Emitir código JavaScript desde WebForms y controles web personalizados

Luis Miguel Blanco Ancos

La dependencia del lado cliente

La llegada de ASP.NET al mundo del desarrollo de aplicaciones para Internet ha supuesto una considerable mejora con respecto al modelo de programación ofrecido por ASP 3 (también conocido como ASP clásico), que imperaba anteriormente.

Los cambios en dicho modelo han sido cuantiosos y profundos, contribuyendo entre otros muchos aspectos a la consecución de un código más efectivo en un menor espacio de tiempo.

No cabe duda de que el paradigma de desarrollo propuesto por ASP.NET es a todas luces más óptimo que el disponible con la anterior versión de la plataforma de desarrollo web de Microsoft, residiendo uno de sus puntos destacables en el conjunto de controles Web, los cuales ofrecen un abanico de posibilidades de programación más amplio y rico en métodos, propiedades y eventos que los tradicionales controles html de ASP clásico; no obstante, como todos sabemos, también podemos seguir utilizando estos últimos en formularios ASP.NET.

Podemos programar gran parte del comportamiento y funcionalidad de los nuevos controles Web desde el lado del servidor, dado que su arquitectura está diseñada para ser gestionada en esta parte del flujo de proceso de la aplicación, lo que ofrece una potencia inusitada con respecto a la programación web empleando ASP clásico.

Sin embargo, el hecho de que ahora tengamos una mayor capacidad de control del lado servidor en nuestras aplicaciones, no quiere decir que esto sea la panacea que vaya a resolver todos nuestros problemas de desarrollo, ni tampoco resulta conveniente su uso desmesurado, ya que seguirán existiendo operaciones cuyo tratamiento sea más adecuado efectuarlo en el lado cliente, dado que su gestión en el servidor provocaría una pérdida de rendimiento.

Es por ello que derivar la ejecución de ciertas tareas para que se procesen en el navegador cliente sigue siendo un factor primordial en aras de conseguir una aplicación equilibrada en lo que al consumo de recursos se refiere, y ágil en cuanto a su ejecución. De esta manera se descarga en muchos casos al servidor de un importante número de operaciones que de otro modo tendría que soportar, o que en determinadas situaciones ni siquiera sería viable su ejecución en el servidor por cuestiones de eficacia de la aplicación.

Por este motivo, tal y como reza el título del artículo, vamos a describir los mecanismos con los que cuenta ASP.NET para la generación de código de script en el

navegador cliente, y más concretamente JavaScript, ya que representa el lenguaje de script más ampliamente utilizado en el desarrollo de aplicaciones para Internet. Como navegador web empleado en la demostración de los diversos ejemplos utilizaremos Internet Explorer.

También haremos un repaso de las técnicas utilizadas para asociar el código JavaScript que generemos con los eventos de los controles de nuestros formularios web, y un pequeño truco para facilitar la generación de este tipo de código desde el servidor.

Dado que este es un artículo que versa sobre cómo aplicar JavaScript, se asume por parte del lector un conocimiento al menos básico de este lenguaje.

Todos los casos tratados se ilustrarán a través de ejemplos repartidos en dos soluciones de Visual Studio .NET, que el lector puede descargar, como material de apoyo para este artículo, desde la dirección www.dotnetmania.com.

Comencemos pues por la primera de estas soluciones, EjArticASPNETJS, en la que podemos encontrar un proyecto de igual nombre con varios webforms, que ilustran algunos de los ejemplos de los próximos apartados.

El formulario de entrada de este proyecto, Inicio.aspx, dispone de varios botones que nos darán paso a cada uno de los webform de ejemplo.

Al estilo antiguo

Si nos encontramos en la situación de ser un programador de ASP clásico que necesita migrar hacia ASP.NET, debemos saber que la creación de código JavaScript podemos seguir realizándola en el modo al que ya estamos acostumbrados, lo que facilita la curva de aprendizaje en lo que a este particular respecta.

Para ello crearemos el formulario WebForm1, y en modo de diseño situaremos dos controles INPUT de tipo texto, cuya finalidad será la de comprobar la cantidad de caracteres que hemos escrito en uno de los controles, mostrando dicho número en el otro.

Seguidamente accederemos al código html del formulario y entre las etiquetas <head></head> escribiremos nuestro código JavaScript, utilizando, naturalmente, las etiquetas <script></script>. Este código consistirá en una función que será llamada al producirse el evento onblur en uno de los controles, y que efectuará el cálculo anteriormente mencionado, como vemos en el siguiente código fuente.

```
<HEAD>
        ....
        ....
        <script language="javascript">
        function CalcularLongitud()
        {
```

```
                var sCadena = new String();
                sCadena = document.Form1.txtCadena.value;
                document.Form1.txtLongitud.value = sCadena.length;
        }
        </script>
</HEAD>
<body>
        <form id="Form1" method="post" runat="server">
                Escribir texto: <INPUT id="txtCadena" type="text"
onblur="CalcularLongitud()">
                <br>
                Longitud texto: <INPUT id="txtLongitud" type="text" size="2">
        </form>
</body>
```

Terminada la creación de este formulario lo ejecutaremos, comprobando, cómo el código cliente que hemos escrito se ejecuta al desencadenarse el evento correspondiente.

Puede darse también el caso de que tengamos un elevado número de funciones en JavaScript, parte de las cuales son de propósito general, que necesitaremos reutilizar en la mayoría de los formularios de la aplicación.

Para no tener que volver a repetir esas funciones en todos los formularios, podemos crear un archivo que contenga este código JavaScript de uso general, y referirnos a él usando el atributo src de la etiqueta script, como hacemos en el formulario WebForm2 del proyecto. Véase el siguiente fuente.

```
//----- archivo FuncionesVarias.js ----------
function PasarMayusculas(oTextBox)
{
        var sTextoControl = new String();
        sTextoControl = oTextBox.value;
        oTextBox.value = sTextoControl.toUpperCase();
}
//--------------------
function CambiarColor(oTextBox)
{
        oTextBox.style.backgroundColor="#66ff99";
}

<!-- código html de WebForm2 -->
....
<HEAD>
        ....
        <script language="javascript" src="FuncionesVarias.js"></script>
</HEAD>
```

```
<body>
        <form id="Form1" method="post" runat="server">
                Nombre:<INPUT id="txtNombre" type="text"
onblur="PasarMayusculas(this)">
                <br>
                Ciudad:<INPUT id="txtCiudad" type="text"
onblur="CambiarColor(this)">
                <br>
                Provincia:<INPUT type="text">
        </form>
</body>
....
```

Lo que hemos hecho aquí ha sido añadir el archivo FuncionesVarias.js al proyecto, y escribir en él las funciones PasarMayusculas() y CambiarColor(), para después conectarlas mediante el evento onblur de los controles del formulario txtNombre y txtCiudad.

Generación de código script al estilo ASP.NET

Dejando a un lado ya la antigua técnica de generación de JavaScript, pasemos a describir los mecanismos de que dispone ASP.NET para realizar todo el proceso de emisión de este tipo de código desde el lado servidor.

Partiendo del hecho de que vamos a trabajar con controles Web, debido a las múltiples ventajas que presentan, no es posible, dada la naturaleza de estos controles, efectuar la conexión entre una función JavaScript y un evento del control de la misma forma que veíamos en el apartado anterior con los controles html. Ahora debemos utilizar un mecanismo distinto, consistente en escribir el código JavaScript en el code-behind del webform, procediendo a su emisión hacia el navegador mediante el método RegisterClientScriptBlock de la clase Page, que tiene la siguiente sintaxis:

Page.RegisterClientScriptBlock(Clave, CodigoScript)

A continuación describimos los parámetros de este método:

- **Clave**. Cadena que identifica de modo unívoco al bloque de código JavaScript que estamos registrando/generando para el formulario. Esto nos permite poder utilizar varias veces este método para registrar diferentes bloques de código script, cada uno con su correspondiente identificador.
- **CodigoScript**. Cadena que contiene el bloque de código JavaScript que vamos a registrar/enviar al navegador web desde el code-behind.

Como ejemplo de esta técnica de creación de código en el cliente web, el formulario WebForm3 de nuestro proyecto de prueba contiene, en su método Load, las instrucciones para generar nuestro código de script pertinente y registrarlo con RegisterClientScriptBlock, como vemos en el siguiente código fuente.

```
Private Sub Page_Load(....) Handles MyBase.Load
....
Dim sJS As String
sJS = "<script language=""javascript"">"
sJS &= "function CambiarMayMin()"
sJS &= "{"
sJS &= "var sTexto = new String();"
sJS &= "sTexto = document.Form1.txtNombre.value;"
sJS &= "if (sTexto.length > 0)"
sJS &= "{"
sJS &= "if (document.Form1.chkMayMin.checked)"
sJS &= "{"
sJS &= "document.Form1.txtNombre.value = sTexto.toLocaleUpperCase();"
sJS &= "}"
sJS &= "else"
sJS &= "{"
sJS &= "document.Form1.txtNombre.value = sTexto.toLowerCase();"
sJS &= "}"
sJS &= "}"
sJS &= "else"
sJS &= "{"
sJS &= "alert(""No hay texto para convertir"");"
sJS &= "}"
sJS &= "}"
sJS &= ""
sJS &= "function ContarCaracteres()"
sJS &= "{"
sJS &= "var sTexto = new String();"
sJS &= "sTexto = document.Form1.txtNombre.value;"
sJS &= "document.Form1.txtNumCaracteres.value = sTexto.length;"
sJS &= "}"
sJS &= "</script>"
' generar el código de script en el navegador web
Me.RegisterClientScriptBlock("CodJS", sJS)
....
```

Al ejecutar este webform, si accedemos a la opción *Ver código fuente* en el navegador, comprobaremos como allí aparece el código script que hemos emitido desde el code-behind de la página, todo ello sin necesidad de haber manipulado directamente el html del webform.

La única pega que podríamos achacar a este resultado sería que el código JavaScript generado se dispone en una línea única, lo que dificulta su lectura en el navegador, y lo que es más grave, cuando existe una gran cantidad de este código se pueden producir errores en su ejecución.

Para remediar este inconveniente tan sólo hemos de añadir, desde el code-behind, un salto de línea de forma explícita a cada una de las líneas del script, utilizando la enumeración/miembro ControlChars.CrLf, como vemos en el siguiente fuente.

```
....
sJS = "<script language=""javascript"">" & ControlChars.CrLf
sJS &= "function CambiarMayMin()" & ControlChars.CrLf
sJS &= "{" & ControlChars.CrLf
sJS &= "var sTexto = new String();" & ControlChars.CrLf
sJS &= "sTexto = document.Form1.txtNombre.value;" & ControlChars.CrLf
sJS &= "if (sTexto.length > 0)" & ControlChars.CrLf
sJS &= "{" & ControlChars.CrLf
....
....
```

Para añadir saltos de línea también podemos utilizar la clase/propiedad Environment.NewLine, que devuelve una cadena con los caracteres de nueva línea para la plataforma de ejecución actual.

Las operaciones que acabamos de practicar simplemente crean código de script y lo depositan en el navegador web, no ejecutan por sí mismas una de estas funciones JavaScript que hemos generado en el navegador. Nos queda pendiente pues una parte muy importante del proceso: conectar una función de script con el evento del control que deberá ejecutarla.

Para lograr este objetivo debemos tomar el control web que necesitamos conectar con el código JavaScript, y añadir a su colección de atributos el nombre del evento y la función de script que actuará como manipulador de dicho evento cuando sea desencadenado. El siguiente fuente, que también escribiremos en el evento Load del formulario, ilustra el modo de llevar a cabo esta acción.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
   '....
   ' generar el código de script en el navegador web
   Me.RegisterClientScriptBlock("CodJS", sJS)
   ' conectar los eventos de los controles con las funciones de script
   Me.chkMayMin.Attributes.Add("onclick", "CambiarMayMin()")
   Me.txtNombre.Attributes.Add("onkeyup", "ContarCaracteres()")
End Sub
```

La colección Attributes, disponible para todos los controles Web, permite que al ser creado en el navegador el código html correspondiente al control, se generen también, dentro de las etiquetas del control, determinados atributos que no están disponibles a través de las propiedades de la clase del control. Como puede comprobar el lector, en este caso, empleamos la mencionada colección para producir el código html que

asigna a un evento del control una función del script. Para el CheckBox chkMayMin de nuestra página, el html resultante sería el siguiente.

```
<input id="chkMayMin" type="checkbox" name="chkMayMin"
onclick="CambiarMayMin()" />
```

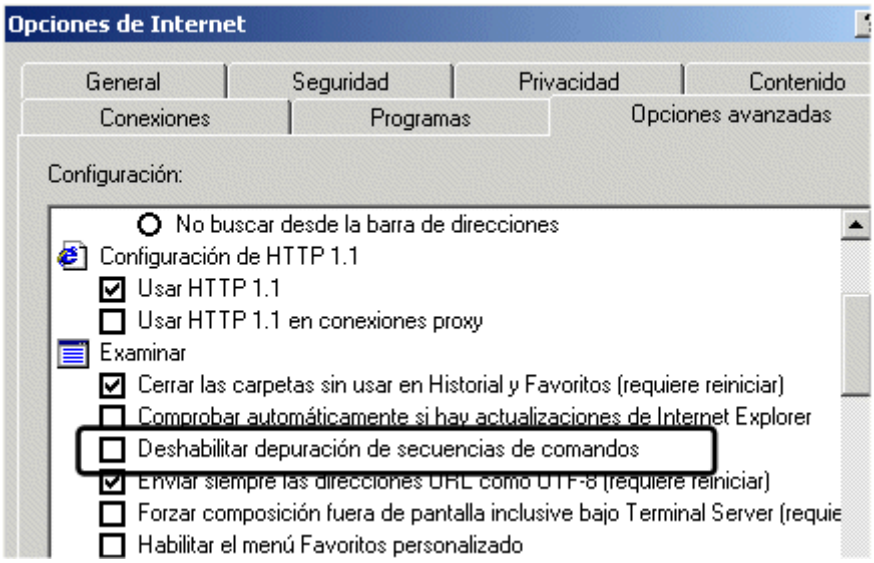
De esta forma, cuando ejecutemos la página WebForm3, cada vez que marquemos o desmarquemos el CheckBox, se ejecutará la función CambiarMayMin en respuesta al evento onclick de este control.

Si a pesar de lo anterior, seguimos prefiriendo escribir nuestro código de script dentro del html del formulario, es totalmente factible hacerlo (aunque no sea la práctica recomendada), ya que la colección Attributes de los controles Web no distingue si el código JavaScript ha sido escrito directamente en el html del webform o desde su code-behind.

Depurando nuestro código JavaScript

Si necesitamos saber con una mayor exactitud cuándo se ejecuta nuestro código de script en el navegador web, podemos hacer que el depurador de Visual Studio .NET supervise dicho código en tiempo de ejecución; para ello debemos seguir unos sencillos pasos que explicamos a continuación.

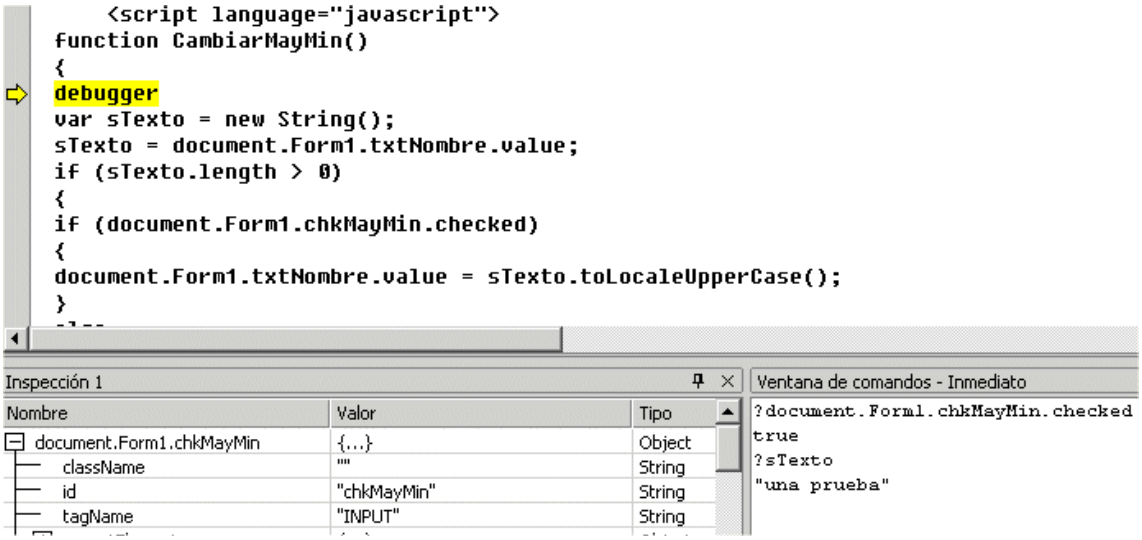
En primer lugar abriremos Internet Explorer, seleccionando su opción de menú *Herramientas + Opciones de Internet*. A continuación haremos clic en la pestaña Opciones avanzadas, y dentro de esta, desmarcaremos la casilla con el título *Deshabilitar depuración de secuencias de comandos*, como vemos en la siguiente figura.



Tras aceptar este cuadro de diálogo pasaremos a la función que necesitemos depurar de nuestro bloque de script, y dentro de su código situaremos la instrucción *debugger*, sin finalizar la línea con punto y coma, como vemos en el siguiente código fuente.

```
....
sJS = "<script language=""javascript"">" & ControlChars.CrLf
sJS &= "function CambiarMayMin()" & ControlChars.CrLf
sJS &= "{" & ControlChars.CrLf
sJS &= "debugger" & ControlChars.CrLf
sJS &= "var sTexto = new String();" & ControlChars.CrLf
....
....
```

Al cargarse ahora en el navegador la página que contiene esta función, cuando la misma sea ejecutada se producirá una parada en el flujo de ejecución, entrando en acción el depurador de igual forma que si estuviéramos depurando el code-behind de una página aspx. Dentro de este contexto podremos evaluar expresiones, visualizar el contenido de variables, propiedades de controles, etc. La siguiente figura muestra un ejemplo de esta situación.



Ejecutando un bloque de script durante la carga de la página

Durante el proceso de creación del webform podemos emitir un bloque de código de script para que sea ejecutado durante esta fase inicial de creación de la página. Todo lo que tenemos que hacer es escribir dicho bloque sin que se encuentre contenido dentro de una función JavaScript, como vemos en el siguiente fuente.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
   Dim sJS As String
   sJS = "<script language=""javascript"">" & ControlChars.CrLf
   sJS &= "alert('comenzamos a ejecutar el webform');"
   sJS &= "</script>"
   Me.RegisterClientScriptBlock("CodJS", sJS)
End Sub
```

Esta práctica sufre empero de un inconveniente. Supongamos que nos encontramos diseñando un formulario que tiene un control DropDownList con la lista de meses del año, y queremos que al comenzar la ejecución de dicho formulario se compruebe cuál es el mes correspondiente a la fecha actual, dejándolo como elemento seleccionado en la lista antes de presentar la página al usuario.

Ante estos requerimientos hemos de tener presente que el método Page.RegisterClientScriptBlock genera el JavaScript justamente después de la etiqueta html de apertura del webform: <form>, lo quiere decir que si usamos este método para incluir un bloque de código que intente manipular uno de los controles del formulario se producirá un error, porque al ser ejecutado el script todavía no se habrán creado los controles del webform.

La solución a este problema la encontramos en el método Page.RegisterStartupScript, que también emite código de script al navegador, pero situándolo en este caso justo antes de la etiqueta html que cierra el webform: </form>.

El modo de uso de este método es igual que RegisterClientScriptBlock, es decir, debemos pasar al método una cadena que será el identificador único del bloque de código, y otra cadena conteniendo el código JavaScript. Veamos este método en acción dentro del evento Load de la página WebForm4 .

```
Me.ddlMeses.Items.Add("Enero")
Me.ddlMeses.Items.Add("Febrero")
Me.ddlMeses.Items.Add("Marzo")
Me.ddlMeses.Items.Add("Abril")
Me.ddlMeses.Items.Add("Mayo")
Me.ddlMeses.Items.Add("Junio")
Me.ddlMeses.Items.Add("Julio")
Me.ddlMeses.Items.Add("Agosto")
Me.ddlMeses.Items.Add("Septiembre")
Me.ddlMeses.Items.Add("Octubre")
Me.ddlMeses.Items.Add("Noviembre")
Me.ddlMeses.Items.Add("Diciembre")

Dim sJS As String
sJS = "<script language=""javascript"">" & ControlChars.CrLf
sJS &= "var dtFecha = new Date();" & ControlChars.CrLf
sJS &= "var nMes = dtFecha.getMonth();" & ControlChars.CrLf
sJS &= "document.Form1.ddlMeses.selectedIndex=nMes;" & ControlChars.CrLf
sJS &= "</script>"

Me.RegisterStartupScript("CodJS", sJS)
```

Ahora ya no se producirán errores en este sentido, puesto que al ser ejecutado el JavaScript, los controles del formulario ya estarán creados, como vemos a continuación, en el código html de la página accesible desde el navegador.

```
<form name="Form1" method="post" action="WebForm4.aspx" id="Form1">
....
....
        <select name="ddlMeses" id="ddlMeses">
                <option value="Enero">Enero</option>
                <option value="Febrero">Febrero</option>
                <option value="Marzo">Marzo</option>
                <option value="Abril">Abril</option>
                <option value="Mayo">Mayo</option>
                <option value="Junio">Junio</option>
                <option value="Julio">Julio</option>
                <option value="Agosto">Agosto</option>
                <option value="Septiembre">Septiembre</option>
                <option value="Octubre">Octubre</option>
                <option value="Noviembre">Noviembre</option>
                <option value="Diciembre">Diciembre</option>
        </select>

        <script language="javascript">
                var dtFecha = new Date();
                var nMes = dtFecha.getMonth();
                document.Form1.ddlMeses.selectedIndex=nMes;
        </script>
</form>
```

Generación de código JavaScript desde controles Web personalizados

Cuando desarrollamos controles Web propios, en la gran mayoría de casos también necesitaremos que los mismos emitan código de script para el navegador web en el que tengan que ejecutarse. Es por ello, que las capacidades de creación de JavaScript en ASP.NET están disponibles para ser usadas tanto desde webforms como desde controles Web.

Como ejemplo para este tipo de casos usaremos la solución EjArticCtlASPNETJS, que también podemos descargar de www.dotnetmania.com. Esta solución la compone el proyecto MisControlesWeb, que consiste en una librería de clases conteniendo un par de controles Web para las diversas pruebas de generación de scripts, y un proyecto con el mismo nombre de la solución, que usaremos como banco de pruebas de los controles personalizados.

El primer control con el que vamos a tratar será CajaNúmero, una clase que hereda de WebControl, y que en tiempo de ejecución crea un TextBox que sólo permite escribir números. Como es natural, dadas las características de nuestro artículo, el elemento de este control que va a comprobar si su contenido es exclusivamente numérico será un bloque de código JavaScript.

Un control no puede, por sí mismo, emitir el código de script que necesite depositar en el navegador, es decir, carece de los métodos RegisterXXX que veíamos en ejemplos anteriores.

Sin embargo un control sí dispone de la propiedad Page, que representa al webform que lo contiene, y por mediación de esta propiedad puede acceder a los métodos RegisterXXX de su formulario para crear los bloques de código de script que necesite. Podemos realizar esta tarea reemplazando el método OnPreRender en la clase de nuestro control, como vemos en el siguiente fuente.

```
Protected Overrides Sub OnPreRender(ByVal e As System.EventArgs)
   Dim sJS As String

   MyBase.OnPreRender(e)
   Me.EnsureChildControls()

   sJS = "<script language=""javascript"">" & ControlChars.CrLf
   sJS &= "function CompruebaCampo()" & ControlChars.CrLf
   sJS &= "{" & ControlChars.CrLf
   sJS &= "if (! (event.keyCode >= 48 && event.keyCode <= 57))" & ControlChars.CrLf
   sJS &= "{" & ControlChars.CrLf
   sJS &= "event.returnValue = false;" & ControlChars.CrLf
   sJS &= "}" & ControlChars.CrLf
   sJS &= "}" & ControlChars.CrLf
   sJS &= "</script>"
   ' registrar el bloque de código script
   ' en la página que contiene al control
   Me.Page.RegisterClientScriptBlock("CodJSCajaNumero", sJS)

   ' oCaja contiene el objeto TextBox,
   ' asociar el evento del TextBox con la función JavaScript
   Me.oCaja.Attributes.Add("onkeypress", "CompruebaCampo()")
End Sub
```

El objeto oCaja representa al TextBox que mostrará nuestro control personalizado, y actúa como control constituyente del control web principal. Para instanciar y generar este TextBox, debemos reemplazar los método CreateChildControls y Render, respectivamente, añadiendo el código que se ocupe de dichas tareas, como vemos en el siguiente fuente.

```
'....
Private oCaja As WebControls.TextBox
'....
Protected Overrides Sub CreateChildControls()
   ' limpiar la colección de controles constituyentes de este control
   Me.Controls.Clear()
   ' crear el TextBox contenido en este control
```

```
   Me.oCaja = New WebControls.TextBox
End Sub

Protected Overrides Sub Render(ByVal output As System.Web.UI.HtmlTextWriter)
   ' generar el control incluyendo
   ' el TextBox constituyente
   MyBase.Render(output)
   Me.oCaja.RenderControl(output)
End Sub
'....
```

De esta forma ya podemos utilizar el control en el webform de pruebas, donde veremos que sólo nos permitirá introducir números.

Comprobar si un bloque de código script ya ha sido registrado

Como ya sabemos, el primer parámetro de los métodos Page.RegisterXXX consiste en una cadena que funciona como identificador del bloque de código que estamos generando en el navegador, lo que impide que registremos en más de una ocasión un bloque de código script con el mismo identificador. Pongamos como ejemplo el siguiente fuente.

```
Dim sJS1 As String
sJS1 = "<script language=""javascript"">"
sJS1 &= "function Primera()"
sJS1 &= "{"
sJS1 &= "alert('esta es la función Primera')"
sJS1 &= "}"
sJS1 &= "</script>"
Me.RegisterClientScriptBlock("CodJS", sJS1)

Dim sJS2 As String
sJS2 = "<script language=""javascript"">"
sJS2 &= "function Segunda()"
sJS2 &= "{"
sJS2 &= "alert('esta es la función Segunda')"
sJS2 &= "}"
sJS2 &= "</script>"
' atención, vamos a usar el mismo identificador
' de bloque de código que en la anterior ocasión
Me.RegisterClientScriptBlock("CodJS", sJS2)
```

Cuando este code-behind sea ejecutado, el código JavaScript generado en el navegador sólo corresponderá a la función Primera(), ya que al intentar registrar el siguiente bloque de script, como también estamos utilizando el identificador “CodJS”, ASP.NET detectará que ya ha sido registrado un script con el mismo identificador, ignorando este último intento de registro.

En este preciso momento se estará preguntando, estimado lector, acerca de la utilidad de lo que acabamos de explicar, y ciertamente, observada desde el contexto de un webform, esta situación no reviste mayores problemas si simplemente registramos todo nuestro código JavaScript con una única llamada a cada uno de los métodos RegisterXXX.

Donde realmente este escenario cobra importancia es cuando nos encontramos desarrollando nuestros propios controles web, sobre todo aquellos que pueden necesitar una considerable cantidad de código de script.

Debemos tener presente que si añadimos a un webform varias copias de nuestro control, cada una de estas instancias pasará por las líneas de código que generan el bloque de JavaScript, pero sólo será en la primera ocasión en que se ejecuten, cuando se emita el consabido código de script hacia el navegador, mientras que en el resto de ocasiones en que el flujo de la aplicación pase por este mismo punto, la llamada a los métodos RegisterXXX no tendrá resultado alguno, consumiendo inútilmente un precioso tiempo de proceso y recursos.

Para optimizar esta situación, la clase Page proporciona los métodos IsClientScriptBlockRegistered e IsStartupScriptRegistered, que sirven para confirmar si un bloque de código script ya ha sido registrado utilizando los métodos RegisterClientScriptBlock o RegisterStartupScript respectivamente.

Ambos métodos reciben como parámetro una cadena que representa al identificador del bloque de código a comprobar, y devuelven un valor lógico que indica si ya existe en la página un script con ese identificador.

Aplicando esta técnica al control CajaNumero, deberemos modificar ligeramente el método OnPreRender del modo que muestra el siguiente fuente.

```
Protected Overrides Sub OnPreRender(ByVal e As System.EventArgs)
   '....
   If Not (Me.Page.IsClientScriptBlockRegistered("CodJSCajaNumero")) Then
      sJS = "<script language=""javascript"">" & ControlChars.CrLf
      sJS &= "function CompruebaCampo()" & ControlChars.CrLf
      '....
      Me.Page.RegisterClientScriptBlock("CodJSCajaNumero", sJS)
   End If
   '....
   Me.oCaja.Attributes.Add("onkeypress", "CompruebaCampo()")
End Sub
```

Empleando macros para automatizar la preparación del código JavaScript

Como hemos mencionado en un apartado anterior, cuando necesitamos añadir una gran cantidad de código de script a un webform podemos escribirlo en un archivo .js

aparte, y establecer una referencia al mismo desde el html del formulario, utilizando el atributo src de la etiqueta <script>.

Esta práctica puede resultar un problema cuando el código JavaScript a generar pertenece a un control web personalizado, ya que en ese caso, además del ensamblado que contiene el control, debemos proporcionar el archivo con su código de script asociado. En resumen, se trata de un modo de trabajo poco efectivo.

Cuando diseñamos y desarrollamos un control propio debemos procurar que su funcionamiento sea lo más autónomo posible; esto quiere decir que si debe generar código de script, tiene que hacerlo el propio control desde el interior de su clase utilizando los métodos RegisterXXX, lo cual puede llegar a resultar un tanto farragoso por la conveniencia, ya explicada en anteriores ejemplos, de concatenar las líneas de código en una variable y añadir los correspondientes saltos de línea.

¿No sería estupendo poder escribir el código JavaScript de igual modo que lo hacemos con el código “normal” del control, es decir, sin tener que concatenar cada línea a una variable que lo contenga ni añadirle el salto de línea final?.

Pues bien, esto es posible si recurrimos a una potente herramienta que nos proporciona el entorno de Visual Studio .NET: las macros. Gracias a una macro, el programador puede dedicarse a escribir exclusivamente el código de script, y encomendar a la macro el trabajo pesado de concatenar todo ese código en una variable, y añadir los caracteres especiales de salto de línea. Procedamos pues.

Primeramente añadiremos al proyecto un nuevo control con el nombre CajaColor, que consistirá en una caja de texto, cuyo color de fondo y contenido cambiará en base a determinadas circunstancias.

A continuación también agregaremos al proyecto un archivo con el nombre GeneracionCodigoJS.js, en el que escribiremos las funciones JavaScript que conectaremos con los eventos del control. El siguiente fuente muestra una función encargada de cambiar el color de fondo de la caja, según la cantidad de caracteres que esta contenga.

```
function CambiarColor(obj)
{
	if (obj.value.length > 4)
	{
		obj.style.backgroundColor = "PaleGreen";
	}
	else
	{
		obj.style.backgroundColor = "White";
	}
}
```

El siguiente paso consiste en seleccionar el menú de Visual Studio .NET *Herramientas + Macros + Explorador de macros*, que abrirá la ventana Explorador de macros, dentro de la cual expandiremos sus nodos de la siguiente forma: *Macros + MyMacros + Module1*. Ver la siguiente figura.

Haciendo clic derecho sobre el nodo Module1, seleccionaremos la opción Editar, con la que entraremos en el entorno de desarrollo de macros (IDE) de VS.NET, quedando situados dentro del editor de código de macros, en el que escribiremos la macro TransfCodigoJSEnBloque, que no es otra cosa que un procedimiento Sub que se ejecuta bajo un contexto muy particular dentro del entorno de desarrollo.

El objetivo de esta macro consiste precisamente en realizar el trabajo que antes hacíamos manualmente, es decir, tomar el código de script y concatenarlo a una variable, incluyendo los caracteres especiales de salto de línea. Debido a que un tratamiento exhaustivo de las macros es algo que queda fuera del ámbito de este artículo, hemos proporcionado las oportunas explicaciones dentro del código para que el lector sepa lo que está ocurriendo en cada momento. Prometemos abordar el interesante tema de las macros en un próximo artículo, veamos ahora el contenido de la macro en el siguiente fuente.

```
Sub TransfCodigoJSEnBloque()
   Dim sLineaCodigo As String
   ' situarnos al final del documento de código
   DTE.ActiveDocument.Selection.EndOfDocument()
   ' añadir dos líneas en blanco
   DTE.ActiveDocument.Selection.NewLine(2)
   ' situarnos al comienzo de línea y añadir una
   ' cadena que usaremos como marca que indica que
   ' hemos llegado al final del código a manipular
DTE.ActiveDocument.Selection.StartOfLine(vsStartOfLineOptions.vsStartOfLineOptions
FirstColumn)
   DTE.ActiveDocument.Selection.Text = "****%%%%****"
   DTE.ActiveDocument.Selection.NewLine()
   ' situarnos al comienzo del código
   DTE.ActiveDocument.Selection.StartOfDocument()
   ' añadir una línea de código que creará un StringBuilder
   ' en el que se irán agregando las líneas de script
   DTE.ActiveDocument.Selection.Text = "Dim sb As New System.Text.StringBuilder"
   DTE.ActiveDocument.Selection.NewLine()
```

```
    ' recorrer y transformar cada línea de código script
    While True
        ' seleccionar la línea de código y asignar a una variable

DTE.ActiveDocument.Selection.StartOfLine(vsStartOfLineOptions.vsStartOfLineOptions
FirstColumn)
        DTE.ActiveDocument.Selection.EndOfLine(True)
        sLineaCodigo = DTE.ActiveDocument.Selection.Text
        ' añadir comillas dobles en donde sea necesario
        ' para evitar errores en la composición de cadenas
        sLineaCodigo = sLineaCodigo.Replace("""", """""")
        ' añadir la línea de código transformada
        DTE.ActiveDocument.Selection.Text = "sb.Append(""" & sLineaCodigo & """ &
ControlChars.CrLf)"
        ' comprobar si hemos llegado al final del código a transformar
        DTE.ActiveDocument.Selection.LineDown()

DTE.ActiveDocument.Selection.StartOfLine(vsStartOfLineOptions.vsStartOfLineOptions
FirstText)
        DTE.ActiveDocument.Selection.EndOfLine(True)
        If CType(DTE.ActiveDocument.Selection, TextSelection).Text = "****%%%%****"
Then
            ' si hemos llegado a la marca de final de código
            ' borrarla y salir del bucle
            DTE.ActiveDocument.Selection.Delete()
            Exit While
        Else

DTE.ActiveDocument.Selection.StartOfLine(vsStartOfLineOptions.vsStartOfLineOptions
FirstText)
        End If
    End While

    ' añadir la línea de código que devuelve el StringBuilder
    ' que contiene el script adecuadamente transformado
    DTE.ActiveDocument.Selection.LineDown()

DTE.ActiveDocument.Selection.StartOfLine(vsStartOfLineOptions.vsStartOfLineOptions
FirstText)
    DTE.ActiveDocument.Selection.Text = "Return sb.ToString()"
End Sub
```

Finalizada la creación de la macro volveremos al proyecto de los controles web, y nos situaremos en el código de la función JavaScript CambiarColor. Desde aquí podemos ver en la ventana Explorador de macros la macro TransfCodigoJSEnBloque recién creada; haciendo doble clic sobre la misma se ejecutará, aplicando la transformación sobre nuestro código de script que vemos en el siguiente fuente.

```
Dim sb As New System.Text.StringBuilder
sb.Append("function CambiarColor(obj)" & ControlChars.CrLf)
sb.Append("{" & ControlChars.CrLf)
sb.Append("    if (obj.value.length > 4)" & ControlChars.CrLf)
sb.Append("    {" & ControlChars.CrLf)
sb.Append("            obj.style.backgroundColor = ""PaleGreen"";          " &
ControlChars.CrLf)
sb.Append("    }" & ControlChars.CrLf)
sb.Append("    else" & ControlChars.CrLf)
sb.Append("    {" & ControlChars.CrLf)
sb.Append("            obj.style.backgroundColor = ""White"";      " &
ControlChars.CrLf)
sb.Append("    }" & ControlChars.CrLf)
sb.Append("}" & ControlChars.CrLf)
sb.Append("" & ControlChars.CrLf)
sb.Append("" & ControlChars.CrLf)

Return sb.ToString()
```

A continuación tomaremos el código resultante tras la ejecución de la macro y lo situaremos como un método del control, al que llamaremos en el momento de registrar los bloques de código de script, lo cual en nuestro ejemplo, hacemos desde el método OnPreRender, como vemos en el siguiente fuente.

```
Private Function GeneraJSCambiarColor() As String
   ' genera el código javascript que cambia el color de la caja de texto
   Dim sb As New System.Text.StringBuilder
   sb.Append("" & ControlChars.CrLf)
   sb.Append("function CambiarColor(obj)" & ControlChars.CrLf)
   sb.Append("{" & ControlChars.CrLf)
   sb.Append("        if (obj.value.length > 4)" & ControlChars.CrLf)
   sb.Append("        {" & ControlChars.CrLf)
   sb.Append("                obj.style.backgroundColor = ""PaleGreen"";          " &
ControlChars.CrLf)
   sb.Append("        }" & ControlChars.CrLf)
   sb.Append("        else" & ControlChars.CrLf)
   sb.Append("        {" & ControlChars.CrLf)
   sb.Append("                obj.style.backgroundColor = ""White"";      " &
ControlChars.CrLf)
   sb.Append("        }" & ControlChars.CrLf)
   sb.Append("}" & ControlChars.CrLf)
   sb.Append("" & ControlChars.CrLf)

   Return sb.ToString()
End Function
'-------------------------------------------------------
```

```
Protected Overrides Sub OnPreRender(ByVal e As System.EventArgs)
   Dim sJS As String
   MyBase.OnPreRender(e)

   If Not (Me.Page.IsClientScriptBlockRegistered("CodJSCajaColor")) Then
      sJS = "<script language=""javascript"">" & ControlChars.CrLf
      sJS &= Me.GeneraJSCambiarColor()
      sJS &= Me.GeneraJSManipulaContenido()
      sJS &= "</script>"
      ' registrar el bloque de código javascript que usará el control
      Me.Page.RegisterClientScriptBlock("CodJSCajaColor", sJS)
   End If
End Sub
```

Una de las ventajas en el empleo de esta técnica consiste en que mantenemos fácilmente la indentación del código de script generado, lo que facilita su lectura desde la opción de visualizar código del navegador.

Conectando eventos y código JavaScript desde el método AddAttributesToRender

Como punto final a este artículo, para conectar los eventos cliente de nuestro control CajaColor con las funciones JavaScript vamos a utilizar una nueva técnica de trabajo, que consiste en reemplazar el método AddAttributesToRender, el cual es llamado durante la creación del control.

En este método podemos centralizar todas las instrucciones relacionadas con la creación de atributos para el control, es decir, la conexión entre los diferentes eventos desencadenados en el lado cliente por el control, y las funciones de script que se ejecutarán en respuesta a dichos eventos. El siguiente fuente muestra un ejemplo de uso de este método.

```
Protected Overrides Sub AddAttributesToRender(ByVal writer As
System.Web.UI.HtmlTextWriter)
   ' conectar los eventos del control
   ' con el código/funciones de javascript correspondientes
   writer.AddAttribute(HtmlTextWriterAttribute.Onchange, "alert('Se ha cambiado el
control');")
   writer.AddAttribute("onkeyup", "CambiarColor(this)")
   writer.AddAttribute("onblur", "ManipulaContenido(this)")
End Sub
```

AddAttributesToRender recibe un parámetro de tipo HtmlTextWriter, que usaremos para asociar los eventos y el código de script llamando a su método AddAttribute.

AddAttribute, por otra parte, recibe dos parámetros: el primero lo utilizamos para especificar el nombre del evento a tratar, bien como una cadena de caracteres o mediante uno de los miembros de la enumeración HtmlTextWriterAttribute; el

segundo parámetro es una cadena con el nombre de la función de script que actuará como manipulador del evento.

<script> Finalizamos </script>

Nuestro periplo termina aquí, no sin antes agradecer al lector la atención que nos haya dedicado, confiando en que todas las experiencias aquí vertidas puedan serle útiles en su quehacer diario como noble artesano del código.