStyle** en **BulletStyle = Custom Image**, se debe especificar la imagen que se va a utilizar como viñeta en la propiedad **BulletImageUrl**. El control permite indicar tres formas para los ítem, determinadas por la propiedad **DisplayMode**. La enumeración **BulletedList DisplayMode** contiene los tres valores mencionados en la Tabla 8. Cuando se utiliza **Bulleted ListDisplayMode = LinkButton**, el control realiza un postback en el momento en que el usuario cliquea. El postback dispara el evento **Click**, y entonces podremos realizar tareas especificas usando un manejador de eventos.

ⓘ RepeatLayout.Flow

Por defecto, el control CheckBoxList se renderiza como una tabla de acuerdo con el valor RepeatLayout.Table, con controles input type="checkbox". Esto no conforma los estándares de accesibilidad, por lo que deberíamos tratar de utilizarlo con RepeatLayout.Flow y aplicar estilos para manejar el aspecto visual.

El evento **Click** del control BulletedList contiene un segundo argumento **BulletedListEvent Args**. El valor de su propiedad **index** corresponde al índice del elemento seleccionado. Veamos cómo utilizar BulletedList con la propiedad **BulletedListDisplayMode = LinkButton**:

```
<asp:BulletedList ID="BulletedList1"
runat="server" Font-Size="12pt">
 <asp:ListItem>A trav&#233;s de
 DineroE-Mail</asp:ListItem>
 <asp:ListItem>A trav&#233;s de
 Dep&#243;sito bancario.</asp:ListItem>
 <asp:ListItem>A trav&#233;s de Giro
 Postal</asp:ListItem>
 <asp:ListItem Selected="True">Con Tarjeta
 de cr&#233;dito</asp:ListItem>
 <asp:ListItem>De contado
 efectivo</asp:ListItem>
</asp:BulletedList>
<br />
<br />
<asp:Label ID="Label1" runat="server"
Text="Label"></asp:Label>
```

El código asociado contiene el manejador del evento Click que se muestra a continuación:

```
protected void BulletedList1_Click(object
sender, BulletedListEventArgs e)
{
// Obtener el item seleccionado
        ListItem itemSeleccionado =
        BulletedList1.Items[e.Index];
      // Mostrar informacion del item
      seleccionado en un Label
      Label1.Text = String.Format("Se
      selecciono el item {0}. Text: {1} |
      Value:
{2}",
e.Index,itemSeleccionado.Text,itemSeleccionado.
Value);
}
```

Practicando ASP.NET

A continuación, veremos mediante ejemplos prácticos, el uso de los controles Button, Hyperlink, ImageButton y LinkButton.

Hasta ahora hemos visto que ASP nos permite desarrollar aplicaciones para Internet y aprendimos cuáles son algunos de los controles u objetos que podemos utilizar cuando desarrollamos aplicaciones. También vimos ejemplos de las líneas de código más comunes que utilizan los controles. Ahora llegó el momento de practicar lo visto sobre ASP, y nada mejor para lograrlo que realizar una serie de prácticas que nos permitirán familiarizarnos con el lenguaje y sus elementos.

Compra de productos utilizando Button

Nuestro primer proyecto Web tiene como objetivo mostrar una aplicación que permita seleccionar productos de una lista y realizar su compra presionando un botón. Lo primero que debemos hacer es abrir Visual Studio (para este ejercicio en particular usaremos C#) y seleccionar en la ventana de nuevo proyecto la opción Crear un proyecto de tipo web. A continuación, colocamos el nombre que llevará nuestro proyecto y seleccionamos dónde lo vamos a guardar.

Una vez finalizada la carga del proyecto, nos encontramos con la pantalla correspondiente a la página Default.aspx, en donde debemos escribir el código correspondiente a la aplicación Web. En la primera parte del código ponemos las líneas que harán que nuestra página, al verse en un navegador, tenga cómo título principal Pedido de productos y, a continuación, un breve texto indicándole al usuario cómo debe proceder para que la aplicación funcione correctamente. Debajo de la página Default veremos una serie de botones; entre ellos, los diseño y código.

Hacemos clic en el de código y, a continuación, veremos el código que agregó Visual Studio en la aplicación. Debemos comenzar a escribir lo siguiente entre y :

```
<h1>Pedido de productos</h1>
<p>Seleccione el/los productos de la lista y
presione el boton "Realizar Pedido" para
continuar.</p>
```

Luego de estas líneas, configuramos los controles y objetos que permitirán la iteración con el usuario. El siguiente código configura el objeto que contendrá la lista de productos, el botón que nos permitirá realizar la compra y, por último, una etiqueta que nos dará un mensaje indicándonos si la operación se realizo de manera correcta:

```
<!– Etiqueta –>
<asp:Label ID="Label1" runat="server"
Text="Productos disponibles"></asp:Label>

<br />

<!– Listado de productos para seleccionar.
Admite seleccion multiple –>
<asp:ListBox ID="ProductosListBox"
runat="server"
  SelectionMode="Multiple"></asp:ListBox>

<br />

<!– Submit button –>
<asp:Button ID="SubmitButton" runat="server"
Text="Realizar Pedido"/>
```

Recordemos que podemos probar el funcionamiento de nuestra página web en todo momento presionando <F5>.

```
<br />

<!– Utilizaremos este control para mostrar
un mensaje –>
<asp:Label ID="Mensaje" runat="server"
style="color:Blue;" />
```

Si probamos la página Web (recordemos que podemos hacerlo presionando <F5>), veremos cómo queda configurada hasta el momento. No aparecerá todavía la lista de productos ni tampoco podremos usar el botón, porque sólo creamos la interfaz para el usuario. A continuación, seguiremos escribiendo el código necesario para lograr que la aplicación funcione como lo deseamos.
La página tendrá un aspecto como se muestra en la Figura 20.

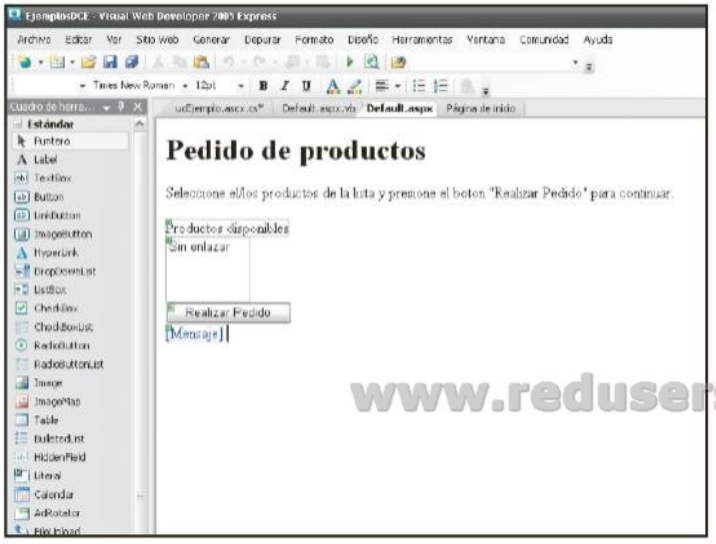


FIGURA 020 | La página vista desde Visual Studio, en tiempo de Diseño.

Para que el control de lista (ListBox) funcione correctamente en nuestro proyecto, debemos realizar su carga en el evento Page_Load. Para acceder a este evento sólo necesitamos hacer doble clic en cualquier sector de la página Default.aspx. A continuación, escribimos:

```
protected void Page_Load(object sender,
EventArgs e) {
if (!IsPostBack) {
            ProductosListBox.DataSource =
            Sistema.ProductosDisponibles();
            ProductosListBox.DataTextField =
            "Nombre";
            ProductosListBox.DataValueField =
            "Id";
            ProductosListBox.DataBind();

      }
}
```

Antes de continuar, analicemos el código escrito. La sentencia If comprueba si nuestra página es cargada por primera vez. En caso de ser así, es necesario configurar el DataSource del objeto ListBox, ya que sin esta propiedad no podríamos mostrar los productos disponibles para que el usuario seleccione.
Luego configuramos la colección List<Producto>, que devuelve los Productos disponibles, que serán traídos a ListBox a través de la propiedad DataSource del control. Más adelante en el curso veremos que esta misma propiedad se utiliza para asignar una fuente de datos (por ejemplo, una base SQL) a fin de que el control se cargue con los datos de ella.
La propiedad dataTextField indica que debe mostrarse como texto el valor de la columna Nombre (más adelante veremos que la lista de productos no sólo incluye nombres o descripciones de productos, por lo que es importante indicar qué tipo de dato queremos mostrarle al usuario en la lista).

La propiedad DataValueField indica que debe utilizarse el valor de la columna Id como valor. Veremos que esta columna es de gran utilidad, ya que nos permite asignar un código único para cada producto. Así, el usuario tiene que seleccionar un producto por su nombre, pero internamente la aplicación relaciona ese producto a través del numero de Id.

Ahora nos encargaremos de programar la acción del botón. Para hacerlo, debemos asignar la acción OnClick con el evento llamado **RealizarPedido**, que contendrá el código que ejecutará el botón al hacer clic.

Volvemos al código de la página Default.aspx y agregamos lo siguiente:

```
…
<!– Submit button –>
<asp:Button ID="SubmitButton" runat="server"
Text="Realizar Pedido"
OnClick="RealizarPedido"/>
…
```

Ahora debemos volver al código correspondiente a Page_Load. Para esto, podemos hacer doble clic en el archivo Default.aspx.cs que muestra el Explorador de soluciones a la derecha de la pantalla. Al código que ya tenemos:

```
…
if (!IsPostBack) {
            ProductosListBox.DataSource =
            Sistema.ProductosDisponibles();
            ProductosListBox.DataTextField =
            "Nombre";
            ProductosListBox.DataValueField =
            "Id";
            ProductosListBox.DataBind();

      }
}
```

Agregaremos lo siguiente:

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

       // Utilizaremos el metodo
       ComprarProductos especializado
       // para obtener una lista de productos
       ListItem.
       Sistema.ComprarProductos
       (ProductosListBox.Items);

       // Dar un mensaje al usuario
       Mensaje.Text = @"Su compra ha sido
       realizada con exito.
         Nos contactaremos con usted";
}
```

Este método representa el manejador de evento para el evento OnClick del botón.
Aquí podremos colocar el código que debe ejecutarse cuando el usuario hace clic sobre el botón para realizar el pedido de los productos.
Para mantener el código flexible, no introduciremos lógica en este evento y, en su lugar, pasamos los elementos elegidos por el usuario en la lista, a una clase especializada que implementa el proceso de compra.

FIGURA 021 | La lista de productos se obtiene del sistema. La compra se realiza al presionar el botón.

LA PROPIEDAD DATAVALUEFIELD INDICA QUE DEBE UTILIZARSE EL VALOR DE LA COLUMNA ID COMO VALOR. VEREMOS QUE ESTA COLUMNA ES DE GRAN UTILIDAD, YA QUE NOS PERMITE ASIGNAR UN CÓDIGO ÚNICO PARA CADA PRODUCTO.

Por otro lado, el método RealizarPedido delega la responsabilidad de la compra al método estático ComprarProductos de la clase Sistema (que desarrollaremos a continuación). Cómo ya dijimos, para hacer nuestro código más flexible o más fácil de actualizar, utilizamos una especialización del método que obtiene una **ListItemCollection**. De esta manera, es muy poco el código que llega a la interfaz. Para que el proyecto funcione, necesitamos agregar las clases **Sistema**, **Producto** y **Medidas**.

Agregar clases al proyecto

Comenzaremos por agregar la clase llamada Sistema, que se encarga de obtener colecciones y tratar con los objetos del sistema entregando a la interfaz lo que necesita.

Para incorporarla, vamos al menú Proyecto y seleccionamos Agregar clase. A continuación, seleccionamos con un clic el elemento clase y, antes de presionar el botón Agregar, le ponemos el nombre Sistema.cs. Veremos que en el explorador de soluciones aparece nuestra nueva clase, en la cual agregamos lo siguiente:


```
public class Sistema {
        public static List<Producto>
        ProductosDisponibles() {
          List<Producto> productos =
          new List<Producto>();
          DataSet ds = new DataSet();
          ds.ReadXml(ConfigurationManager.
          AppSettings["XmlDataSource"].ToString());
          foreach (DataRow dr in
          ds.Tables[0].Rows) {
                  Producto producto =
```

Esta colección será cargada con los datos de un documento XML, que armaremos más adelante, y la utilizaremos como fuente de datos. Además utilizamos un DataSet para acceder a esa fuente. Es importante tener en cuenta que el path o ruta absoluta a la fuente de datos está guardado en el archivo web.config del proyecto.

Cada DataSet contiene una colección de tablas en Tables. Por ejemplo, si nos referimos a Tables[0], estaremos tomando la primera Table de la colección.

Por otro lado, cada Table contiene una colección de filas (Rows) cuyos elementos están modelados como DataRow. El siguiente fragmento de código itera las filas creando un objeto Producto y lo agrega a la colección de Productos por mostrar. Para armar la colección, utilizamos un bucle, que permite inicializar un producto para luego cargarlo; cada Row se utiliza como un array asociativo. Podemos indicar el nombre de la columna para obtener los datos correspondientes. Antes de asignarlo a una propiedad de la clase Producto, convertimos el valor al tipo en cuestión. A continuación, vemos el código que nos permite realizar esto en la clase Sistema:

```
                new Producto();
                producto.Id =
                Convert.ToInt32(dr["Id"]);
                producto.Nombre =
                Convert.ToString
                (dr["Nombre"]);
                producto.Precio =
                Convert.ToDouble
                (dr["Precio"]);

                // Producto.Medidas es un
                tipo referenciado.
                producto.Medidas.Alto =
                Convert.ToDouble(dr["Alto"]);
                producto.Medidas.Alto =
                Convert.ToDouble
                (dr["Ancho"]);
                producto.Medidas.Alto =
                Convert.ToDouble
                (dr["Peso"]);

                // una vez que producto esta
                cargado, lo agregamos a
                // la coleccion asi la vamos
                llenando productos.
                Add(producto);
            }

            // Una vez cargada la coleccion
            de Productos,
            // la devolvemos
            return productos;
        }
```

A continuación, programaremos el método que realiza la compra de los productos seleccionados en la interfaz utilizando como entrada una ListItemCollection:

```
        public static void ComprarProductos
        (ListItemCollection items) {

            foreach (ListItem li in items) {
                if (li.Selected) {

                    // crear el producto
                    utilizando el metodo
                    // CargarProducto
                    Producto producto =
                        CargarProducto
                        (Convert.ToInt32
                        (li.Value));

                    // Registrar la compra
                    aqui
                }
            }
        }
```

La siguiente parte del código de la clase Sistema permite realizar la carga de un producto reutilizando el método ProductosDisponibles:

```
        private static Producto
        CargarProducto(int id) {

            // Ob
            List<Producto>
            productosDelSistema =
                ProductosDisponibles();

            // Buscar entre los Productos
            del sistema foreach (Producto
            p in productosDelSistema) {

                if (p.Id == id) { // si lo
                encuentra ...

                    // crear un producto,
                    llenarlo y devolverlo
                    // saliendo del metodo
                    Producto producto =
                    new Producto();
                    producto.Id = id;
                    producto.Nombre =
                    p.Nombre;
```

```
                    producto.Precio =
                    p.Precio;
                    producto.Medidas =
                    p.Medidas;
                    return producto;

                }
            }
            return null;
        }

    }
```

Hasta aquí hemos programado la clase Sistema; ahora comenzaremos con el desarrollo de Producto. Antes de comenzar a escribir su código, vamos a crearla de la misma manera en que hicimos con la anterior. Como nombre, le pondremos Producto.cs.

Una ver creada la clase le agregamos el siguiente código:

```
public class Producto {

        private string nombre;
        private double precio;
        private Medidas medidas;
        private int id;

        public int Id {
            get { return id; }
            set { id = value; }
        }

        public Medidas Medidas {
            get {
                // comprobar que medidas este
                // inicializado antes
                de devolverlo.
                // Si no esta inicializada,
                // se inicializa
                if (medidas == null)
                    medidas = new Medidas();

                return medidas;
            }

            set { medidas = value; }
        }

        public double Precio {
            get { return precio; }
            set { precio = value; }
        }

        public string Nombre {
            get { return nombre; }
            set { nombre = value; }
        }

        public Producto() { }
        public Producto(string nombre,
        double precio) {
            this.Precio = precio;
            this.Nombre = nombre;

            // Medidas no se inicializa aqui,
            // sino que se implementa la
            // inicializacion en la propiedad.
        }

        public Producto(string nombre,
        double precio,
                Medidas medidas) {
            this.Nombre = nombre;
```

NO INTRODUCIREMOS LÓGICA EN ESTE EVENTO. EN SU LUGAR, PASAMOS LOS ELEMENTOS ELEGIDOS, A UNA CLASE ESPECIALIZADA.

```
            this.Precio = precio;
            this.Medidas = medidas;
        }
    }
```

Para finalizar con las clases de nuestro proyecto sólo nos falta crear la última, llamada Medidas.cs. Lo haremos de la misma manera en que trabajamos con las dos anteriores.
Luego que terminemos de desarrollar esta última clase, nos dedicaremos a realizar el archivo de contenido de datos. El código que deberá tener la clase Medidas es el siguiente:

```
public class Medidas{
    private double alto;
    private double peso;
    private double ancho;

    public double Alto {
        get { return alto; }
        set { alto = value; }
    }

    public double Ancho {
        get { return ancho; }
        set { ancho = value; }
    }

    public double Peso {
        get { return peso; }
        set { peso = value; }
    }

    public Medidas() { }
    public Medidas(double alto,
    double ancho,
        double peso) {
        this.Alto = alto;
        this.Ancho = ancho;
        this.Peso = peso;
    }
}
```

> El código XML es de suma utilidad para transportar datos o respuestas a consultas realizadas a servidores.

Para que la aplicación funcione correctamente, debemos agregar la siguiente sentencia dentro del archivo de configuración **web.config**. Este archivo podrá abrirse desde el Explorador de soluciones, haciendo doble clic en él.

```
<appSettings>
  <add key="XmlDataSource"
        value="C:\fasciculos\
        ControlesDeAccion\Productos.xml"/>
</appSettings>
```

Recordemos que value debe tener como ruta el lugar donde está guardado el archivo XML que contiene los datos. En el siguiente paso crearemos este archivo.

Archivo de datos XML

Si bien más adelante veremos en detalle cómo está compuesto y cómo se crea un archivo con código XML, es un buen momento para comenzar a familiarizarnos con este formato. El código XML es de suma utilidad para transportar datos o respuestas a consultas realizadas a servidores.
A continuación, generaremos una sentencia que crea una variable a nivel de aplicación llamada **XmlDataSource**, utilizada en el método **ProductosDisponibles** de la clase Sistema. Llegó el momento de crear nuestra fuente de datos.
Para agregar un archivo con formato XML en nuestra aplicación, deberemos ir a Agregar un ítem nuevo al proyecto, desde el menú Proyec-

FIGURA 022 | El mensaje es escrito en el manejador del evento luego de realizar la compra.

to/Agregar nuevo elemento. A continuación, elegimos Archivo Xml, y lo grabamos con el nombre **Productos.xml**. Abrimos el archivo y copiamos el siguiente fragmento de código:

```
<?xml version="1.0" encoding="utf-8" ?>
<Productos>
  <Producto>
    <Id>1</Id>
    <Nombre>Computadora
    de escritorio</Nombre>
    <Precio>1200</Precio>
    <Alto>100</Alto>
    <Ancho>100</Ancho>
    <Peso>4</Peso>
  </Producto>
  <Producto>
    <Id>2</Id>
    <Nombre>Notebook</Nombre>
    <Precio>2200</Precio>
    <Alto>70</Alto>
    <Ancho>70</Ancho>
    <Peso>2</Peso>
  </Producto>
  <Producto>
    <Id>3</Id>
    <Nombre>Servidor 1</Nombre>
    <Precio>4300</Precio>
    <Alto>150</Alto>
    <Ancho>120</Ancho>
    <Peso>6</Peso>
  </Producto>
  <Producto>
    <Id>4</Id>
    <Nombre>Notebook Ultra Liviana</Nombre>
    <Precio>3200</Precio>
    <Alto>59</Alto>
    <Ancho>62</Ancho>
    <Peso>1.6</Peso>
  </Producto>
</Productos>
```

Resumen

Hemos explicado cómo utilizar un control web Button para ejecutar el procesamiento de los datos de un formulario. Vimos cómo escribiendo un manejador de evento para el evento OnClick, es posible realizar acciones para responder al clic del botón. El método RealizarPedido contiene muy poco código, y aun así la aplicación gana flexibilidad. De esta manera, es posible realizar diferentes cambios en el proceso sin afectar la interfaz.

Es muy importante que la ruta especificada en el archivo web.config (en la declaración de la variable XmlDataSource) coincida con la del archivo xml.

Ya tenemos nuestra aplicación lista para probar. Lo único que nos queda es realizar la compilación presionando la tecla <F5> y probar cómo responde el proyecto. En la lista podemos seleccionar los productos y, cuando presionemos el botón Realizar pedido, veremos un mensaje indicando que nuestro pedido fue registrado correctamente.

Controles ricos

Estos controles combinan diversas características y elementos HTML para brindar comportamientos avanzados a nuestro sitio.

Concepto

Los llamados controles ricos o "Rich controls" están incluidos en la lista de controles estándar de ASP.NET, pero poseen un comportamiento avanzado que brinda una experiencia de usuario notablemente mejorada. Toda la lógica necesaria para el funcionamiento del control se encuentra encapsulada, y sus propiedades son accesibles tanto en el diseñador visual como por código.

Por qué son "ricos"

El término hace referencia a la llamada "riqueza dinámica"; es decir que, a diferencia del resto de controles de servidor "estáticos", poseen algún tipo de interactividad (en el caso del Calendario) o dinamismo (en el caso del AdRotator).

Calendar

El control Calendar permite al usuario seleccionar una fecha a partir de un pequeño calendario generado dinámicamente por el servidor. El siguiente código representa un control Calendar dentro de una página ASPX:

```
<form id="form1" runat="server">
<div>
  <asp:Calendar ID="Calendar1" runat=
  "server"></asp:Calendar>
</div>
</form>
```

Todo el JavaScript necesario para el comportamiento de Calendar es generado automáticamente por ASP.NET.

A medida que lo configuremos, se actualizará el código correspondiente al calendario, como veremos a continuación.

Características

Calendar es un control sumamente configurable, que permite al desarrollador adaptarlo a casi cualquier necesidad. Entre sus propiedades fundamentales podemos encontrar: DayNameFormat, que establece el formato del Día por mostrar dentro del calendario entre un grupo de opciones predefinidas (Corto, Iniciales, Primeras dos letras, Completo); SelectionMode, para establecer qué podrá seleccionar el usuario en el calendario

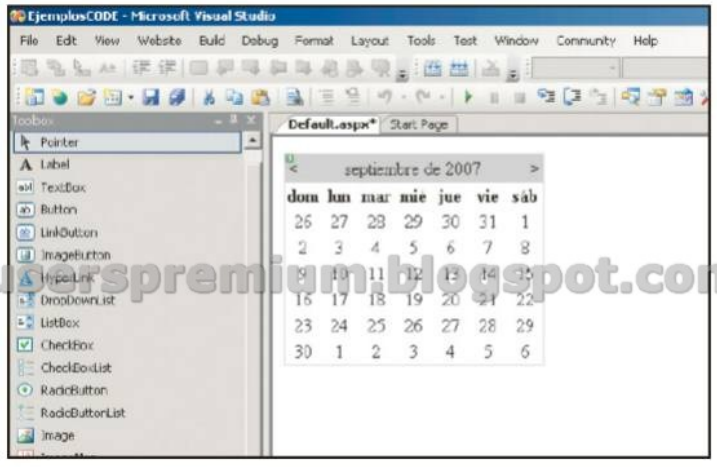


FIGURA 023 | El calendario configurable de ASP.NET.

FIGURA 024 | Calendario con selección de semana, primer día establecido en lunes.

(también incluye un grupo de opciones: Día, Semana, Mes); y FirstDayOfWeek, que permite seleccionar el primer día de la semana (por defecto, es domingo).
A estas propiedades se suman las de Estilo, que sirven para ajustar todas las características visuales del calendario. Podemos mencionar, a modo de ejemplo: SelectedDayStyle, TodayDayStyle, DayHeaderStyle, etc.

El siguiente código corresponde a un calendario establecido para selección de semana, y con el primer día configurado en lunes:

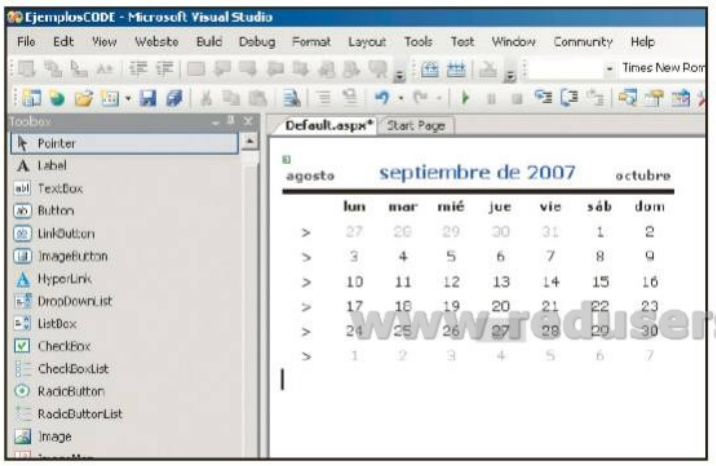


FIGURA 025 | Calendario con AutoFormato, una solución rápida y atractiva.

```
<form id="form1" runat="server">

<div>
  <asp:Calendar ID="Calendar1" runat=
  "server" FirstDayOfWeek="Monday"
  SelectionMode="DayWeek"></asp:Calendar>
</div>

</form>
```

AutoFormato

Esta opción permite darles estilos "prediseñados" a ciertos controles, y es útil a la hora de hacer desarrollos rápidos. En este caso, nos servirá para dotar a nuestro calendario de un atractivo visual importante.

Eventos

El principal evento de este control es el llamado SelectionChanged, que se dispara cuando el usuario selecciona una fecha del calendario. En el siguiente código de ejemplo capturamos la selección y la mostramos en una etiqueta, previamente colocada en el formulario:

```
protected void
Calendar1_SelectionChanged(object sender,
EventArgs e)
{
  lblFecha.Text = c
  alFecha.SelectedDate.ToString;
   }
```

Internacionalización

Un punto para tener en cuenta son las variables de internacionalización; es decir, la configuración regional del servidor en donde esté instalada la aplicación Web. El control Calendar se configura de manera automática, para estar acorde con la configuración regional del servidor, mostrando los días y los meses en el lenguaje correspondiente.

AdRotator

Desde la versión 1.1 de ASP.NET está disponible este control, destinado a la mezcla automática de publicidad para banners y similares. El objetivo de AdRotator es brindar un espacio para publicidad, con rotación automática, completamente configurable y encapsulado. El siguiente código es el resultado autogenerado en la página al insertar desde el diseñador visual un AdRotator:

```
<form id="form1" runat="server">
<div>
  <asp:AdRotator ID="arPropaganda"
  runat="server" />
   </div>
</form>
```

Configuración

Existen dos maneras fundamentales de configurar un AdRotator. La primera es crear un archivo XML con un formato preestablecido y setear la propiedad AdvertisementFile apuntando a él. La segunda alternativa es establecer una fuente de datos (DataSource) para nuestro AdRotator con los datos de configuración, también respetando el formato predeterminado.

Archivo XML

El siguiente fragmento de código será utilizado para configurar nuestro AdRotator, y debemos guardarlo en un archivo (Config.xml) dentro del directorio de la aplicación Web:

```
<Advertisements>
  <Ad>
    <ImageUrl>www.redusers.com/noticias/
    wp-content/themes/3k2redux-klein/logo-
    redusers-noticias.gif</ImageUrl>
    <NavigateUrl>www.redusers.com
    </NavigateUrl>
    <AlternateText>Red
    Users</AlternateText>
    <Keyword>Users</Keyword>
    <Impressions>80</Impressions>
  </Ad>
  <Ad>
    <ImageUrl>http://i2.microsoft.com/
    h/all/i/ms_masthead_10x7b_ltr.jpg
    </ImageUrl>
    <NavigateUrl>www.microsoft.com.ar
    </NavigateUrl>
    <AlternateText>Microsoft
    Argentina</AlternateText>
    <Keyword>Microsoft</Keyword>
    <Impressions>50</Impressions>
  </Ad>
</Advertisements>
```

AdRotator es ideal para el intercambio de banners entre sitios.

En este caso, nuestro AdRotator poseerá dos banners intercambiables, con diferente nivel de importancia (establecido en la propiedad deno-

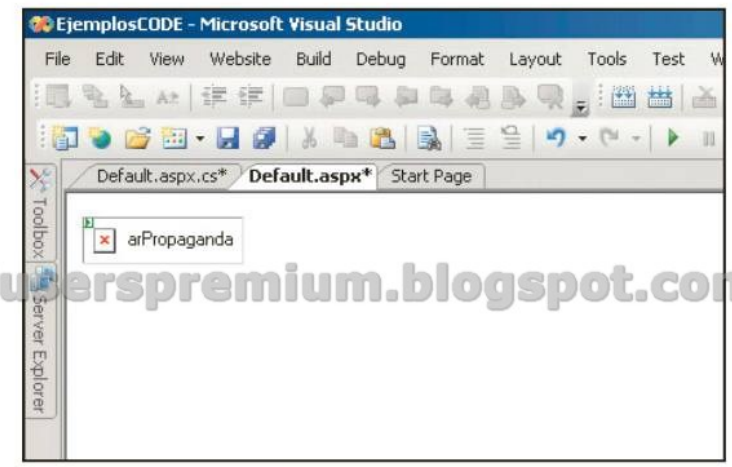


FIGURA 026 | AdRotator y su configuración por Default.

El control XML evita la programación de acceso y la lectura de archivos XML.

minada Impressions), con su propia imagen (establecida en ImageUrl), dirección de destino (NavigateUrl) y palabra clave (Keyword).
Una vez guardado el fragmento como archivo XML, podremos establecer la propiedad del AdRotator para que "apunte" a él. El código de AdRotator se verá de la siguiente manera:

```
<form id="form1" runat="server">
<div>
  <asp:AdRotator ID="arPropaganda"
  runat="server" AdvertisementFile=
    "~/Config.xml" />
</div>
</form>
```

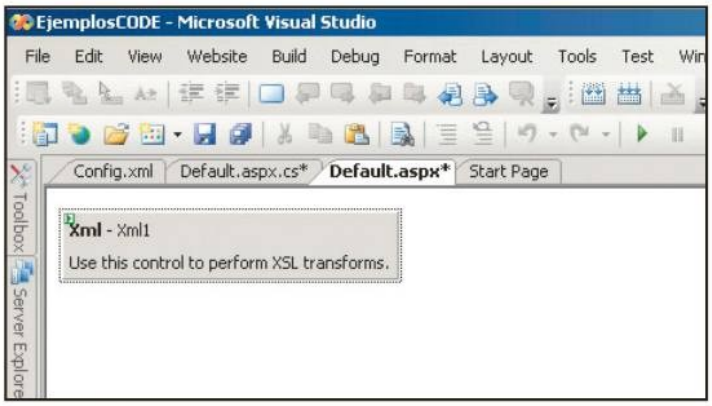


FIGURA 027 | Un control XML dentro de una página ASPX.

Transformaciones XML

El control XML permite aplicar transformaciones XSLT a un archivo XML, indicando el origen correspondiente en la propiedad TransformSource. Si ambos archivos (XML y XSLT) son válidos, el control XML renderizará en nuestro HTML el resultado de la transformación.

Eventos

El evento AdCreated del control AdRotator nos informa cada vez que se obtiene un Ad del archivo de configuración. Podemos capturarlo y realizar acciones customizadas.

```
protected void arPropaganda_AdCreated(object
sender, AdCreatedEventArgs e)
    {
    //INTRODUCIR CODIGO
    }
```

XML

Para visualizar texto con formato XML, existe un Rich Control con el mismo nombre, capaz de renderizar en el HTML todo el contenido de un archivo, sin escribir una sola línea de código.

DocumentSource

Ésta es la propiedad más importante del control XML, que permite indicar el archivo por renderizar. Debemos recordar que, sin esta propiedad seteada, no veremos ningún código en la pantalla, y se generará una etiqueta HTML vacía.

Ejemplo

En el próximo ejemplo deseamos mostrar en nuestro sitio el siguiente código XML, para lo cual tendremos que insertarlo previamente en un archivo y configurar el control para que lo reconozca como documento.
Código XML del archivo Datos.xml:


```
<?xml version="1.0" encoding="utf-8"?>
<Datos>
 <Empleado>
  <Nombre>Juan</Nombre>
```

```
  <Apellido>Perez</Apellido>
  <Edad>34</Edad>
 </Empleado>
 <Empleado>
  <Nombre>Jose</Nombre>
  <Apellido>Lopez</Apellido>
  <Edad>45</Edad>
 </Empleado>
</Datos>
```

Código generado por Visual Web Developer, con la propiedad DocumentSource establecida:

```
<form id="form1" runat="server">
  <asp:Xml ID="Xml1" runat="server"
  DocumentSource="~/Datos.xml"></asp:Xml>
</form>
```

Aplicar transformaciones

Una manera sencilla de visualizar el código XML con formato es aplicando una transformación, es decir, interpretarlo para darle estilo de presentación. El siguiente código corresponde al archivo de transformación, Trans.xsl:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/
1999/XSL/Transform" version="1.0">
 <xsl:output method="html"/>
 <xsl:template match="/">
  <html>
   <head>
    <title>Registro de Empleados</title>
   </head>
   <body>
    <xsl:apply-templates />
   </body>
  </html>
 </xsl:template>
 <xsl:template match="Empleado">
  <h4>Empleado</h4>
  <hr color="white"/>
  <xsl:apply-templates />
 </xsl:template>
 <xsl:template match="Nombre">
  <b>Nombre: </b>
  <xsl:apply-templates />
  <br/>
 </xsl:template>
 <xsl:template match="Apellido">
  <b>Apellido:</b>
  <xsl:apply-templates />
  <br/>
 </xsl:template>
 <xsl:template match="Edad">
  <b>Edad:</b>
  <xsl:apply-templates />
  <br/>
 </xsl:template>
</xsl:stylesheet>
```

Una vez que creamos y guardamos nuestro archivo con la transformación, establecemos la propiedad TransformSource apuntando a él. El código de servidor del control se verá de la siguiente manera:

```
<asp:Xml ID="Xml1" runat="server"
DocumentSource="~/Datos.xml"
TransformSource="~/Trans.xsl"></asp:Xml>
```

El resultado será HTML con formato, basándose en las especificaciones de la transformación. El cuerpo HTML generado poseerá las etiquetas renderizadas de XSL:

```
 <body>
     <h4>Empleado</h4>
     <hr color="white">
        <b>Nombre: </b>Juan<br>
        <b>Apellido:</b>Perez<br>
        <b>Edad:</b>34<br>
  <h4>Empleado</h4>
  <hr color="white">
        <b>Nombre: </b>Jose<br>
        <b>Apellido:</b>Lopez<br>
        <b>Edad:</b>45<br>
</body>
```

Controles de usuario

Estos controles nos permiten definir nuestra propia combinación de código de servidor, HTML y Script.

Reutilizar, reutilizar, reutilizar

Una de las principales ventajas de la programación orientada a objetos es la llamada reutilización de código, mediante la cual podemos aplicar la misma solución a un problema cada vez que se nos presente, sea cual sea el proyecto en el que aparezca.

Un user control es una clase en sí misma, con su representación en código HTML y su propio código fuente para manejo de eventos, encapsulado en un archivo con extensión ASCX.

Características

Un control de usuario puede estar programado en cualquier lenguaje soportado por el .NET Framework, y poseer propiedades públicas y privadas, lo que nos permite configurarlo desde el Diseñador Visual de Visual Studio.

Todo user control debe tener definida su interfaz mediante código HTML (que puede incluir etiquetas de servidor), sin incluir los encabezados, como BODY o FORM, presentes en una página ASPX.

A su vez, debe tener definido el lenguaje, mediante la propiedad Language, como vemos en el siguiente código:

```
'USER CONTROL EN VISUAL BASIC.NET
<%@ Control Language="vb" %>

//USER CONTROL EN C#
<%@ Control Language="cs" %>
```

Creación de controles de usuario

Para crear un control de usuario seleccionamos la opción Agregar nuevo elemento, del menú contextual de nuestro Proyecto Web, tal como se ve en la Figura 28.

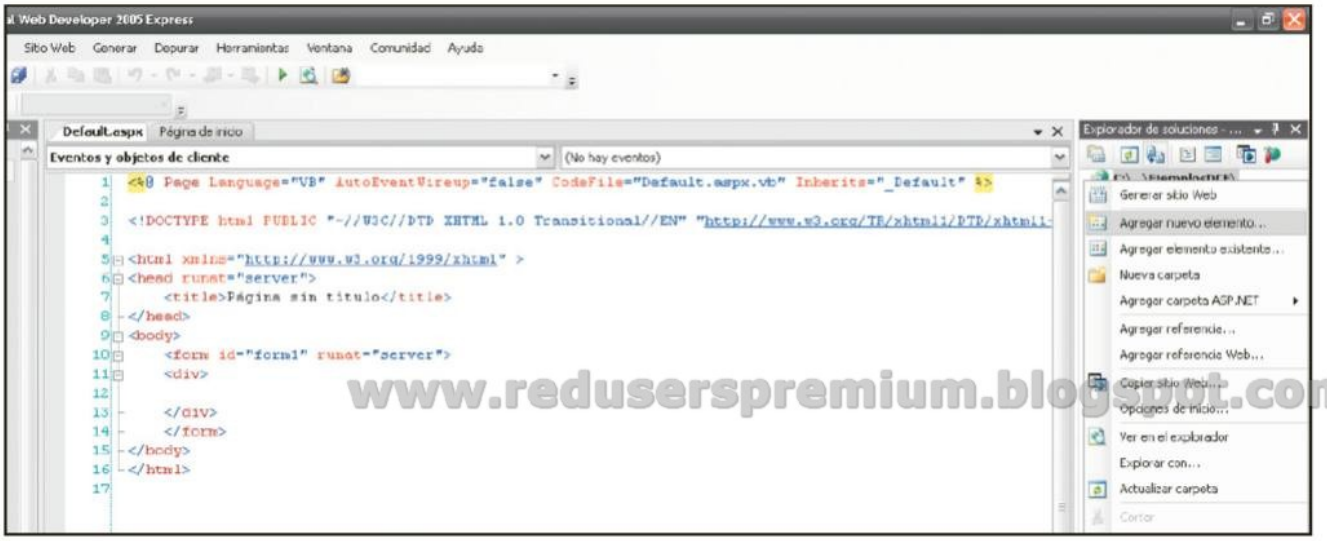


FIGURA 028 | Al hacer clic con el botón derecho del mouse en un proyecto Web, se despliega el menú contextual correspondiente.

Una vez abierto el cuadro de diálogo Agregar nuevo elemento, seleccionamos de la lista el elemento Control de usuario Web, y tras ponerle un nombre (en este ejemplo utilizaremos ucEjemplo.ascx), presionamos Agregar. Es importante seleccionar siempre la opción Colocar el código en un archivo independiente, para facilitar la codificación correspondiente. El formulario también nos solicita elegir un lenguaje; en este caso, optamos por C#, pero podría haber sido cualquiera de los dos que figuran en el combo de selección.

Finalizados estos pasos, se abrirá en el editor el código fuente (vacío) de nuestro user control, listo para ser completado con componentes visuales o Script:

```
<%@ Control Language="C#"
AutoEventWireup="true"
CodeFile="ucEjemplo.ascx.cs" Inherits=
"ucEjemplo" %>
```

Si cambiamos a la vista de diseño, veremos sólo una pantalla vacía, a la cual podremos arrastrar cualquiera de los controles mencionados anteriormente o, incluso, otros controles de usuario. Para nuestro ejemplo, vamos a utilizar dos controles TextBox (txtNumUno y txtNumDos), un control Label(lblResultado) y un control Button(btnSumar), con el objetivo de crear una pequeña calculadora que sume los valores ingresados en los cuadros de texto y muestre el resultado en la etiqueta. Nuestro user control se vería como muestra la Figura 30.

El código se verá de la siguiente manera:

```
<%@ Control Language="C#"
  AutoEventWireup="true"
CodeFile="ucEjemplo.ascx.cs" Inherits=
"ucEjemplo" %>
```

```
<asp:TextBox ID="txtNumUno"
runat="server"></asp:TextBox><br />
<asp:TextBox ID="txtNumDos"
runat="server"></asp:TextBox><br />
<asp:Button ID="btnSumar" runat="server"
```

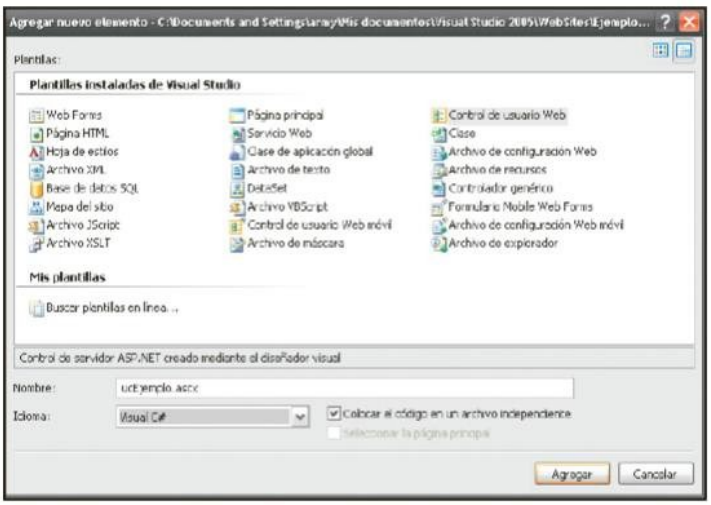


FIGURA 029 | Con esto finalizamos la creación de un nuevo UserControl. El lenguaje utilizado en este caso será C#.

El prefijo uc"NombreDeControl" es un estándar de nomenclatura para controles de usuario.

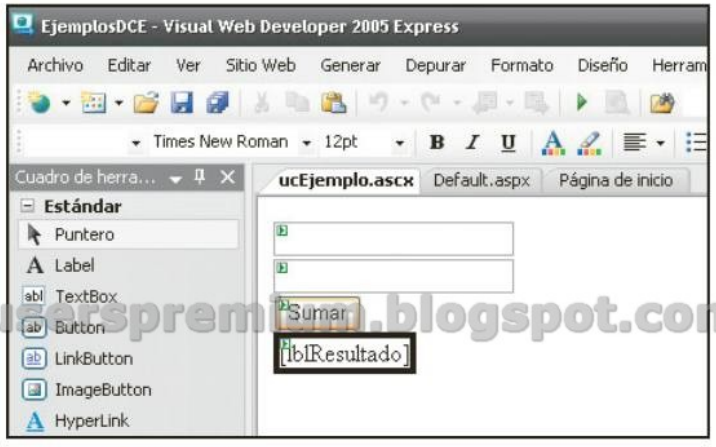


FIGURA 030 | Aquí vemos los controles configurados que componen nuestro control de usuario.

> Un mal manejo de errores puede convertir la tarea de depuración en una pesadilla.

```
Text="Sumar" /><br />
<asp:Label ID="lblResultado" runat="server"
BorderStyle="Solid"></asp:Label><br />
```

Cabe notar que asignamos las propiedades Text del botón btnSumar y el borde de la etiqueta lblResultado, para dar visibilidad a los datos.

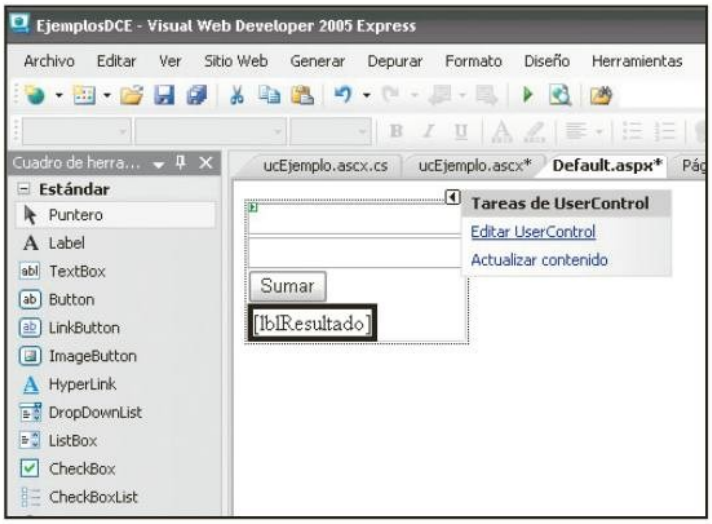


FIGURA 031 | Resultado de arrastrar nuestro control de usuario en Default.aspx, dentro del proyecto Web.

ⓘ Importancia de los prefijos

El uso de prefijos (txt para los cuadros de texto, lbl para las etiquetas, btn para los botones, etc.) simplifica la tarea de programar código, ya sea que tengamos 10 o 100 controles en una página.

Codificando

Es momento de escribir el código necesario para realizar la suma de los valores ingresados. Para hacerlo, capturamos el evento Click de nuestro btnSumar, y realizamos una sencilla rutina de suma entre los cuadros de texto, para establecer luego el resultado en la etiqueta correspondiente:

```
    protected void btnSumar_Click(object
    sender, EventArgs e)

    {
        int res;
        res = Int32.Parse(txtNumUno.Text) +
        Int32.Parse(txtNumDos.Text);
        lblResultado.Text = res.ToString();
    }
```

Todas las variables declaradas en los controles de usuario serán independientes de la página Web que los consuma, y no se mezclarán con distintas instancias del mismo control (en caso de que haya más de uno por página).

Consumir el control

Para utilizar concretamente nuestro user control de ejemplo, debemos incluirlo en una página ASPX, como si se tratara de un control cualquiera. Para hacerlo, arrastramos desde el Explorador de Soluciones el control ucEjemplo dentro de Default.aspx en nuestro proyecto Web.

En el código de la página podemos observar cómo se incluyen automáticamente dos etiquetas: una en el encabezado, para registrar el sufijo "uc1" en el resto de la página; y otra en donde está incrustado el user control:

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="Default.aspx.vb" Inherits="_
Default" %>
```

```
<%@ Register Src="ucEjemplo.ascx"
TagName="ucEjemplo" TagPrefix="uc1" %>

<!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>Página sin título</title>
</head>

<body>
    <form id="form1" runat="server">
    <div>
        <uc1:ucEjemplo id="UcEjemplo1"
        runat="server">
        </uc1:ucEjemplo></div>
    </form>
</body>

</html>
```

Probar el control

Es hora de probar el control de usuario creado, para lo cual ejecutamos nuestro sitio Web presionando la tecla <F5> o, directamente, haciendo clic en el comando Ejecutar de Visual Studio.

> Una vez programado el control, es posible incrustarlo en cualquier página en que lo deseemos.

Try & Catch y excepciones

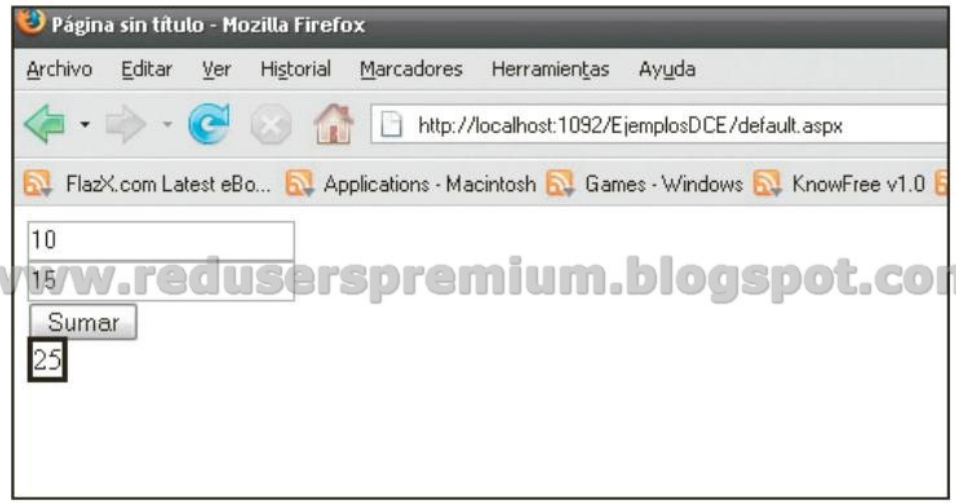
El mecanismo para capturar errores provisto por .NET, conocido como Try & Catch, es prácticamente "obligatorio" a la hora de programar controles de usuario.

En el supuesto escenario de tener varias instancias de nuestro ucEjemplo dentro de un mismo formulario ASPX, la captura de errores podría volverse un tormento si no está bien hecha.

Para esto, añadimos un bloque Try & Catch en el evento Click del btnSumar, con el objetivo de asegurarnos de que, ante cualquier error, se informe al usuario en la etiqueta lblResultado de lo sucedido:

```
    protected void btnSumar_Click(object
    sender, EventArgs e)
    {
        try
        {
```

FIGURA 032 | Probamos la suma de valores; funciona correctamente sumando los dos ingresados.



```
            int res;
            res = Int32.Parse(txtNumUno.Text)
            + Int32.Parse(txtNumDos.Text);
            lblResultado.Text =
            res.ToString();
        }
        catch (Exception ex)
        {
            lblResultado.Text = ex.Message +
            this.ID;
        }
    }
```

Propiedades

Al igual que los controles "de fábrica" de Visual Studio, los de usuario pueden tener propiedades públicas para configurar su comportamiento o estilo visual. A continuación, veremos cómo crear una propiedad enlazada con el color de fuente de la etiqueta de resultado en nuestro ucEjemplo. Para comenzar, generamos un nuevo Enum público con dos valores (Black y Red), fuera del ámbito de la clase, en el archivo ucEjemplo.ascx:

```
public enum Letra{Red,Black}
```

A continuación, creamos un campo privado llamado tipoLetra y una propiedad pública, TipoLetra, del tipo de dato Letra, dentro de la clase ucEjemplo:

```
    private Letra tipoLetra;
    public Letra TipoLetra
    {
        get
        {
            return tipoLetra;
        }
```

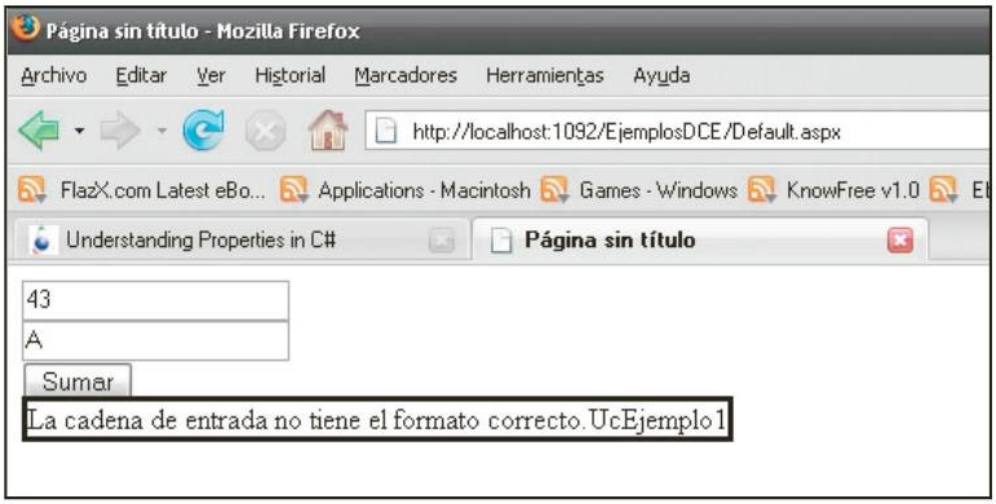


FIGURA 033 | Surge un error al ingresar un número en uno de los campos de texto, y esto es informado mediante el user control.

AL IGUAL QUE LOS CONTROLES "DE FABRICA" DE VISUAL STUDIO, LOS DE USUARIO PUEDEN TENER PROPIEDADES PÚBLICAS PARA CONFIGURAR SU COMPORTAMIENTO O ESTILO VISUAL.

```
        set
        {
            tipoLetra = value;
        }
    }
```

Una vez que guardamos este cambio, pasamos otra vez a la página por default, donde tenemos "incrustado" nuestro control de usuario. Si hacemos clic en él, en la ventana de Propiedades observaremos que se agregó la llamada TipoLetra, la cual nos permite seleccionar entre las opciones programadas.

Después de seleccionar la opción, el generador de Visual Studio escribe el código necesario para indicar al control cómo debe ser configurado:

```
<form id="form1" runat="server">

    <div>
        <uc1:ucEjemplo id="UcEjemplo1"
        runat="server" TipoLetra="Black">
        </uc1:ucEjemplo></div>

</form>
```

> El uso de propiedades corresponde a uno de los principios de la orientación a objetos: la encapsulación.

Más adelante, veremos cómo reflejar lo seleccionado en cada instancia de nuestro control de usuario a la hora de renderizarlo en HTML.

Acceder desde código

Como toda clase o control, sus propiedades son accesibles desde el código fuente.
A continuación, cambiaremos la propiedad TipoLetra en el evento Load de la página Default (en Visual Basic .NET), como se observa en la Figura 35.

Usos

Los controles de usuario se utilizan para una gran variedad de soluciones. Algunos casos prácticos son las grillas dinámicas, capaces de cargar contenido desde la base de datos

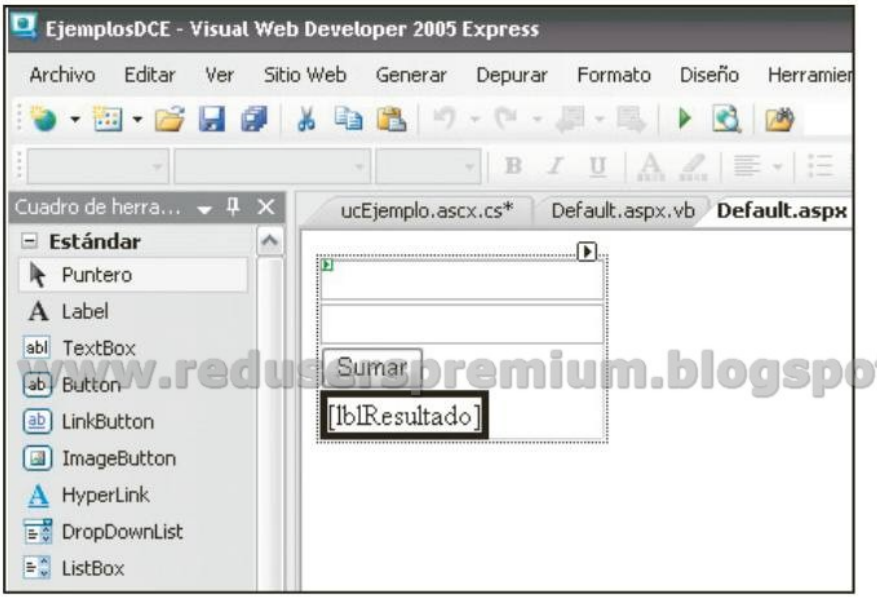


FIGURA 034 | Aquí podemos observar cómo responde el Diseñador Visual a un cambio hecho desde el código fuente.

según la fila/columna visible; o los famosos controles AJAX, parte de la Web 2.0 y una de las apuestas fuertes en ASP.NET para el futuro. También podemos recurrir a la múltiple combinación de controles de usuario en un mismo sitio, y así minimizar las líneas de código por escribir.

Encapsular funcionalidad

Uno de los principales objetivos a la hora de diseñar/codificar controles de usuario es dar la posibilidad de ser completamente independientes de la página en donde son ejecutados; es decir, incluir todas las propiedades públicas necesarias para que cualquier cambio al control no implique la modificación del código fuente.

Imágenes, scripts y demás

Cada control generado puede tener sus propias imágenes incluidas, y si bien debemos recordar que la visualización depende del espacio donde sea incrustado, un user control posee su propio HTML, separado del resto de la página.

En Internet

En la Web existe una gran cantidad de controles de usuario disponibles para la plataforma ASP.NET, tanto gratuitos como pagos, que pueden solucionar prácticamente cualquier problema que se nos plantee.

Siempre debemos estimar si el costo de desarrollar nuestro control de usuario será menor al de comprarlo o, al menos, investigar algunos minutos para ver si existe de manera gratuita en algún lado. Los controles pagos van desde gráficos vectoriales para presentación de datos y reportes, hasta chats en vivo con video incluido, pasando por foros, paneles, wizards, y más.

ⓘ Repositorio de controles

Es aconsejable destinar un directorio en nuestra carpeta de código fuente o proyectos a los controles de usuario, y depositar una copia de cada uno de ellos una vez finalizado el componente, incluyendo una breve descripción de para qué sirve y, si es necesario, un breve instructivo en formato DOC o TXT acerca de su modo de uso.

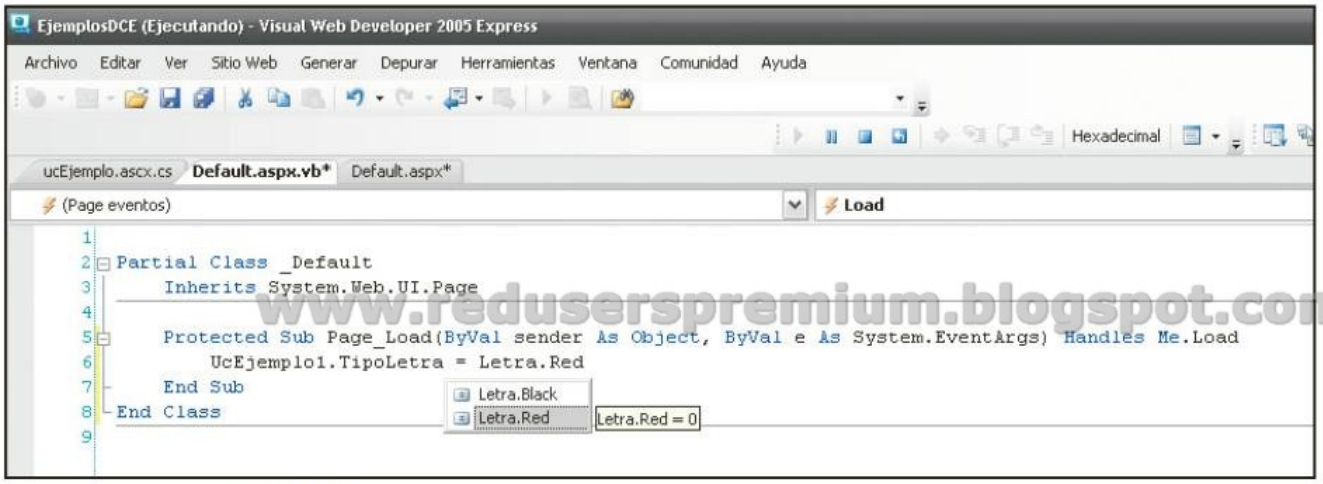


FIGURA 035 | Cambiamos la propiedad desde el código fuente. IntelliSense nos trae automáticamente las opciones Red y Black, definidas en el Enum con anterioridad.

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

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

8

Controles ricos

Campos de Texto - Calendario
Casillas de Verificación

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Controles de usuario

Creación - Instalación - Personalización

ISBN 978-987-1347-43-8
00008
9 789871 347438