



ASP.NET 3.5

CONTROLES WEB

CLIPSA



**DISTRIBUIDO POR:**

**CENTRO DE INFORMÁTICA PROFESIONAL S.L.**

C/ URGELL, 100  
08011 BARCELONA  
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10  
48014 BILBAO  
TFNO: 94 448 31 33

[www.cipsa.net](http://www.cipsa.net)

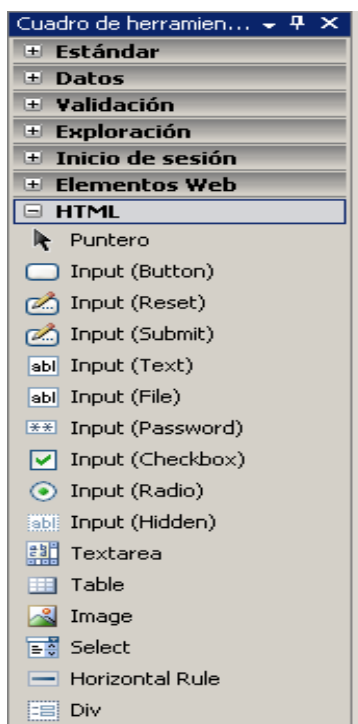
**RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR**

CLIPSA

# Controles Web

ASP.NET introduce un nuevo modelo de creación de páginas Web. En ASP las páginas se diseñaban en HTML convencional y después se añadía el código de servidor ASP. Esto provocaba que el diseño y la lógica de la página estuvieran entremezclados.

Una de las soluciones que aporta ASP.NET para evitar esta situación es el uso de controles de servidor. Estos controles se crean y manipulan como cualquier otro objeto de programación, se ejecutan en el servidor Web, mantienen su estado entre solicitudes de la misma página y generan de forma automática su código HTML para el navegador del cliente.



ASP.NET introduce dos tipos de controles de servidor: los controles HTML de servidor, y los controles Web de ASP.NET.

Los controles convencionales de HTML pueden emplearse perfectamente en una página ASP.NET. Sin embargo, estos controles resultan poco eficientes en comparación con los controles de servidor.

En Visual Studio para añadir controles HTML convencionales a nuestra página Web basta expandir la pestaña Controles HTML en la ventana Cuadro de Herramientas.

Los controles HTML pueden ejecutarse en el equipo del cliente o ser ejecutados en el servidor de la aplicación Web ( *control HTML de servidor* ).

## 1.1.- Control HTML de Cliente

Estos son los controles clásicos HTML. Todas las características del control y el código de sus eventos aparecen incrustados en el HTML que se envían al ordenador del cliente para que se ejecuten en su navegador.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Página sin título</title>
<script language="javascript" type="text/javascript">
function Text1_onclick( element ) {
    element.style.backgroundColor = 'silver'
}
</script>
</head>
<body>
<form id="form1" runat="server">
<div>
<input id="Text1" type="text" onclick="return Text1_onclick(this)"/></div>
</form>
</body>
```

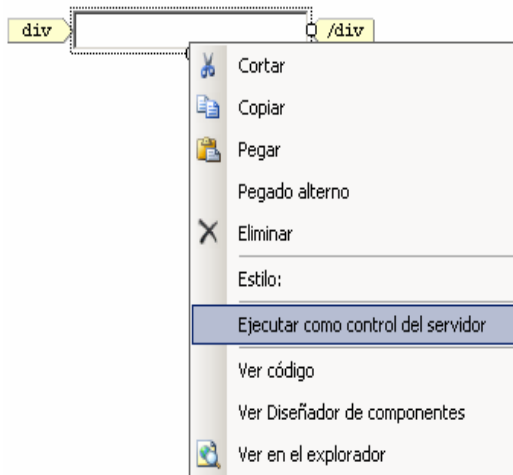
## CONTROLES WEB ASP.NET

`</html>`

En el ejemplo se observa el código de un control de cliente de tipo `InputBox`, junto con el código asociado a su evento `click` codificado en un script de javascript dentro también de la propia HTML. El script se ejecuta en el navegador del cliente y modifica el estilo del color de fondo del `InputBox` al ser pulsado.

Los controles HTML de cliente se envían al navegador tal y como aparecen en la página. Estos controles no mantienen su estado entre las recargas de la página, no son accesibles desde el código de servidor, y sus eventos se ejecutan exclusivamente en el navegador del cliente.

### 1.2.- Control HTML de Servidor



Los controles HTML pueden hacerse funcionar como controles de servidor para que sus propiedades y comportamiento para que mantengan su valor entre las recargas de la página, y puedan gestionarse desde el código de servidor sin depender del navegador del cliente.

Para modificar el modo de ejecución de un control HTML se selecciona la opción “Ejecutar como control de servidor” del menú contextual del control en la vista diseño

Esto provoca la aparición del atributo “`runat`” en el código HTML del control:

```
<input id="Text1" type="text" runat="server"/></div>
```

Los controles HTML de servidor, a diferencia de los de cliente, sí son accesibles desde el código de servidor mediante variables de instancia con el mismo nombre que el valor de su atributo “`Id`”. Las clases que representan los controles HTML del lado de servidor se agrupan en el espacio de nombre: *System.Web.UI.HtmlControls*.

namespace **System.Web.UI.HtmlControls**

Miembro de **System.Web**

#### Resumen:

El espacio de nombres `System.Web.UI.HtmlControls` contiene clases que permiten crear controles de servidor HTML en una página de formularios Web Forms. Los controles de servidor HTML se ejecutan en el servidor y se asignan directamente a etiquetas HTML estándar compatibles con la mayoría de los exploradores. Estas clases permiten controlar mediante programación los elementos HTML de una página de formularios Web Forms.

*Descripción del espacio de nombres del Examinador de Objetos del IDE del Visual Studio*

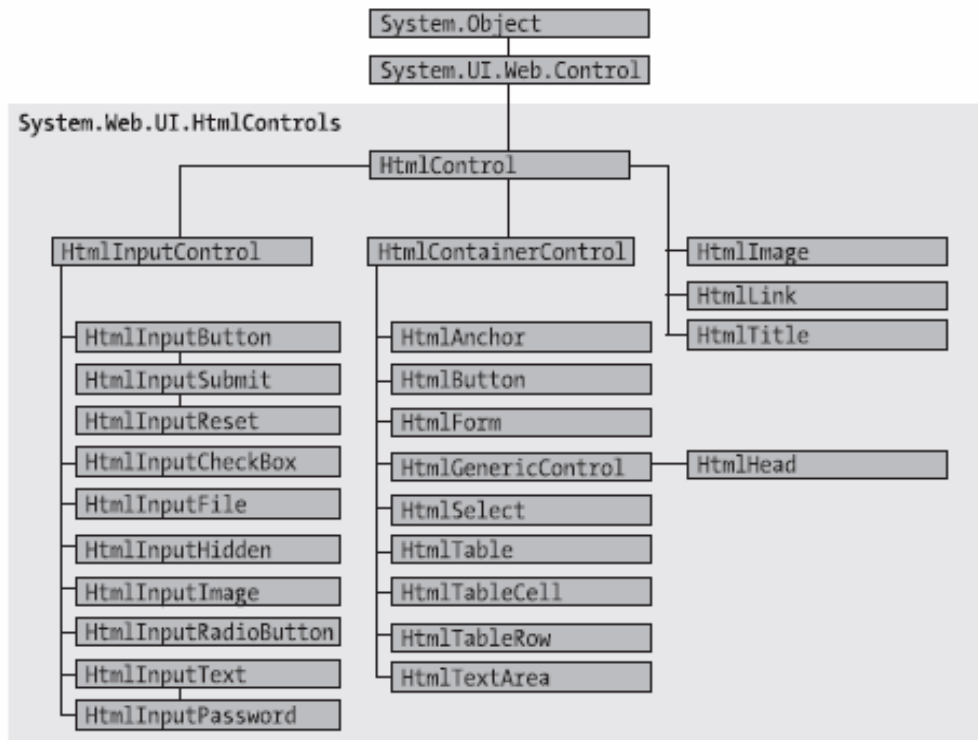
Si se desea modificar por ejemplo el color del fondo de un control HTML de tipo `InputBox`, es posible hacerlo mediante el siguiente código:

```
this.Text1.Style.Add(HtmlTextWriterStyle.BackgroundColor, "silver");
```

En esta línea de código, se alteran las propiedades del control HTML añadiéndole un estilo nuevo en tiempo de ejecución. Es importante resalta que el nombre de la instancia que representa el objeto en el código de servidor coincide con el valor de la propiedad *ID* del control expuesta en el código HTML de la propia página.

```
<input id="Text1" type="text" runat="server"/></div>
```

Cada uno de los controles HTML convencionales tiene una clase que lo representa dentro del espacio de nombres *System.Web.UI.HtmlControls*:



**Jerarquía de clases de controles HTML en el espacio de nombres *System.Web.UI.HtmlControls***

Los controles HTML de servidor proveen dos eventos que pueden atenderse desde el código de servidor: *ServerClick* y *ServerChange*.

El evento *ServerClick* se produce al hacer clic sobre el control, y es generado por la mayoría de los controles HTML. Dado que suele provocar además la recarga de la página; se trata de un evento que se trata inmediatamente. Este evento es producido por los controles: *HtmlAnchor*, *HtmlButton*, *HtmlInputButton*, *HtmlInputImage*, *HtmlInputReset*

El evento *ServerChange* responde a cambios en el texto o en la selección de controles de texto o listas. Debe tenerse en cuenta que este evento no se trata inmediatamente como el caso de *ServerClick*, dado que las acciones que lo provocan no producen recarga de la página y no son atendidos hasta que otra acción provoca la recarga de la misma.

Los controles que generan este evento son: *HtmlInputText*, *HtmlInputCheckBox*, *HtmlInputRadioButton*, *HtmlInputHidden*, *HtmlSelect*, *HtmlTextArea*.

### 2.- CONTROLES WEB DE ASP.NET

Los controles Web de ASP.NET son equivalentes a los controles HTML, pero proporcionan un amplio rango de eventos, propiedades y métodos que pueden ser manipulados desde código de servidor.

La mayoría de los controles Web son similares a los controles que pueden verse en un formulario de una típica aplicación de Windows, tales como botones, cajas de texto, casillas de verificación, ...etc.

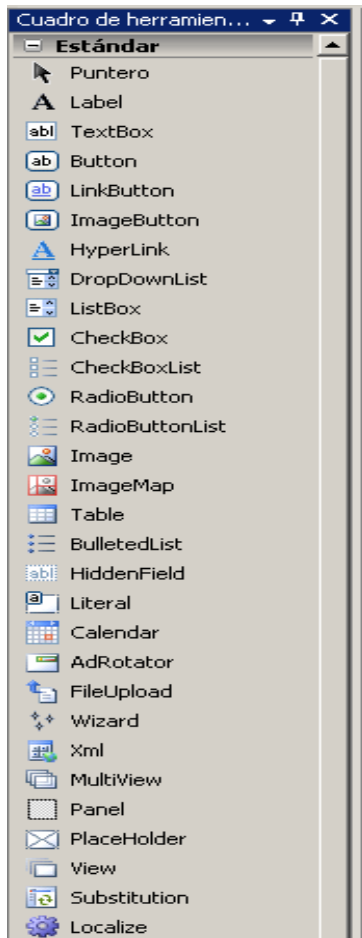
Cada control Web está representado por una clase cuyo nombre coincide con la del control equivalente para formularios, sin embargo; ambas clases están definidas en espacios de nombres diferentes. Por ejemplo; la clase `TextBox` definida en el espacio de nombres *System.Windows.Forms* representa un control de caja de texto para su uso en formularios de aplicaciones de escritorio convencionales. Existe sin embargo otra clase `TextBox` definida en el espacio de nombres *System.Web.UI.WebControls*, que representa también un control caja de texto pero para uso en páginas ASP.NET. Ambos controles también poseen además de mismos nombres, métodos y propiedades similares.

Los controles Web son sin embargo controles de servidor que los navegadores Web no reconocen, por lo que deben sustituirse por controles HTML equivalentes. El siguiente listado muestra los controles Web básicos y los controles HTML equivalentes en que se convierten al ser enviados al navegador del cliente:

Control Class	Underlying HTML Element
Label	<span>
Button	<input type="submit"> or <input type="button">
TextBox	<input type="text">, <input type="password">, or <textarea>
CheckBox	<input type="checkbox">
RadioButton	<input type="radio">
Hyperlink	<a>
LinkButton	<a> with a contained <img> tag
ImageButton	<input type="image">
Image	<img>
ListBox	<select size="X"> where X is the number of rows that are visible at once
DropDownList	<select>
CheckBoxList	A list or <table> with multiple <input type="checkbox"> tags
RadioButtonList	A list or <table> with multiple <input type="radio"> tags
BulletedList	An <ol> ordered list (numbered) or <ul> unordered list (bulleted)
Panel	<div>
Table, TableRow, and TableCell	<table>, <tr>, and <td> or <th>



Existen multitud de controles Web, de manera que pueden englobarse en diferentes conjuntos según su función:



- Controles Básicos
- Controles de tipo Lista
- Controles de Validación
- Controles Avanzados
- Controles de Acceso a Datos

Los controles Web pueden añadirse a las páginas seleccionándolos y arrastrándolos desde el Cuadro de Herramientas a la página en el IDE del Visual Studio.

Estos controles aparecen reunidos dentro de la pestaña "Estandar".

Cuando un control Web se añade al diseño de una página ASP.NET, automáticamente aparece reflejado en el código HTML de la misma:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Página sin título</title>
</head>
<body>
<form id="form1" runat="server">
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
</form>
</body>
</html>
```

Una característica fundamental de los controles Web es que sólo existen en el servidor Web. Por ello, éstos controles se transforman en código HTML, CSS, y Javascript cuando la página es procesada. Este proceso se denomina renderizado.

La renderización de controles Web en controles HTML durante el procesado de la página es adaptativa y se realiza en función de las capacidades y tipo de navegador Web empleado que solicitó la página.

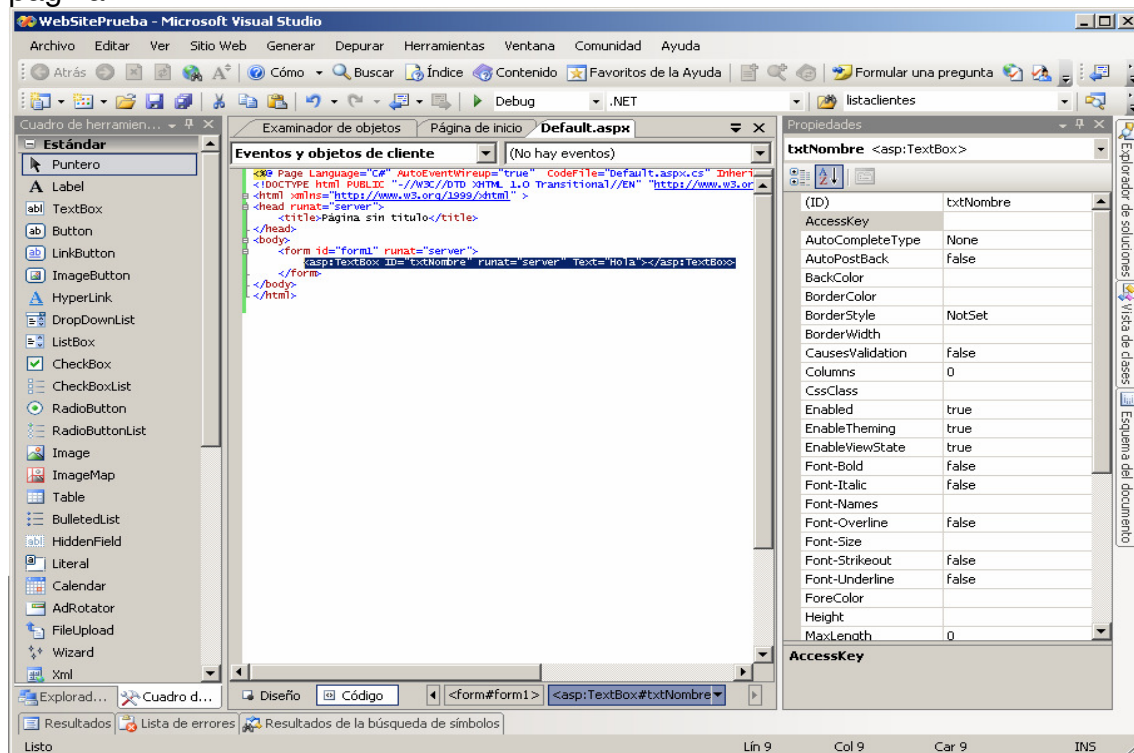
Los controles Web aparecen reflejados en el diseño de la página mediante etiquetas del tipo `<asp:XXXXX>`. Por ejemplo; el control web TextBox (equivalente al control HTML InputBox ), se representa con la etiqueta `<asp:TextBox>`

```
<asp:TextBox ID="txtNombre" runat="server" Text="Hola"></asp:TextBox>
```

Las diferentes propiedades del control aparecen como atributos de la propia etiqueta. El atributo Text muestra el valor de la propiedad Text del control TextBox que designa el texto mostrado inicialmente en el campo de texto.

## CONTROLES WEB ASP.NET

Las propiedades de los controles Web pueden ser asignadas en el diseño mediante la ventana de propiedades del Visual Studio seleccionando el control en la vista de diseño de la página, o su etiqueta en la vista de código de la página:



**Control Web seleccionado en vista de diseño con sus propiedades mostradas en la ventana de propiedades**

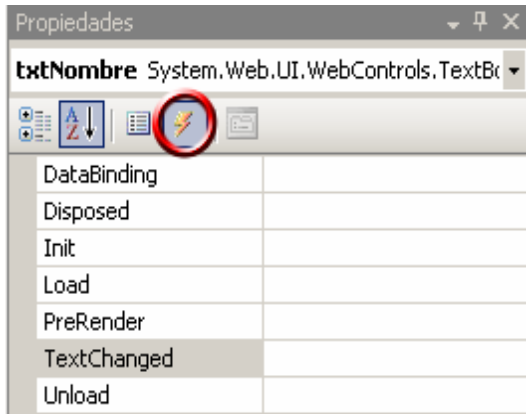
La propiedad principal de los controles Web es *ID*, que determina el nombre del identificador de cada control. El nombre de la variable de instancia asociada a un control coincide con el nombre de la variable que lo referencia desde el código de servidor.

Por ejemplo: Si queremos cambiar desde código el contenido de la caja de texto que acabamos de definir cuyo *ID* es "txtNombre", debemos incluir el siguiente código en el fichero de código asociado a la página:

```
protected void Page_Load(object sender, EventArgs e)
{
    txtNombre.Text = "HOLA";
}
```

La variable de instancia "txtNombre" referencia al control TextBox cuya propiedad ID tiene el valor "txtNombre". Esta instancia referencia a un objeto de la clase TextBox dentro del espacio de nombres *System.Web.UI.WebControls*. No confundir con la clase TextBox del espacio de nombres *System.Windows.Forms*, que se emplea en los formularios de las aplicaciones Windows convencionales fuera del entorno Web.

Los controles Web también tienen la capacidad de producir eventos al igual que los controles HTML. En estos controles aunque los eventos se desencadenan en el navegador del cliente, el evento y el método de respuesta se implementan y ejecutan en el servidor exclusivamente.

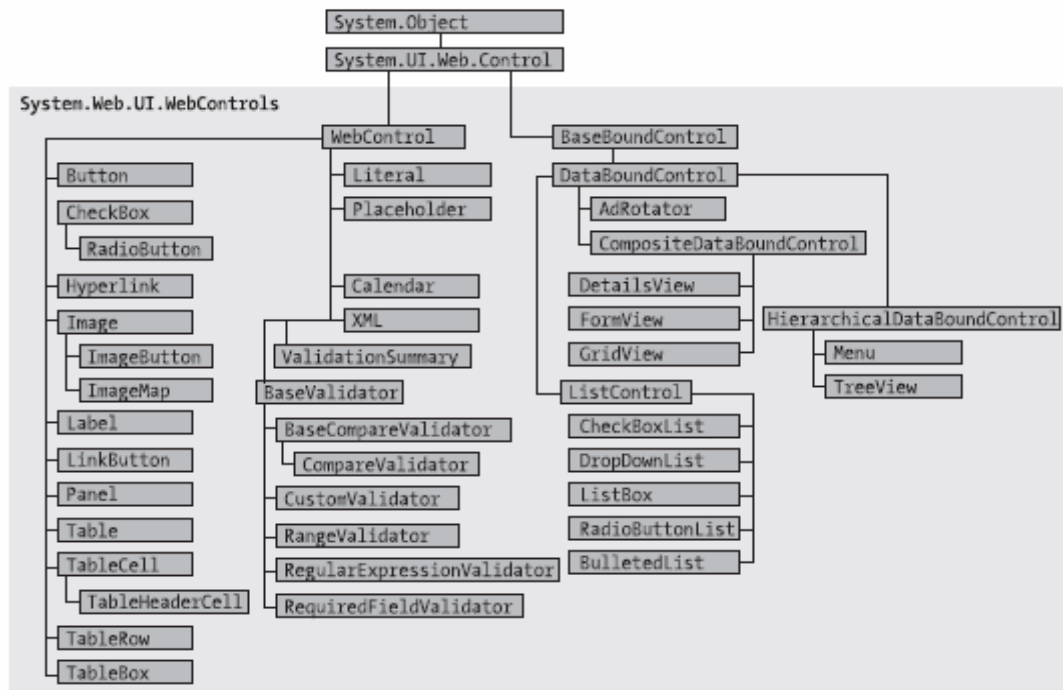


Los eventos que un control Web detecta pueden verse pulsando el icono con forma de rayo en la ventana de propiedades de un control.

Seleccionando cualquiera de ellos y haciendo doble clic provocaremos la codificación del evento. Con ello se crea el método correspondiente para atenderlo en el código de servidor de la página.

## 2.1.- JERARQUIA DE CONTROLES WEB

Los controles Web están definidos todos dentro del espacio de nombres `System.Web.UI.WebControls`.



*Jerarquía de clases de controles HTML en el espacio de nombres `System.Web.UI`*

La clase padre de la jerarquía de controles Web es *WebControl*, la cual a su vez hereda de otra clase superior llamada *Control* definida fuera del espacio de nombres. La clase *Control* define propiedades básicas como *Id*, *EnableViewState*, *Parent*, al igual que métodos y eventos que son comunes a todos los controles de servidor y a las propias páginas Web.

La clase *WebControl* hereda de *Control*, y define por su parte propiedades, métodos y eventos comunes exclusivamente a la mayoría de los controles Web excepto *Literal*, *Placeholder*, *Xml* y *Repeater*. A partir de las propiedades, métodos y eventos definidos por *WebControl* cada control añade aparte los suyos propios específicos.

## CONTROLES WEB ASP.NET

La siguiente lista muestra las principales propiedades definidas por la clase `WebControl` que son comunes a todos los controles Web.

- **BackColor, ForeColor, BorderColor:** Controlan el color de los bordes, el fondo y el texto que aparece inscrito en los controles.
- **BorderWidth, BorderStyle:** Permite indicar el grosor y el tipo de borde mostrado alrededor del control.
- **Enable:** Permite indicar si el control es interactivo. Si se deshabilita asignando el valor `False`, el control es visible pero no es manipulable por el usuario.
- **Visible:** Permite indicar si el control es visible o no para el usuario en el navegador. Si se deshabilita asignando un valor `False`, el control no se inscribe en el código HTML enviado como respuesta al navegador del cliente.
- **Font:** Permite indicar el tipo de fuente a emplear en el texto mostrado por el control.
- **Height, Width:** Permite indicar el alto y ancho del control.
- **Page:** Contiene la instancia de la página que contiene el control.
- **TabIndex:** Permite indicar un valor que determina el orden en que cada control recibe el foco al pulsar el usuario la tecla tabulador.
- **ToolTip:** Permite indicar un mensaje que es mostrado cuando el ratón pasa por encima del control en el navegador del usuario.

La siguiente lista muestra como ejemplo los eventos definidos por la clase `System.Web.UI.Control`. Estos eventos están presentes en todos los controles Web al igual que en la propia clase `Page` ya todas las clases heredan igualmente de la clase `Control`.

	Nombre	Descripción
⚡	<code>DataBinding</code>	Se produce cuando el control de servidor se enlaza a un origen de datos. (Se hereda de <code>Control</code> ).
⚡	<code>Disposed</code>	Se produce cuando un control de servidor se libera de la memoria, lo que constituye la última fase del período de duración de un control de servidor cuando se solicita una página ASP.NET. (Se hereda de <code>Control</code> ).
⚡	<code>Init</code>	Tiene lugar al inicializar el control de servidor, que es el primer paso en su ciclo de vida. (Se hereda de <code>Control</code> ).
⚡	<code>Load</code>	Se produce cuando el control de servidor se carga en el objeto <code>Page</code> . (Se hereda de <code>Control</code> ).
⚡	<code>PreRender</code>	Se produce una vez que se carga el objeto <code>Control</code> , pero antes de su representación. (Se hereda de <code>Control</code> ).
⚡	<code>Unload</code>	Se produce cuando el control de servidor se descarga de la memoria. (Se hereda de <code>Control</code> ).

## 2.2.- CONTROLES WEB BASICOS

A continuación se enumeran los controles Web fundamentales:

### **2.2.1.- Label ( *System.Web.UI.WebControls.Label* )**

Se emplea para mostrar texto manipulando el valor de su propiedad *Text*, que es la única que no hereda de la clase *WebControl*. Durante la generación del HTML de respuesta para el usuario; este control genera una etiqueta `<span>` con los estilos ( atributo *style* ), asociados para mostrar las características de color, tipo, tamaño de fuentes... etc, fijadas para el control Web. Si se desea emplear una clase definida en una hoja de estilos CSS, se debe asignar el nombre a la propiedad *CssClass*.

#### Control Web

```
<asp:Label ID="Label1"
    runat="server"
    Font-Bold="True"
    Font-Names="Arial Black"
    ForeColor="#C00000"
    Text="HOLA MUNDO">
</asp:Label>
```

#### Código HTML

```
<span id="Label1"
    style="color:#C00000;font-family:Arial Black;font-weight:bold;">HOLA MUNDO
</span>
```

### **2.2.2.- TextBox ( *System.Web.UI.WebControls.TextBox* )**

Se empleo para la entrada y visualización de texto al usuario. Se puede configurar para que permita una sola línea de texto, o varias mediante la propiedad *TextMode*, *Rows* y *Columns*. Se puede configurar para que no permita en un momento dado la escritura por parte del usuario mediante la propiedad *ReadOnly*, y para que admite un máximo de caracteres mediante *MaxLength*.

Este control implementa la interfaz *ITextControl* junto con los controles *Label*, y *DropDownList*, incluyendo todos ellos la propiedad *Text*, que hace referencia al contenido visual del control. Además de los eventos heredados de *WebControl*, incluye el evento propio *TextChanged* que se dispara al cambiar el texto que contiene. Este evento no provoca la recarga de la página, a menos que se asigne el valor 'true' a la propiedad *AutoPostBack* del control.

#### Control Web

```
<asp:TextBox ID="txtNombre"
    runat="server"
    BackColor="#E0E0E0">Hola Mundo!
</asp:TextBox>
```

#### Código HTML

```
<input name="txtNombre" type="text" value="Hola Mundo!" id="txtNombre"
    style="background-color:#E0E0E0;" />
```

### 2.2.3.- HiddenField ( *System.Web.UI.WebControls.HiddenField* )

El control HiddenField es un control sin apariencia que resulta invisible en el diseño de la página. Se emplea para almacenar información que se desea conservar entre distintas instancias de la misma página.

Su principal propiedad es *Value* que permite obtener y establecer el valor almacenado en el control. Este valor debe ser una cadena de texto. El control también posee el evento *ValueChanged* que se ejecuta cuando cambia el valor del control. Este evento no provoca una recarga inmediata de la página, y el control no posee la propiedad *AutoPostBack*.

#### Control Web

```
<asp:HiddenField ID="HiddenField1"
runat="server"
value="012345" />
```

#### Control HTML

```
<input type="hidden"
name="HiddenField1"
id="HiddenField1"
value="caca" />
```

Los campos ocultos son empleados por ASP.NET para mantener la información de estado y control de eventos entre instancias distintas de una misma página. Estos aparecen en el código de respuesta de las páginas ASP.NET con los nombres “\_\_VIEWSTATE” y “\_\_EVENTVALIDATION”.

#### Código HTML

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDWULLTEZmJiYNDk3NDFkZFpw608vhNpOY3cqMPg2v+J6cD/B" />
<input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/wEWAwLmgIqTBwLS0bLrBgLS0fbZDPHoalQExwgaNRE1/hBEhkPb8qY0" />
```

### 2.2.4.- Button ( *System.Web.UI.WebControls.Button* )


Los botones son controles que envían el formulario al servidor provocando las recargas de las páginas, y el procesamiento de los eventos pendientes en el lado del servidor. Existen tres tipos de controles de tipo botón en ASP.NET: *Button*, *LinkButton*, e *ImageButton*.

El control ***LinkButton*** se representa como un hipervínculo en el navegador del cliente. La URL de la página destino del hipervínculo se indica mediante la propiedad *PostBackURL* del control.

El control ***ImageButton*** se representa mediante una imagen de mapa de bits, admitiendo un texto explicativo mediante la propiedad *AlternateText*.



Además de los eventos heredados de WebControls, los tres tipos de controles botón implementan la interfaz `IButtonControl`, por lo que comparten las siguientes propiedades y eventos:

Nombre	Descripción
 <code>CausesValidation</code>	Obtiene o establece un valor que indica si al hacer clic en el botón se produce la validación de la página.
 <code>CommandArgument</code>	Obtiene o establece un argumento opcional que se propaga al evento <code>Command</code> .
 <code>CommandName</code>	Obtiene o establece el nombre de comando que se propaga al evento <code>Command</code> .
 <code>PostBackUrl</code>	Obtiene o establece la dirección URL de la página Web destino del envío desde la página actual cuando se hace clic en el control de botón.
 <code>Text</code>	Obtiene o establece el título de texto que se muestra para el botón.
 <code>ValidationGroup</code>	Obtiene o establece el nombre del grupo de controles cuya validación causa el control de botón cuando devuelve datos al servidor.

Los eventos específicos de los tres controles son *Click* y *Command*.

El evento `Click` responde a la pulsación del botón cuando las propiedades `CommandName` y `CommandArgument` no se emplean.

El evento `Command` responde también a la pulsación del botón incluyendo un segundo parámetro de tipo `CommandEventArgs` que permite recuperar los valores de las propiedades `CommandName` y `CommandArgument` del botón origen del evento:

```
protected void btnOk_Click(object sender, EventArgs e)
{
}

protected void btnOk_Command(object sender, CommandEventArgs e)
{
    String ButtonCommandName = e.CommandName;
    String ButtonCommandArguments = e.CommandArgument.ToString();
}
```

El uso del evento `Command` puede emplearse para administrar con el mismo método varios botones de una página Web utilizando las propiedades `CommandName` y `CommandArgument` para identificar el control origen del evento.

En el caso del control `ImageButton` el evento `Click` devuelve como segundo parámetro un objeto de tipo `ImageClickEventArgs` que permite obtener las coordenadas de la imagen del botón en las que se hizo clic el usuario.

```
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    int coordenadaX = e.X;
    int coordenadaY = e.Y;
}
```

## Control Web

```
<asp:Button ID="btnOk"
    runat="server"
    OnClick="btnOk_Click"
    Text="Button"
    CommandArgument="argumento"
    CommandName="nombre"
    OnCommand="btnOk_Command" />

<asp:ImageButton ID="ImageButton1"
    runat="server"
    OnClick="ImageButton1_Click"
    OnCommand="ImageButton1_Command" />
```

# CONTROLES WEB ASP.NET

## Código HTML

```
<input type="submit"
       name="btnOk"
       value="Button"
       id="btnOk" />

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

### 2.2.5.- HyperLink ( *System.Web.UI.WebControls.HyperLink* )

El control HyperLink genera siempre un hipervínculo HTML común para el cliente. La URL de la página de destino debe indicarse mediante la propiedad *NavigateURL*. La propiedad *Text* que permite declarar el texto que muestra el vínculo en el navegador.

La propiedad *ImageUrl* permite indicar la dirección URL de un fichero de imagen que aparezca en el hipervínculo. En este caso de declararse ambas propiedades; la imagen se muestra sobre el texto, salvo cuando la imagen se descarga; o no está disponible.

#### Control Web

```
<asp:HyperLink ID="HyperLink1" runat="server"
NavigateUrl="~/Test.aspx">HyperLink</asp:HyperLink>
```

#### Código HTML ( generado por un control Hyperlink )

```
<a id="HyperLink1" href="Test.aspx">HyperLink</a>
```

El control HyperLink cumple la misma función que el control LinkButton; ambos provocan un salto a otra página. La diferencia está en que LinkButton provoca una llamada al servidor para que envíe la página de destino. En cambio; el control HyperLink provoca un salto directo a la dirección URL indicada en su propiedad *NavigateURL* sin enviar de información al servidor. Por este motivo el control HyperLink no posee eventos de servidor como *Click* y *Command*.

#### Código HTML ( generado por un control LinkButton )

```
<script type="text/javascript">
//
var theForm = document.forms['form1'];
if (!theForm) {
    theForm = document.form1;
}
function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}
//]]&gt;
&lt;/script&gt;

&lt;a id="LinkButton1" href="javascript:WebForm_DoPostBackWithOptions(
    new WebForm_PostBackOptions(&amp;quot;LinkButton1&amp;quot;, &amp;quot;&amp;quot;, false,
    &amp;quot;&amp;quot;, &amp;quot;Test.aspx&amp;quot;, false, true))"&gt;LinkButton&lt;/a&gt;</pre></div><div data-bbox="138 857 862 909" data-label="Text"><p>El control LinkButton genera código JavaScript para forzar el refresco de la página en el servidor desde el navegador del cliente, por lo que no funcionará adecuadamente si el navegador de cliente no tiene habilitado Javascript.</p></div><div data-bbox="138 936 166 955" data-label="Page-Footer"><p>16</p></div>
```

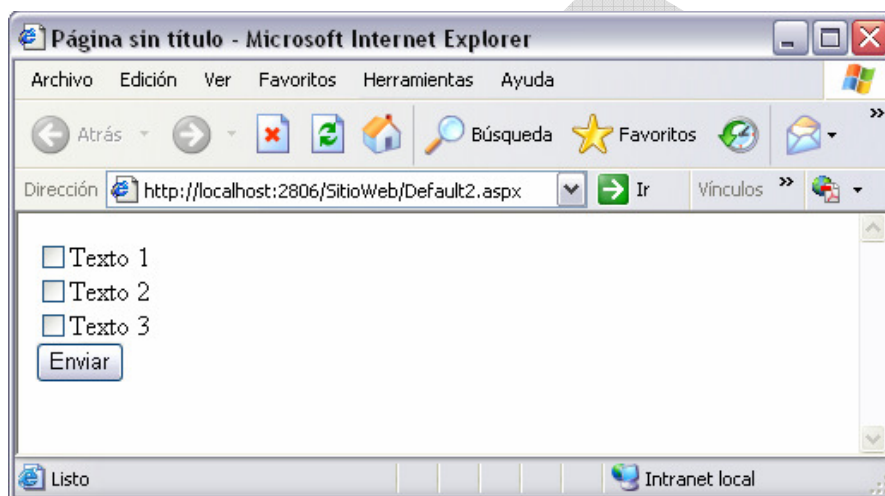


### **3.- CONTROLES WEB DE SELECCIÓN DE VALORES**

Existen varios controles Web en ASP.NET que están diseñados para permitir al usuario seleccionar un valor entre varios mostrados. Estos controles son *CheckBox* y *RadioButton*.

#### **3.1.- CheckBox ( *System.Web.UI.WebControls.CheckBox* )**

El control *CheckBox* permite realizar una selección sobre un conjunto de elementos representados por controles *CheckBox*. Cada control sólo admite dos estados; seleccionado o no seleccionado. La propiedad *Text* del control permite asociarle un texto descriptivo. La propiedad *Checked* indica con un valor booleano si el control está o no seleccionado.



El control posee además el evento propio *CheckedChanged* que se activa cuando se detecta desde el servidor un cambio en el estado del control. Este evento no supone una recarga automática de la página salvo que se asigne el valor TRUE a la propiedad *AutoPostBack*.

```
protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
{
    if (CheckBox1.Checked)
        Response.Write("CheckBox1 seleccionado.");
    else
        Response.Write("CheckBox1 no seleccionado.");
}
```

El control *CheckBox* se convierte al procesarse la página ASP.NET, en un control `<input>` de tipo "checkbox", seguido de un control `<label>` para el texto identificativo.

#### **Control Web**

```
<asp:CheckBox ID="CheckBox1"
    runat="server"
    OnCheckedChanged="CheckBox1_CheckedChanged"
    Text="Texto 1" />
```

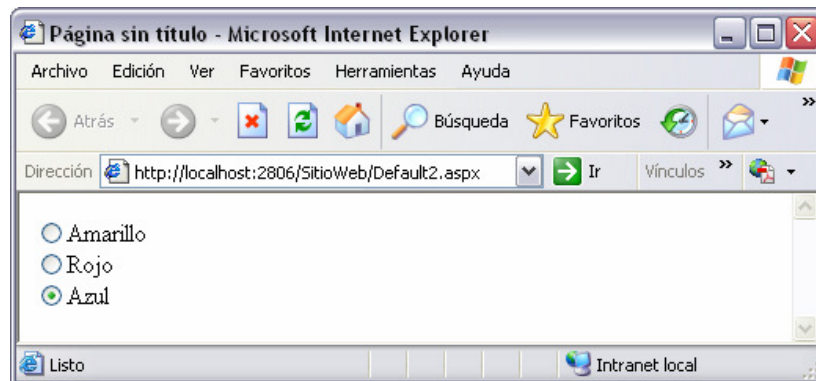
#### **Código HTML**

```
<input id="CheckBox1"
    type="checkbox"
    name="CheckBox1" />
<label for="CheckBox1">Texto 1</label>
```

### **3.2.- RadioButton ( *System.Web.UI.WebControls.RadioButton* )**

Este control es semejante al control CheckBox con la diferencia de que los controles RadioButton se agrupan de forma que sólo un control del grupo puede estar seleccionado al mismo tiempo.

El agrupamiento de varios RadioButton se lleva a cabo estableciendo el mismo valor en la propiedad *GroupName* de los controles. Al igual que el control CheckBox también posee las propiedades *Checked* y *Text* para determinar su estado de selección y el texto descriptivo respectivamente.



El evento *CheckedChanged* también está presente en este control, y no produce la recarga automática de la página, a menos que se asigne el valor TRUE a su propiedad *AutoPostBack*.

El control RadioButton se convierte al ser procesado en un control HTML `<Input>` de tipo "radio", seguido de un control `<Label>` con el texto descriptivo.

#### **Control Web**

```
<asp:RadioButton ID="RadioButton1" runat="server" GroupName="Colores" Text="Amarillo" />  
<asp:RadioButton ID="RadioButton2" runat="server" GroupName="Colores" Text="Rojo"/>  
<asp:RadioButton ID="RadioButton3" runat="server" GroupName="Colores" Text="Azul" />
```

#### **Código HTML**

```
<input id="RadioButton1" type="radio" name="Colores" value="RadioButton1" /><label  
for="RadioButton1">Amarillo</label>  
<input id="RadioButton2" type="radio" name="Colores" value="RadioButton2" /><label  
for="RadioButton2">Rojo</label>  
<input id="RadioButton3" type="radio" name="Colores" value="RadioButton3" /><label  
for="RadioButton3">Azul</label>
```






## 4.- CONTROLES WEB DE TIPO LISTA

En ASP.NET se distinguen cinco tipos de controles Web diseñados para la selección de uno o varios elementos dentro de una lista de opciones. Estos son:

- \* ***ListBox***
- \* ***CheckBoxList***
- \* ***RadioButtonList***
- \* ***DropDownList***
- \* ***BulletedList***





Todos estos controles heredan de la misma clase padre *ListControl*. Esta clase contiene los principales métodos y propiedades de todos los controles de tipo lista disponibles en ASP.NET.

Cada control de tipo lista contiene internamente una colección con los elementos que el usuario pueda seleccionar. Estos elementos son instancias de la clase *System.Web.UI.WebControls.ListItem* que define las siguientes propiedades:

Nombre	Descripción
 <b>Attributes</b>	Obtiene una colección de pares de nombre y valor de atributo para la clase <b>ListItem</b> que ésta no admite directamente.
 <b>Enabled</b>	Obtiene o establece un valor que indica si el elemento de lista está habilitado.
 <b>Selected</b>	Obtiene o establece un valor que indica si el elemento está seleccionado.
 <b>Text</b>	Obtiene o establece el texto mostrado en el control de lista para el elemento representado por el control <b>ListItem</b> .
 <b>Value</b>	Obtiene o establece el valor asociado a <b>ListItem</b> .

Cada elemento de un control lista se diferencia básicamente por un texto a mostrar ( propiedad *Text* ), y una cadena de texto con la información que representa ese elemento ( propiedad *Value* ).

Las propiedades *SelectedIndex*, *SelectedItem*, y *SelectedValue* permiten obtener el elemento, o conjunto de elementos seleccionados por el usuario:

 <b>Items</b>	Obtiene la colección de elementos del control de lista.
 <b>SelectedIndex</b>	Obtiene o establece el índice ordinal inferior de los elementos seleccionados en la lista.
 <b>SelectedItem</b>	Obtiene el elemento seleccionado con el índice inferior en el control de lista.
 <b>SelectedValue</b>	Obtiene el valor del elemento seleccionado en el control de lista o selecciona el elemento en el control de lista que contiene el valor especificado.

## CONTROLES WEB ASP.NET

El conjunto de los elementos mostrados en un control de tipo Lista se agrupan en una colección de la clase `ListItemCollection`. Esta colección es accesible mediante la propiedad `Items` de la clase `ListControl`. La clase `ListItemCollection` define los siguientes métodos para manipular los elementos `ListItem` contenidos.

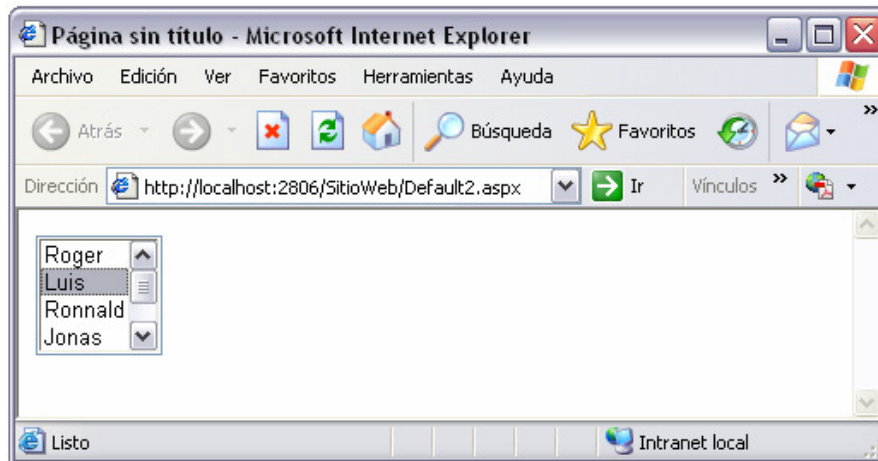
	Nombre	Descripción
◆	<code>Add</code>	Sobrecargado. Agrega un <code>ListItem</code> al final de la colección.
◆	<code>AddRange</code>	Agrega a la colección los elementos de la matriz de objetos <code>ListItem</code> .
◆	<code>Clear</code>	Quita todos los objetos <code>ListItem</code> de la colección.
◆	<code>Contains</code>	Determina si la colección contiene el elemento especificado.
◆	<code>CopyTo</code>	Copia los elementos de <code>ListItemCollection</code> en el objeto <code>System.Array</code> especificado, empezando por el índice especificado.
◆	<code>Equals</code>	Sobrecargado. Determina si dos instancias de <code>Object</code> son iguales. (Se hereda de <code>Object</code> ).
◆	<code>FindByText</code>	Busca en la colección un <code>ListItem</code> cuya propiedad <code>Text</code> contenga el texto especificado.
◆	<code>FindByValue</code>	Busca en la colección un <code>ListItem</code> cuya propiedad <code>Value</code> contenga el valor especificado.
◆	<code>GetEnumerator</code>	Devuelve un objeto <code>System.Collections.IEnumerator</code> implementado que contiene todos los objetos <code>ListItem</code> de <code>ListItemCollection</code> .
◆	<code>GetHashCode</code>	Actúa como función hash para un tipo concreto. (Se hereda de <code>Object</code> ).
◆	<code>GetType</code>	Obtiene el objeto <code>Type</code> de la instancia actual. (Se hereda de <code>Object</code> ).
◆	<code>IndexOf</code>	Determina el valor del índice que representa la posición del <code>ListItem</code> especificado en la colección.
◆	<code>Insert</code>	Sobrecargado. Inserta un <code>ListItem</code> en la colección, en la ubicación del índice especificado.
◆ S	<code>ReferenceEquals</code>	Determina si las instancias de <code>Object</code> especificadas son la misma instancia. (Se hereda de <code>Object</code> ).
◆	<code>Remove</code>	Sobrecargado. Quita un objeto <code>ListItem</code> de la colección.
◆	<code>RemoveAt</code>	Quita el <code>ListItem</code> en el índice especificado de la colección.
◆	<code>ToString</code>	Devuelve una clase <code>String</code> que representa la clase <code>Object</code> actual. (Se hereda de <code>Object</code> ).

La clase `ListControl` implementa además un evento adicional para todos los controles de tipo lista; el evento `SelectedIndexChanged`. Este evento se produce cuando el usuario selecciona un elemento de la lista y no produce la recarga automática de la página, a menos que se asigne el valor `TRUE` a su propiedad `AutoPostBack`.

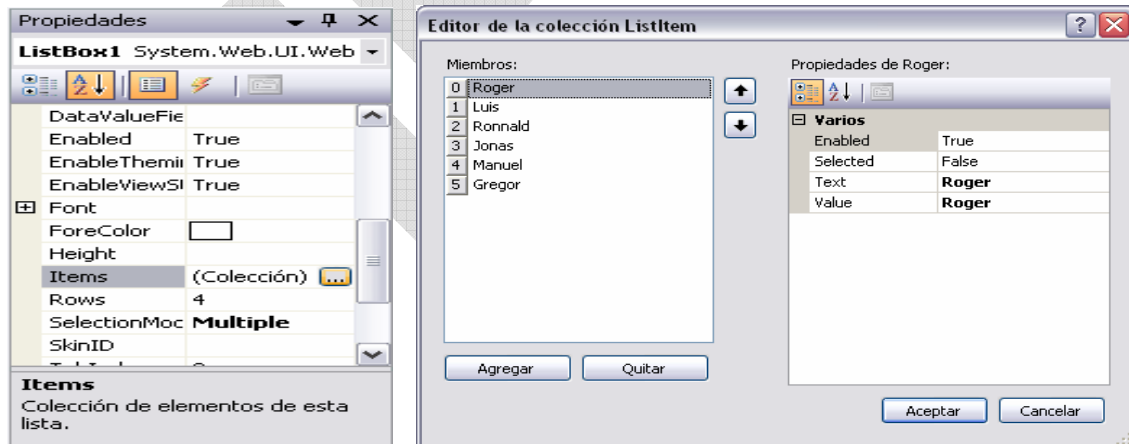
```
protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
}
}
```

#### 4.1.- ListBox ( *System.Web.UI.WebControls.ListBox* )

Los controles ListBox muestran una serie de elementos dentro de un recuadro para su selección. En caso de haber más elementos de los que se visualizar se muestra automáticamente una barra de desplazamiento. Mediante el valor de la propiedad *SelectionMode* se puede emplear el control para la selección de un único elemento ( *single* ), o de varios ( *multiple* ).



El control ListBox puede rellenarse de elementos en la fase de diseño de la página empleando el IDE del Visual Studio. Para ello, sólo basta seleccionar la propiedad *Items* de la ventana de propiedades del control, e ir añadiendo elementos en la ventana de edición de la colección:



De esta manera, los elementos de la lista quedan definidos ya en el diseño quedando inscritos en propio código del diseño de la página:

##### Control Web

```
<asp:ListBox ID="ListBox1" runat="server">
  <asp:ListItem Value="665748938">Roger</asp:ListItem>
  <asp:ListItem Value="668392093">Yuri</asp:ListItem>
  <asp:ListItem Value="689950392">Sassa</asp:ListItem>
  <asp:ListItem Value="669403920">Ivan</asp:ListItem>
</asp:ListBox>
```

## CONTROLES WEB ASP.NET

Hay muchas ocasiones en las que es necesario añadir las opciones de un control `ListBox` mediante código en vez de en diseño. Para ello, los objetos `Listltem` pueden crearse e insertarse en la colección del control mediante código en el evento *Init* del propio control.

```
String[] nombres = { "Roger", "Petrov", "Ivan", "Alex", "Yuri", "Natasha" };
String[] numeros = { "5554895", "5559403", "5553921", "5553920", "5559201", "5550990" };

protected void ListBox_Init(object sender, EventArgs e)
{
    ListItem elemento;
    for (int indx = 0; indx < nombres.Length; indx++) {
        elemento = new ListItem(nombres[indx], numeros[indx]);
        ListBox1.Items.Add(elemento);
    }
}
```

Por cada elemento que debe contener el control `ListBox`; se crea una instancia de la clase `Listltem`, y después se añade a la colección del control empleando la propiedad *Item* de `ListBox`, y el método *Add* de `ListltemCollection`.

La propiedad *SeletedIndex* devuelve un valor numérico que indica la posición en la colección del elemento seleccionado por el usuario. En caso de no haber ningún elemento seleccionado, retorna el valor -1. La propiedad *Selectedltem* devuelve en cambio el elemento `Listltem` seleccionado directamente. En caso de no haberse seleccionado ningún elemento retorna null. Finalmente; la propiedad *SelectedValue* retorna la cadena de texto de la propiedad *Value* del elemento seleccionado. En caso no haber ningún elemento seleccionado en el `ListBox` retorna una cadena de texto vacía:

```
if (ListBox1.SelectedIndex != -1)
{
    Response.Write("Has seleccionado a : " + ListBox1.SelectedItem.Text + " y su numero es: " +
        ListBox1.SelectedValue);
} else {
    Response.Write("No has seleccionado a nadie!");
}
```

En caso de tratarse de un control `ListBox` que permite selección múltiple, debe emplearse el método *GetSelectedIndices()*. Este método retorna una matriz de valores enteros con los índices de los elementos seleccionados dentro de la colección del control. En caso de no haber ningún elemento seleccionado la matriz devuelta no contendrá ningún valor.

```
if (ListBox1.GetSelectedIndices().Length > 0)
{
    int[] indices = ListBox1.GetSelectedIndices();
    foreach (int indice in indices)
    {
        Response.Write("Has seleccionado a : " + ListBox1.Items[indice].Text + " y su numero es: "
            + ListBox1.Items[indice].Value);
        Response.Write("<br />");
    }
}
```

El control Web `ListBox` se convierte siempre en un control *Select* de HTML. Los distintos elementos aparecerán como subelementos “*Option*” embebidos dentro del control *Select*. El atributo “multiple” de la etiqueta *Select* es el equivalente en HTML a la propiedad *SelectionMode* del control Web.

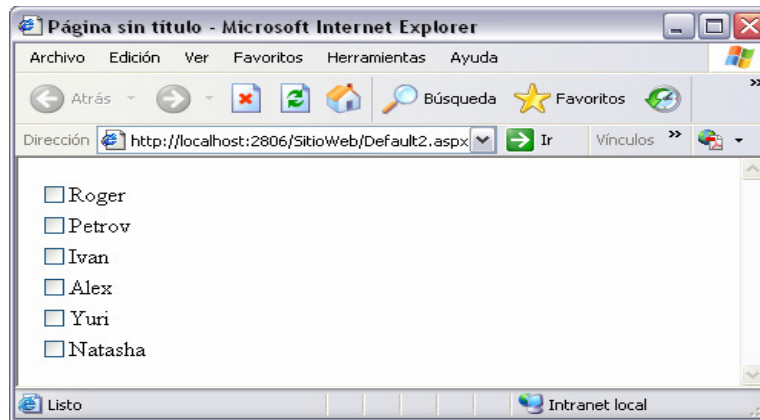
### Código HTML

```
<select size="4" name="ListBox1" id="ListBox1" multiple="multiple">
    <option value="Roger">Roger</option>
    <option value="Luis">Luis</option>
    <option value="Gregor">Gregor</option>
</select>
```

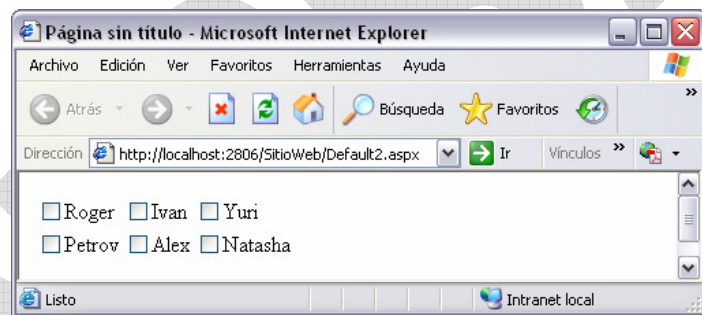


#### 4.2.- CheckBoxList ( *System.Web.UI.WebControls.CheckBoxList* )

El control CheckBoxList es un control muy semejante a ListBox, pero con la diferencia de que sus elementos son controles CheckBox.



La propiedad *RepeatDirection* controla si los elementos aparecen extendiéndose en vertical o en horizontal. La propiedad *RepeatColumns* permite indicar un número de columnas entre las que se reparten los elementos para su visualización.



**Control CheckBoxList con RepeatColumns ajustado a tres columnas**

Las mecánicas para añadir elementos a un CheckBoxList es idéntica a la de un control ListBox tanto por diseño como por código.

En un control CheckBoxList no hay ninguna limitación para el número de elementos que pueden marcarse; podrían marcarse todos o ninguno. Para conocer los elementos marcados se comprueba el valor de la propiedad *Selected* de cada objeto ListItem en la colección de elementos del control.

```
protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
{
    foreach (ListItem item in this.CheckBoxList1.Items)
    {
        if (item.Selected)
        {
            Response.Write("Seleccionado: " + item.Text + " Tlf: " + item.Value);
            Response.Write("<br />");
        }
    }
}
```

Cuando un usuario marca o desmarca la casilla de verificación de un elemento en una CheckBoxList, se activa el evento *SelectedIndexChanged*. Este evento no provoca la recarga de la página salvo que se modifique el valor de la propiedad *AutoPostBack*.

# CONTROLES WEB ASP.NET

## Control Web

```
<asp:CheckBoxList ID="CheckBoxList1"
    runat="server"
    OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged"
    RepeatColumns="2" RepeatDirection="Horizontal">
    <asp:ListItem Value="5557485">Roger</asp:ListItem>
    <asp:ListItem Value="5557789">Petrov</asp:ListItem>
    <asp:ListItem Value="5553322">Ivan</asp:ListItem>
    <asp:ListItem Value="5559302">Yuri</asp:ListItem>
    <asp:ListItem Value="5554432">Andrei</asp:ListItem>
    <asp:ListItem Value="5554322">Sasa</asp:ListItem>
    <asp:ListItem Selected="True" Value="5554930">Sergei</asp:ListItem>
</asp:CheckBoxList>
```

## Código HTML

No existe una etiqueta equivalente en HTML al control Web CheckBoxList, por lo que el control se convierte en una serie de controles <Input> de tipo "checkbox" ordenados en las celdas de una tabla sin bordes.

```
<table id="CheckBoxList1" border="0">
  <tr>
    <td>
      <input id="CheckBoxList1_0"
        type="checkbox"
        name="CheckBoxList1$0" />
      <label for="CheckBoxList1_0">Roger</label>
    </td>
    <td>
      <input id="CheckBoxList1_1"
        type="checkbox"
        name="CheckBoxList1$1" />
      <label for="CheckBoxList1_1">Petrov</label>
    </td>
  </tr>
  <tr>
    <td>
      <input id="CheckBoxList1_2"
        type="checkbox"
        name="CheckBoxList1$2" />
      <label for="CheckBoxList1_2">Ivan</label>
    </td>
    <td>
      <input id="CheckBoxList1_3"
        type="checkbox"
        name="CheckBoxList1$3" />
      <label for="CheckBoxList1_3">Yuri</label>
    </td>
  </tr>
  <tr>
    <td>
      <input id="CheckBoxList1_4"
        type="checkbox"
        name="CheckBoxList1$4" />
      <label for="CheckBoxList1_4">Andrei</label>
    </td>
    <td>
      <input id="CheckBoxList1_5"
        type="checkbox"
        name="CheckBoxList1$5" />
      <label for="CheckBoxList1_5">Sasa</label>
    </td>
  </tr>
</table>
```



#### 4.3.- RadioButtonList ( *System.Web.UI.WebControls.RadioButtonList* )

El control RadioButtonList es equivalente al control CheckBoxList pero sus elementos son controles RadioButton en vez de CheckBox. Un RadioButtonList sólo puede tener un elemento seleccionado en cada momento.



Las propiedades como *RepeatColumns* y *RepeatDirection* también son miembros de la clase RadioButtonList, y permiten controlar la visualización de los elementos de igual manera que en un CheckBoxList.

Este control también presenta el evento *SelectedIndexChanged* que se activa cuando el usuario selecciona un elemento. El evento no provoca la recarga de la página salvo que se modifique el valor de la propiedad *AutoPostBack*.

La mecánica para declarar elementos en un control RadioButtonList por diseño o por código es idéntica a la del resto de controles CheckBoxList y ListBox.

Para obtener el elemento marcado por el usuario pueden emplearse las propiedades *SelectedIndex*, *SelectedItem* o *SelectedValue*.

##### Control Web

```
<asp:RadioButtonList ID="RadioButtonList1"
    runat="server"
    RepeatDirection="Horizontal">
    <asp:ListItem>Activo</asp:ListItem>
    <asp:ListItem>Inactivo</asp:ListItem>
    <asp:ListItem>Deshabilitado</asp:ListItem>
</asp:RadioButtonList>
```

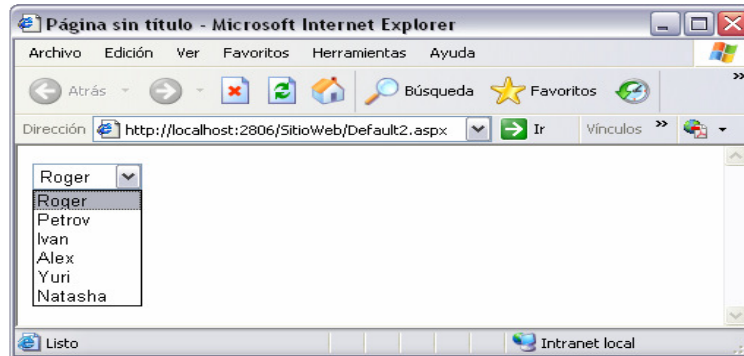
##### Código HTML

Al igual que en el caso del control CheckBoxList; no existe una equivalente en HTML para el control RoundButtonList. Este control Web se convierte en un conjunto de etiquetas <Input> de tipo "radio" organizadas en una tabla sin bordes.

```
<table id="RadioButtonList1" border="0">
  <tr>
    <td>
      <input id="RadioButtonList1_0" type="radio" name="RadioButtonList1" value="Activo"/>
      <label for="RadioButtonList1_0">Activo</label>
    </td>
    <td>
      <input id="RadioButtonList1_1" type="radio" name="RadioButtonList1" value="Inactivo" />
      <label for="RadioButtonList1_1">Inactivo</label>
    </td>
  </tr>
</table>
```

### 4.4.- DropDownList ( *System.Web.UI.WebControls.DropDownList* )

Este control Web muestra un único elemento junto con un botón que permite desplegar una lista de elementos para seleccionar uno. Su aspecto y comportamiento es semejante al del control ComboBox y no admite la selección múltiple.



El control posee la propiedad *Text* que muestra el valor de la propiedad *Text* del objeto *ListItem* seleccionado. Este campo es modificable por el usuario y permite realizar búsquedas entre los elementos contenidos por el valor de su propiedad *Text*.

La mecánica para incluir elementos en el control bien por diseño o por código es idéntica al del resto de controles de tipo lista.

Cuando se selecciona un elemento en el control se activa primero el evento *SelectedItemChanged*. La selección de un elemento provoca que la propiedad *Text* del control refleje de la propiedad *Text* del elemento seleccionado y que se active a continuación el evento *TextChanged*.

```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e) { }  
protected void DropDownList1_TextChanged(object sender, EventArgs e) { }
```

El control posee también las propiedades *SelectedIndex*, *SelectedItem* y *SelectedValue* para comprobar el elemento seleccionado por el usuario.

#### Control Web

```
<asp:DropDownList ID="DropDownList1"  
    runat="server" OnInit="DropDownList1_Init"  
    OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged"  
    OnTextChanged="DropDownList1_TextChanged">  
    <asp:ListItem Value="5557489">Yuri</asp:ListItem>  
    <asp:ListItem Value="5558493">Ivan</asp:ListItem>  
    <asp:ListItem Value="5553322">Jhonny</asp:ListItem>  
</asp:DropDownList>
```

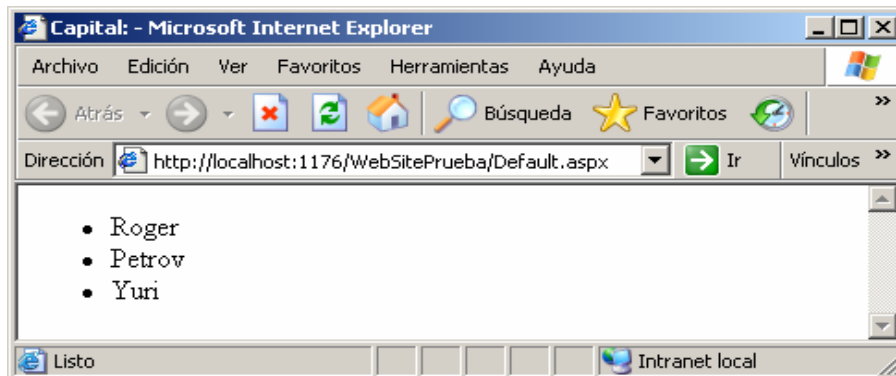
El control Web *DropDownList* se convierte en HTML en la etiqueta *<select>*, seguida de una etiqueta *option* por cada uno de los elementos de la lista.

#### Código HTML

```
<select name="DropDownList1" id="DropDownList1">  
    <option selected="selected" value="5557489">Yuri</option>  
    <option value="5558493">Ivan</option>  
    <option value="5553322">Jhonny</option>  
    <option value="5550990">Natasha</option>  
</select>
```

#### 4.5.- **BulletedList ( *System.Web.UI.WebControls.BulletedList* )**

Este control Web permite representar una lista de elementos jerarquizados empleando dibujos, símbolos o valores numéricos.



El símbolo que precede a cada elemento puede modificarse empleando la propiedad *BulletStyle*. Esta propiedad admite los valores presentes en el enumerador *System.Web.UI.WebControls.BulletStyle*:

Nombre de miembro	Descripción
<b>Circle</b>	El estilo de viñeta es un círculo vacío.
<b>CustomImage</b>	El estilo de viñeta es una imagen personalizada.
<b>Disc</b>	El estilo de viñeta es un círculo relleno.
<b>LowerAlpha</b>	El estilo de viñeta es una letra minúscula (a, b, c,...).
<b>LowerRoman</b>	El estilo de viñeta es un número romano en minúsculas (i, ii, iii, ...).
<b>NotSet</b>	El estilo de viñeta no se ha establecido. El explorador que representa el control <b>BulletedList</b> determinará el estilo de viñeta que se va a mostrar.
<b>Numbered</b>	El estilo de viñeta es un número (1, 2, 3, ...).
<b>Square</b>	El estilo de viñeta es un cuadrado relleno.
<b>UpperAlpha</b>	El estilo de viñeta es una letra mayúscula (A, B, C, ...).
<b>UpperRoman</b>	El estilo de viñeta es un número romano en mayúsculas (I, II, III, ...).

También es posible asignar una imagen que haga de símbolo asignando el valor *CustomImage* a la propiedad *BulletStyle*, y la URL de la imagen en la propiedad *BulletedImageUrl* del control.

Cada uno de los elementos que conforman la lista puede ser un simple texto, un hipervínculo, o un control *LinkButton*. Este comportamiento se controla mediante la propiedad *DisplayMode*, asignándole un valor perteneciente al enumerador *System.Web.UI.WebControls.BulletedListDisplayMode*.

Nombre de miembro	Descripción
<b>HyperLink</b>	Muestra el contenido de los elementos de la lista como hipervínculos.
<b>LinkButton</b>	Muestra el contenido de los elementos de la lista como botones de vínculo.
<b>Text</b>	Muestra el contenido de los elementos de la lista como texto.

## CONTROLES WEB ASP.NET

Recordar que la diferencia entre un hipervínculo ( *Hyperlink* ), y un LinkButton radica en que el primero produce solicitud directa de la URL indicada en el control desde el cliente, y el segundo provoca una recarga de la página siendo el servidor quien reconduce la solicitud a la URL indicada.

Al igual que el resto de controles heredados de ListControl, éste posee una colección de elementos ListItem accesible a través de la propiedad *Items*. La mecánica para incluir elementos en diseño o por código es idéntica al del resto de controles de tipo lista.

Este control no posee las propiedades *SelectedIndex*, *SelectedItem*, *SelectedValue*. Es posible detectar la selección de un elemento empleando el evento Click si los elementos son de tipo LinkButton exclusivamente. Para ello la propiedad DisplayMode debe valer BulletedListDisplayMode.HyperLink.

```
protected void BulletedList1_Click(object sender, BulletedListEventArgs e)
{
    String seleccionado_texto = BulletedList1.Items[e.Index].Text;
    String seleccionado_valor = BulletedList1.Items[e.Index].Value;
}
```

Este evento recibe el parámetro 'e' de tipo BulletedListEventArgs. Esta clase posee una propiedad llamada Index, que posee el índice del elemento LinkButton que ha pulsado el usuario.

### Control Web

```
<asp:BulletedList ID="BulletedList1"
    runat="server"
    BulletStyle="Disc"
    OnClick="BulletedList1_Click"
    DisplayMode="LinkButton">
    <asp:ListItem Value="5558439">Roger</asp:ListItem>
    <asp:ListItem Value="5556678">Petrov</asp:ListItem>
    <asp:ListItem Value="5552435">Yuri</asp:ListItem>
</asp:BulletedList>
```

El control BulletList es equivalente a las etiquetas de listas ordenadas <OL> y listas no ordenadas <UL> presentes en HTML. Por otro lado, dependiendo del tipo seleccionado para los elementos ( texto, hipervínculos, o LinkButtons ), éstos serán convertidos al HTML correspondiente.

### Código HTML

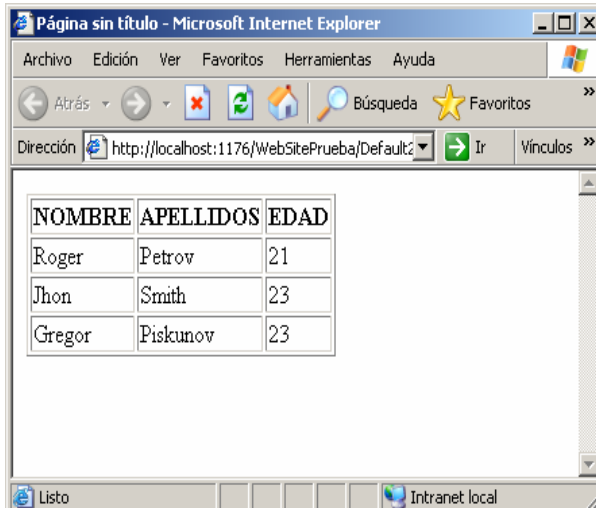
```
<script type="text/javascript">
//
var theForm = document.forms['form1'];
if (!theForm) {
    theForm = document.form1;
}
function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}
//]]&gt;
&lt;/script&gt;
&lt;ul id="BulletedList1" style="list-style-type:disc;"&gt;
&lt;li&gt;
    &lt;a href="javascript:__doPostBack('BulletedList1','0')"&gt;Roger&lt;/a&gt;
&lt;/li&gt;&lt;li&gt;
    &lt;a href="javascript:__doPostBack('BulletedList1','1')"&gt;Petrov&lt;/a&gt;
&lt;/li&gt;&lt;li&gt;
    &lt;a href="javascript:__doPostBack('BulletedList1','2')"&gt;Yuri&lt;/a&gt;
&lt;/li&gt;
&lt;/ul&gt;</pre></div><div data-bbox="138 937 165 954" data-label="Page-Footer"><p>28</p></div>
```

## 5.- CONTROLES WEB DE DISPOSICION

### 5.1.- Tablas ( *System.Web.UI.WebControls.Table* )

Las tablas son un elemento importante en el diseño de las páginas Web, ya que constituyen una de las principales formas de organizar la disposición de los elementos que las integran.

Si no se desea generar o manipular las tablas en tiempo de ejecución desde servidor se puede emplear el control HTML Table:



NOMBRE	APELLIDOS	EDAD
Roger	Petrov	21
Jhon	Smith	23
Gregor	Piskunov	23

#### Código HTML

```
<table border="1">
  <tr>
    <th>NOMBRE</th>
    <th>APELLIDOS</th>
    <th>EDAD</th>
  </tr>
  <tr>
    <td>Roger</td>
    <td>Petrov</td>
    <td>21</td>
  </tr>
  <tr>
    <td>Jhon</td>
    <td>Smith</td>
    <td>23</td>
  </tr>
  <tr>
    <td>Gregor</td>
    <td>Piskunov</td>
    <td>23</td>
  </tr>
</table>
```

ASP.NET dispone de su propia versión de control tabla. El control Web Tabla permite la generación y modificación de la estructura de una tabla desde código de servidor. Los elementos propios de una tabla ( filas, celdas.. etc ), existen igualmente como objetos para el control Web Tabla:

#### Control Web

```
<asp:Table ID="Table1" runat="server" GridLines="Both">
  <asp:TableRow runat="server" BackColor="#E0E0E0">
    <asp:TableCell runat="server">NOMBRE</asp:TableCell>
    <asp:TableCell runat="server">APELLIDOS</asp:TableCell>
    <asp:TableCell runat="server">EDAD</asp:TableCell>
  </asp:TableRow>
  <asp:TableRow runat="server">
    <asp:TableCell runat="server">Roger</asp:TableCell>
    <asp:TableCell runat="server">Petrov</asp:TableCell>
    <asp:TableCell runat="server">21</asp:TableCell>
  </asp:TableRow>
  <asp:TableRow runat="server">
    <asp:TableCell runat="server">Jhon</asp:TableCell>
    <asp:TableCell runat="server">Smith</asp:TableCell>
    <asp:TableCell runat="server">23</asp:TableCell>
  </asp:TableRow>
  <asp:TableRow runat="server">
    <asp:TableCell runat="server">Gregor</asp:TableCell>
    <asp:TableCell runat="server">Piskunov</asp:TableCell>
    <asp:TableCell runat="server">23</asp:TableCell>
  </asp:TableRow>
</asp:Table>
```

La estructura del control Web Tabla viene dada por el conjunto de controles Web que contiene en un determinado orden. Así, por ejemplo; las filas de una tabla están representadas por el control Web *TableRow*. Las celdas de las filas de una tabla están representadas por el control *TableCell*.

## CONTROLES WEB ASP.NET

En la siguiente lista podemos ver las equivalencias entre las diferentes etiquetas que componen la estructura de una tabla en HTML, y los objetos equivalentes en ASP.NET:

Control Web	Etiqueta HTML	Descripción
Table	<table>	Control padre de la tabla. La propiedad <i>Rows</i> referencia una colección de <i>TableRow</i> con las filas presentes.
TableRow	<tr>	Control padre de las filas de Tabla. La propiedad <i>Cells</i> referencia a una colección de <i>TableCell</i> con las celdas de cada fila.
TableCell	<td>	Control celda que contiene el contenido a mostrar en cada celda. La propiedad <i>Text</i> contiene texto, la propiedad <i>Controls</i> puede contener más controles Web.
TableHeaderCell	<th>	Control derivado de <i>TableCell</i> . Representa las celdas del encabezado de la tabla.
TableHeaderRow	<thead>	Control del encabezado de la tabla.
TableFooterRow	<tfoot>	Control del elemento pie de fila.

Conociendo los diferentes controles y la forma en que se relacionan, se puede manipular el diseño y contenido de un control Web tabla desde el código de una página ASP.NET:

```
protected void Page_Load(object sender, EventArgs e)
{
    // Generar Tabla
    int numRows = 3;
    int numcells = 2;
    // Generar cabecera
    TableHeaderRow oHeader = new TableHeaderRow();
    for (int i = 0; i < numcells; i++)
    {
        TableHeaderCell c = new TableHeaderCell();
        c.Text = "Fila n." + i.ToString();
        oHeader.Cells.Add(c);
    }
    // Generar Filas
    this.Table1.Rows.Add(oHeader);
    for (int j = 0; j < numRows; j++)
    {
        TableRow r = new TableRow();
        for (int i = 0; i < numcells; i++)
        {
            TableCell c = new TableCell();
            c.Text = "Fila: " + j.ToString() + " Celda: " + i.ToString();
            r.Cells.Add(c);
        }
        Table1.Rows.Add(r);
    }
    // Generar Pie
    TableFooterRow tFooterRow = new TableFooterRow();
    for (int i = 0; i < numcells; i++)
    {
        TableCell c = new TableCell();
        c.Text = "Pie n." + i.ToString();
        tFooterRow.Cells.Add(c);
    }
    this.Table1.Rows.Add(tFooterRow);
}
```

Es importante recordar que cualquier adición de controles realizada en las filas o celdas de la tabla mediante programación **no** se conservará en los envíos al servidor. Esto se debe a que las filas y las celdas de la tabla son en sí mismas controles y no propiedades del control Table que se almacenen como parte de *viewstate*.

Para almacenar los cambios en la tabla, se deben reconstruir las filas y celdas después de cada devolución de datos. De hecho, si se prevén modificaciones sustanciales, se recomienda utilizar otro tipo de controles como *DataList*, *DataGrid* o *GridView* en lugar del control Table.



## 5.2.- Panel ( *System.Web.UI.WebControls.Panel* )

Este control se utiliza como contenedor de otros controles. Su principal utilidad es agrupar ciertos controles de una página para controlar su aspecto o visibilidad conjuntamente. Este control resulta especialmente útil si desea generar, ocultar o mostrar conjuntamente un grupo de controles desde código.

Las principales propiedades del control panel son las siguientes:

Nombre	Tipo	Descripción
BackImageURL	String	URL de un fichero de imagen a mostrar como fondo del panel
Direction	ContentDirection	Dirección del texto contenido: <i>LeftToRight, RightToLeft, NotSet</i>
HorizontalAlign	HorizontalAlign	Alineamiento horizontal del contenido del panel: <i>Center, Justify, Left, Right, None</i> .
ScrollBars	ScrollBars	Especifica la presencia y ubicación de barras de desplazamiento asociadas al panel: <i>Auto, Both, Horizontal, None, Vertical</i>
Wrap	Boolean	Indica si el contenido se ajusta a las dimensiones del panel ( <i>true</i> ), o no ( <i>false</i> ).

La propiedad *GroupingText* permite mostrar el contenido del control bordeado por una línea y un texto titular con el valor de la propiedad. Esto afecta a la manera en que el control Web se convierte a HTML. Si no se indica ningún valor a la propiedad *GroupingText*, el control se traduce a HTML empleando capas `<div>`. Si se asigna algún valor el control se convierte a HTML empleando la etiqueta `<fieldset>` y el valor en `<legend>`.

### Control Web

```
<asp:Panel ID="Panel1" runat="server" Height="50px" width="125px">
  <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
</asp:Panel>
```

### Control HTML sin GroupingText

```
<div id="panel1" style="height:50px;width:125px;">
  <input name="TextBox1" type="text" id="TextBox1" />
</div>
```

### Control HTML con GroupingText="Grupo"

```
<fieldset>
  <legend>
    Grupo
  </legend>
  <input name="TextBox1" type="text" id="TextBox1" />
</fieldset>
```

El control Panel posee una colección interna de tipo *ControlCollection* (*System.Web.UI.ControlCollection*) que alberga las referencias de todos los controles Web que contiene. Esta colección puede fijarse en diseño, o alterarse por código para añadir o manipular los controles contenidos en el Panel dinámicamente en ejecución. Esta colección es accesible mediante la propiedad *Controls*.

```
TextBox txt = new TextBox(); // Creo un control de tipo TextBox
txt.ID = "txt0001"; // Asignación del ID en la página en HTML
this.Panel1.Controls.Add(txt); // Añado a la colección de controles del panel.
```

Al igual que en el control Table, los controles añadidos por código no se conservan en los envíos al servidor. Esto se debe a que los controles de un panel no son propiedades del mismo y no se almacenan con su estado de vista.

## 6.- CONTROLES WEB DE IMÁGENES

Las imágenes son un aspecto importante en el diseño de cualquier página Web. En ASP.NET existen varios controles Web destinados a mostrar imágenes: *Image* y *ImageMap*.

### 6.1.- Image ( *System.Web.UI.WebControls.Image* )

El control Image es relativamente simple. Se utiliza para mostrar una imagen en una página Web, con la posibilidad de mostrar un texto si la imagen no se encontrase disponible. Este control dispone de las siguientes propiedades propias:

- *AlternateText*: Cadena de texto a mostrar si la imagen no está disponible, o mientras se descarga.
- *ImageAlign*: Permite indicar el tipo de alineamiento del control imagen con respecto al diseño circundante en la página. Los valores posibles son los siguientes:

Alineación	Descripción
NotSet	No se ha establecido la alineación.
Left	La imagen está alineada en el borde izquierdo de la página Web con ajuste de texto a la derecha.
Right	La imagen está alineada en el borde derecho de la página Web con ajuste de texto a la izquierda.
Baseline	El borde inferior de la imagen está alineado con el borde inferior de la primera línea de texto.
Top	El borde superior de la imagen está alineado con el borde superior del elemento más alto que se encuentra en la misma línea.
Middle	El centro de la imagen está alineado con el borde inferior de la primera línea de texto.
Bottom	El borde inferior de la imagen está alineado con el borde inferior de la primera línea de texto.
AbsBottom	El borde inferior de la imagen está alineado con el borde inferior del elemento más grande que se encuentra en la misma línea.
AbsMiddle	El centro de la imagen está alineado con el centro del elemento más grande que se encuentra en la misma línea.
TextTop	El borde superior de la imagen está alineado con el borde superior del texto más alto que se encuentra en la misma línea.

- *ImageUrl*: Cadena de texto con la dirección URL de la imagen a representar.

A pesar de la propiedad *ImageAlign*; lo más recomendable es emplear tablas para organizar el diseño de imágenes y demás elementos en la página.

Este control no posee eventos propios a excepción de los heredados de la clase padre WebControl. Si se quiere detectar si el usuario hace clic encima de una imagen debe emplearse el control Web *ImageButton*.

El control Image se convierte en HTML en un la etiqueta <img> clásica.

Control Web:

```
<asp:Image ID="Image1" runat="server" ImageUrl="~/Imágenes/sap.jpg" />
```

Código HTML:

```

```



## 6.2.- ImageMap ( *System.Web.UI.WebControl.ImageMap* )

El control Web ImageMap proporciona una superficie con una imagen en la que se definen diferentes zonas activas. Cada zona activa puede detectar si el usuario hace clic sobre ella y generar un evento.

La propiedad *HotSpots* de ImageMap hace referencia a una colección del tipo *HotSpotsCollection* ( *System.Web.UI.WebControls.HotSpotCollection* ) que contiene todas las zonas activas definidas en un control ImageMap.

Cada zona activa de un control ImageMap puede ser una instancia de la clase *RectangleHotSpot*, *CircleHotSpot*, o *PolygonHotSpot*. Las tres clases heredan a su vez de la clase padre *HotSpot* (*System.Web.UI.WebControl.HotSpot*), que define las siguientes propiedades comunes a todas ellas:

Nombre	Descripción
 <a href="#">AccessKey</a>	Obtiene o establece la tecla de acceso que permite desplazarse rápidamente a la región <b>HotSpot</b> .
 <a href="#">AlternateText</a>	Obtiene o establece el texto alternativo para mostrar en un objeto <b>HotSpot</b> de un control <b>ImageMap</b> cuando la imagen no está disponible o se representa en un explorador que no admite imágenes.
 <a href="#">HotSpotMode</a>	Especifica o establece el comportamiento de un objeto <b>HotSpot</b> en un control <b>ImageMap</b> cuando se hace clic en <b>HotSpot</b> .
 <a href="#">NavigateUrl</a>	Obtiene o establece la dirección URL a la que desplazarse cuando se hace clic en un objeto <b>HotSpot</b> .
 <a href="#">PostBackValue</a>	Obtiene o establece el nombre del objeto <b>HotSpot</b> que se pasará en los datos de evento cuando se haga clic en <b>HotSpot</b> .
 <a href="#">TabIndex</a>	Obtiene o establece el índice de tabulación de la región <b>HotSpot</b> .
 <a href="#">Target</a>	Obtiene o establece la ventana o el marco de destino donde debe mostrarse el contenido de la página Web a la que se vincula al hacer clic en un objeto <b>HotSpot</b> que se desplace a una dirección URL.

Adicionalmente a estas propiedades, cada clase posee unas propiedades propias para definir la posición y extensión de la zona activa dentro del control ImageMap:

- **CircleHotSpot:** Posee las propiedades *Radius*, *X* e *Y*; que indican el radio y la posición relativa al control ImageMap de la circunferencia que conforma el área activa.
- **RectangleHotSpot:** Posee las propiedades *Top*, *Left*, *Bottom*, *Right*; que indican la posición relativa del margen superior izquierda e inferior derecho del rectángulo que conforma el área activa.
- **PolygonHotSpot:** Posee la propiedad *Coordinates* que contiene una cadena de texto con secuencia de números separados por comas. Cada par de números representan las coordenadas de un vértice del polígono. Por ejemplo: "128,185,335,157,510,224,510,383,228,383" define una zona activa poligonal con cinco vértices. La coordenada x del primer vértice es 128; la coordenada y del primer vértice es 185.

## CONTROLES WEB ASP.NET

Las zonas activas pueden comportarse como un enlace a otra página, o provocar una recarga de la página cuando el usuario hace clic sobre ellas. El comportamiento depende del valor que se asigne a la propiedad *HotSpotMode* de cada zona activa. Los posibles valores para esta propiedad están definidos en el tipo enumerado *System.Web.UI.WebControls.HotSpotMode*, y son los siguientes:

Valor	Descripción
NotSet	<b>HotSpot</b> utiliza el comportamiento establecido por la propiedad <i>HotSpotMode</i> del control <b>ImageMap</b> . Si el control <b>ImageMap</b> no define el comportamiento, el objeto <b>HotSpot</b> se desplaza a una dirección URL.
Inactive	El objeto <b>HotSpot</b> no tiene comportamiento.
Navigate	El objeto <b>HotSpot</b> se desplaza a una dirección URL.
PostBack	El objeto <b>HotSpot</b> genera una devolución de datos al servidor.

Si el valor asignado es “*Navigate*”, el área activa provocará un salto a la página indicada en su propiedad *NavigateUrl*.

Si el valor es “*PostBack*”, el área activa provocará una recarga de la página y la ejecución en el servidor del evento Click del control ImageMap. Cada área deberá identificarse con un valor diferente asociado a su propiedad *PostBackValue*. El método controlador del evento Click de un control ImageMap, posee un segundo parámetro de tipo *ImageMapEventArgs* que permite obtener el valor de la propiedad *PostBackValue* del área seleccionada por el usuario.

```
protected void ImageMap1_Click(object sender, ImageMapEventArgs e)
{
    string area = e.PostBackValue;    // Devuelve el valor de la propiedad PostBackValue del
                                     // área pulsada por el usuario.
}
```

### Control Web:

```
<asp:ImageMap ID="ImageMap1" runat="server" ImageUrl="~/Imagenes/sap.jpg" OnClick="ImageMap1_Click">
  <asp:CircleHotSpot Radius="50" X="100" Y="100" PostBackValue="1" />
  <asp:RectangleHotSpot Bottom="100" Left="10" Right="100" Top="10" PostBackValue="2" />
  <asp:PolygonHotSpot Coordinates="1,1,1,100,100,100,100,1" PostBackValue="3" />
  <asp:CircleHotSpot HotSpotMode="Navigate" Radius="25" X="25" Y="50" NavigateUrl="~/Test.aspx" />
</asp:ImageMap>
```

El control ImageMap se convierte en HTML en un control <img> junto con un control <map> con las etiquetas <shape> correspondientes a cada zona activa definida.

### Código HTML:

```

<map name="ImageMapImageMap1" id="ImageMapImageMap1">
  <area shape="circle" coords="100,100,50" href="" title="" alt="" />
  <area shape="rect" coords="10,10,100,100" href="" title="" alt="" />
  <area shape="poly" coords="1,1,1,100,100,100,100,1" href="" title="" alt="" />
  <area shape="circle" coords="25,50,25" href="Test.aspx" title="" alt="" />
</map>
```

## **7.- CONTROLES WEB DE VALIDACION**

Los controles de validación son un conjunto de controles Web que se ocupan de validar la información de un control Web de entrada de datos ( p.ej: un textbox ), y mostrar un mensaje de error informativo. También es posible definir un control Web especial que muestre agrupados los mensajes de error de los controles de validación de una página ( *ValidationSummary* ).

Los controles de validación pueden realizar las comprobaciones en el lado del cliente, o en el lado del servidor.

La validación en el lado del cliente implica que los controles de validación generarán código HTML y Javascript para ejecutar las comprobaciones en el navegador Web del cliente de modo que hasta que los valores del formulario no sean válidos; no se envía ninguna información al servidor.

La validación en el lado del servidor implica que el formulario envía los datos al servidor de cualquier manera, y es el servidor el que comprueba si están correctos. En este caso, los controles de validación se ejecutan en el servidor íntegramente, y no se envía ningún script de comprobación al navegador del cliente.

De manera predeterminada, la validación se realiza al hacer clic en un control de tipo botón, como Button, ImageButton o LinkButton con la propiedad *CausesValidation = true*. Esta propiedad suele estar establecida en false para un botón cancel o clear con el fin de impedir que se realice la validación cuando se hace clic en el botón. También puede realizarse la validación manualmente invocando desde el código el método *Validate()*.

Una vez ejecutados la validación puede comprobarse si son correctos consultando el valor lógico retornado por la propiedad *IsValid()*.

La plataforma ASP.NET ofrece cinco controles Web de validación:

- ***RequiredFieldValidator***: Comprueba que un campo no quede vacío.
- ***RangeValidator***: Comprueba que el valor esté dentro de un rango
- ***CompareValidator***: Comprueba que dos controles tengan el mismo valor.
- ***RegularExpressionValidator***: Comprueba que el valor de un control sea conforme a una expresión regular dada.
- ***CustomValidator***: Comprueba el valor con un código a medida que devuelva si el dato es o no válido.

El control *ValidationSummary*, está relacionado con los controles Web de validación. Este control no realiza ninguna validación, sino que se ocupa de mostrar agrupados los mensajes de error de los controles de validación presentes en una página ASP.NET.

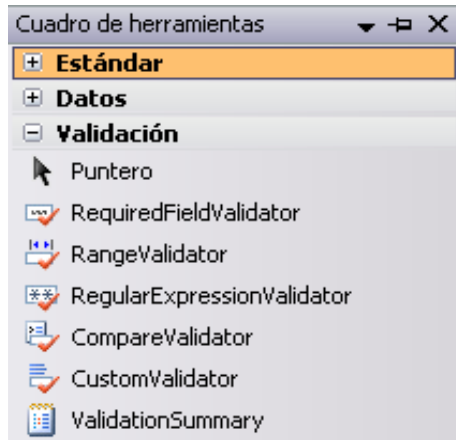
## CONTROLES WEB ASP.NET

Todos los controles Web de validación heredan de la clase padre abstracta *BaseValidator* ( *System.Web.UI.WebControls.BaseValidator* ) que proporciona los métodos y propiedades comunes. Las más destacables son las siguientes:

Propiedad	Descripción
<b><i>ControlToValidate</i></b>	Id. de programación del control de salida que evaluará el control de validación. Si no es un Id. legítimo, se iniciará una excepción.
<b><i>Display</i></b>	<p>Modo en que se muestra el control de validación especificado. Esta propiedad puede ser uno de los siguientes valores:</p> <p><b><i>None</i></b> — El control de validación jamás se muestra en línea. Utilice esta opción cuando desee mostrar el mensaje de error sólo en un control <i>ValidationSummary</i></p> <p><b><i>Static</i></b> — El control de validación muestra un mensaje de error si se produce un error en la validación. Se asigna un espacio en la página Web para el mensaje de error incluso si el control de entrada supera la validación. No cambia el diseño de la página cuando el control de validación muestra su mensaje de error. Como el diseño de página es estático, si hay varios controles de validación para el mismo control de entrada, éstos deberán ocupar distintas ubicaciones en la página.</p> <p><b><i>Dynamic</i></b> — El control de validación muestra un mensaje de error si se produce un error en la validación. El espacio para el mensaje de error se asigna dinámicamente en la página cuando se produce un error en la validación. De este modo, varios controles de validación pueden compartir la misma ubicación física en la página.</p>
<b><i>EnableClientScript</i></b>	Indica si está habilitada la validación en el cliente. Para deshabilitar la validación en el cliente en los exploradores que admitan esta función, establezca la propiedad <i>EnableClientScript</i> en false.
<b><i>Enabled</i></b>	Indica si está habilitado el control de validación. Para impedir que el control de validación valide un control de entrada, establezca esta propiedad en false.
<b><i>ErrorMessage</i></b>	Mensaje de error que se va a mostrar en el control <i>ValidationSummary</i> si se produce un error en la validación. Si no está establecida la propiedad <i>Text</i> del control de validación, también se muestra este texto en el control de validación cuando se produce un error en la validación. Se utiliza normalmente la propiedad <i>ErrorMessage</i> para proporcionar diferentes mensajes para el control de validación y el control <i>ValidationSummary</i> .
<b><i>IsValid</i></b>	Indica si el control de entrada especificado por la propiedad <i>ControlToValidate</i> se determina como válido.
<b><i>ValidationGroup</i></b>	Establece el nombre de un grupo de validación. Los grupos de validación permiten comprobar la validación de ciertos controles de una página independientemente de otros.
<b><i>Text</i></b>	Si esta establecida esta propiedad, se muestra este mensaje en el control de validación cuando se produce un error en la validación. Si no está establecida esta propiedad, el texto especificado en la propiedad <i>ErrorMessage</i> se muestra en el control.

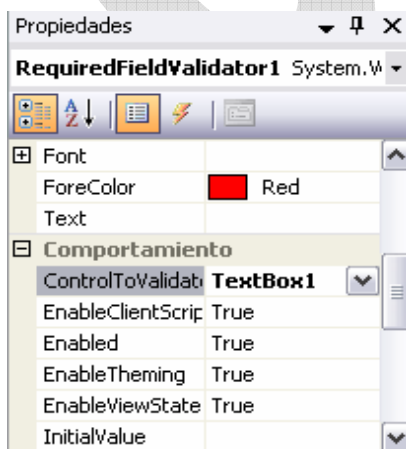
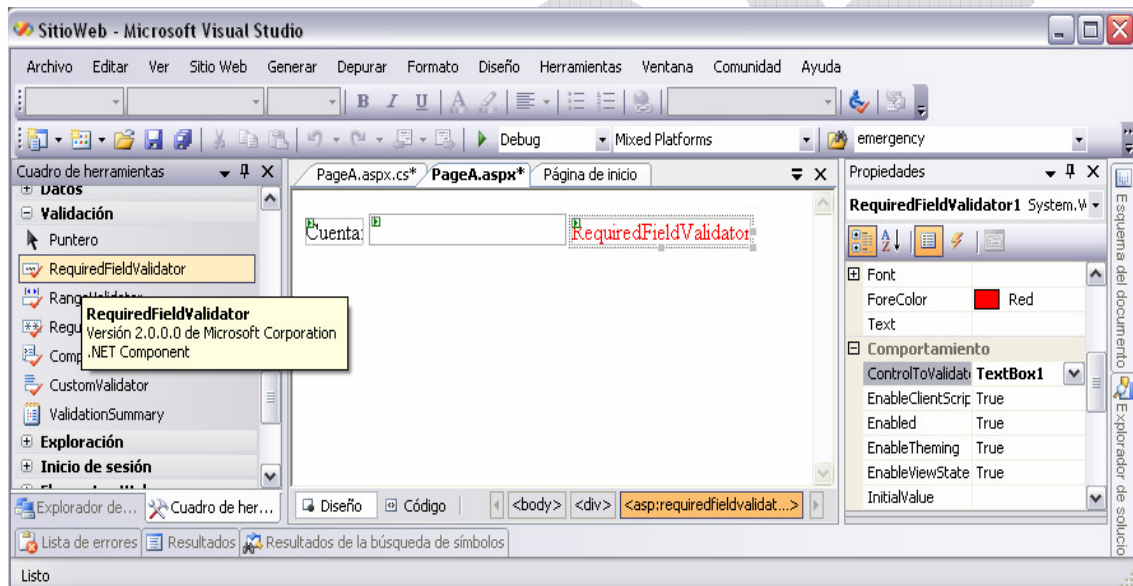
## 7.1.- INSERCIÓN DE CONTROLES DE VALIDACIÓN

Los controles de validación pueden situarse en cualquier parte del diseño de una página; pero se recomienda situarlos al lado del control que validan.



Los controles Web de validación se encuentran en la categoría "Validación" dentro del cuadro de herramientas del IDE del Visual Studio.

Para añadirlos a una página Web, el procedimiento es exactamente el mismo que con cualquier otro control Web; simplemente señalar el control deseado y arrastrarlo hasta el diseño de página.



Una vez situado en la página un control de validación, debe indicársele el control que va a validar.

Seleccionando el control, buscamos la propiedad *ControlToValidate* y seleccionamos en la lista desplegable a su derecha el nombre del control a verificar.

## 7.2.- DETECCION DE ERRORES DE VALIDACION

Los controles Web de validación por sí mismos sólo muestran mensajes de error, y en caso de ejecutarse en el lado del cliente ( *EnableClientScript = true* ) bloquean el envío de datos al servidor hasta verificarse las validaciones.

Net Desarrollo SAC - Ing. Fernando Luque Sánchez - Validación de Datos en ASP .Net - Microsoft Internet Explorer

Dirección: http://localhost/ASPValidacion/ASPValidar.aspx

Autor: Ing. Fernando Luque Sánchez

Registro (obviamente simulado...je,je,je... )

Nombre: María Fernanda Luque Villacorta

Dirección: \*

Edad: 23 (mínimo 18 años)

Email: \*

Password: ... Repetir Password: ...

Fecha: 01/10/2005 Formato: mm/dd/aa

Matricula: ...

Teléfono: ... Formato: (mm)-mm-nnnnnnnn \*

Valor: 350 (Entre 230 y 890)

Aceptar Cancelar

Resumen de Errores

- Falta Ingreso de Dirección
- Falta Ingreso de Email
- Falta Ingreso de Teléfono

En el caso de que la validación se lleve a cabo en el lado del servidor en vez de en el navegador del cliente ( *EnableClientScript = false* ), se puede detectar desde código si alguna validación no es correcta, o qué control de validación no se verifica.

Ejemplo: Código de atención al evento de Click del botón de envío de datos de un formulario:

```
protected void btnEnviadDatos_Click(object sender, EventArgs e)
{
    // Comprobacion si la página es válida.
    if (!IsValid)
    {
        // Comprobacion de validez de un control de validacion del formulario.
        if (!ControlValidation.IsValid) // Si el control de validacion
        {
            // ...
        }
    }
}
```

La propiedad *IsValid*, pertenece a la propia instancia de la clase Page que representa la página en si misma. Esta propiedad retorna el valor cierto cuando todos los controles de validación en la página son válidos. En caso de que devuelve el valor falso, se puede emplear la propiedad *IsValid* de cada uno de los controles de validación para comprobar si son válidos, y localizar la validación con el error.

Es importante destacar, que la detección y manejo de los errores desde código en el servidor funciona cuando las validaciones se realizan en el lado del servidor.

### 7.3.- CONTROLES WEB DE VALIDACION

#### 7.3.1.- RequiredFieldValidator

Este control de validación se asegura de que el usuario no omita introducir o seleccionar datos en un control. Puede vincularse a controles caja de texto para que el usuario no los deje vacíos, o indicar un valor inicial para que el usuario introduzca uno distinto. Puede también vincularse a controles de selección tales como ListBox, ComboBox, RadioButton, CheckBox para que el usuario tenga que hacer una selección.

Es importante destacar que este control no comprueba la validez de la información introducida por el usuario, tan sólo comprueba que el usuario introduzca alguna información en los campos vinculados.

Ejemplo de diseño:



En este ejemplo se muestra una caja de texto, un control RequiredFieldValidator, y un control Button. El objetivo de este ejemplo es que el usuario no pueda dejar la caja de texto ( TextBox1 ) con el valor inicial “sin datos”. Para ello se ajustan las propiedades del control RequiredFieldValidator a los siguientes valores:

<b>ControlToValidate</b>	TextBox1
<b>InitialValue</b>	“sin datos”
<b>ForeColor</b>	Color.Red
<b>Text</b>	“Campo vacío”

Para que el control RequiredFieldValidator reconozca “sin datos” como valor inicial que debe considerarse como vacío, debe introducirse en su propiedad *InitialValue*. El mensaje de error que el control debe mostrar en caso de error “Campo vacío” está asignado en su propiedad *Text*, y el color del mensaje de error se asigna en la propiedad *ForeColor*.



Tal y como se ve, queda un sospechoso espacio en blanco entre la caja de texto y el botón del formulario. Este es el espacio lo ocupa el control de validación para mostrar su mensaje de error. Si la propiedad *Display* del control se ajusta a “*static*” el control reserva espacio para su mensaje de error. Si su valor es “*dynamic*” el control no ocupa espacio en el diseño de la página hasta el momento en el que tenga que mostrar su mensaje de error.

La asignación dinámica de espacio para los mensajes de error de los controles de validación en general es potente; pero puede hacer que los controles de alrededor se desplacen estropeando el diseño. Debe tenerse esto en cuenta a la hora del diseño.



## CONTROLES WEB ASP.NET

Si ejecutamos la página, y pulsamos el botón "Button" sin modificar el valor inicial de la caja de texto obtendremos:

NOMBRE:  Campo vacío

La validación se realiza en el momento de la recarga de la página al pulsar el botón. Esto es debido a que el botón causa la validación de la página (*CausesValidation=true*), y que el control de validación se ejecuta del lado del servidor (*EnableScriptClient=false*). Al no verificarse la validación el control de validación muestra la cadena de texto contenida en su propiedad *Text*.

Si el control de validación se ejecuta en el lado del cliente (*EnableClientScript=true*), se produce la comprobación en el navegador del cliente sin recargar la página. En esta situación no puede controlarse la validez de la página desde código de servidor.

### Control Web:

```
<asp:RequiredFieldValidator
  ID="rfv_Nombre"
  runat="server"
  ControlToValidate="TextBox1"
  EnableScriptClient="true"
  ErrorMessage="RequiredFieldValidator"
  InitialValue="sin datos">Campo vacío
</asp:RequiredFieldValidator>
```




### 7.3.2.- CompareValidator (System.Web.UI.WebControls.CompareValidator)

Este control de validación compara el valor de un control Web ( *ControlToValidate* ) con un determinado valor, o el de otro control Web de la página. Para ello se emplean las propiedades *ControlToCompare* y *ValueToCompare*.

- **ControlToCompare:** Propiedad que hace referencia al control Web cuyo valor será comparado.
- **ValueToCompare:** Propiedad que contiene el valor con el que comparar. Se emplea cuando el campo debe ser comparado con un valor determinado, en vez de contra otro control.

Las propiedades anteriores determinan qué comparar. Las propiedades *Operator* y *Type* determinan cómo deben compararse. La propiedad *Operator* determina el operador de comparación a aplicar según el valor del enumerador *ValidateCompareOperation* que se la asigne. Los valores definidos son los siguientes:

Operación	Descripción
<b>Equal</b>	Comparación de igualdad entre los valores del control de entrada que se va a validar y otro control o un valor constante.
<b>NotEqual</b>	Una comparación de desigualdad entre los valores del control de entrada que se validan y otro control o un valor constante.
<b>GreaterThan</b>	Una comparación de superioridad entre los valores del control de entrada que se valida y otro control o un valor constante.
<b>GreaterThanEqual</b>	Comparación de superioridad o igualdad entre los valores del control de entrada que se va a validar y otro control o un valor constante.
<b>LessThan</b>	Comparación de inferioridad entre los valores del control de entrada que se va a validar y otro control o un valor constante.
<b>LessThanEqual</b>	Comparación de inferioridad o igualdad entre los valores del control de entrada que se va a validar y otro control o un valor constante.
<b>DataTypeCheck</b>	<p>Una comparación entre el tipo de datos del valor especificado en el control de entrada que se valida y el tipo de datos especificado por la propiedad <a href="#">BaseCompareValidator.Type</a>. Se produce un error en la validación si el valor no se puede convertir al tipo de datos especificado.</p> <div> <p> <b>Nota:</b></p> <p>Las propiedades <a href="#">ControlToCompare</a> y <a href="#">ValueToCompare</a> se omiten cuando se utiliza este operador.</p> </div>

La propiedad *Type* indica el tipo de valores que se van a comparar. Los valores admitidos son:

Tipo de datos	Descripción
<b>String</b>	Especifica un tipo de datos de cadena.
<b>Integer</b>	Especifica un tipo de datos de entero de 32 bits con signo.
<b>Double</b>	Especifica un tipo de datos de número de punto flotante de precisión doble.
<b>Date</b>	Especifica un tipo de datos de fecha.
<b>Currency</b>	Especifica un tipo de datos de moneda.

## CONTROLES WEB ASP.NET

### Comprobación de tipo de datos

Supongamos que queremos comprobar si el valor introducido en una caja de texto es un valor numérico. Ejemplo:

CODIGO POSTAL:

Interesa que el usuario introduzca un valor numérico, es decir; el campo no puede considerarse válido si el usuario introduce alguna letra. Podemos emplear el control CompareValidator ajustando la propiedad Operator a DataTypeCheck, y la propiedad Type a Integer.

```
<asp:CompareValidator
  ID="CompareValidator1"
  runat="server"
  ControlToValidate="TextBox1"
  Display="dynamic"
  Operator="DataTypeCheck"
  Type="Integer">Datos no válidos.</asp:CompareValidator>
```

Si el usuario introduce un valor con alguna letra, se muestra el error de validación al pulsar el botón de envío:

CODIGO POSTAL:  Datos no válidos.

Dado que el valor introducido por el usuario consta de un carácter, no se evalúa como de tipo Integer, y se produce el error de validación mostrando el mensaje correspondiente a la propiedad Text. En este caso, se ha fijado la propiedad DisplayMode a Dynamic para que el control no ocupe espacio en el diseño de la página hasta mostrar el error.

### Comprobación con otro control

Supongamos que queremos comprobar que el valor de una caja de texto sea idéntico al de otra para realizar una confirmación de cambio de contraseña:

Contraseña   
Repetir Contraseña:

Interesa que se valide que el usuario introduzca el mismo valor en ambas cajas de texto. Podemos emplear el control CompareValidator ajustando la propiedad Operator a Equal para comprobar que los datos de ambos controles sean iguales, y la propiedad Type a String.

```
<asp:CompareValidator
  ID="CompareValidator1"
  runat="server" ErrorMessage="CompareValidator"
  ControlToValidate="TextBox1"
  Display="dynamic"
  ControlToCompare="TextBox2"
  Type="String">Contraseña no coincide.</asp:CompareValidator>
```

Uno de los controles es referenciado en la propiedad ControlToValidate como control al que se vincula el control de validación. El control con el que comparar el valor se indica en la propiedad ControlToCompare.

### 7.3.3.- RangeValidator (System.Web.UI.WebControls.RangeValidator)

En muchas ocasiones es necesario comprobar que los datos introducidos por el usuario en un campo están dentro de un determinado rango. El rango puede estar definido entre dos valores numéricos, caracteres o fechas que pueden ser constantes, o tomados a su vez de otros controles.

Para definir el rango de validación el control dispone de las propiedades propias: *MaximumValue* y *MinimumValue* que indican el valor máximo y mínimo entre los que se define el rango de valores considerados válidos. La propiedad *Type* indica el tipo de datos que van a validarse:

Tipo de datos	Descripción
<b>String</b>	Especifica un tipo de datos de cadena.
<b>Integer</b>	Especifica un tipo de datos de entero de 32 bits con signo.
<b>Double</b>	Especifica un tipo de datos de número de punto flotante de precisión doble.
<b>Date</b>	Especifica un tipo de datos de fecha.
<b>Currency</b>	Especifica un tipo de datos de moneda.

Supongamos que queremos comprobar que el valor numérico de un campo de edad esté entre 18 y 65:

Edad:

Emplearemos un control RangeValidator asignando los valores 18 y 65 a las propiedades *MinimumValue* y *MaximumValue* respectivamente. A la propiedad *Type* le asignaremos el valor *Integer*.

```
<asp:RangeValidator
  ID="RangeValidator1"
  runat="server"
  ControlToValidate="TextBox1"
  Display="Dynamic"
  MaximumValue="65" MinimumValue="18"
  Type="Integer">Edad fuera de rango</asp:RangeValidator>
```

Si introducimos un valor fuera del rango fijado y hacemos clic sobre el botón de envío se mostrará automáticamente el mensaje de error asociado a la propiedad *Text* del control.

Edad:  Edad fuera de rango

Es importante fijarse que si dejamos el valor de la caja de texto vacío no se producirá error de validación. Para controlar esa situación deberíamos emplear un control *RequiredFieldValidator* vinculado a la caja de texto.

### 7.3.4.- RegularExpressionValidator

En algunas ocasiones es necesario comprobar si un dato cumple un determinado formato o estructura. Este es el caso de una dirección de correo electrónico, un número de DNI, un código postal o un número de teléfono..., en definitiva; valores que siguen un formato definido. En estos casos puede resultar útil emplear el control `RegularExpressionValidator` empleando una expresión regular que determine el formato esperado.

Una expresión regular es una cadena de texto que describen de forma codificada un determinado formato. Consta de dos tipos de literales: caracteres y metacaracteres. Los literales son caracteres que deben aparecer en cualquier cadena que cumpla con el formato y los metacaracteres símbolos especiales que indican propiedades del formato. Ejemplos:

`\d{5}`

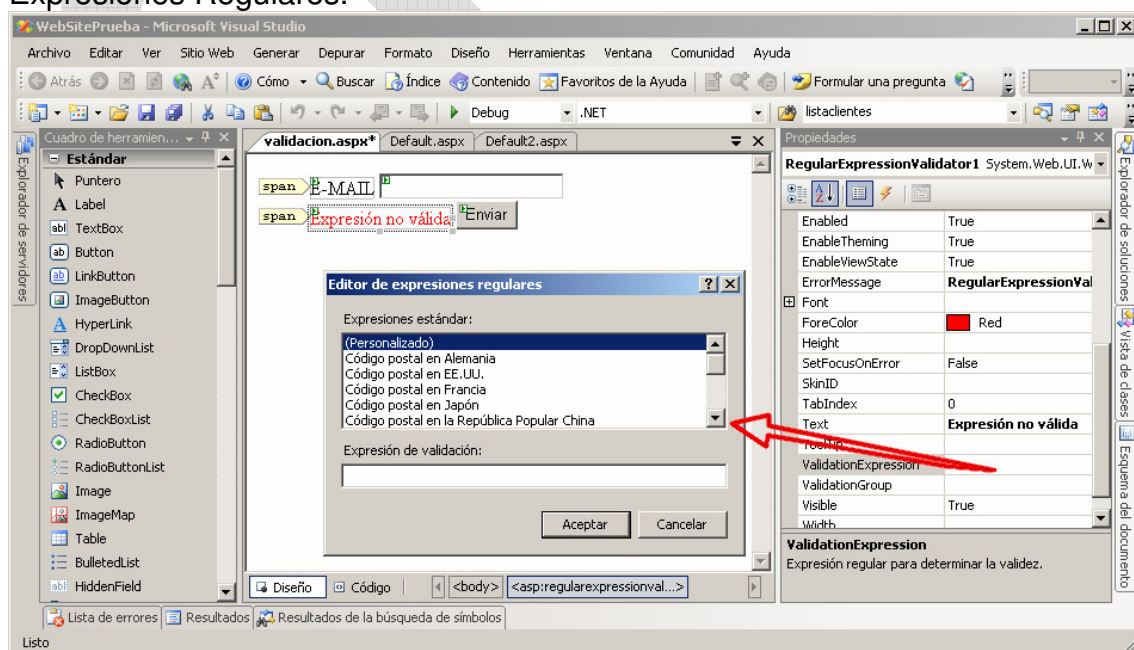
formato consistente en 5 valores de tipo numérico.

`[0-9]{5}|[0-9]{5}-[0-9]{4}`

formato consistente en 5 valores entre 0 y 9, o 5 valores entre 0 y 9 seguidos de un guión y otros 4 valores entre 0 y 9.

El control de validación `RegularExpressionValidator` verifica que el valor de un control sea conforme al formato descrito por una expresión regular indicada en su propiedad `ValidationExpression`. Las expresiones regulares para validar distintos tipos de datos pueden consultarse en Internet o construirlas manualmente. No obstante; el IDE del Visual Studio ofrece ya unas cuantas expresiones regulares.

Si insertamos un control `RegularExpressionValidator` en una pagina ASP.NET, y hacemos click sobre el cuadro de la propiedad `ValidationExpression` en la ventana de sus propiedades, se nos abrirá la ventana del Editor de Expresiones Regulares.

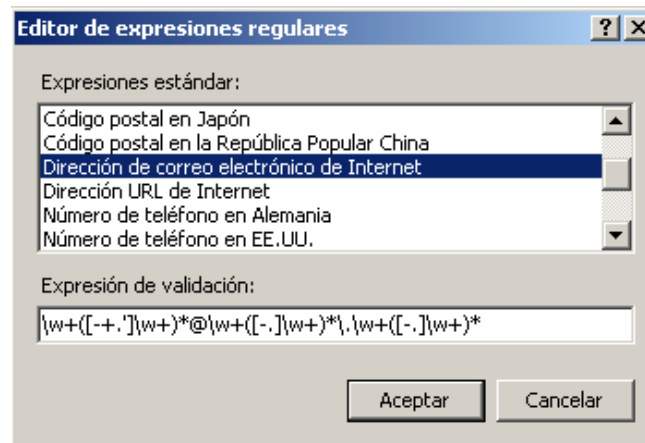


El editor de expresiones regulares ofrece un conjunto de expresiones regulares comunes útiles para validar diferentes tipos de datos.

Supongamos tenemos la siguiente sección de un formulario en la que el usuario debe introducir su dirección de correo electrónico.

E-MAIL

Para validar que la dirección sea válida ( al menos en su formato ), deberemos emplear el control de validación `RegularExpressionValidator`. La expresión regular para validar direcciones de e-mail podemos seleccionarla en el editor:



El código del control quedará así:

```
<asp:RegularExpressionValidator
  ID="RegularExpressionValidator1"
  runat="server"
  ControlToValidate="TextBox1"
  Display="Dynamic"
  EnableClientScript="False"
  ValidationExpression="\w+([-+.' ]\w+)*@\w+([-.] \w+)*\.\w+([-.] \w+)*">Expresión no
válida</asp:RegularExpressionValidator>
```

El control validará al hacer Clic sobre el botón de envío de la página que el valor introducido en el control `TextBox` al que está vinculado cumpla con la expresión regular, provocando un error de validación en caso contrario. Ej:

E-MAIL  Expresión no válida

## 7.3.5.- CustomValidator (*System.Web.UI.WebControls.CustomValidator*)

La validación de algunos datos puede requerir a veces implementar un método de validación específico mediante código. El control CustomValidator permite definir un código que valide el contenido de un control indicado en la propiedad *ControlToValidate*. Al igual que en el caso del resto de controles de validación, éste puede funcionar del lado del cliente o del servidor dependiendo el valor de la propiedad *EnableClientScript*.

Supóngase que implementamos un control de validación personalizado que valide que la entrada de una caja de texto sea un valor no nulo, de tipo numérico y par.

La propiedad *ClientValidationFunction* debe tener por valor el nombre de la función contenida en un script de Javascript encargada de realizar la validación en el navegador del cliente en caso de que funcione del lado del cliente. (*EnableClientScript=True*)

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<script type="text/javascript" language="javascript">
  function validador(source, args) {
    if (args.value % 2 == 0)
      args.IsValid = true;
    else
      args.IsValid = false;
    return;
  }
</script>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:CustomValidator ID="CustomValidator1" runat="server"
        ErrorMessage="CustomValidator"
        Text="Error detectado" ValidateEmptyText="True"
        ClientValidationFunction="validador"
        ControlToValidate="TextBox1"
        OnServerValidate="CustomValidator1_ServerValidate"></asp:CustomValidator>
      <asp:Button ID="Button1" runat="server" Text="Button" />
    </div>
  </form>
</body>
</html>
```

Si el control funciona del lado del servidor (*EnableClientScript=false*), la función de validación se debe implementar como respuesta al evento *ServerValidate*.

```
protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs args)
{
  int valor;
  if (int32.TryParse(args.Value, out valor)) {
    args.IsValid = (valor % 2 == 0); // IsValid = TRUE si la validacion OK.
  }
  else args.IsValid = false;
}
```

Este evento retorna un parámetro de tipo *ServerValidateEventArgs* que contiene el valor a validar en la propiedad *Value*, y el resultado booleano de la validación en la propiedad *IsValid*.

En versiones anteriores de .NET Framework, si el control de destino tenía un valor de cadena vacía; el validador no evaluaba el control de destino y devolvía simplemente que la validación había tenido éxito. La propiedad *ValidateEmptyText* indica si el control de validación debe o no evaluar el control indicado en *ControlToValidate* cuando presenta una cadena vacía por valor.



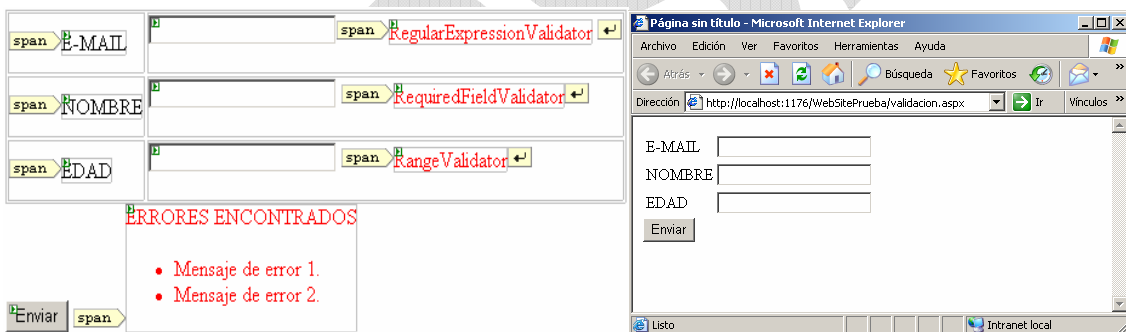
## 7.3.6.- ValidationSummary (System.Web.UI.WebControls.ValidationSummary)

El control ValidationSummary no es propiamente un control de validación. La función de este control mostrar los mensajes de error de varios controles de validación en una lista con puntos, una lista sencilla, o en un simple párrafo. La forma de visualizar los mensajes depende del valor asignado a la propiedad *DisplayMode*. Esta propiedad admite valores de la enumeración *ValidationSummaryDisplayMode*:

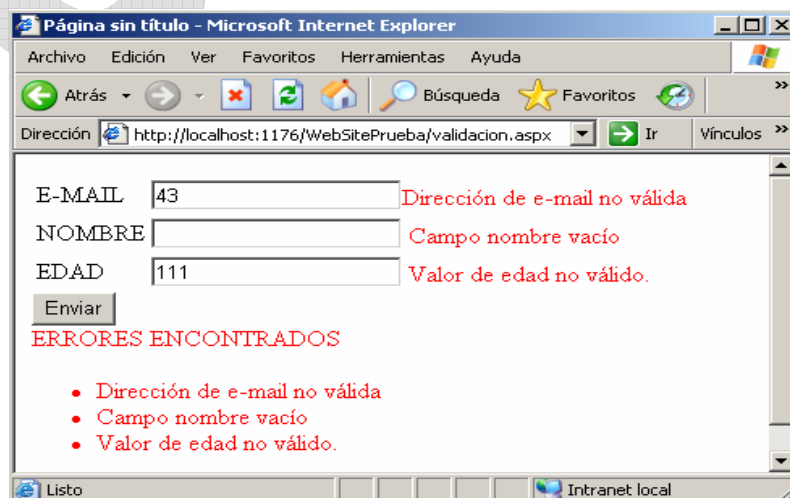
Nombre de miembro	Descripción
<b>List</b>	Resumen de validación que aparece en una lista.
<b>BulletList</b>	Resumen de validación que aparece en una lista con viñetas.
<b>SingleParagraph</b>	Resumen de validación que aparece en un único párrafo.

Todos los controles de validación vistos disponen de la propiedad *ErrorMessage*. Esta propiedad contiene un mensaje de error asociado al control de validación al igual que Text; pero con la diferencia de que es para ser mostrado en el control ValidationSummary presente en la página.

Supongamos el siguiente formulario. A la izquierda vemos el diseño de la página, y a la derecha el resultado en el navegador.



El control ValidationSummary aparece en la parte inferior del diseño de la página, pero no se muestra en el navegador si no hay ningún error de validación:

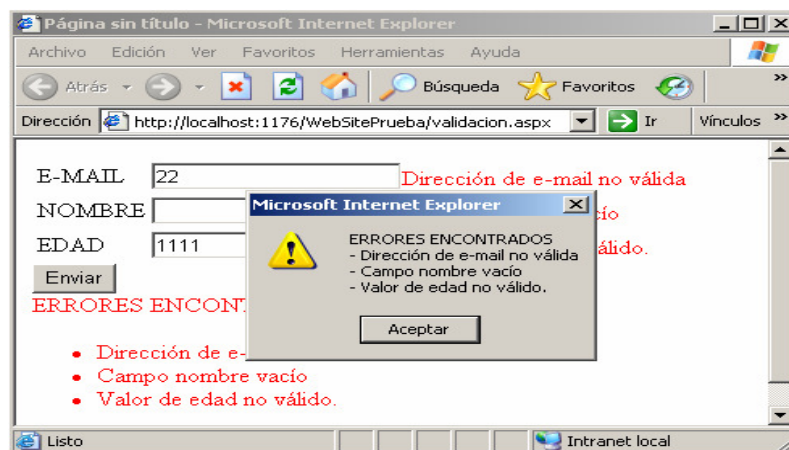




## CONTROLES WEB ASP.NET

En el momento que se produce un error en uno de los controles de validación, el control `ValidationSummary` puede mostrar los mensajes de error en el cuerpo de la propia página Web y/o en un cuadro de dialogo de alerta. Ambos comportamientos se activan mediante las propiedades `ShowSummary` y `ShowMessageBox`.

La propiedad `ShowSummary` posee un valor booleano que indica si el control debe mostrar los mensajes de error en el cuerpo de la página. La propiedad `ShowMessageBox` posee otro valor booleano que indica si los mensajes deben mostrarse en un cuadro de dialogo de alerta. Ambas propiedades son independientes y pueden activarse juntas o alternativamente.



Código de ejemplo:

```
<table>
<tr>
<td><asp:Label ID="Label1" runat="server" Text="E-MAIL"></asp:Label></td>
<td><asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:RegularExpressionValidator
ID="RegularExpressionValidator1"
runat="server" ErrorMessage="Dirección de e-mail no válida"
ControlToValidate="TextBox1" Display="Dynamic" EnableClientScript="False"
ValidationExpression="\w+([-+.'\]\w+)*@\w+([-.\]\w+)*\.\w+([-
.]\w+)*"></asp:RegularExpressionValidator>
</td>
</tr>
<tr>
<td><asp:Label ID="Label2" runat="server" Text="NOMBRE"></asp:Label></td>
<td><asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator
ID="RequiredFieldValidator1" runat="server"
ErrorMessage="Campo nombre vacío" ControlToValidate="TextBox2"
Display="Dynamic"
EnableClientScript="False"></asp:RequiredFieldValidator><br />
</td>
</tr>
<tr>
<td><asp:Label ID="Label5" runat="server" Text="EDAD"></asp:Label></td>
<td><asp:TextBox ID="TextBox6" runat="server"></asp:TextBox>
<asp:RangeValidator ID="RangeValidator1" runat="server"
ErrorMessage="Valor de edad no válido." ControlToValidate="TextBox6"
Display="Dynamic" EnableClientScript="False"
MaximumValue="45" MinimumValue="18" Type="Integer"></asp:RangeValidator>
</td>
</tr>
</table>
<asp:Button ID="Button1" runat="server" Text="Enviar" />
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
EnableClientScript="False" HeaderText="Errores detectados."
ShowMessageBox="True" />
```

Para evitar que los controles muestren sus mensajes de error propios cuando se dispone de un control `ValidationSummary` en la página, hay que asignar el valor "none" a la propiedad `Display` de cada control.

## **7.4.- GRUPOS DE VALIDACION**

Los grupos de validación permiten englobar varios controles de validación de modo que se evalúen conjuntamente de forma independiente a otros presentes en una página. Esto resulte útil en situaciones en las que se desea validar una página por partes. Aquellos controles que deseemos agrupar en un mismo grupo de validación deben compartir el mismo valor en la propiedad `ValidationGroup`.

Para validar los controles de un grupo de validación basta asignar el valor del grupo a la propiedad `ValidationGroup` de un control `Button`, `LinkButton`.. etc, que provoque la recarga de la página con validación (`CausesValidation=true`).

### **Consideraciones sobre la validación del lado del cliente y del servidor**

Todos los controles de validación poseen la propiedad `EnableClientScript`. Como ya se ha mencionado, esta propiedad indica si las comprobaciones deben realizarse en el propio navegador del cliente, o en el servidor.

En la validación en el lado del cliente, los controles de validación presentes en la página ASP.NET generan scripts al ser enviados al usuario. Estos scripts son pequeños códigos en Javascript que son incrustadas en el código HTML de la página, y que se ejecutan en el navegador del cliente con dos objetivos:

- Realizar las validaciones correspondientes de forma independiente al servidor, y mostrar los mensajes de error en el momento en que se produzcan. No obstante, el control `ValidationSummary` sólo se activa al validarse la página.
- Bloquear el envío de datos al servidor mientras no se cumplen todas las validaciones. Esto impide cualquier acción en caso de error por parte del servidor, puesto que no es informado. De esta manera; el servidor no debería recibir nunca datos inválidos.

Los inconvenientes de la validación en el navegador del cliente es que depende de muchos factores: el navegador del usuario, la versión del mismo, su configuración de seguridad. Estos factores pueden hacer que los scripts de Javascript no funcionen y la validación tampoco. Se recomienda por lo tanto mantener siempre un grado de seguridad en el servidor, y no hacer que la validación dependa exclusivamente del navegador del cliente. Esto haría vulnerable a la aplicación Web a errores y ataques.

La validación en el lado del servidor es totalmente independiente del navegador del usuario y eso permite mucha mayor libertad de acción al tratar los errores por código en el servidor. Sin embargo, tiene el inconveniente de que implica el envío de datos al servidor y la recarga de la página cada vez que se validan los datos. Esto puede provocar mayor lentitud en la respuesta de la página.

CLIPSA

## Eventos de Controles Web

Como ya se ha visto, los métodos de respuesta a los eventos de un control Web y de la propia página se implementan en el código de servidor de la página ASP.NET. Sin embargo los sucesos que provocan estos eventos suceden en el navegador del cliente y no en el servidor. Para que se ejecuten los eventos en el servidor correspondientes a las acciones del usuario sobre la página en el navegador es necesario que la página se recargue. Esto también se denomina operación de *PostBack*.

La recarga de la página implica que el navegador vuelve a solicitar la misma página al servidor enviando una serie de datos que informan al servidor de las acciones realizadas por el usuario para que ejecute los eventos correspondientes en el código de la misma.

ASP.NET ofrece dos opciones para la ejecución de los eventos ligados a los controles de la misma.

- Eventos con recarga. Los eventos con recarga van ligados a acciones que provocan la solicitud de la página en el momento de darse provocando la recarga de la misma. Por ejemplo; la pulsación de un botón.
- Eventos sin recarga. Los eventos sin recarga van ligados a acciones que no provocan la recarga de la página. Por ejemplo; la selección de un elemento en una lista. Los eventos correspondientes a estas acciones se ejecutan cuando otro suceso provoca la recarga de la página.

### 8.1 – EVENTOS CON RECARGA. ( *PostBack* )

Las acciones sobre algunos controles provocan la recarga inmediata de la página. Este es el caso de la pulsación del control Button por ejemplo:

Sea:



Donde tenemos un control TextBox1 y un control Button1 cuyo evento Click está asociado al siguiente método:

```
protected void Button1_Click(object sender, EventArgs e)
{
    this.TextBox1.Text = "HOLA HOLA";
}
```

Al pulsar el usuario el botón se provoca la recarga de la página enviando la información al servidor. Para ello se emplea una estructura formulario de HTML ( etiquetas `<form>..</form>` ).

Si nos fijamos en el código de diseño de una página ASP.NET cualquiera; veremos que siempre aparece un formulario dentro del cuerpo de la página conteniendo a todos los controles Web:

## CONTROLES WEB ASP.NET

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default2.aspx.cs"
Inherits="Default2" %>
<!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></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>

      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Button" />

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

Si vemos el código HTML resultante enviado al usuario se aprecian también las etiquetas `<form>` y `</form>` del diseño:

```
<!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><title>
</title></head>
<body>
  <form name="form1" method="post" action="Default2.aspx" id="form1">
    <div>
      <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwULLTE0MDM4MzYxMjNkZkhwiwe/C7o2aH3bAw+3AfMSoEDF" />
    </div>
    <div>
      <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/wEWAwk60KyvCwLs0bLrBgkM54rGBnJMIB3qf9h/vnwXxdoekmT6IkiR" />
    </div>
    <div>
      <input name="TextBox1" type="text" id="TextBox1" />
      <input type="submit" name="Button1" value="Button" id="Button1" />
    </div>
  </form>
</body>
</html>
```

La razón de este formulario que contiene los controles de la página es permitir el envío al servidor de información sobre las acciones del usuario al recargarse la página. Si nos fijamos en el atributo *action* de la etiqueta *form* vemos que su valor es el nombre de la propia página. Esto hace que la página se solicite así misma, es decir; se recargue.

El control Web `<asp:Button>` de ASP.NET se convierte al renderizarse para el navegador del cliente en un control *Input* de HTML de tipo de *Submit* (*input type="submit"*). La pulsación de este control provoca el envío del formulario y la recarga de la página indicada en la URL del atributo *action*. Al mismo tiempo, con el formulario se envían también los datos de los controles contenidos en el mismo. Finalmente, el servidor recibe la petición junto con la información del estado de los controles, identifica el evento que se produjo en el navegador y el control que lo originó, y ejecuta el método del evento Click del control Button. A esto se le denomina evento con devolución de datos.

Otros controles capaces de provocar la recarga de la página son: *Calendar*, *DataGrid*, *DataLit*, *FileUpload*, *GridView*, *ImageButton*, *ImageMap*, *LinkButton*, *Menu*, *Repeater*.

## 8.2.- EVENTOS SIN RECARGA

Estos son eventos que no provocan la recarga automática de la página, por lo que su código en el lado del servidor no se ejecuta en el mismo momento en que suceden, si no cuando otro evento provoca la recarga de la página.

Supongamos la siguiente página donde se tiene un control ListBox provisto de las estaciones del año, un control TextBox donde se desea que se muestre la estación seleccionada, y un control Button:



En el código de servidor de la página se captura el evento `SelectIndexChanged` de la lista para detectar la selección de un elemento y mostrar en la caja de texto la estación seleccionada.

```
protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    this.TextBox1.Text = this.ListBox1.SelectedValue;
}
```

Si ejecutamos la página y seleccionamos un elemento en la lista veremos que la caja de texto no refleja el cambio. Esto es debido a que el control *SelectIndexChanged* no provoca la recarga de la página, y por lo tanto; el evento asociado no se ejecuta. Para que el código del servidor asociado al evento *SelectIndexChanged* se ejecute debe darse la recarga de la página para que el servidor sea informado. Por ello; si pulsamos el botón; veremos que la caja de texto aparece entonces con la estación seleccionada en la lista:

Sin embargo; si queremos que al seleccionar un elemento en la lista aparezca en el momento la estación en la caja de texto es necesario forzar la recarga de la página. Esto es posible asignando el valor `True` a la propiedad `AutoPostBack` del control en la ventana de propiedades.

Al habilitar la propiedad `AutoPostBack`, cualquier acción sobre el control provoca la recarga de la página para informar al servidor y ejecutar así el código del evento correspondiente en el momento. Si ejecutamos la página y seleccionamos una estación veremos que la página se recarga ( exactamente igual que cuando pulsamos el botón ), y a continuación reaparece la caja de texto esta vez con el nombre de la estación seleccionada por el usuario.

Para que esto sea posible, el código HTML enviado al navegador como respuesta incluye una sección de código Javascript que se encarga de provocar el envío del formulario en el momento que el usuario cambia la selección del control `<select>` que representa al `ListBox` en el navegador.

El código siguiente muestra la adhesión del código Javascript para provocar la recarga de la página en el momento en que se produce el evento onChange del control HTML <select> al cambia la selección la opción seleccionada.

```
<!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><title>

</title></head>
<body>
    <form name="form1" method="post" action="Default.aspx" id="form1">
        ...
        ...
        ...
        <select size="4" name="ListBox1"
onchange="javascript:setTimeout('__doPostBack(\'\'ListBox1\','\'\''), 0)" id="ListBox1">
            <option value="Primavera">Primavera</option>
            <option value="Verano">Verano</option>

            <option selected="selected" value="Otoño">Otoño</option>
            <option value="Invierno">Invierno</option>

        </select>
        <input name="TextBox1" type="text" value="Otoño" id="TextBox1" />
        <input type="submit" name="Button1" value="Button" id="Button1" />
    </form>
</body>
</html>
```

A continuación se muestra una pequeña lista con la relación de eventos, controles y cuales de ellos provocan la recarga automática de la página.

Event	Web Controls That Provide It	Always Posts Back
Click	Button, ImageButton	True
TextChanged	TextBox (fires only after the user changes the focus to another control)	False
CheckedChanged	CheckBox, RadioButton	False
SelectedIndexChanged	DropDownList, ListBox, CheckBoxList, RadioButtonList	False

Algunos de los controles más comunes con eventos que no producen la recarga de la página son: CheckBox, CheckBoxList, DropDownList, ListBox, RadioButton, RadioButtonList, TextBox.



### 8.3.- ENVIO DE EVENTOS AL SERVIDOR

Los controles Web ASP.NET poseen dos propiedades únicas que favorecen su uso dentro del entorno de una aplicación Web frente a los controles HTML del lado del cliente. Por un lado; mantienen el estado a través de las recargas de la misma página, y por otro; provocan eventos en el servidor como respuesta a acciones en el navegador. El mecanismo por el que ambas propiedades son posibles radica en el uso de Javascript y campos ocultos en el código HTML de respuesta enviado al navegador del cliente.

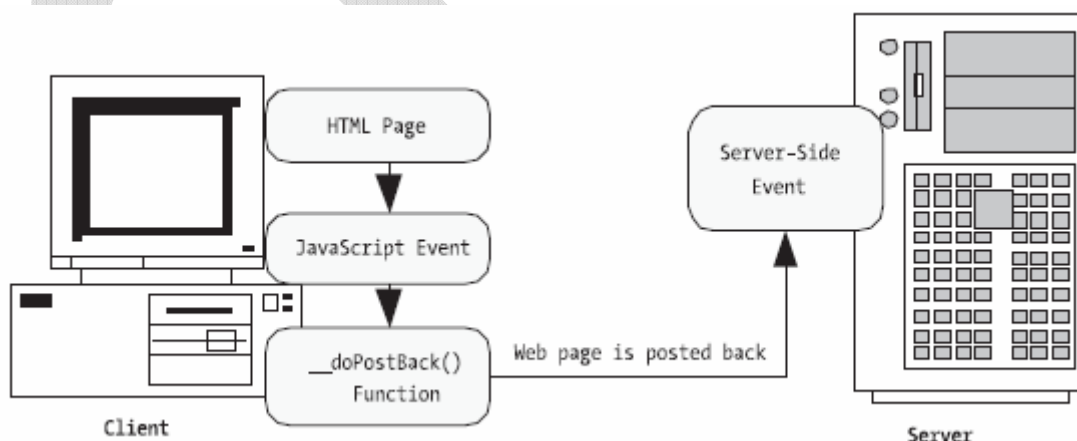
Cuando una página ASP.NET contiene uno o más controles Web con la opción `AutoPostBack` habilitada, el código HTML de respuesta lleva incluido una función especial escrita en Javascript llamada `_doPostBack()`. La misión de esta función es provocar la recarga de la página al ser invocada.

```
<script language="text/javascript">
<!--
function __doPostBack(eventTarget, eventArgument) {
var theform = document.Form1;
theform.__EVENTTARGET.value = eventTarget;
theform.__EVENTARGUMENT.value = eventArgument;
theform.submit();
}
// -->
</script>
```

Adicionalmente a esta función, también se añaden dos campos ocultos donde la función `AutoPostBack` almacena la información relativa a las acciones del usuario. Esta información es recuperada por el servidor al realizarse la recarga y en base a ella se ejecutan los eventos correspondientes en el código de servidor.

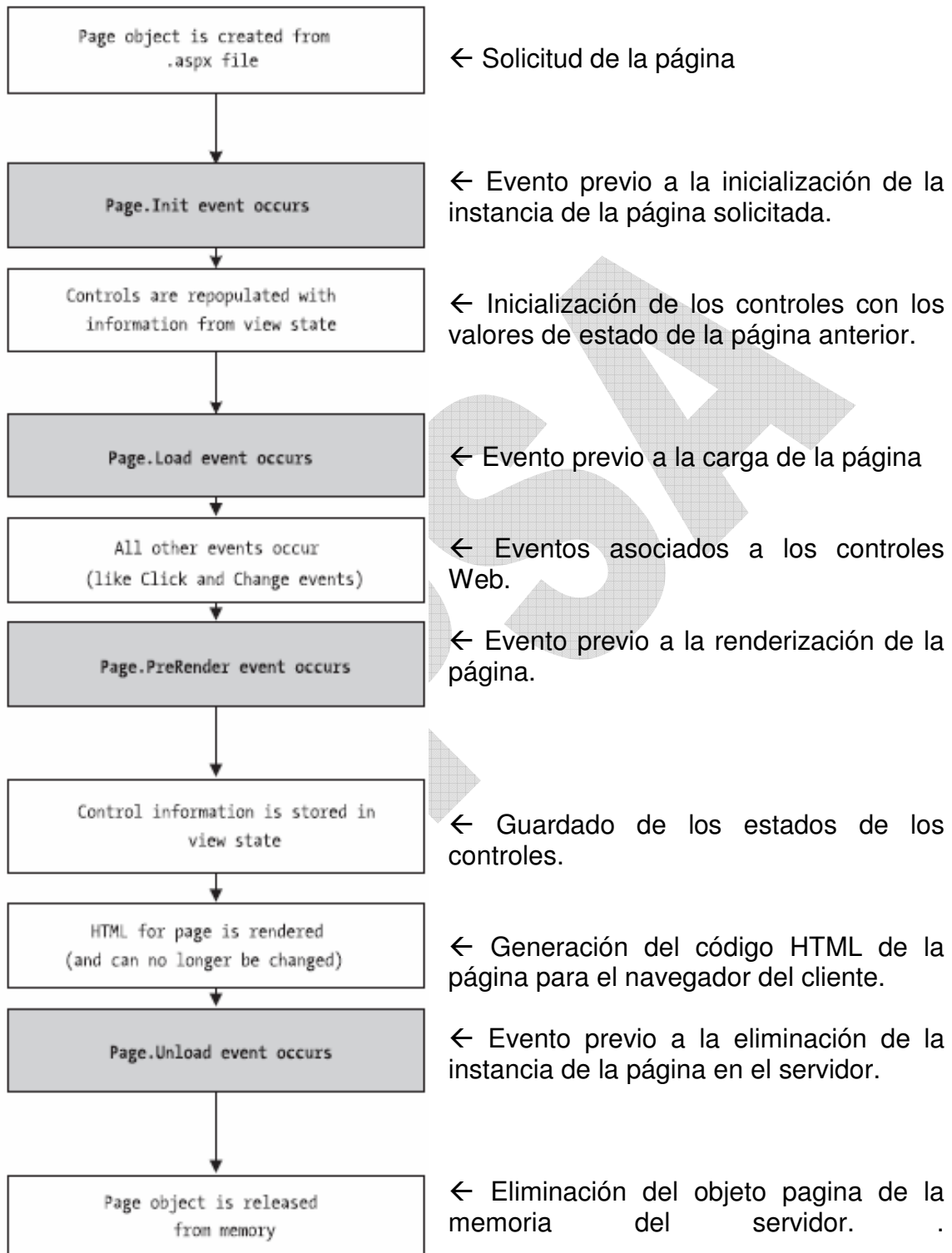
```
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
```

El siguiente gráfico muestra esquemáticamente el funcionamiento de este mecanismo:



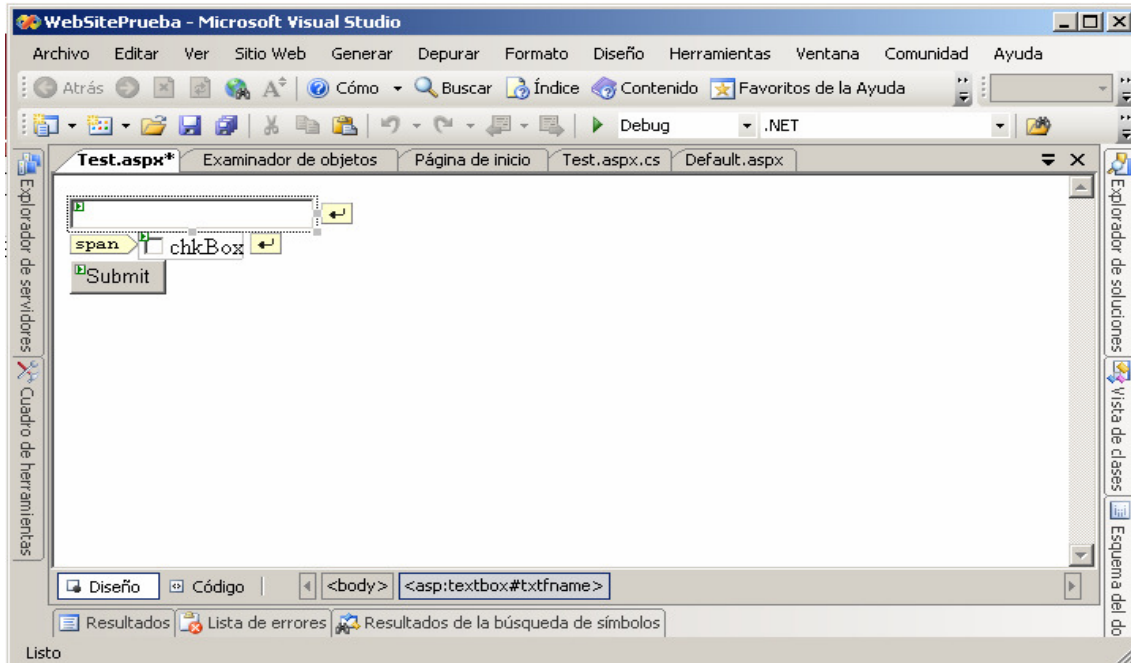
### 8.4.- EVENTOS DE CONTROLES WEB

La ejecución de los eventos asociados a los controles Web se integra dentro de la secuencia de eventos propios de la página Web siguiendo el siguiente orden:



## Ejemplo

Vamos a diseñar una página ASP.NET para apreciar los eventos de página y el orden en que estos se suceden al procesarse la misma. Para ello, debemos dotar a la página de un control Web TextBox, un CheckBox y un Button:



### test.aspx

```
<%@ Page language="c#" Trace="false" CodeFile="Test.aspx.cs" AutoEventWireup="true"
Inherits="Test.Input" %>
<!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>prueba</title></head>
<body>
<form id="Test" method="post" runat="server">
  <asp:textbox id="txtBox" runat="server"
    OnTextChanged="txtBox_TextChanged"></asp:textbox>
  <br />
  <asp:checkbox id="chkBox" runat="server" text="chkBox"
    OnCheckedChanged="chkBox_CheckedChanged"></asp:checkbox>
  <br />
  <asp:button id="btnSubmit" runat="server" text="Submit"
    OnClick="btnSubmit_Click"></asp:button>
</form>
</body>
</html>
```

A continuación; seleccionaremos los eventos de los siguientes controles:

Button	→ Evento Click
TextBox	→ Evento TextChanged
CheckBox	→ Evento CheckedChanged

Con ello se crearán los siguientes métodos en el código asociado a la página:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
}

protected void txtBox_TextChanged(object sender, EventArgs e)
{
}

protected void chkBox_CheckedChanged(object sender, EventArgs e)
{
}
```

## CONTROLES WEB ASP.NET

Finalmente, añadiremos los métodos delegados siguientes para atender los eventos originados por la página. En cada método se incluye una llamada a un método auxiliar ( *mostrarMensaje* ) que inscribe un mensaje en la respuesta HTML para el usuario según se genera empleando la propiedad Response.

### test.aspx.cs

```
namespace Test
{
    public partial class Input : System.Web.UI.Page
    {
        private void mostrarMensaje(string message)
        {
            Response.Write("--> " + message + "<br />");
        }

        // Eventos de inicializacion de pagina
        protected void Page_PreInit(object sender, EventArgs e)
        {
            this.mostrarMensaje("Evento PreInit");
        }
        protected void Page_Init(object sender, EventArgs e)
        {
            this.mostrarMensaje("Evento Init");
        }
        protected void Page_InitComplete(object sender, EventArgs e)
        {
            this.mostrarMensaje("Evento InitComplete");
        }

        // Eventos de carga de pagina
        protected void Page_PreLoad(object sender, EventArgs e)
        {
            this.mostrarMensaje("Evento PreLoad");
        }
        protected void Page_Load(object sender, EventArgs e)
        {
            this.mostrarMensaje("Evento Load");
            if (ViewState["dato"] == null)
            {
                ViewState["dato"] = "valor";
                this.mostrarMensaje("dato almacenado en viewstate");
            }
            else
            {
                this.mostrarMensaje("dato presente en viewstate" + ViewState["dato"].ToString());
            }
        }
        protected void Page_LoadComplete(object sender, EventArgs e)
        {
            this.mostrarMensaje("Evento LoadComplete");
        }
    }
}
```

En el método del evento Load, se incluye código para comprobar, añadir, y consultar el valor “dato” almacenado en el viewstate de la página. Esto permitirá observar los eventos de carga y guardado del estado de página.

```
// Eventos de render de pagina. ( Generacion del código respuesta para el usuario )
protected void Page_PreRender(object sender, EventArgs e)
{
    this.mostrarMensaje("Evento PreRender");
}
protected void Page_PreRenderComplete(object sender, EventArgs e)
{
    this.mostrarMensaje("Evento PreRenderComplete");
}

// Evento de descarga de instancia de pagina
protected void Page_Unload(object sender, EventArgs e)
{
    //this.mostrarMensaje("Guardado del ViewState"); // Ya no existe Response.
}

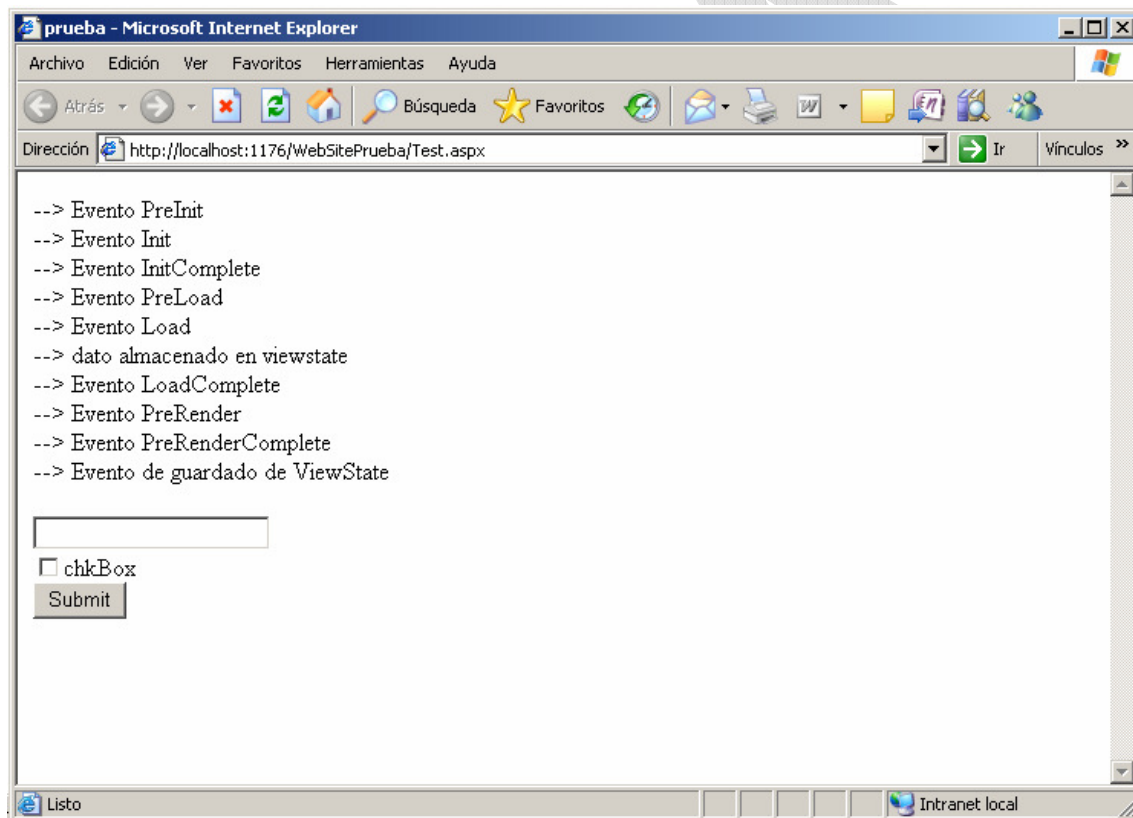
// Carga y Evento de guardado de ViewState
protected override void LoadViewState(object o)
{
    this.mostrarMensaje("Carga del ViewState");
    base.LoadViewState(o);
}
protected void Page_SaveStateComplete(object sender, EventArgs e)
{
    this.mostrarMensaje("Evento de guardado de ViewState");
}
}
```

Modificamos los métodos de eventos de los controles Web de la página, incluyendo una llamada al método *mostrarMensaje* para registrar el momento en el que éstos son ejecutados.

```
// Eventos de Controles con devolución de datos ( Provoca la recarga de la página )
protected void btnSubmit_Click(object sender, EventArgs e)
{
    this.mostrarMensaje("Evento Click de control Web Button.");
}

// Eventos de Controles sin devolución de datos ( No provocan recarga de la página )
protected void txtBox_TextChanged(object sender, EventArgs e)
{
    this.mostrarMensaje("Evento TextChanged de control Web TextBox.");
}
protected void chkBox_CheckedChanged(object sender, EventArgs e)
{
    this.mostrarMensaje("Evento CheckedChanged de control Web CheckBox.");
}
}
```

Si ejecutamos la página; obtendremos el siguiente resultado:



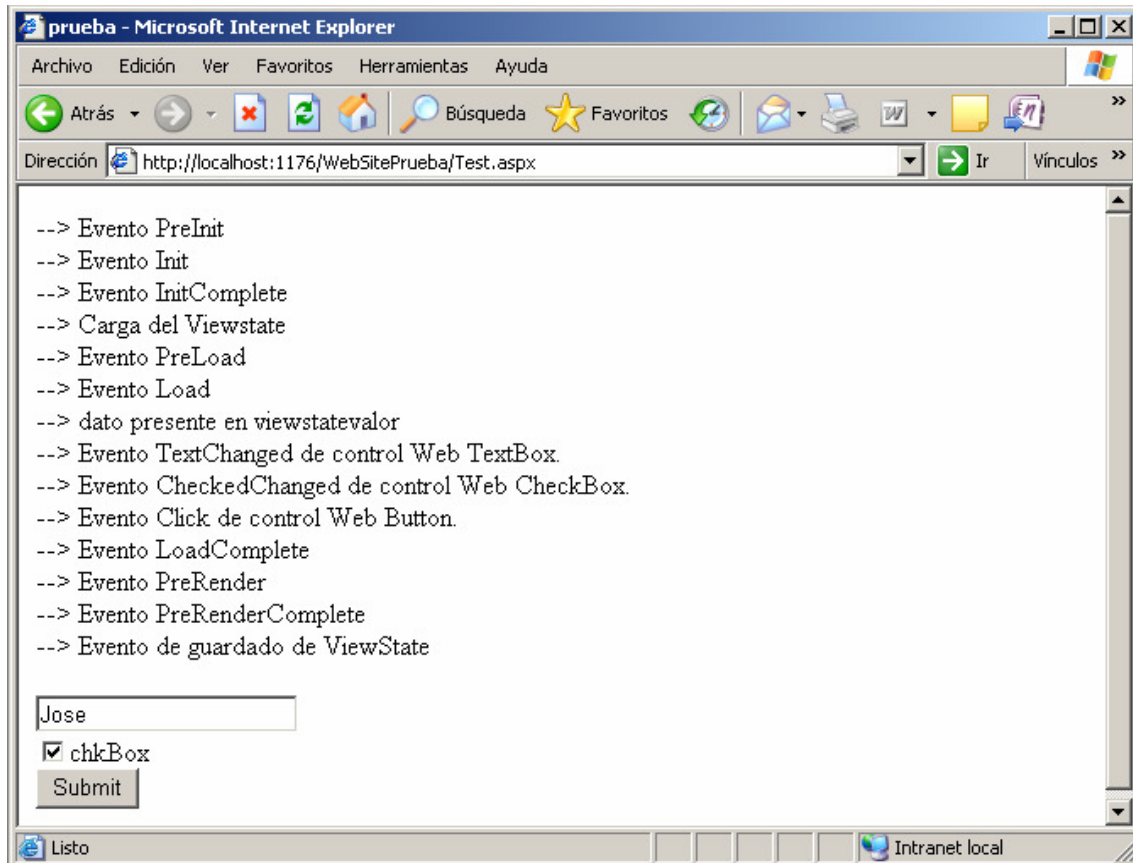
Los mensajes inscritos por cada uno de los eventos del proceso de página, aparecen justo por encima del diseño de la misma debido a que se ejecutan antes que se inscriba el código HTML de la página y sus controles en el proceso de renderización.

Los mensajes aparecen por tanto en el orden en que se han ido ejecutando los distintos eventos de la página.

## CONTROLES WEB ASP.NET

A continuación, escribimos un texto cualquiera en la caja de texto y marcamos la casilla de verificación. Estas acciones no producen ningún efecto inmediato ya que los eventos *textChanged* y *checkedChanged* sólo se ejecutan en el servidor, y para ello es necesario la recarga de la página.

Sin embargo, la pulsación del botón “*Submit*” sí que produce la recarga de la misma y la ejecución de todos los eventos pendientes.



Es particularmente interesante el orden en que se ejecutan los eventos de los controles Web contenidos en la página.

Todos ellos se ejecutan después del evento Load. Primero son aquellos eventos asociados al cambio en el estado de los controles ( el caso de *TextChanged* y *CheckedChanged* ), que no producen la recarga de la página.

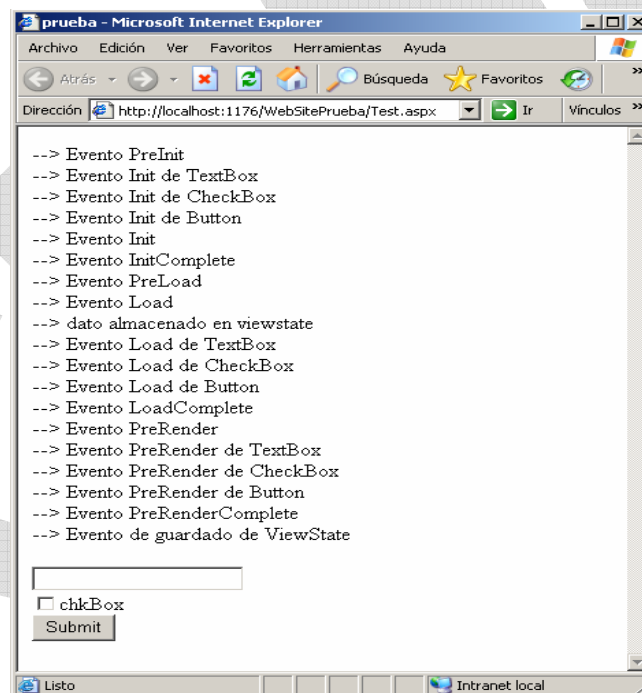
Después se ejecuta el evento del control que produjo la recarga de la propia página. ( *Click* ). Una vez que se han ejecutado los eventos de todos los controles, se produce el evento *LoadComplete*.

Algunos de los eventos de la clase *Page*, tales como *Load*, *Init*, *PreRender*, *Unload*; son realmente eventos heredados de la clase base *System.Web.UI.Control*. Estos eventos son heredados también por la clase *System.Web.UI.WebControl*, padre de todos los controles Web de ASP.NET. Por lo tanto; los controles Web también disponen de estos eventos.

Es importante comprobar la relación entre estos eventos básicos de los controles Web con los de la propia página Web; modificaremos el anterior ejemplo añadiendo los métodos para atender los eventos *Load*, *Init*, y *PreRender* de los controles Web.

```
protected void txtBox_Init(object sender, EventArgs e) {
    this.mostrarMensaje("Evento Init de TextBox");
}
protected void txtBox_Load(object sender, EventArgs e) {
    this.mostrarMensaje("Evento Load de TextBox");
}
protected void txtBox_PreRender(object sender, EventArgs e) {
    this.mostrarMensaje("Evento PreRender de TextBox");
}
protected void chkBox_Init(object sender, EventArgs e) {
    this.mostrarMensaje("Evento Init de CheckBox");
}
protected void chkBox_Load(object sender, EventArgs e) {
    this.mostrarMensaje("Evento Load de CheckBox");
}
protected void chkBox_PreRender(object sender, EventArgs e) {
    this.mostrarMensaje("Evento PreRender de CheckBox");
}
protected void btnSubmit_Init(object sender, EventArgs e) {
    this.mostrarMensaje("Evento Init de Button");
}
protected void btnSubmit_Load(object sender, EventArgs e) {
    this.mostrarMensaje("Evento Load de Button");
}
protected void btnSubmit_PreRender(object sender, EventArgs e) {
    this.mostrarMensaje("Evento PreRender de Button");
}
```

Si ejecutamos la página obtendremos la siguiente resultado:



En el estado de inicialización de la página; los eventos *Init* de los controles Web se ejecutan antes que el de la página.

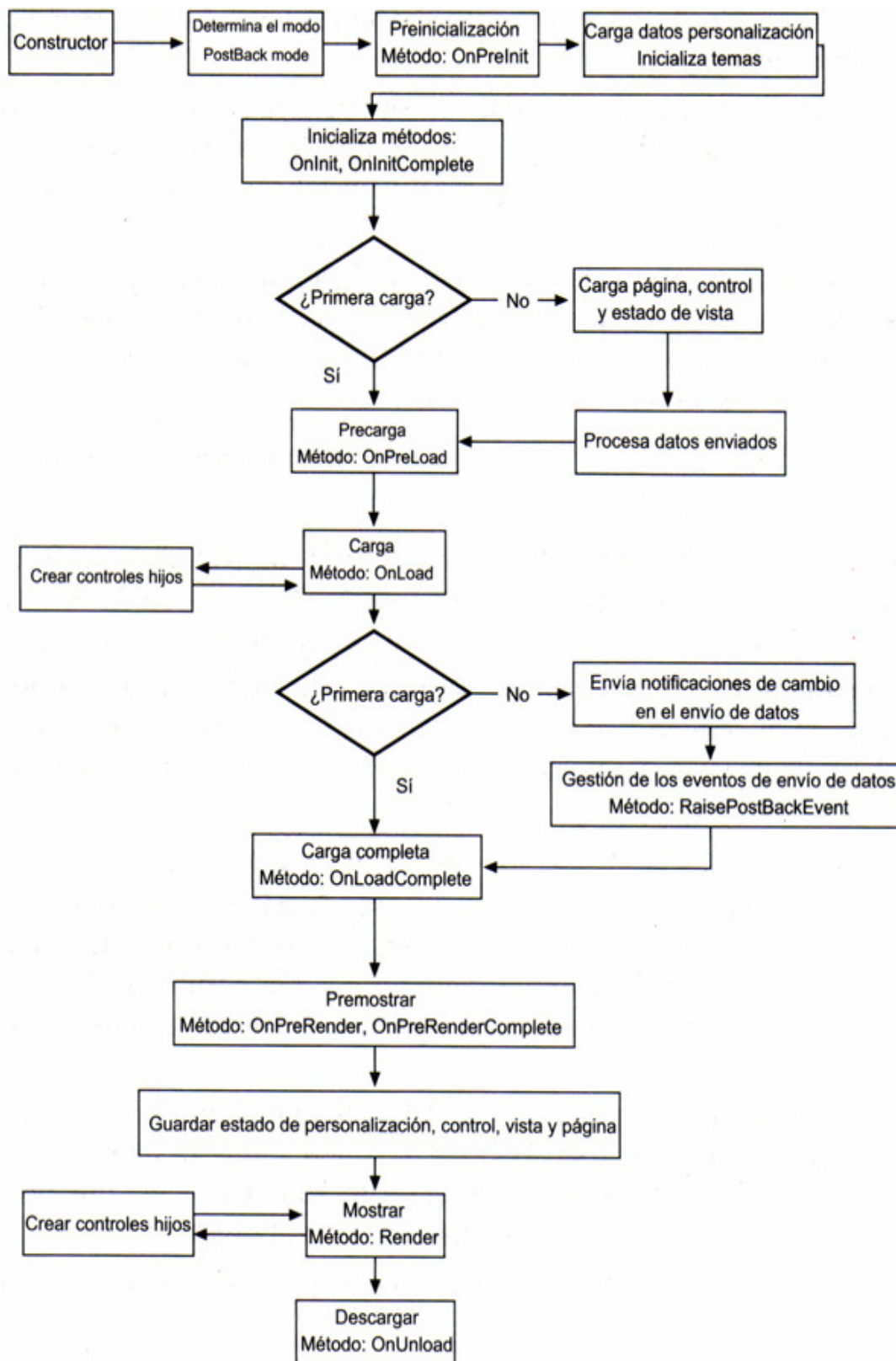
En el estado de carga; el evento *Load* de la página es el primero en ejecutarse seguido del de los controles Web.

Finalmente; en la fase previa a la generación de la respuesta al usuario, el evento de la página se ejecuta el primero seguido del de cada uno de los controles Web.



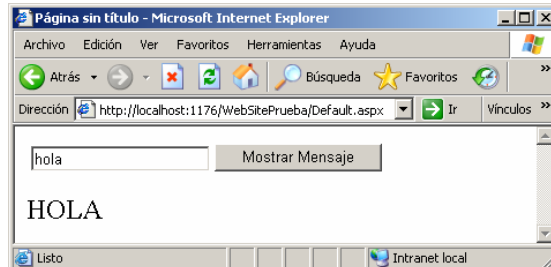
## CONTROLES WEB ASP.NET

Para finalizar se muestra la descripción esquematizada de cada uno de los estados junto con los eventos que producen:



## EJERCICIOS

**1.-** Crea una página ASP.NET provista de un formulario con una caja de texto y un botón. Al pulsarse el botón deberá mostrarse un mensaje con el texto introducido en letras mayúsculas. Si el usuario no escribe ningún texto, se mostrará el mensaje “VACIO”.

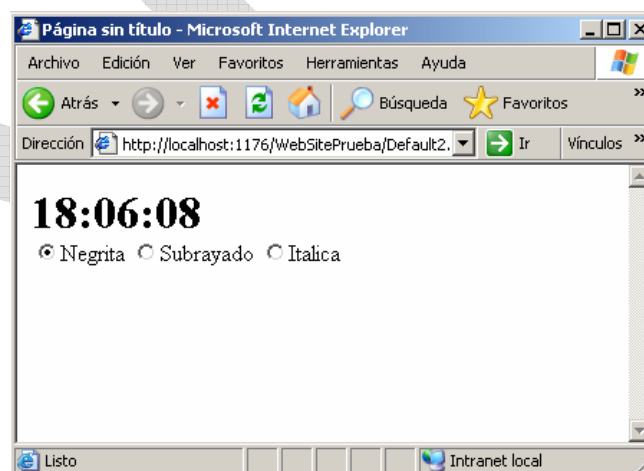


**2.-** Crea una página ASP.NET que muestre el texto “ESTO ES ASP.NET” en color negro y tres controles Button con los nombres “Rojo”, “Verde” y “Azul”. La página debe funcionar de manera que al pulsar cada uno de los botones, el texto del mensaje cambie al color correspondiente.

**3.-** Crea una página ASP.NET con dos cajas de texto. En una se podrá introducir texto mientras que la otra estará deshabilitada. Al escribir el usuario un texto en una de las cajas de texto, éste deberá aparecer automáticamente en la otra al cambiar de foco.

**4.-** Crea una página ASP.NET provista de tres botones de opción y un mensaje que muestre la hora actual. Los botones de opción deben mostrar las opciones de estilos de letra: “Negrita”, “Itálica” y “Subrayado”.

Al seleccionar cada uno de los botones de opción el mensaje con la hora deberá aplicar el estilo de fuente correspondiente.

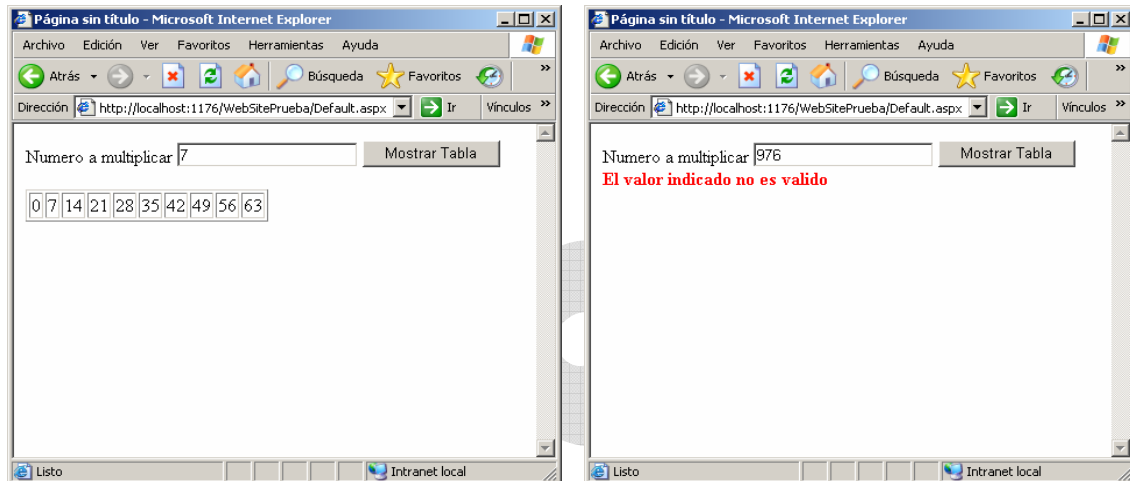


## CONTROLES WEB ASP.NET

**5.-** Repite el ejercicio anterior, pero ahora empleando tres botones de opción situados en un panel con el nombre “Control de colores”. Añade un botón con el título “Cambiar Color” que haga que el texto cambie al color indicado.

Deshabilita finalmente el botón “Cambiar Color”, y haz que el texto cambie de control al seleccionar simplemente un botón de opción.

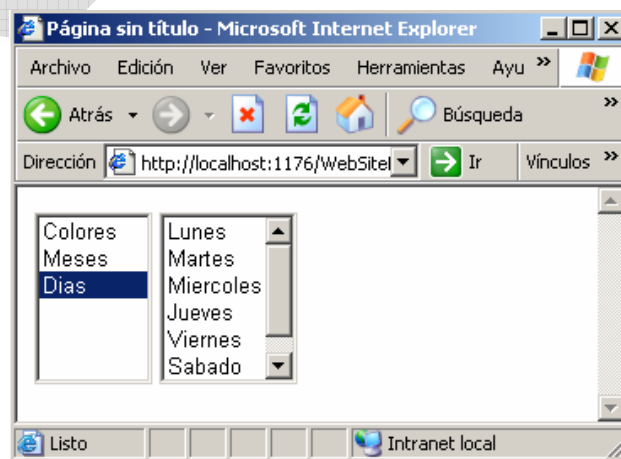
**6.-** Crea una página ASP.NET provista de un formulario con una caja de texto en la que el usuario deberá introducir un valor numérico, y un botón “Mostrar Tabla” que al ser pulsado mostrará la tabla de multiplicar del número indicado.



Si el usuario introduce un valor no numérico, o que no está entre 0 y 10; deberá mostrarse un mensaje de error con el mensaje en rojo “El valor indicado no es válido”.

**7.-** Crea una página ASP.NET provista de dos listas. La primera lista contendrá los siguientes elementos: “Colores”, “Meses”, “Días”. La segunda lista deberá rellenarse dinámicamente en función de la selección en la primera lista.

Si se seleccionan colores; aparecerán las opciones “Rojo”, “Verde”, “Azul”. Si seleccionan “Meses”, aparecerán los nombres de los meses del año. Si se seleccionan “Días”, aparecerán los días de la semana.



**8.-** Crear una página ASP.NET con un formulario de alta para un sitio Web. El formulario debe constar con los siguientes campos: Nombre, Correo electrónico, Teléfono de contacto, Contraseña, y confirmación de contraseña.

El formulario debe estar provisto de un botón de envío. Deben validarse los siguientes requisitos: El campo nombre debe ser obligatorio. El teléfono debe ser un valor de 9 cifras exclusivamente. El correo electrónico debe validarse conforme a la expresión regular correspondiente. Los campos de contraseña son obligatorios y deben coincidir.

Si se detectan campos inválidos deben mostrarse los correspondientes mensajes de error en una lista en la parte baja del formulario. Si todos los campos son correctos deberá mostrarse a continuación una página con el mensaje “Registro realizado con éxito, Gracias.”

*\* Observa las diferencias de funcionamiento empleando la comprobación desde servidor, y en cliente.*

**9.-** Crea la siguiente página que representa el formulario de selección de componentes para una tienda de venta on-line de equipos informáticos.

El precio base de los equipos es de 400€. Un equipo puede tener 1,2,4 o 8 Gb de RAM, con un incremento de 30€, 45€, 60€, y 90€ respectivamente sobre el precio base. Si se le añade un monitor TFT se incrementa 200€. Si se añade un disco duro de 300Gb de incrementa 50€. Si se añade una regrabadora de DVD se incrementa 40€. El cliente puede elegir pagarlo por crédito con un sobrecargo del 3%, o bien al contado con una rebaja del 1%.

Al pulsar el botón “Calcular Precio” deberá mostrarse el mismo en función de las selecciones.