Accesibilidad Web con .NET

Internet llega cada vez a más usuarios, por lo que crear sitios Web accesibles se está convirtiendo en algo imprescindible. Para los programadores de sitios Web, los controles de ASP.NET facilitan mucho la programación, pero éstos no son accesibles al renderizarse en la página. En este artículo veremos por qué no lo son y cómo corregir esta situación.

David Provencio

Especialista en accesibilidad a proyectos de MOSS

Cuando se está desarrollando un sitio Web, se debe poner énfasis en hacerlo accesible para el mayor rango de usuarios posible, incluyendo en muchos casos a personas discapacitadas o con problemas de percepción o comprensión.

Para evitar que estos usuarios tengan dificultades a la hora de navegar por páginas Web existen numerosas ayudas técnicas, como son: lectores de pantalla, zooms de pantalla o de letra, añadidos para la personalización de fuentes y colores de la página, etc. Todas estas ayudas técnicas disponibles pierden considerablemente su efectividad si la página no está bien construida en lo que a accesibilidad se refiere.

La accesibilidad Web se puede definir de la siguiente forma: *consiste en ofrecer la información de un sitio Web sin excluir a personas con deficiencias visuales, auditivas, motrices y/o cognitivas o que utilicen dispositivos con limitaciones de visualización o de velocidad.*

Para promover la accesibilidad Web, y que tanto desarrolladores como editores de contenidos sepan aplicarla, el **W3C** (*World Wide Web Consortium*) ha creado un grupo de trabajo denominado **WAI** (*Web Accessibility Initiative*), el cual ha definido unas **pautas** a seguir para que los desarrolladores puedan mejorar fácilmente la accesibilidad de sus sitios. Cada pauta contiene varios **puntos de verificación** que se deben comprobar para saber cómo de accesible es un sitio Web.

Muchos de estos aspectos se pueden verificar mediante validadores automáticos de accesibilidad Web, mientras que otros, como son el caso de los contrastes de color, el uso incorrecto del espacio de la página, el tamaño de los textos, las imágenes o textos parpadeantes o en movimiento, etc. se deben verificar de manera manual.

Además, es necesario revisar la **gramática** de nuestras páginas para que cumpla con el esquema **XHTML Strict 1.0**, que es el recomendado por el W3C, y que no permite etiquetas ni atributos referentes al estilo de la página, por lo que el estilo se debe expresar única y exclusivamente mediante hojas de estilo en cascada (CSS). De esta manera se asegura que la página esté bien formada y que el usuario pueda personalizar el estilo (colores, fuentes, etc.) si lo necesita.

> Un elemento indispensable para que una página Web sea accesible es que esté bien construida. Para comprobarlo, se puede utilizar el validador de gramática del W3C: http://validator.w3.org "(figura 1)"

Es importante resaltar que una Web accesible beneficia a todos los usuarios, ya que si nuestras páginas están bien construidas es más probable que se visualicen correctamente en todo tipo de navegadores y dispositivos, como por ejemplo telé-

fonos móviles o PDA, aumentando así el rango de posibles usuarios.

Accesibilidad en páginas ASP.NET

A continuación, vamos a describir los principales problemas que se presentan en relación con la accesibilidad al crear un sitio Web con ASP.NET.

En primer lugar, al crear un proyecto Web con Visual Studio, éste genera la página por defecto según el esquema XHTML Transitional 1.0, por lo que, siguiendo la recomendación del W3C, se debe modificar el **DocType** de la página para que sea XHTML Strict 1.0, tal y como se muestra en el listado 1.

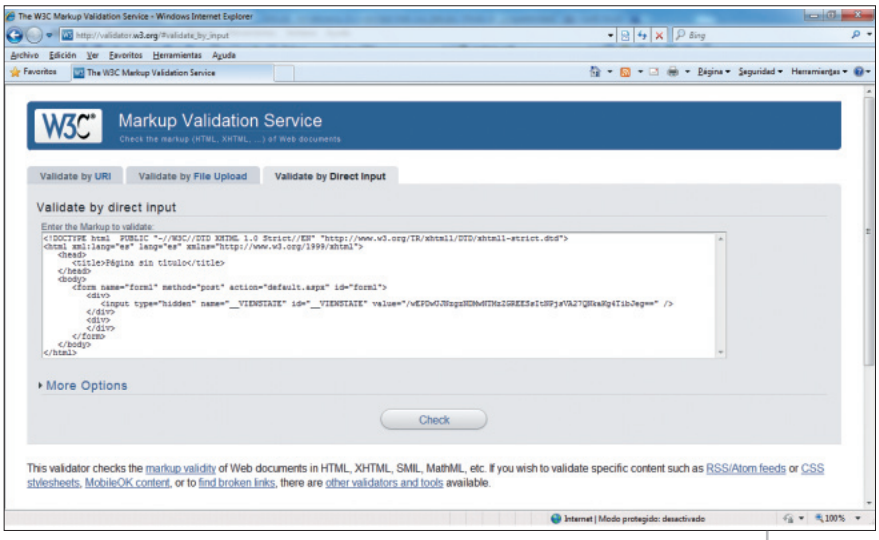

Figura 1. Validador de gramática del W3C

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

Listado 1. DocType para XHTML Strict 1.0.

Con XHTML Strict 1.0 se consigue un marcado totalmente limpio ya que, como se ha comentado antes, XHTML Strict 1.0 no admite atributos referidos a la presentación, como son **width**, **border** o **style**, por nombrar algunos. El hecho de que el estilo se exprese únicamente mediante CSS permitirá a los usuarios, por ejemplo, establecer su propia hoja de estilos CSS para aumentar el tamaño de letra o modificar los contrastes de colores de la página.

Para validar si nuestra página ASP.NET cumple con la **gramática** que hemos indicado en el **DocType**, basta con ejecutar la aplicación, copiar el código fuente de la página que se renderiza en el navegador, y luego entrar en el validador de gramática del W3C (**http://validator.w3.org/#validate_by_input**, figura 1), pegar el código fuente en el cuadro de texto habilitado al efecto y pulsar el botón "Check".

Al realizar la comprobación, obtendremos el siguiente error: "*Attribute 'name' exists, but can not be used for this element*", debido a que el elemento **form** no admite el atributo **name** en XHMTL Strict 1.0.

Revisando el código en Visual Studio, se ve que el elemento **form** no contiene el atributo **name** (**<form id="form1" runat="server">**), debido a que éste se añadirá en tiempo de ejecución, cuando el servidor vaya a generar la página que enviará al navegador. Para solucionar esto, se debe añadir al archivo **Web.config** el elemento **xhtmlConformance** asociándole el valor **Strict**, de tal forma que las páginas de ASP.NET se generen cumpliendo automáticamente con la gramática XHTML Strict 1.0 (listado 2). Al volver a pasar la validación, veremos que ahora sí obtenemos código válido XHTML Strict 1.0.

```
<configuration>
   <system.web>
      <xhtmlConformance mode="Strict" />
   </system.web>
</configuration>
```

Listado 2. Preparación del fichero Web.config para que genere código XHTML Strict 1.0.

Uso de adaptadores de controles

En el apartado anterior, hemos obtenido una página ASP.NET en blanco que cumple las normas de accesibilidad. A continuación, introduciremos en dicha página controles de ASP.NET y veremos cómo hacerlos accesibles.

Los controles de ASP.NET facilitan mucho las cosas a la hora de programar en comparación con los controles HTML estándar pero, como se verá a continuación, el renderizado que tienen en la página en muchos casos no es accesible. Por ejemplo, vamos a añadir a nuestra página **.aspx** un control ASP.NET de tipo **Image**, como se ve en el listado 3. El código en el navegador, una vez procesada la página por .NET Framework, queda como se muestra en el listado 4.

```
<asp:Image ID="Image1" runat="server"
   ImageUrl="~/wai.bmp"
   AlternateText="Logo de la WAI" />
```

Listado 3. Código ASP.NET de un control de tipo Image

```
<img id="Image1" src="wai.bmp"
   alt="Logo de la WAI"
   style="border-width:0px;" />
```

Listado 4. Código HTML resultante de renderizar un control de tipo Image

Al pasar el validador de gramática del W3C, vemos que el código renderizado no es válido, ya que el elemento

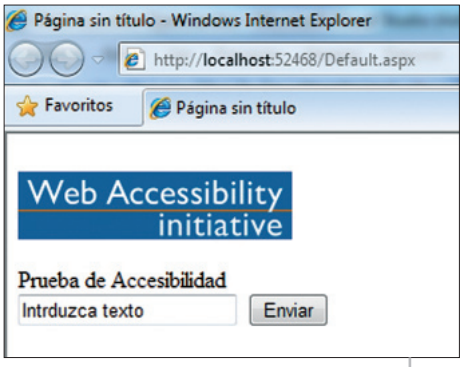

Figura 2. Página .aspx con un control de tipo Imagen

img incluye el atributo **style**. Este atributo no es admitido en XHTML Strict 1.0, ya que define estilo directamente en la página, lo cual no permite a las ayudas técnicas establecer su propio formato y hacer la página legible y accesible para los usuarios que dependen de estas ayudas. Si se quiere definir estilo en el elemento **img**, se debe utilizar una hoja de estilos, aplicando el atributo **class** al elemento.

Para eliminar el atributo **style** del control, utilizaremos los adaptadores de controles (*Control Adapters*) de .NET. La clase **ControlAdapter** está incluida desde la versión 2.0 de .NET Framework, y permite sobrescribir el renderizado de cualquier control justo antes de que se éste se envíe al navegador.

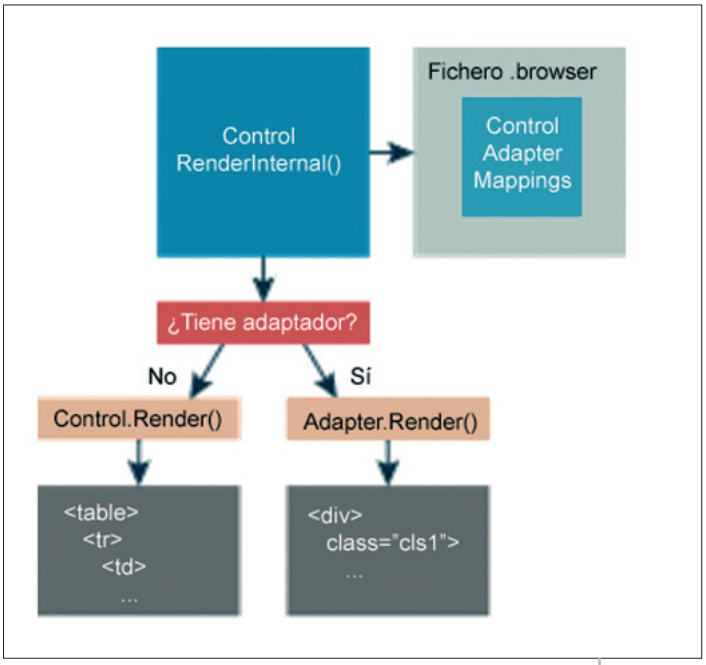

Figura 3. Esquema de funcionamiento del control adapters

El funcionamiento de los *control adapters* es el siguiente: cuando el navegador solicita una página **.aspx**, .NET comprueba si existe algún adaptador para las clases de los controles que se están utilizando en la página, y en caso de que existan, los aplica inmediatamente antes de generar el HTML correspondiente.

Para que .NET Framework conozca si existe un *control adapter* para un tipo de control concreto, se hace un mapeo que se configura mediante ficheros **.browser**. En el caso del ejemplo anterior, para resolver el problema del atributo **style** en el control de tipo imagen, hemos creado el archivo **accesible.browser**, que se muestra en el listado 5.

```
<browsers>
  <browser refID="Default">
    <controlAdapters>
      <adapter controlType="System.Web.UI.WebControls.Image"
          adapterType="MisAdaptadores.ImagenAccessible" />
    </controlAdapters>
  </browser>
</browsers>
```

Listado 5. Archivo .browser con el mapeo del adaptador ImagenAccesible para un control de tipo Imagen

Asimismo, hemos creado una clase denominada **ImagenAccesible** (listado 6), que hereda de **System.Web.UI.Adapters.ControlAdapter**. Esta clase es el adaptador en sí, y a través de ella se va a obtener el HTML del control y combinarlo con el del resto de la página para su envío al navegador. Como se ve en el listado 5 (elemento **<adapter>**), la clase **System.Web.UI.WebControls.Image** se mapea a la clase **ImagenAccessible**; de manera que en el atributo **controlType** se indica la clase a adaptar y en **adapterType** se establece el adaptador correspondiente a ese tipo de control. En nuestro ejemplo, el adaptador lo que hace es sustituir el atributo **style** y su valor por una cadena vacía. De esta manera, el atributo **style** no aparecerá en el HTML resultante.

Se pueden crear uno o varios ficheros para indicar los mapeos, y los ficheros **.browser** deberán estar en la carpeta especial **App_Browsers**. Por su parte, las clases herederas de **ControlAdapter** deberán estar en la carpeta especial **App_Code** (figura 4).

CSS-Friendly Control Adapters

La manera de resolver problemas de accesibilidad que hemos presentado en el ejemplo anterior para un control ASP.NET sencillo, como es **Image**, es apli-

```
namespace MisAdaptadores
{
   public class ImagenAccessible :
      System.Web.UI.Adapters.ControlAdapter
   {
      protected override void Render(HtmlTextWriter output)
      {
         System.IO.StringWriter sw =
            new System.IO.StringWriter();
         HtmlTextWriter hw = new HtmlTextWriter(sw);

         base.Render(hw);
         string html = sw.ToString();

         output.Write(html.Replace(
            "style=\"border-width:0px;\"", string.Empty));
      }
   }
}
```

Listado 6. Código del adaptador (ImagenAccesible.cs)

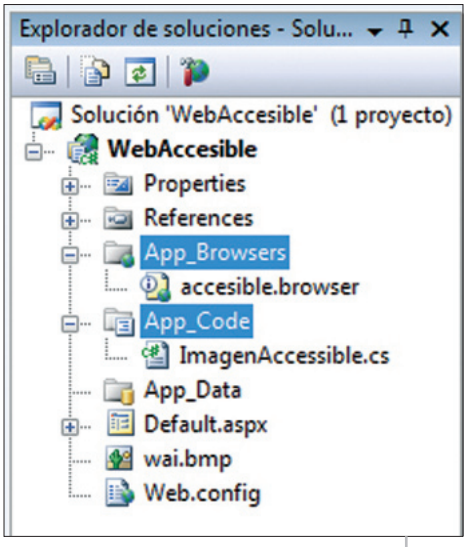

Figura 4. Carpetas especiales App_Browsers y App_Code

cable a todos los controles de .NET. No obstante, para facilitar esta tarea en controles más complejos, existe toda una serie de *control adapters* ya creados, denominados *CSS-Friendly Control Adapters*, que permiten que controles como, por ejemplo, el **GridView** o el **TreeView** sean accesibles.

> Los CSS-Friendly Control Adapters son adaptadores de controles ya creados, que permiten hacer accesibles muchos controles de ASP.NET. Se pueden descargar de http://www.ASP.NET/CssAdapters

```
<browser refID="IE">
   <controlAdapters>
      <adapter controlType="System.Web.UI.WebControls.Image"
               adapterType=" MisAdaptadores.ImagenAccessibleIE" />
   </controlAdapters>
</browser>

<browser refID="MozillaFirefox">
   <controlAdapters>
      <adapter controlType="System.Web.UI.WebControls.Image"
               adapterType=" MisAdaptadores.ImagenAccessibleFireFox" />
   </controlAdapters>
</browser>
```

Listado 7. Archivo .browser con el mapeo de dos adaptadores para distintos navegadores.

Por ejemplo, para el caso de un **GridView**, por defecto al renderizarse un control de este tipo en una página se crean varias tablas anidadas. Por accesibilidad, es recomendable utilizar tablas solo para presentar datos y no para maquetar la página, por lo que se deben eliminar las tablas anidadas y dejar exclusivamente la que contiene los datos. Aplicando el *control adapter* para el **GridView** que forma parte de los *CSS-Friendly Control Adapters*, se renderizará únicamente una tabla, la que contiene los datos del **GridView**. En la figura 5 se puede comparar los resultados de renderizar un **GridView** con y sin la aplicación del control.

La tabla 1 muestra un listado de los *CSS-Friendly Control Adapters* actualmente disponibles. Estos adaptadores se ofrecen con código fuente, que se puede modificar para adecuarlos a las necesidades concretas del proyecto Web en el que estemos trabajando.

Beneficios de los adaptadores de controles

Los *control adapters* son una buena solución para salvar problemas de accesibilidad, consiguiendo que los controles se rendericen de manera accesible. De igual

```
<browser refID="Default">
  <controlAdapters>
     <adapter controlType=
      "Microsoft.SharePoint.Publishing.WebControls.SummaryLinkFieldControl"
        adapterType=" MisAdaptadores.Mi_SummaryLinkFieldControl_Adapter" />
  </controlAdapters>
</browser>

<browser refID="Default">
  <controlAdapters>
     <adapter controlType="System.Web.UI.WebControls.WebParts.WebPartZone"
              adapterType=" MisAdaptadores.Mi_WebPartZone_Adapter" />
  </controlAdapters>
</browser>
```

Listado 8. Archivo .browser con el mapeo de un adaptador para un WebControl de MOSS 2007 y otro para WebPartZone.

Figura 5. HTML de un GridView, aplicando el adaptador y sin aplicarlo.

forma, es posible crear una clase que herede de **`System.Web.UI.Adapters.PageAdapter`** para crear un adaptador para una página **`.aspx`** completa, permitiendo de esta manera incluir la lógica que queramos para el renderizado de la página.

Una vez resueltos los problemas de gramática, se realizará una revisión de la accesibilidad, verificando las pautas de accesibilidad de la WAI. En la página de KAW (Kit de Accesibilidad Web), **`http://www.e-kaw.org`**, se ofrecen diversas herramientas, tanto para los puntos de verificación automáticos como para los manuales.

Los *control adapters* posibilitan además la personalización del renderizado en función del navegador, permitiendo mapear distintos adaptadores para un mismo control dependiendo del navegador en el que se vaya a mostrar la página. Esto se establece mediante el atributo **`refID`** (listado 7).

Por último, los *control adapters* también son muy útiles para conseguir adaptar el renderizado de controles de los que no se dispone de código fuente. Por ejemplo, se pueden utilizar para adaptar controles Web de MOSS 2007, o para adaptar *WebParts* de MOSS 2007 de forma que sean accesibles (listado 8).

Conclusión

En este artículo hemos presentado diversas técnicas destinadas a resolver los problemas de accesibilidad que se presentan al desarrollar con ASP.NET, y en particular cómo utilizar los adaptadores de controles con este objetivo. Esperamos que este artículo le resulte útil para su trabajo.

Web Control	CSS Friendly Control Adapter
System.Web.UI.WebControls.Menu	CSSFriendly.MenuAdapter
System.Web.UI.WebControls.TreeView	CSSFriendly.TreeViewAdapter
System.Web.UI.WebControls.DetailsView	CSSFriendly.DetailsViewAdapter
System.Web.UI.WebControls.FormView	CSSFriendly.FormViewAdapter
System.Web.UI.WebControls.DataList	CSSFriendly.DataListAdapter
System.Web.UI.WebControls.GridView	CSSFriendly.GridViewAdapter
System.Web.UI.WebControls.ChangePassword	CSSFriendly.ChangePasswordAdapter
System.Web.UI.WebControls.Login	CSSFriendly.LoginAdapter
System.Web.UI.WebControls.LoginStatus	CSSFriendly.LoginStatusAdapter
System.Web.UI.WebControls.CreateUserWizard	CSSFriendly.CreateUserWizardAdapter
System.Web.UI.WebControls.PasswordRecovery	CSSFriendly.PasswordRecoveryAdapter

Tabla 1. Lista de CSS-Friendly Control Adapters disponibles