Desarrollo de aplicaciones para sistemas móviles con tecnología.NET

Manuel-Jesús Prieto Martín [1]

[1] Telefónica Investigación y Desarrollo, Spain
mjprieto@telefonica.es

Resumen: En este capítulo se presente una introducción de la tecnología .NET para el desarrollo de aplicaciones móviles. Las capacidades de los terminales de acceso a Internet que tenemos hoy en día, van desde las pantallas sencillas hasta las pantallas de alta calidad de ordenadores de sobremesa, pasando incluso por las televisiones. Para diseñar una web que sea capaz de servir información e interactuar con todos estos dispositivos de forma adecuada y satisfactoria, tendremos que utilizar herramientas adecuadas o de lo contrario el esfuerzo de desarrollo será enorme. Microsoft ha creado el Mobile Internet Toolkit para el desarrollo de sistemas móviles. Este marco de trabajo se engloba dentro de la tecnología ASP.NET, y está orientado al desarrollo de entornos de información para Internet con capacidades de acceso para dispositivos móviles. En este capítulo describimos la tecnología y se muestra como aplicarla.

Palabras clave: .Net; ASP; programación móvil

Abstract. This chapter provides an introduction to .NET technology for mobile application development. The capabilities of today's Internet access terminals range from simple screens to high quality desktop screens and even televisions. To design a website that is capable of serving information and interacting with all these devices in an adequate and satisfactory way, we will have to use appropriate tools or else the development effort will be enormous. Microsoft has created the Mobile Internet Toolkit for the development of mobile systems. This framework is part of ASP.NET technology and is aimed at developing information environments for the Internet with access capabilities for mobile devices. In this chapter we describe the technology and show how to apply it.

Keywords: .Net; ASP; mobile programming

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

1 INTRODUCCIÓN

Hoy en día nos encontramos con una cantidad de dispositivos diferentes con capacidades muy diversas, y todos ellos con acceso a Internet. Esto nos obliga a tener en cuenta todos estos dispositivos a la hora de crear nuestras aplicaciones o nuestros entornos web. Cuando estos entornos están diseñados como sistemas de provisión de información genéricos (portales de noticias o páginas corporativas) deberemos contemplar todos estos dispositivos para no perder usuarios potenciales o hacer nuestra web más completa. Cuando afrontamos el diseño de un sistema destinado a ser un servicio corporativo, será también útil contemplar todos estos dispositivos, para poder así potenciar nuestros sistemas y crear un entorno más potente, profesional y con mayores capacidades.

Las capacidades de los terminales de acceso a Internet que tenemos hoy en día van desde las mínimas pantallas en blanco y negro de algunos terminales móviles hasta las pantallas de alta calidad de ordenadores de sobremesa, pasando incluso por las televisiones. Para diseñar un web que sea capaz de servir información e interactuar con todos estos dispositivos de forma adecuada y satisfactoria, tendremos que utilizar herramientas adecuadas o de lo contrario el esfuerzo de desarrollo será enorme. Cuando hace unos años se comenzó a extender la tecnología WAP, las empresas que deseaban ofrecer información bajo esta tecnología tendían a tener una duplicidad de información y sistemas, una orientada al mundo web y otra orientada al mundo WAP. Este hecho es por si bastante preocupante, ya que la duplicidad de sistemas e información provoca que el esfuerzo de mantenimiento del sistema sea mayor. Cuando a parte de este hecho, queremos ofrecer un buen servicio a todos los terminales, tendremos que adaptar nuestra información a cada uno de ellos, lo que hace crecer definitivamente de forma exponencial el esfuerzo. Algunas familias de terminales, especialmente los terminales móviles, presentan un rango tan amplio de capacidades que la presentación de información de manera adecuada en todos ellos obliga a tomar en consideración el uso de un entorno de trabajo a que nos ayude a afrontar este reto, o de lo contrario será un trabajo ímprobo la consecución de este objetivo. El problema no radica sólo en que los terminales tengan un tamaño de pantalla muy diferente o capacidades cromáticas diversas, lo que implicaría una adaptación de los contenidos, sino que además presentan unos formatos de presentación de información muy diferentes (WML, cHTML o HTML) lo que implica un proceso de adaptación mucho más profundo.

Como medida de apoyo para el desarrollo de sistemas con las características antes descritas y basados en tecnología ASP, Microsoft ofrece lo que denominó *Mobile Internet Toolkit*. Este marco de trabajo se engloba dentro de la tecnología ASP.NET, y está orientado al desarrollo de entornos de información para Internet con capacidades de acceso para dispositivos móviles [1-3].

1.1 LA TECNOLOGÍA ASP.NET

ASP.NET es la evolución de la tecnología ASP (*Active Server Pages*). Se podría definir como un marco de trabajo para la creación de aplicaciones web. En la primera época de Internet, el contenido de la misma estaba alojado en páginas HTML (*Hypertext Markup Language*) estáticas, es decir, páginas cuyo contenido estaba determinado desde el momento de la creación de la misma y siempre se mostraba la misma información con la misma estructura. A partir de 1993 el contenido de ciertas páginas comenzó a ser dinámico, es decir, se generaba en el momento en el que se solicitaba la información. Dentro de este modelo, se desarrollaron gran número de tecnologías y soluciones, entre ellas, el marco de trabajo ASP, creado por la empresa Microsoft.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

ASP se basa en un lenguaje de marcado similar a HTML o WML, pero que contiene partes de código, delimitadas por <%....%>. Los ficheros ASP son salvados e instalados dentro de una aplicación adecuada (servidor WEB con capacidad ASP), de tal manera que cuando se solicita la página en cuestión, las partes de código especial serán interpretadas y la versión definitiva de la página será creada y provista [4,5].
La tecnología ASP.NET es una evolución de ASP, con el mismo objetivo final, pero con una mayor funcionalidad y extensibilidad que su antecesor. Como parte del marco de trabajo más amplio y potente, conocido como .NET, ASP.NET tiene, entre otras, la capacidad de crear aplicaciones WEB utilizando cualquiera de los lenguajes de programación que integran .NET.
A grandes rasgos, el funcionamiento de ASP.NET es el siguiente. El cliente solicitará al servidor una página con extensión *aspx*. El servidor localizará esta página dentro de su sistema y la compilará, generando una versión compilada de la página. Esta página, una vez compilada, será ejecutada dentro del marco de trabajo del servidor y el resultado generado por la misma será remitido por el servidor al cliente. Las siguientes veces que se solicite esta página al servidor, este ejecutará directamente la versión compilada de la misma.
El código fuente de la página puede ser escrito en cualquier de los lenguajes soportados por ASP.NET, es decir, puede ser Visual Basic, C# o C++. Cuando la página es compilada, se genera un código intermedio, denominado código manejado (managed code) que estará en un lenguaje común. Este código será compilado a su vez por el JIT (Just In Time Compiler) de tal forma que finalmente tenemos código nativo.

1.2 WEB FORMS o FORMULARIOS WEB

Dentro de la tecnología ASP.NET, la base para la construcción de las páginas son los denominados controles para formularios web o *web forms*. Estos controles son abstracciones o representaciones de los diferentes elementos visuales que nos podemos encontrar dentro de las páginas WEB. El concepto es simple, pero es funcionalmente muy potente ya que podemos programar dichos controles dentro del entorno .NET, de tal forma que los eventos de las aplicaciones WEB pueden ser capturados y tratados.
Los controles que componen los formularios WEB de ASP.NET(concepto diferente al de los formularios WEB clásicos) se configuran componiendo páginas que serán mostradas al usuario cuando este lo solicite. Esta filosofía nos permite trabajar separando perfectamente la lógica de negocio de nuestra aplicación de lo que sería la interfaz de usuario de la misma. Los controles encapsulan totalmente el código y por lo tanto, de forma diferente a lo que ocurre con la tecnología ASP antigua, el código de presentación no se ve invadido por código de negocio. Esto propicia una mejora en el proceso de desarrollo de las aplicaciones, a parte de una mayor reutilización de los componentes [6-10].
El siguiente código fuente muestra un ejemplo de cuál es el formato básico de una página ASP.NET. El código fuente completo de la página será el siguiente, en el que se muestra tanto el código HTML como el código correspondiente propiamente a ASP.NET.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

<%@ Page Language="C#" %>

```
<script runat="server">
   private void Page_Load(object sender, System.EventArgs e){
      Título.Text = "Master en Comercio Electrónico";
      correo.Text = "Manuel J. Prieto";
   }

</script>
<html>
<head>
</head>
<body>
   <form runat="server">
      <div align="center"><asp:Label id="Título" runat="server"
        forecolor="Blue" font-size="Larger" font-names="Arial Black"
        font-bold="True"></asp:Label>
      </div>
      <div align="center">
         <asp:Image id="Image1" runat="server"
           ImageUrl="esccolmt.gif ">
         </asp:Image>
      </div>
      <div align="center">
         <asp:HyperLink id="correo" runat="server"
           NavigateUrl="mailto:mjpm@tid.es">
         </asp:HyperLink>
      </div>
   </form>
</body>
</html>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Como se puede ver en el código anterior, dentro de una página ASP.Net, tendremos elementos diferenciados mediante etiquetas XML y que indica la existencia de los controles. También se puede comprobar cómo se incluye dentro de una página código ejecutable similar a la forma de programar existente en el ASP clásico.
El resultado de este código una vez ejecutado y servido al usuario que lo ha demandado será el siguiente:

```
<html>
<head>
</head>
<body>
   <form name="_ctl0" method="post" action="NewFile.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwxMzY1NTgxMDMyO3Q8O2w8aT-
wxPjs+O2w8dDw7bDxpPDE+O2k8NT47PjtsPHQ8cDxwPGw8VGV4dDs+O2w8TWFzdGVyIGVuIENvb
WVyY2lvIEVsZWN0csOzbmljbzs+Pjs+Ozs+O3Q8cDxwPGw8VGV4dDs+O2w8TWFudWVsIEouIFByaW
V0bzs+Pjs+Ozs+Oz4+Oz4+Oz5FIGHXkhhcruJ24mPX+bfLT7SsYQ==" />

      <div align="center"><span id="TÃtulo"><b><font face="Arial
        Black" color="Blue">Master en Comercio
        ElectrÃ³nico</font></b></span>
      </div>
      <div align="center">
         <img id="Image1" src="/esccolmt.gif " border="0" />
      </div>
      <div align="center">
      <a id="correo" href="mailto:mjpm@tid.es">Manuel J. Prieto</a>
      </div>
   </form>
</body>
</html>
```

Los controles, que en último término se corresponden con un elemento visual dentro de la página, se pueden programar de tal forma que controlaremos exactamente qué información se va a mostrar en dicho control y qué repercusiones tiene cada una de las acciones del usuario sobre nuestro sistema. En el ejemplo anterior, se muestra cómo se establece el valor de unas variables de forma programática.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Una gran ventaja de los controles, es que al fin y al cabo son clases del sistema, de tal forma que podemos crear nuestras propias versiones de estas clases extendiendo a las mismas y así tendremos un comportamiento especial de nuestra WEB, gracias nuestras propias versiones de los controles. Esto sería ir un paso más allá de la *simple* programación de los eventos de los controles estándar.
Los controles que provee ASP.NET son muy variados, incluyendo:
Controles HTML. Estos controles se corresponden con los elementos disponibles dentro del lenguaje HTML.
Controles de ASP.NET. Estos controles permiten la introducción de datos de forma genérica. Un ejemplo de este tipo de controles sería un *check box*. Estos controles están pensados para ser usados de forma genérica. Dentro de este tipo de controles se englobarán lo controles que componen el denominado *Mobile Internet Toolkit*.
Controles de validación. Estos son controles destinados a comprobar los datos introducidos por el usuario en el resto de controles. Estos controles simplificarán nuestro código y evitarán ciertos errores. Por ejemplo, con este tipo de controles se podrá validar la información introducida por el usuario en un campo, de tal forma que esta se corresponda con un determinado formato (como una fecha), tenga un determinado tipo de datos (como una cadena), que no esté en blanco la información de dicho campo o que se corresponda con una determinada expresión regular [11-15].

1.3 MOBILE WEB FORMS

Dentro del modelo visto anteriormente de los *Web Forms*, tenemos una extensión del mismo que nos permite desarrollar páginas orientadas a los clientes móviles, es decir, a los terminales móviles como un teléfono o un PDA. Estos terminales, como se vio en la introducción, presentan unas características dispares y soportan una serie de formatos de página diversos.
El modelo propuesto dentro del marco de trabajo ASP.NET para trabajar con terminales móviles, consiste en la abstracción de la interfaz de usuario mediante el uso de componentes que modelan los elementos visuales básicos. En entorno de ejecución nos provee de forma automática la transformación de este componente abstracto en un formato adecuado y entendible para el terminal concreto que realiza la petición.
Dentro de un mismo lenguaje de marcado, diferentes terminales contienen diferentes características dentro de la interfaz de usuario, como puede ser el número de botones disponibles o las teclas especiales del terminal. Estos detalles también los tiene en cuenta el motor de ASP.NET, de tal forma que se adapta lo mejor posible a cada terminal concreto, con el objetivo final de proveer una experiencia del usuario satisfactoria.
En el código siguiente, se muestra un ejemplo muy sencillo de lo que sería un *Mobile Web Form*.

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage" Language="C#" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile" %>
<mobile:Form runat="server">
   Hola, mundo móvil!<br />
   <mobile:Label runat="server" Text="Etiqueta" />
</mobile:Form>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Como vemos, en la página configurada en el ejemplo anterior se mostrará un texto literal y un control correspondiente al MIT. Las primeras líneas corresponden a las etiquetas obligatorias para la indicar que se trata de página orientada al mundo móvil. También cabe destacar dentro de la cabecera del formulario, la etiqueta *Language*, que nos permitirá especificar el lenguaje de programación usado para escribir el código correspondiente a la página. El resto del código de la página es muy sencillo. Como se puede comprobar, se define un formulario a través de la etiqueta *mobile:Form* y se introduce la información dentro del mismo. Cualquier página que trabaje con controles del MIT debe tener al menos un formulario. Por último, vemos que dentro del propio formulario tenemos un texto literal estático, alguna etiqueta HTML y un control que nos permite especificar un texto y que no es más que un control que simula una etiqueta de texto. La siguiente imagen muestra el resultado del formulario anterior una vez servida la página al dispositivo.

En el párrafo anterior hemos indicado que las páginas móviles basadas en MIT deben tener al menos un formulario. Dentro de los lenguajes de marcado orientados a móviles, es común tener lo que podríamos definir como varias páginas lógicas dentro de una única página física (modelo del *mazo de cartas* de WML). Por esto, a diferencia de las páginas ASP.NET estándar, en las que cada página tiene un único formulario, en el caso de los *Mobile Forms*, es común tener varios formularios en una única página.

Dentro de una página con varios formularios móviles, disponemos de la propiedad *ActiveForm* para indicar cuál de los formularios incluidos en la página debe ser el formulario activo en cada momento. Por supuesto, también se pueden navegar entre los formularios a través de enlaces. El siguiente ejemplo nos muestra una página con varios formularios y nos presenta un sistema de navegación entre dichos formularios basado en enlaces [16-18].

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<mobile:Form id="FirstForm" runat="server">
   <mobile:Label id="Label1" runat="server" StyleReference="title">
      Form #1 de la página
   </mobile:Label>
   <mobile:Link id="Link1" runat="server" NavigateURL="#SecondForm">
     Siguiente Form
   </mobile:Link>
</mobile:Form>
<mobile:Form id="SecondForm" runat="server">
   <mobile:Label id="Label2" runat="server" StyleReference="title">
     Form #2 de la página
   </mobile:Label>
   <mobile:Link id="Link2" runat="server" NavigateURL="#FirstForm">
     Primer Form
   </mobile:Link>
</mobile:Form>
```

El resultado del ejemplo anterior es el siguiente.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

1.4 PROGRAMACIÓN DENTRO DE ASP.NET

Como se ha comentado anteriormente, la programación utilizada dentro de los proyectos desarrollados con ASP.NET puede realizar con cualquier de los lenguajes soportados dentro de .NET. Dentro de este documento se utilizará C# como lenguaje de programación.

Una vez que tenemos seleccionado el lenguaje de programación, deberemos decidir la técnica de programación a utilizar. Tendremos dos posibles métodos sobre los que elegir: código incrustado (*inline code*) o tener ficheros separados con el código. En este documento se utilizarán las dos técnicas indistintamente.

Dentro del modelo de programación de ASP.NET, podemos atrapar los diferentes eventos que se producen. Supongamos que dentro de una página ASP.NET tenemos un control correspondiente a un botón. Podemos indicar el código que se ejecutará al seleccionar dicho botón. Dicho código se ejecutará por supuesto en el servidor. El siguiente código muestra un ejemplo del método a escribir para capturar el evento de pulsación de un botón [19-24].

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Boton.aspx.cs" Inherits="MasterSalamanca.Boton" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="num" runat="server">

                  No se ha pulsado el bot&#243;n

                </mobile:Label>

                <mobile:Command id="comando" runat="server"

                     StyleReference="subcommand">

                  Ok

                </mobile:Command>

        </mobile:Form>

</body>
```

El código que se ejecuta en el servidor al pulsar un botón, es el que se presenta a continuación.

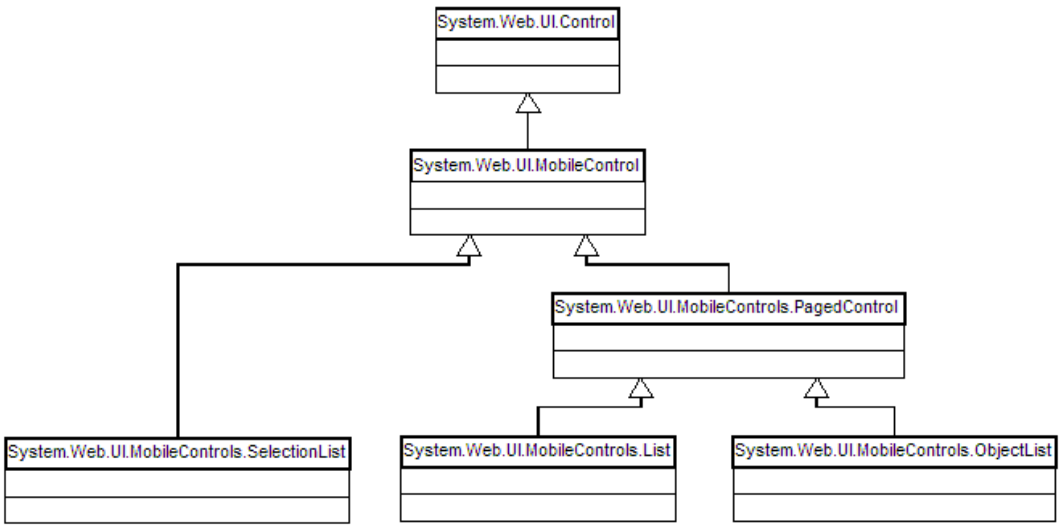
```
private void comando_Click(object sender, System.EventArgs e)

                {

                        num.Text = "Se ha pulsado el botón";

                }
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El resultado final en el terminal es el que se puede ver en la siguiente imagen.

2 LISTAS DE ELEMENTOS

Dentro de las aplicaciones móviles, es habitual el uso de los listados de elementos como principal forma de presentación de información. Esto es así debido a las importantes restricciones de la pantalla de los terminales y sobre todo a la pobre interfaz que presentan los dispositivos para la introducción de información por parte del usuario. Dentro del MIT disponemos de tres tipos diferentes de listas, modeladas cada una de ellas por una clase. Las clases de las que hablamos serán *SelectionList*, *ObjectList* y *List*. El siguiente diagrama de clases presenta la composición de dichas clases.



ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El control más simple es el correspondiente a *SelectionList*, ya que este, tal y como podemos descubrir en el diagrama de clases, no dispone de la capacidad de paginación. El control *SelectionList* es el único de los controles de listas que permite selección múltiple. Este control puede ser transformado en última instancia en una lista desplegable, en una serie de *radio buttons* o en otros elementos similares. Cualquier *SelectionList* debe ir acompañado de un botón de comando que nos permite enviar la selección del usuario al servidor.
El control *List* nos permitirá crear una lista estática o bien una lista interactiva. Cuando se utiliza este control de forma estática, el funcionamiento es similar a *SelectionControl*, es decir, se indican una serie de elementos y se acompaña el control *List* por un control correspondiente a un botón de comando, con el objetivo de enviar la selección de la lista al servidor. En cambio, si el control *List* funciona en modo interactivo, podremos utilizar cada uno de los ítems de la lista como un enlace, que generará un evento al ser seleccionado. Es decir, en este caso, los propios elementos de la lista son el punto de comunicación con el servidor y por lo tanto no es necesario un botón de comando auxiliar. Los ítems que componen la lista pueden ser especificados tanto de forma estática, como de forma dinámica a través de los que se denomina *data binding*.
Por último, el control *ObjectList* presenta la particularidad de no permitir la especificación estática de los elementos de la lista, debiendo realizar esta operación en todos los casos a través del enlace entre la lista y una fuente de datos. Esta fuente de datos será típicamente un conjunto de objetos (por ejemplo, un *array*) y se indicará qué campo del objeto se utilizará como clave, de tal forma que se mostrará dicha información en la lista. Una vez seleccionado un determinado elemento dentro de la lista, se mostrará el resto de información contenida en el objeto [25-30].

2.1 EL CONTROL SELECTIONLIST

Como hemos comentado, este control es el más sencillo dentro de los controles orientados a la generación de listas de elementos. En el siguiente ejemplo vemos un uso muy sencillo de este control en el que se presenta una lista de cuatro elementos, indicados estáticamente y se acompaña dicha lista de un botón destinado al envío de los datos al servidor.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="Sys-
tem.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="EjSelecionList1.aspx.cs" Inherits="MasterSalamanca.MobileWeb-
Form1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:SelectionList id="lista" runat="server" SelectType="ListBox">

                        <Item Value="uno" Text="uno" Selected="True"></Item>

                        <Item Value="dos" Text="dos"></Item>

                        <Item Value="tres" Text="tres"></Item>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                    <Item Value="cuatro" Text="cuatro"></Item>
                </mobile:SelectionList>
                <mobile:Command id="enviar" runat="server">Enviar</mobile:Command>
            </mobile:Form>
</body>
```

En el fragmento de código anterior se añaden cuatro elementos a la lista, esto se realiza a través de la etiqueta *item*. Otro elemento a destacar, es la propiedad *SelectType* que nos permitirá especificar la apariencia de la lista una vez generado el código en el adecuado lenguaje de marcado. Los posibles valores que puede tomar dicha propiedad son los siguientes:

- *DropDown* – Se corresponde con una lista desplegable
- *ListBox* – Se corresponde con una lista normal de elementos
- *Radio* – Cada elemento de la lista será un *radio button* seleccionable
- *MultiSelectListBox* – En este caso se permiten selecciones múltiples dentro de una interfaz similar al caso de *ListBox*
- *CheckBox* – Cada elemento de la lista se podrá seleccionar mediante lo que se denomina *checkbox*, y en este caso, la selección también puede ser múltiple

Las siguientes imágenes muestran el resultado final en el terminal.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

A continuación, vamos a ampliar el código de ejemplo anterior para explicar cómo se capturaría la selección de un elemento de la lista, una vez enviada esta información. En este caso, al formulario anterior le vamos a añadir un segundo formulario, que será el que muestre el resultado. El código completo de la página con los dos formularios es el siguiente.

```
<%@ Page language="c#" Codebehind="EjSelecionList1.aspx.cs" Inherits="MasterSalamanca.MobileWeb-
Form1" AutoEventWireup="false" %>

<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="Sys-
tem.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:SelectionList id="lista" runat="server" SelectType="ListBox">

                        <Item Value="uno" Text="uno" Selected="True"></Item>

                        <Item Value="dos" Text="dos"></Item>

                        <Item Value="tres" Text="tres"></Item>

                        <Item Value="cuatro" Text="cuatro"></Item>

                </mobile:SelectionList>

                <mobile:Command id="enviar" runat="server">Enviar</mobile:Command>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        </mobile:Form>

        <mobile:Form id="Form2" runat="server">
                <mobile:Label id="result" runat="server"></mobile:Label>
        </mobile:Form>
</body>
```

El siguiente fragmento de código muestra el código C# que se ejecutará al pinchar sobre el botón enviar del primer formulario. Como se puede comprobar en el código de la página (gracias a la propiedad *Codebehind*), el código asociado a la misma estará alojado en el fichero *EjSelecionList1.aspx.cs*.

```
using System;

namespace MasterSalamanca
{
        public class MobileWebForm1 : System.Web.UI.MobileControls.MobilePage
        {
                protected System.Web.UI.MobileControls.SelectionList lista;
                protected System.Web.UI.MobileControls.Command enviar;
                protected System.Web.UI.MobileControls.Form Form2;
                protected System.Web.UI.MobileControls.Label result;
                protected System.Web.UI.MobileControls.Form Form1;

                private void enviar_Click(object sender, System.EventArgs e)
                {
                        this.ActiveForm = Form2;
                        string seleccion = lista.Selection.Value;
                        result.Text = "El elemento seleccionado es: " + seleccion;
                }
        }
}
```

El resultado de la ejecución de este ejemplo en un terminal, sería muestra en las siguientes pantallas.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

En el caso de que tuviéramos una selección múltiple, el código C# que nos permitiría recuperar la información de todos los elementos seleccionados, sería el que se presenta a continuación. En este caso, sólo se muestra el método concreto, no se muestra toda la clase, ya que sería similar a lo visto en el último ejemplo. El único cambio que habría que realizar sería la inclusión de la línea que nos permite utilizar las clases *MobileListItemCollection* y *MobileListItem*, tal y como se muestra también en el listado siguiente.

```
Using System.Web.UI.MobileControls;
….
private void enviar_Click(object sender, System.EventArgs e)
{
 this.ActiveForm = Form2;
 MobileListItemCollection items = lista.Items;
 string seleccion = “”;
 foreach (MobileListItem item in items)
 {
   if (item.Selected)
   {
     seleccion += item.Value + “<br/>”;
   }
 }
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
  result.Text = "Los elementos seleccionados son: " + seleccion;
}
```

2.1.1 CONSTRUCCIÓN DE LISTAS CON DATA BINDING

Como se ha comentado anteriormente en varias ocasiones, es posible asignar o indicar los elementos de una lista de forma dinámica mediante el uso de lo que se denomina *data binding*. A pesar de que este ejemplo se basa en un control de tipo *SelectionList*, lo que veremos sobre *data binding* es totalmente aplicable al resto de controles de tipo lista. La fuente de datos que establecerá los elementos de la lista debe ser un objeto de tipo *IEnumerate* o *IListSource*. Dentro de las librerías de clases incluidas en el marco de trabajo de .NET, tenemos varias que implementan la interfaz *IEnumerate*, como por ejemplo *Array*, *ArrayList* o *MobileListItemCollection*. En el caso de la interfaz *IListSource*, sólo dos clases de .NET la implementan: *DataSet* y *DateTable*. *DataSet* no es más que una colección de objetos de tipo *DataTable*, y estos elementos están relacionados con la tecnología ADO.NET de acceso a bases de datos [31-34].

El siguiente ejemplo muestra cómo crear a una lista de tipo *SelectionList*, que es completada a partir de la información generada de forma programática. En primer lugar, vamos a ver el código correspondiente al propio control *SelectionList*. En este caso no se muestra el resto de código de la página, porque no es relevante y ya está explicado en anteriores ejemplos.

```
<mobile:SelectionList id=lista runat="server"
  SelectType="MultiSelectListBox"
  DataTextField="Campo" DataValueField="Valor">
</mobile:SelectionList>
```

En este fragmento de código, caben destacar las propiedades *DataTextField* y *DataValueField*, que nos permiten indicar a través de qué métodos seremos capaces de obtener la información correspondiente al texto y al valor, respectivamente, de cada uno de los ítems de la lista.

El código de las clases C# que servirán de soporte para la lista y para la página que contiene dicha lista, será el siguiente.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.Mobile;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.MobileControls;
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace MasterSalamanca
{
	class Elemento
	{
		private String campo, valor;

		public Elemento (String campo, String valor)
		{
			this.campo = campo;
			this.valor = valor;
		}

		public String Campo{get{return this.campo;}}
		public String Valor{get{return this.valor;}}
	}


	public class EjSelectionList2 : System.Web.UI.MobileControls.MobilePage
	{
		protected System.Web.UI.MobileControls.SelectionList lista;
		protected System.Web.UI.MobileControls.Form Form1;

		private void Page_Load(object sender, System.EventArgs e)
		{
			if (!IsPostBack)
			{
				ArrayList datos = new ArrayList();
				datos.Add(new Elemento("A", "1"));
				datos.Add(new Elemento("AB", "3"));
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                    datos.Add(new Elemento("ABC", "4"));
                    lista.DataSource = datos;
                    lista.DataBind();
                }
            }
        }
}
```

Acabamos de presentar dos clases, que servirán para crear la lista de forma dinámica. La primera clase se ha denominado *Elemento* y no es más que lo que podríamos denominar un contenedor de datos. Esta clase presenta dos atributos y dos métodos destinados a permitir recuperar la información almacenada en dichos atributos. En este ejemplo tan sencillo, los métodos simplemente retornan la información de los atributos, pero es obvio que se podrían realizar operaciones más complejas para así determinar qué información será mostrada en la lista.

En segundo lugar, tendremos la clase *EjSelectionList2*, que será la clase que soporte la lógica de la página ASP.NET. En el ejemplo que nos ocupa, lo único que se ha incluido en esta clase es una serie de líneas de código dentro del método *Page_Load*. Este método, tal y como indica su nombre, se ejecuta al cargar la página, es decir, cada vez que la página en cuestión sea solicitada, se ejecutará este método como lo que podríamos definir como un proceso de iniciación de dicha página. Lo que hacemos en este método en primer lugar, es comprobar si la página está siendo accedida por primera vez o se trata de una recarga debida a que el usuario ha retrocedido en la secuencia de navegación. Esto se hace gracias a *IsPostBack*. Si la página no es recargada, se genera la información y se incluye la misma dentro de la lista. Como vemos, lo que hacemos es crear un objeto de tipo *ArrayList*, que tal y como se ha indicado implementa la interfaz *IEnumerate* y se introducen objetos de tipo *Elemento* en esta matriz. Una vez hecho esto, se asigna el objeto de tipo *ArrayList* a la lista y por lo tanto esta quedará asociada desde el punto de vista de la fuente de datos, a la matriz de datos indicada [35-40].

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

2.2 EL CONTROL LIST

El control *List*, ya ha sido presentado en la introducción de los controles de tipo lista y tal y como se dijo entonces, es muy similar al control *SelectionList*, si bien, presenta la capacidad de paginación ausente en este último. El control *List*, puede ser utilizado para generar una lista estática o una lista interactiva. En este último caso, no hay posibilidad de hacer selección múltiple, ya que cada elemento de la lista sirve como enlace para la ejecución de una acción. En este caso, es obvio que no se necesita el botón de comando adicional. Por defecto, el control lista generará una lista en modo interactivo. Para modificar este comportamiento habrá que establecer la propiedad *ItemsAsLinks* con valor *false*. El siguiente ejemplo muestra un ejemplo sencillo sobre el comportamiento de un control de tipo *List*, funcionando en modo interactivo. En el siguiente fragmento de código muestra únicamente la definición de la lista.

```
<mobile:Form id="Form1" runat="server">
        <mobile:List id="lista" runat="server">
                <Item Value="Elto1" Text="Elto1"></Item>
                <Item Value="Elto2" Text="Elto2"></Item>
                <Item Value="Elto3" Text="Elto3"></Item>
                <Item Value="Elto4" Text="Elto4"></Item>
        </mobile:List>
</mobile:Form>
```

Como vemos definimos una lista de forma estática (sin inclusión programática de datos) y con comportamiento dinámico, es decir, cada ítem de la lista actuará como un enlace. El siguiente fragmento

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

de código muestra cómo sería el método que responde a la selección por parte del usuario de uno de los ítems de la lista.

```
private void List1_ItemCommand(object sender, System.Web.UI.MobileControls.ListCommandEventArgs e)
{
        this.ActiveForm = Form2;
        String seleccion = e.ListItem.Value;
        result.Text = "Ha seleccionado: " + seleccion;
}
```

Las siguientes imágenes muestran el resultado final.

2.2.1 PAGINACIÓN

Como se ha comentado en varias ocasiones anteriormente, el control *List* presenta la capacidad de paginación de la lista de forma automática. Para que la paginación sea posible, debemos indicar el número de ítems a mostrar por página y el atributo *Paginate* del formulario debe ser establecido con el valor *true*.

Para realizar una paginación más eficiente, se puede realizar el proceso de tal forma que sólo se suministre al control aquella información que va a mostrar, es decir, la información referente a la página que está generando en cada momento.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

2.3 EL CONTROL OBJECTLIST

Este control es quizás el más potente de todos los controles destinados a listas ya que permite un mayor número de tipos de datos como fuente y presenta un manejo más potente de los comandos o acciones a asociar con cada ítem de la lista. Como ya se mencionó anteriormente, el control *ObjectList* presenta las mismas capacidades de paginación que el control *List*. Las listas basadas en *ObjectList* no admiten, en cambio, la provisión estática de los elementos que componen dicha lista.

Durante el estudio del control *List*, hemos visto cómo se puede asociar un control a una fuente de datos. En este caso, con el control *ObjectList*, vamos a comprobar la potencia del mismo manejando fuentes de una forma más compleja. Veremos cómo, partiendo de una fuente de datos con elementos que se componen de varios campos, el control mostrará un listado de un determinado campo de los diferentes elementos de la fuente y al seleccionar uno de ellos, se mostrará una pantalla con la información completa del elemento en cuestión de la fuente de datos [41-43].

El siguiente ejemplo muestra un ejemplo sencillo de cómo funciona este control. En primer lugar, lo que hacemos es definir una clase contenedora de datos. Esta clase tendrá campos de varios tipos de datos. El código de esta clase es el siguiente.

```
class ElementoComplejo
        {
                private String id;
                private String campo1, campo2;
                private int campo3;
                private double campo4;
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        public ElementoComplejo(String id, String campo1, String campo2, int campo3, double
campo4)
        {
                this.id = id;
                this.campo1 = campo1;
                this.campo2 = campo2;
                this.campo3 = campo3;
                this.campo4 = campo4;
        }

        public String Id{get{return this.id;}}
        public String Campo1{get{return this.campo1;}}
        public String Campo2{get{return this.campo2;}}
        public int Campo3{get{return this.campo3;}}
        public double Campo4{get{return this.campo4;}}
}
```

El formulario de la página, como es de esperar, tendrá un control de tipo *ObjectList*, que se llamará *lista*. Lo único que se presenta en este documento, es el método *Page_Load* asociado al formulario, ya que el resto del código del mismo es irrelevante.

```
private void Page_Load(object sender, System.EventArgs e)
        {
                if (!IsPostBack)
                {
                        ArrayList datos = new ArrayList();
                        for (int i=0; i<10; i++)
                        {
                                ElementoComplejo ec = new ElementoComplejo(
                                        "id" + i, "c1 - " + i, "c2 - " + i,
                                        i, i);
                                datos.Add(ec);
                        }

                        lista.DataSource = datos;
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                    lista.LabelField = "Id";
                    lista.DataBind();
                }


            }
```

Se puede visualizar más de un ítem en el listado de todos los elementos que componen la lista. Esta visualización se hará de forma tabular. Para realizar esta operación, lo único que tendríamos que hacer es añadir la siguiente línea al método *Page_Load*, que como se puede imaginar al verla, establece los campos *Id* y *Campo1* del elemento de datos de tipo *ElementoComplejo*, como elementos a visualizar en el listado.

```
lista.TableFields = "Id;Campo1";
```

Como se ha comentado ya, el control *ObjectList* permite una gestión más compleja de los comandos asociados a los diferentes ítems de la lista. En concreto, nos permite asociar más de un comando a un único ítem. Es decir, en todos los ejemplos que hemos visto hasta ahora, al seleccionar un ítem, se disparaba una determinada acción. En este caso, podremos también seleccionar qué acción se realizará, partiendo del hecho de que existe un elemento seleccionado, podremos realizar varias operaciones sobre este registro o elemento de la lista seleccionado.
El siguiente código muestra cómo sería el código correspondiente a la página. Es similar a ejemplos vistos anteriormente, pero podemos comprobar que tenemos la definición de dos comandos.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="Sys-
tem.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="ObjectListEj1.aspx.cs" Inherits="MasterSalamanca.ObjectListEj1"
AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:ObjectList id="lista" runat="server" LabelStyle-StyleReference="title" Command-
Style-StyleReference="subcommand">

                        <Command Name="Campo1" Text="Muestra campo 1"></Command>

                        <Command Name="Campo2" Text="Muestra campo 2"></Command>

                </mobile:ObjectList>

        </mobile:Form>

        <mobile:Form id="Form2" runat="server">

                <mobile:Label id="Label1" runat="server">Label</mobile:Label>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                <mobile:Label id="Label2" runat="server">Label</mobile:Label>
        </mobile:Form>
</body>
```

El código C# asociado a esta página, es muy similar, por lo que único que se mostrará en el siguiente listado, será el método encargado de gestionar la selección de los diferentes comandos.

```
protected void Elto_OnItemCommand(Object sender, ObjectListCommandEventArgs  args)
{
        this.ActiveForm = Form2;
        if (args.CommandName == "Campo1")
        {
                Label1.Text = "Campo 1";
                Label2.Text = args.ListItem["Campo1"];
        }
        else
        {
                Label1.Text = "Campo 2";
                Label2.Text = args.ListItem["Campo2"];
        }
}
```

Las siguientes imágenes muestran cómo sería el resultado final de estos ejemplos.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

3 EL CONTROL CALENDAR

El control *Calendar*, como su nombre indica, nos permite realizar de forma sencilla y flexible, la gestión de introducción y selección de fechas por parte del usuario. En aquellos casos en el que el terminal de acceso lo permita, el resultado será gráfico. El uso de este control para que el usuario indique fechas presenta dos ventajas importantes. La primera de ella es la uniformidad en las fechas,

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

lo que evita la necesidad de formateo de las fechas, o el control de posibles errores. Por otro lado, facilita el proceso de cara al usuario, ya que este no tiene que teclear la fecha de forma explícita.
A la hora de introducir este control dentro de un formulario, el programador podrá especificar gran cantidad de propiedades del control, que modificarán el comportamiento del mismo. El siguiente código muestra un ejemplo muy sencillo. En primer lugar, tenemos el código correspondiente al formulario que contiene el control *Calendar*.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Calendario.aspx.cs" Inherits="MasterSalamanca.Calendario" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Calendar id="cal" runat="server" SelectionMode="DayWeek" SelectedDate="2004-01-01"></mobile:Calendar>

                <mobile:Label id="Label1" runat="server">Label</mobile:Label>

        </mobile:Form>

</body>
```

El código que hay detrás de la página anterior es muy sencillo, y lo que hará será mostrar por pantalla, dentro de una etiqueta, la fecha seleccionada. El control *Calendar*, como es de esperar, nos permite recuperar la fecha en una gran variedad de formatos.

```
private void cal_SelectionChanged(object sender, System.EventArgs e)

{

        Label1.Text = cal.SelectedDate.ToShortDateString();

}
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

4 EL CONTROL PHONECALL

Estamos contemplado el desarrollo de servicios de provisión de información para terminales móviles. Típicamente, estos terminales móviles serán teléfonos móviles, lo que nos lleva a tener en nuestras manos un elemento natural de comunicación, de comunicación vocal. Muchos de nuestros servicios podrán ser mucho más completos y efectivos, si enlazan el proceso con una llamada estándar de voz.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

El control *PhoneCall* permitirá la inicialización automática de llamadas si el terminal lo soporta. En otro caso, el control mostrará un enlace que puede ser seleccionado por el usuario con el fin de realizar la llamada.
El siguiente ejemplo muestra el uso sencillo de este control.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="PhoneCallEj1.aspx.cs" Inherits="MasterSalamanca.PhoneCallEj1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">


<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

   <mobile:Form id=Form1 runat="server">

<mobile:PhoneCall id=telef runat="server" PhoneNumber="923 12 34 56" AlternateUrl="http://gsii.usal.es">Llamar a</mobile:PhoneCall></mobile:Form>

</body>
```

5 EL CONTROL ADROTATOR

Dentro del marketing o publicidad *online*, es muy común la presentación dentro de las páginas de lo que se denominan *banners* y que se podría traducir por pancarta o banderola. Para hacer más efectivo este sistema publicitario, habitualmente es requerida una rotación en el mensaje mostrado, de tal manera que la cantidad de usuarios que contemplan los anuncios es mayor y también es mayor el número de mensajes publicitarios que se le muestran a un mismo usuario. Este comportamiento es tan común dentro de los entornos web, que *Microsoft* provee un control dentro del MIT que nos permite realizar estas operaciones de forma automática. Este control es el denominado *AdRotator*.
El control nos permite introducir dentro de nuestras páginas, mensajes publicitarios gráficos. La información sobre los mensajes publicitarios se debe proveer a través de un fichero XML, con el siguiente formato [44].

```
<?xml version="1.0"?>

<Advertisements>

  <Ad>

    <ImageUrl>[Imagen a mostrar]</ImageUrl>

    <MonoImageUrl>[Imagen monocromática a mostrar. Será típicamente un fichero WBMP para navegadores WML]</MonoImageUrl>

    <NavigateUrl>[URL a mostrar al seleccionar el enlace]</NavigateUrl>

    <AlternateText>[Texto a mostrar si no se puede ]</AlternateText>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
  <Keyword>[Categoría del banner]</Keyword>
  <Impressions>[Indica cuantas veces debe ser mostrado el anuncio con respecto al resto]</Impressions>
 </Ad>
 <Ad>
 ……
</Advertisements>
```

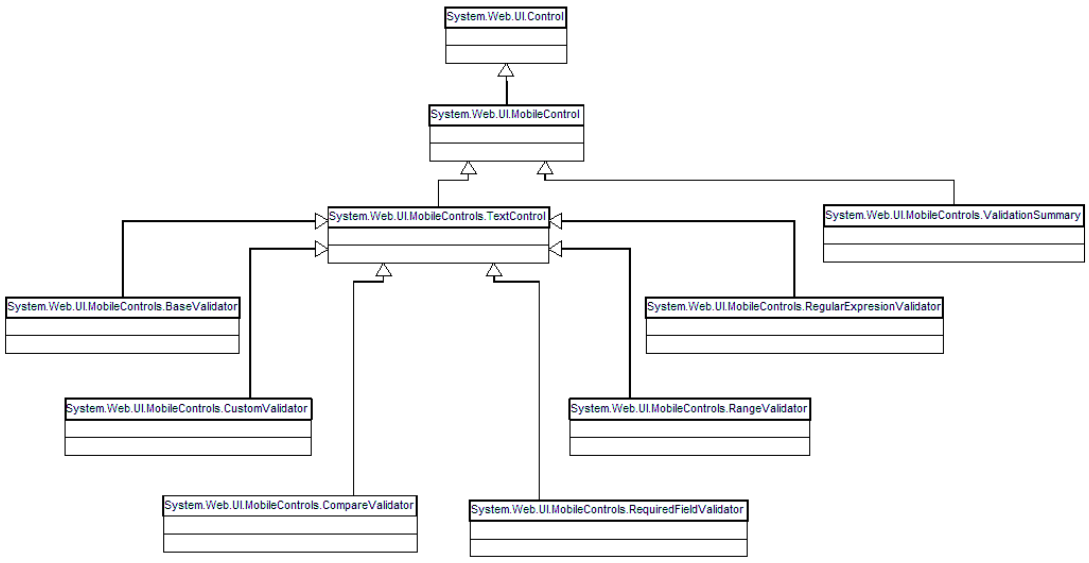
6 CONTROLES DE VALIDACIÓN

Dentro de cualquier aplicación, las validaciones de la información proporcionada por el usuario son muy importante, pero en el caso de las aplicaciones web es más importante si cabe, ya que la información, a pesar de ser errónea, es posible que sea enviada al servidor y esto supone un consumo de red. Este hecho, dentro de los entornos móviles, también cobra especial importancia, ya que las comunicaciones son más lentas y caras.

Algunas de las validaciones típicas a realizar serán el chequeo de que ciertos campos han sido completados, que tienen un formato correcto (correo electrónico o teléfono) o que dos campos contienen la misma información (común cuando se tiene que confirmar una clave de acceso, por ejemplo).

La realización de todas estas operaciones de validación, suele ser un trabajo costoso, tanto si hace en la parte del servidor, como si se hace en la parte del cliente (no siempre se puede comprobar todo en esta parte cliente). Para facilitar esta labor, ASP.NET proporciona un método flexible basado en lo que se denominan, controles de validación. Dentro del MIT, también nos encontramos con algunos controles de este tipo, que debido a las peculiaridades de los clientes a los que está orientado el MIT, no presentará procesos de validación en el cliente y todos ellos se realizarán en la parte del servidor.

La siguiente imagen presenta el diagrama de clases de los controles que proporciona el MIT para la realización de validaciones [45].



ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

6.1 EL CONTROL REQUIREDFIELDCONTROL

El control *RequiredFieldControl* es el más simple de los controles de validación y será el más común de todos. Este control simplemente comprueba que el usuario haya introducido alguna información dentro de un determinado campo.

El siguiente ejemplo muestra un sencillo ejemplo, en el que se le presenta al usuario lo que podría ser una pantalla de *login* o identificación y este debe especificar su nombre y clave de acceso. En caso de que alguno de los dos campos esté sin completar, se mostrará el error correspondiente. Por lo tanto, tendremos que introducir un validador para cada uno de los campos, el nombre y la clave de acceso. En caso de que ambos casos estén completos, iremos al segundo formulario de la página, que contiene simplemente un texto de bienvenida. El código de la página se muestra a continuación.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Valid1.aspx.cs" Inherits="MasterSalamanca.Valid1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="Label1" runat="server" BreakAfter="False">Nombre: </mobile:Label>

                <mobile:TextBox id="nombre" title="Nombre:" runat="server" EnableViewState="False"></mobile:TextBox>

                <mobile:Label id="Label2" runat="server" BreakAfter="False">Clave: </mobile:Label>

                <mobile:TextBox id="clave" runat="server" Password="True"></mobile:TextBox>

                <mobile:RequiredFieldValidator id="validaClave" runat="server" ErrorMessage="Indique su clave de acceso" ControlToValidate="clave"></mobile:RequiredFieldValidator>

                <mobile:RequiredFieldValidator id="validaNombre" runat="server" ErrorMessage="Indique su nombre" ControlToValidate="nombre"></mobile:RequiredFieldValidator>

                <mobile:Command id="ok" runat="server">ok</mobile:Command>

        </mobile:Form>

        <mobile:Form id="Form2" runat="server">

                <mobile:Label id="msg" runat="server">Bienvenido...</mobile:Label>

        </mobile:Form>

</body>
```

El código correspondiente al botón *ok* de la página, es muy sencillo y lo único que hará será comprobar que la página es válida, y en caso de que sea así, activar el segundo formulario.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
if (Page.IsValid)
{
        ActiveForm = Form2;
}
```

La siguiente imagen muestra la pantalla de error, resultado de la validación.

6.2 EL CONTROL COMPAREVALIDATOR

El control *CompareValidator*, tal y como mencionamos anteriormente, nos permite comparar dos campos relacionados. La posible comparación de estos campos, no se limita a comparaciones de igualdad, sino que también podremos, por ejemplo, comprobar si un campo tiene un valor mayor que el otro [46].

Vamos a ampliar el ejemplo anterior, de tal forma que el usuario tendrá que identificarse indicando la clave de acceso dos veces para la comprobación de la misma. Es decir, tendremos un campo más, y comprobaremos que todos los campos *nombre* y *clave* están completos, y que los campos correspondientes a la clave y la confirmación de esta, contienen la misma información.

La página tendría ahora el siguiente código.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Valid2.aspx.cs" Inherits="MasterSalamanca.Valid2" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
        <mobile:Form id="Form1" runat="server">
                <mobile:Label id="Label1" runat="server" BreakAfter="False">Nombre: </mobile:Label>
                <mobile:TextBox id="nombre" title="Nombre:" runat="server" Ena-
bleViewState="False"></mobile:TextBox>
                <mobile:Label id="Label2" runat="server" BreakAfter="False">Clave: </mobile:Label>
                <mobile:TextBox id="clave" runat="server" Password="True"></mobile:TextBox>
                <mobile:Label id="Label3" runat="server" BreakAfter="False">Confirmaci&#243;n
Clave:</mobile:Label>
                <mobile:TextBox id="clave2" runat="server" Password="True"></mobile:TextBox>
                <mobile:RequiredFieldValidator id="validaNombre" runat="server" ControlToVali-
date="nombre" ErrorMessage="Indique su nombre"></mobile:RequiredFieldValidator>
                <mobile:RequiredFieldValidator id="validaClave" runat="server" ControlToVali-
date="clave" ErrorMessage="Indique su clave de acceso"></mobile:RequiredFieldValidator>
                <mobile:CompareValidator id="validClaves" runat="server" ControlToValidate="clave2"
ErrorMessage="Las claves indicadas no coinciden" ControlToCompare="clave"></mobile:CompareValida-
tor>
                <mobile:Command id="ok" runat="server">ok</mobile:Command>
        </mobile:Form>
        <mobile:Form id="Form2" runat="server">
                <mobile:Label id="msg" runat="server">Bienvenido...</mobile:Label>
        </mobile:Form>
</body>
```

El código correspondiente al control del botón que permite activar el segundo formulario sería exactamente el mismo que el visto en el ejemplo correspondiente al control *RequiredFieldValidator*.

6.3 EL CONTROL RANGEVALIDATOR

El control *RangeValidator*, tal y como indica su nombre, nos permite comprobar que el valor indicado por un usuario dentro de un determinado campo está dentro de un rango de valores predeterminado. En este caso, vamos a ver un ejemplo en el que se le presenta al usuario una página en la que este debe indicar su fecha de nacimiento. Esta fecha no debe ser menor que el 1 de Enero de 1900, ni mayor que la fecha actual. Estos valores mínimo y máximo, que delimitan el rango a evaluar, se establecerán de forma programática y pertenecerán a un control de tipo *RangeValidator*. El código de la página es el siguiente.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="Sys-
tem.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>
<%@ Page language="c#" Codebehind="Valid3.aspx.cs" Inherits="MasterSalamanca.Valid3" Au-
toEventWireup="false" %>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" content="C#">
<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">
<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
        <mobile:Form id="Form1" runat="server">
                <mobile:Label id="Label1" runat="server">Fecha de Nacimiento: </mobile:Label>
                <mobile:TextBox id="fecha" runat="server"></mobile:TextBox>
                <mobile:RangeValidator id="validFecha" runat="server" ErrorMessage="La fecha no puede
ser inferior a 1900 ni mayor que la actual" ControlToValidate="fecha" Type="Date"></mobile:RangeValida-
tor>
                <mobile:Command id="ok" runat="server">ok</mobile:Command>
        </mobile:Form>
        <mobile:Form id="Form2" runat="server">
                <mobile:Label id="msg" runat="server">Bienvenido...</mobile:Label>
        </mobile:Form>
</body>
```

El código para establecer los valores del rango, será el siguiente.

```
private void Page_Load(object sender, System.EventArgs e)
{
        validFecha.MinimumValue = "1/1/1900";
        validFecha.MaximumValue = System.DateTime.Today.Day +
                "/" + System.DateTime.Today.Month + "/" +
                System.DateTime.Today.Year;
}
```

El resultado de la validación sería el siguiente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

6.4 EL CONTROL REGULAREXPRESSIONVALIDATOR

Este control, tal y como su propio nombre indica, nos permite realizar comprobaciones del cumplimiento o no de las restricciones establecidas por una expresión regular, por parte de la información especificada por el usuario. Evidentemente, este control es sumamente potente y con él podemos comprobar correos electrónicos, teléfonos, cuentas bancarias o cualquier otra información que tenga un cierto formato reconocido.
El código correspondiente al formulario es el siguiente.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Valid4.aspx.cs" Inherits="MasterSalamanca.Valid4" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="Label1" runat="server" BreakAfter="False">Teclee su correo electr&#243;nico</mobile:Label>

                <mobile:TextBox id="correo" runat="server"></mobile:TextBox>

                <mobile:RegularExpressionValidator id="RegularExpressionValidator1" runat="server" ErrorMessage="El correo indicado es incorrecto." ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*" ControlToValidate="correo"></mobile:RegularExpressionValidator>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
                <mobile:Command id="ok" runat="server">Ok</mobile:Command>
        </mobile:Form>
</body>
```

En el código anterior, vemos que la expresión regular correspondiente al correo electrónico, indicada como valor para el atributo *ValidationExpression*.

6.5 EL CONTROL CUSTOMVALIDATOR

Este control, a diferencia del resto, no provee ningún sistema de validación automático. Este control nos permite crear nuestra propia validación, que el sistema invocará de forma automática. Para realizar la validación, deberemos sobrescribir el método que se encarga de gestionar el evento *ServerValidate*. El método que debemos sobrescribir es el siguiente.

```
void ServerValidate(Object source, ServerValidateEventArgs args)
```

En el ejemplo a realizar para ilustrar el uso de este control, vamos a realizar una validación que nos permite comprobar que el número indicado por el usuario es un número par.
El código correspondiente al formulario será el que se muestra a continuación.

```
<%@ Page language="c#" Codebehind="Valid5.aspx.cs" Inherits="MasterSalamanca.Valid5" AutoEventWireup="false" %>

<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">
<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
        <mobile:Form id="Form1" runat="server">
                <mobile:Label id="Label1" runat="server" BreakAfter="False">Escriba un n&#250;mero par</mobile:Label>
                <mobile:TextBox id="num" runat="server"></mobile:TextBox>
                <mobile:CustomValidator id="CustomValidator1" runat="server" ErrorMessage="Por favor, indique un número par" ControlToValidate="num"></mobile:CustomValidator>
                <mobile:Command id="ok" runat="server">Ok</mobile:Command>
        </mobile:Form>
</body>
```

Del código asociado a este control, cabe destacar únicamente el método de validación y la línea del método de inicialización de la página que establece el método *ServerValidate* como el método encargado de gestionar la validación correspondiente al control *CustomValidator1*, que será el control de validación para comprobar que el usuario introduce un número par.

```
private void InitializeComponent()
{
        this.CustomValidator1.ServerValidate += new System.Web.UI.WebControls.ServerValidateEventHandler(this.ServerValidate);
        this.Load += new System.EventHandler(this.Page_Load);

}

void ServerValidate(object source, System.Web.UI.WebControls.ServerValidateEventArgs args)
{
        try
        {
                int n = Int32.Parse(num.Text);
                if (n%2 == 0)
                {
                        args.IsValid = true;
                }
                else
                {
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
                    args.IsValid = false;
                }
            }
            catch (FormatException e)
            {
                args.IsValid = false;
            }
}
```

6.6 EL CONTROL VALIDATIONSUMMARY

El control *ValidationSummary*, agrupará todos los mensajes de error producidos por todos los controles de validación incluidos en una página. Así, podremos mostrar todos los mensajes de error juntos o tratar esta información de aquella forma que creamos más adecuada.

Evidentemente, este control deberá ir acompañando a otros controles de validación. Una vez realizadas las validaciones por parte de cada uno de estos controles de validación, el control *ValidationSummary* presentará un listado conjunto de todos los mensajes de error.

El control *ValidationSummary* se puede incluir tanto dentro del propio formulario que contiene el resto de los controles de validación, como en un formulario a parte.

El siguiente ejemplo, es una versión mejorada del ejemplo en que se solicitaba al usuario su nombre y la clave de acceso por duplicado. En este caso, para mostrar los mensajes de error lo que vamos a hacer es tener un formulario adicional, de tal forma que todos los mensajes de error se muestren conjuntamente.

El código correspondiente a la página que contiene tres formularios es el siguiente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Valid2.aspx.cs" Inherits="MasterSalamanca.Valid2" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

<mobile:Label id=Label1 runat="server" BreakAfter="False">Nombre: </mobile:Label>

<mobile:TextBox id=nombre title=Nombre: runat="server" EnableViewState="False"></mobile:TextBox>

<mobile:Label id=Label2 runat="server" BreakAfter="False">Clave: </mobile:Label>

<mobile:TextBox id=clave runat="server" Password="True"></mobile:TextBox>

<mobile:Label id=Label3 runat="server" BreakAfter="False">Confirmaci&#243;n Clave:</mobile:Label>

<mobile:TextBox id=clave2 runat="server" Password="True"></mobile:TextBox>

<mobile:RequiredFieldValidator id=validaNombre runat="server" ErrorMessage="Indique su nombre" ControlToValidate="nombre"></mobile:RequiredFieldValidator>

<mobile:RequiredFieldValidator id=validaClave runat="server" ErrorMessage="Indique su clave de acceso" ControlToValidate="clave"></mobile:RequiredFieldValidator>

<mobile:CompareValidator id=validClaves runat="server" ErrorMessage="Las claves indicadas no coinciden" ControlToValidate="clave2" ControlToCompare="clave"></mobile:CompareValidator>

<mobile:Command id=ok runat="server">ok</mobile:Command>

        </mobile:Form>

        <mobile:Form id="Form2" runat="server">

<mobile:Label id=msg runat="server">Bienvenido...</mobile:Label>

        </mobile:Form>

        <mobile:Form id="Form3" runat="server">

<mobile:ValidationSummary id=errores runat="server" FormToValidate="Form1"></mobile:ValidationSummary>

        </mobile:Form>

</body>
```

El código del botón encargado de enviar la información del primer formulario, comprobará si la página es válida. En caso de no serlo, nos mostrará el formulario que contiene el control *ValidationSummary* para mostrar todos los mensajes de error.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
private void ok_Click(object sender, System.EventArgs e)
{
        if (Page.IsValid)
        {
                ActiveForm = Form2;
        }
        else
        {
                ActiveForm = Form3;
        }
}
```

El resultado de la validación dentro del terminal es el siguiente.

7 APARIENCIA Y HOJAS DE ESTILO

Como hemos visto hasta este punto, el entorno MIT se encarga de *renderizar* en último término la página configurada en el formulario, de tal forma que pueda ser visualizada correctamente en el terminal concreto que ha realizado la petición. Si bien este sistema es automático, el propio MIT nos provee un procedimiento orientado a proporcionar al usuario un mayor control sobre el resultado final.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Para controlar a más bajo nivel el proceso de *renderizado*, tenemos dentro de los formularios móviles algunas propiedades que denominaremos, propiedades de estilo. Estas propiedades nos permiten modificar aspectos visuales de nuestra página, como puede ser la fuente de los textos o el color de los mismos. Por supuesto, todas estas características de visualización controladas con las propiedades de estilo sólo tienen efecto en aquellos terminales que soportan dichas características. Los terminales que no presenten pantalla en color evidentemente no podrán mostrar texto en color.
Las propiedades de estilo dentro de MIT, presentan ciertas similitudes con las hojas de estilo habituales (CSS), si bien, estas últimas presentan muchas más capacidades y son mucho más ricas.
A parte de estas capacidades de concreción en lo que a la apariencia se refiere, también tenemos ciertos estilos predefinidos que podemos aplicar a nuestros textos. Estos formatos predefinidos, son especificados a través del atributo *StyleReference*. Los terminales presentarán estos formatos predefinidos de la forma más adecuada. Los formatos predefinidos son aplicables únicamente a los controles *Label*. Los posibles valores a especificar son los siguientes:

- Title
- Error
- SubCommand

El siguiente ejemplo muestra cómo se especifican los formatos de texto. El siguiente ejemplo muestra varios casos, en algunos se indica un formato predefinido a través de *StyleReference* y en otros casos se hace de forma específica.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="Formatos.aspx.cs" Inherits="MasterSalamanca.Formatos" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="Label1" runat="server" Font-Bold="True" Font-Size="Small">Mobile Internet Toolkit</mobile:Label>

                <mobile:Label id="Label2" runat="server" Font-Bold="True" Font-Size="Large" Font-Italic="True" Font-Name="Book Antiqua" ForeColor="Blue">Mobile Internet Toolkit</mobile:Label>

                <mobile:Label id="Label4" runat="server" Font-Italic="True" Font-Name="Forte" ForeColor="Red">Mobile Internet Toolkit</mobile:Label>

                <mobile:Label id="Label3" runat="server" StyleReference="error">Mobile Internet Toolkit</mobile:Label>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
            <mobile:Label id="Label5" runat="server" StyleReference="subcommand">Mobile Internet
Toolkit</mobile:Label>
            <mobile:Label id="Label6" runat="server" StyleReference="title">Mobile Internet
Toolkit</mobile:Label>
        </mobile:Form>
</body>
```

La siguiente captura muestra el resultado en el terminal de los diferentes formatos.

7.1 HOJAS DE ESTILO

Las hojas de estilo, nos permiten definir estilos a aplicar a los diferentes controles. Un estilo no será más que un conjunto de características referentes a la visualización del resultado de los controles. Para utilizar dentro de nuestras páginas una hoja de estilo, lo único que tendremos que hacer es introducir dentro de dicha página un control de tipo *StyleSheet*.

Dentro del control *StyleSheet* incluido en la página, definiremos los diferentes estilos que componen la hoja. Estos estilos serán los mismos que se pueden aplicar directamente sobre un control, pero de esta forma, los definiremos una vez y los utilizaremos las veces que deseemos. De esta forma, también se hace mucho más fácil el proceso de modificación de los estilos de una página.

Una vez que hemos definido un estilo, se puede aplicar a los controles a través del atributo *StyleReference* de los mismos, tal y como hemos visto anteriormente.

El siguiente ejemplo define dos estilos dentro de una hoja y aplica dichos estilos a una serie de controles.

```
<%@ Page language="c#" Codebehind="HojasEstilo.aspx.cs" Inherits="MasterSalamanca.HojasEstilo" AutoEventWireup="false" %>
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="Sys-
tem.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="Label1" runat="server" StyleReference="Estilo 1">Mobile Internet
Toolkit</mobile:Label>

                <mobile:Label id="Label2" runat="server" StyleReference="Estilo 2">Mobile Internet
Toolkit</mobile:Label>

                <mobile:TextBox id="TextBox1" runat="server" StyleReference="Estilo 1"></mobile:Text-
Box>

        </mobile:Form>

        <mobile:StyleSheet id="StyleSheet1" runat="server">

                <mobile:Style Font-Size="Large" Font-Name="Gill Sans Ultra Bold" Font-Bold="True"
BackColor="Yellow" ForeColor="Red" Alignment="Center" Name="Estilo 1"></mobile:Style>

                <mobile:Style Font-Size="Large" Font-Italic="True" BackColor="Aqua" Alignment="Left"
Name="Estilo 2"></mobile:Style>

        </mobile:StyleSheet>

</body>
```

Es lógico que una aplicación web mantenga una apariencia similar en todos sus elementos, de tal forma que de la sensación de consistencia. Bien, para hacer más sencillo este proceso, el MIT no provee un sistema por el que podemos utilizar hojas de estilo definidas de forma externa, es decir, no es necesario que estén incrustadas en la propia página, tal y como se mostraba en el último ejemplo.
Este sistema, se basa en la definición de la hoja estilo en un fichero separado. El proceso comienza con la definición de un control propio (un fichero *ascx*), que tendrá un control correspondiente a una hoja de estilo incrustada. Esta hoja de estilo, definirá los formatos generales a aplicar en la aplicación. Una vez que tenemos esto, introduciremos en nuestra página un control del tipo *StyleSheet* y en este control, estableceremos el atributo *ReferencePath*, de tal forma que haga referencia al control creado con los estilos generales. A partir de este momento, podremos utilizar los estilos generales para los controles incluidos dentro de la página.
El resultado del anterior ejemplo en terminales con diferentes capacidades será el siguiente.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

8 CREACIÓN DE CONTROLES

A parte de los controles pertenecientes al MIT que hemos visto y estudiado hasta este punto, el marco de trabajo nos permite la creación de nuestros propios controles, encapsulando así ciertos comportamientos que necesitemos. Una vez creados estos controles, pueden ser usados dentro de los formularios, de forma similar a cómo se hace el uso de controles propios del MIT.

8.1 CONTROLES DE USUARIO

Los controles de usuario nos permitirán reutilizar de forma sencilla y rápida, lógica creada para otras páginas. Para crear un control de usuario, usaremos el mismo formato de fichero que utilizamos para la creación de páginas basadas en formularios, pero en este caso, el nombre del fichero no tendrá extensión *aspx*, sino que su extensión será *ascx*.

Para definir un control de usuario, simplemente introduciremos dentro del mismo, aquellos controles (pertenecientes al MIT o no) que deseemos agrupar en el control de usuario que estamos creando. Una vez que tengamos este hecho finalizado, podremos utilizar el control de usuario como un único elemento, de tal forma que en realidad estamos modelando varios controles unidos. Es sencillo imaginar la gran mejora desde el punto de vista la reutilización de código y desde el punto de vista del mantenimiento de nuestra aplicación.

El siguiente código muestra un ejemplo de un control de usuario, en el que se presentan dos etiquetas con un texto fijo. Este ejemplo nos permite tener una posible cabecera para todas las páginas de nuestra aplicación. El código correspondiente al control de usuario es el siguiente.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Control Language="c#" AutoEventWireup="false" Codebehind="UserControl1.ascx.cs" Inherits="MasterSalamanca.UserControl1" TargetSchema="http://schemas.microsoft.com/Mobile/WebUserControl" %>

<mobile:Label id="Label2" runat="server" StyleReference="title" Alignment="Center">MOBILE INTERNET TOOLKIT</mobile:Label>

<mobile:Label id="Label1" runat="server" Alignment="Center" Font-Italic="True">Manuel J. Prieto</mobile:Label>
```

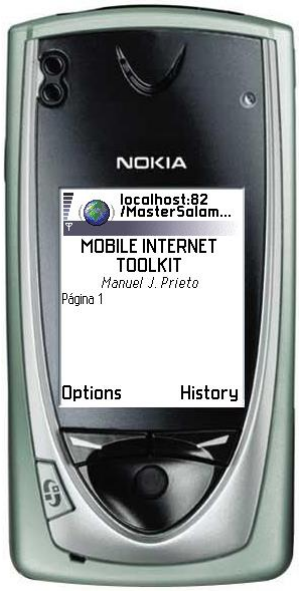
Una vez que tenemos este control definido, podremos utilizarlo dentro de cualquier página, insertándolo dentro de un formulario. Por cierto, en el ejemplo anterior se puede comprobar que los controles de usuario no necesitan contener un formulario, como si ocurre en las páginas estándar. El siguiente ejemplo muestra un ejemplo de utilización del control de usuario definido en el ejemplo anterior.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="UsoControlUsuario.aspx.cs" Inherits="MasterSalamanca.UsoControlUsuario" AutoEventWireup="false" %>

<%@ Register TagPrefix="uc1" TagName="UserControl1" Src="UserControl1.ascx" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <uc1:UserControl1 id="UserControl11" runat="server"></uc1:UserControl1>

                <mobile:Label id="Label1" runat="server">P&#225;gina 1</mobile:Label>

        </mobile:Form>

</body>
```

El resultado final será el siguiente.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241



8.2 CONTROLES PROPIOS

Los controles propios (*custom controls*), son codificados directamente como clases. Esta codificación se realizará en cualquiera de los lenguajes de programación soportados por ASP.NET y será compilada. Esta forma de creación de controles es más complicada que la vista en el punto anterior, lo que hemos denominado controles de usuario, pero también es significativamente más potente.

Para la creación de controles propios, podremos partir de los controles propios del MIT, o crear el control partiendo desde cero. Evidentemente es mucho más sencillo y apropiado, en la mayoría de los casos, partir de un control ya creado y aprovechar así parte de su lógica y su comportamiento. En concreto, las técnicas por las que podemos optar para crear nuestros controles es el siguiente:

Herencia – En este caso nuestro control extenderá un control ya existente. Para realizar esto, evidentemente tendremos que extender la clase que define el control del que vamos a partir. Al extender la clase en cuestión, lo que haremos será añadir nuestros propios atributos, métodos o cualquier otra cosa que consideremos necesario, o modificar alguno de los asistentes.

Composición – En este caso lo que haremos será crear un control propio, basado en la combinación de varios controles ya existentes.

Composición dependiente de dispositivo – Este tipo de controles se compondrán de manera diferente, según el dispositivo que realice la petición. Por ejemplo, un control puede ser compuesto con una serie de imágenes para un determinado grupo de dispositivos, o puede ser compuesto como una lista de elementos textuales en el caso de que el grupo de dispositivos sea otro.

Partiendo de cero – En este caso, nuestro control será creado a partir directamente de la clase *System.Web.UI.MobileControl*, y proveerá de forma completa el propio control todos los comportamientos y funcionalidades.

Evidentemente, las dos primeras técnicas expuestas son mucho más sencillas que las dos últimas.

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

8.2.1 USO DE LA HERENCIA

Esta forma de creación de controles, basada en la extensión o modificación de controles ya existentes, es muy útil y sencilla. Sobre todo, cabe destacar que el proceso de *renderización* o creación del contenido final que será enviado al terminal, ya está implementado y podemos utilizarlo de forma totalmente transparente.

En el siguiente ejemplo, vamos a crear un control que extiendo un control de tipo lista estándar. El nuevo control, en nuestro caso, presentará la capacidad de ordenar el listado de ítems en función de algún criterio concreto. Este ejemplo no se muestra de forma completa, pero sería un control de tipo lista con un comportamiento de personalización al usuario embebido. El código correspondiente a la clase que extiende la lista es el siguiente.

```
namespace Shadow
{
	using System;
	using System.Data;
	using System.Drawing;
	using System.Web;
	using System.Web.Mobile;
	using System.Web.UI.MobileControls;
	using System.Web.UI.WebControls;
	using System.Web.UI.HtmlControls;
	public class MobileWebUserControl5 : System.Web.UI.MobileControls.List
	{
		public MobileWebUserControl5()
		{
		}
		private void Page_Load(object sender, System.EventArgs e)
		{
			// Cojo el userId inicializado por el constructor de "GeneralPage" y construyo
			// la página
			// Cambio el orden de los elementos
			// Borro la lista anterior y añado la nueva
		}
	}
}
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

En el ejemplo anterior no se modifican los elementos, pero se presenta la arquitectura, de tal forma que el objetivo es que el lector comprenda como funcionaría un posible control que extienda al control *List* que ya conocemos.

8.2.2 CONTROL BASADO EN COMPOSICIÓN

Como ya se ha indicado, los controles de composición se crean a partir de la unión o composición de uno o más controles. En este caso, de forma similar a lo que ocurre con la herencia, son los controles ya existentes los encargados de crear el contenido que se envía al terminal en último término.
Cuando se crea un control de composición, generalmente es apropiado partir o heredar de un control de tipo *Panel*. Esto es debido a que el control *Panel* es tratado de forma especial por el entorno de ejecución del MIT. Por ejemplo, el uso de un control *Panel* como control base, nos asegura que el nuevo control no va a ser dividido en varias páginas durante el proceso de ejecución.
El resultado del uso de este tipo de controles, es similar a los controles de usuario. En este caso, lo que se hace es incluir los controles de forma programática. El siguiente ejemplo muestra un ejemplo sencillo de cómo sería el método principal en el proceso de creación de un control compuesto.

```
protected override void CreateChildControls()
{
        texto1 = new Label();
        texto2 = new TextBox();
        texto1.Text = "Su nombre";
        texto2.Text = "Escriba aquí";
        Controls.Add(texto1);
        Controls.Add(texto2);

        ChildControlsCreated = true;
}
```

8.2.3 CREACIÓN AVANZADA DE CONTROLES

Cuando un control necesita un resultado muy concreto, es decir, debemos controlar de forma muy cercana el proceso de *rederizado* del control. Este proceso se realiza dentro del método *Render*, y por lo tanto será en este punto dónde deberemos trabajar.
En el caso de los dispositivos móviles, como se ha indicado en repetidas ocasiones a lo largo de este documento, el proceso de *renderizado* es especialmente delicado, ya que deberemos adaptarnos lo mejor posible al terminal con el que se está trabajando en cada momento.
El entorno de ejecución del MIT provee lo que se denomina adaptadores al dispositivo, que nos permiten trabajar con varios grupos de terminales o dispositivos. Cuando se realiza una petición por parte de un determinado terminal, los adaptadores al dispositivo aplicados a cada control, son seleccionados

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

por parte del entorno de ejecución, de tal forma que sean los más adecuados para el terminal en cuestión. Los adaptadores de dispositivo, implementan una serie de métodos y propiedades, en los que el control relegará las funciones que necesitan de una gestión dependiente del dispositivo.
Para configurar nuestros propios adaptadores, tendremos que configurar dichos adaptadores para el control en cuestión, dentro del fichero *Web.config*. Este fichero configura totalmente la aplicación, entre otros puntos, los adaptadores que trabajarán sobre los controles.

9 ADAPTACIÓN AL DISPOSITIVO

Se ha hablado en varias ocasiones a lo largo de este documento sobre el proceso de adaptación al dispositivo. A parte de la adaptación automática que realiza el propio entorno, podremos configurar lo que denominaremos filtros de dispositivo para nuestra aplicación. Los filtros de dispositivo se definen dentro del fichero *Web.config* y estos filtros serán utilizados para seleccionar el contenido más adecuado para cada terminal. Un terminal puede encajar dentro de varios filtros. Los filtros se definen dentro de la sección *deviceFilters* del fichero *Web.config*. El siguiente listado muestra la definición de una serie de filtros.

```
<system.web>
  <deviceFilters>
    <filter name="IsColor" compare="IsColor" argument="true" />
    <filter name="IsPocketIE" compare="Browser" argument="Pocket IE" />
    <filter name="IsHTML" compare="PreferredRenderingMIME"
       argument="text/html" />
  </deviceFilters>
</system.web>
```

Dentro de la definición de un filtro, el nombre identifica el mismo y este nombre será el utilizado dentro de los formularios para hacer referencia a dicho filtro. El resto de atributos indican los datos necesarios para hacer la comprobación de aplicabilidad del filtro. Las características válidas para comprobar si un filtro es aplicable o no, están definidas dentro de la clase *System.Web.UI.Mobile.Controls.MobileCapabilities*.
Para definir un contenido específico para un determinado grupo de dispositivos, podemos utilizar la etiqueta *<DeviceSpecific>*. El siguiente ejemplo muestra cómo indicar un contenido específico, según el terminal desde el que se realiza la petición.

```
<mobile:Label runat="server" >
  <DeviceSpecific>
    <Choice Filter="IsPocketIE" Text="Corriendo en Pocket IE" />
    <Choice Filter="IsHTML" Text="Corriendo en un terminal HTML" />
    <Choice Text="Corriendo en un terminal genérico" />
  </DeviceSpecific>
```

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
<mobile:Label>
```

Cuando se realiza la petición de la página, el marco de trabajo comprueba cada una de las etiquetas *Choice*, en el orden en el que están definidos dentro del formulario. Si el filtro indicado corresponde con la petición, el filtro será seleccionado y será aplicado. En el ejemplo anterior, podemos comprobar cómo el último *Choice* no tiene filtro, por lo que será seleccionada como elemento por defecto. Es decir, en caso de que ninguno de los otros filtros sea aplicable, se aplicará este filtro. Este filtro por defecto es opcional.
Como queda plasmado en el ejemplo anterior, cuando se selecciona un filtro, lo que se hace es establecer el valor del atributo *Text* del control. Esta operación se denomina sobreescritura de propiedades. Esta técnica nos permite cambiar las características del control en función del terminal que hace la petición. Así, podemos cambiar el texto, tal y como acabamos de ver, podemos cambiar una imagen o incluso podríamos cambiar la apariencia del control, variando el valor del estilo del control.

9.1 USO DE PLANTILLAS

Con el objetivo de poder realizar personalización de una forma más sencilla dentro de los controles, las páginas compuestas por formularios, proveen una funcionalidad que permite el uso de plantillas. Las plantillas se pueden utilizar sobre los siguientes controles:

- Form
- Panel
- List
- ObjectList

Dentro de las plantillas aplicables a los formularios, tenemos la capacidad de modificar la cabecera y el pie del formulario. Estas operaciones se realizarán a través de las etiquetas *HeaderTemplate* y *FooterTemplate*. También tenemos la posibilidad de utilizar la etiqueta *ScriptTemplate*, de tal forma que insertaremos contenido inmediatamente después de la etiqueta *head* en HTML o *card* en WML. El siguiente ejemplo muestra un ejemplo de cómo sería el uso de plantillas con el fin antes descrito. Hay que destacar, que en este caso no se tiene en cuenta el dispositivo que hace la petición.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="PlantillasEj.aspx.cs" Inherits="MasterSalamanca.PlantillasEj1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">
```


ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

```
            <mobile:TextView id="TextView1" runat="server">Este es el texto de la página</mobile:TextView>
            <mobile:DeviceSpecific id="DeviceSpecific1" runat="server">
                <Choice>
                    <HeaderTemplate>
                        <mobile:Label id="Label1" runat="server">Cabecera de la p&#225;gina</mobile:Label>
                    </HeaderTemplate>
                    <FooterTemplate>
                        <mobile:Label id="Label2" runat="server">Pie de la p&#225;gina</mobile:Label>
                    </FooterTemplate>
                </Choice>
            </mobile:DeviceSpecific>
        </mobile:Form>
</body>
```

El resultado sería el siguiente en el terminal.

Podemos ampliar un poco más el ejemplo anterior, de tal manera que tengamos en consideración desde qué terminal se realiza la petición a la hora de crear el contenido. En el siguiente ejemplo se muestra esta funcionalidad, de tal forma que se cambia la cabecera en el caso de que el terminal corresponda con un determinado grupo de terminales.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="Sys-
tem.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>
<%@ Page language="c#" Codebehind="PlantillaEj2.aspx.cs" Inherits="MasterSalamanca.PlantillaEj2" Au-
toEventWireup="false" %>
<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">
<meta name="CODE_LANGUAGE" content="C#">
<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">
<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">
        <mobile:Form id="Form1" runat="server">
                <mobile:DeviceSpecific id="DeviceSpecific1" runat="server">
                        <Choice Filter="isPocketIE" Xmlns="http://schemas.microsoft.com/mo-
bile/html32template">
                                <HeaderTemplate>
                                        Esto es un POCKET IE
                                </HeaderTemplate>
                        </Choice>
                        <Choice Xmlns="http://schemas.microsoft.com/mobile/html32template">
                                <HeaderTemplate>
                                        <mobile:Label id="Label1" runat="server">Cabecera
gen&#233;rica</mobile:Label>
                                </HeaderTemplate>
                                <FooterTemplate>
                                        <mobile:Label id="Label2" runat="server">Pie
gen&#233;rico</mobile:Label>
                                </FooterTemplate>
                        </Choice>
                </mobile:DeviceSpecific>
        </mobile:Form>
</body>
```

Como hemos indicado cuando enumerábamos los controles que aceptaban plantillas, estas también pueden aplicarse a los controles *List* y *ControlList*. Estas plantillas se utilizarán normalmente para mejorar la presentación en algunos terminales. En estos casos, además de la cabecera y el pie de la lista, también podemos modificar el modo en que se mostrarán los elementos de la lista. Así, si estamos trabajando con un terminal que soporta HTML, podremos mostrar la lista utilizando una tabla. En la cabecera de la lista escribiremos las etiquetas correspondientes a la creación de la tabla, cada

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241

ítem podrá ser una fila de la tabla y para cerrar la tabla al final, usaremos el elemento de la plantilla que nos permite controlar el pie.
Trabajar con plantillas en el caso de los controles tipo *Panel*, lo que nos permite es insertar bloques de lenguaje de marcado final, dentro de la aplicación. Esto mismo se puede hacer usando los elementos de la plantilla del formulario, pero en el caso del control *Panel*, el elemento *ContentTemplate* reemplaza cualquier otro control o contenido que estuviera dentro del *Panel*.

10 GESTIÓN DEL ESTADO DE LA APLICACIÓN

Para mantener el estado de la aplicación, podemos usar cualquiera de las técnicas disponibles dentro de ASP.NET. En este documento veremos alguno de ellos. Por ejemplo, los formularios WEB disponibles dentro de ASP.NET son capaces de mantener su propio estado durante las diferentes peticiones del cliente. Cuando una propiedad adquiere un valor, bien programáticamente o bien mediante una acción del usuario, dicho valor es guardado automáticamente como parte del estado del control.
Normalmente, esta técnica se basa en el envío al cliente de una variable oculta que es retornada al servidor como parte de la respuesta. Si bien esto es el funcionamiento habitual dentro de ASP.NET, en el caso del trabajo con terminales móviles, el estado de la aplicación no se envía al cliente, debido entre otras cosas, a las limitaciones de ancho de banda que se sufren en las comunicaciones móviles. En lugar de esto, lo que hacen es salvar el estado como parte de la sesión del usuario dentro del servidor.
El siguiente ejemplo muestra de forma sencilla este tipo de funcionamiento. Tenemos dos etiquetas dentro de un formulario y vemos cómo el contenido de las etiquetas se mantiene entre peticiones.

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls" Assembly="System.Web.Mobile, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" %>

<%@ Page language="c#" Codebehind="GestionEstado1.aspx.cs" Inherits="MasterSalamanca.GestionEstado1" AutoEventWireup="false" %>

<meta name="GENERATOR" content="Microsoft Visual Studio 7.0">

<meta name="CODE_LANGUAGE" content="C#">

<meta name="vs_targetSchema" content="http://schemas.microsoft.com/Mobile/Page">

<body Xmlns:mobile="http://schemas.microsoft.com/Mobile/WebForm">

        <mobile:Form id="Form1" runat="server">

                <mobile:Label id="val" runat="server" Visible="False">1</mobile:Label>

                <mobile:Label id="Label1" runat="server">Label</mobile:Label>

                <mobile:Command id="Reload" runat="server" StyleReference="subcommand">Reload</mobile:Command>

        </mobile:Form>

</body>
```

El código que tenemos detrás de esta página para que funcione, es el que se muestra a continuación, si bien, se han eliminado las partes sin relevancia.

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

```
private void Page_Load(object sender, System.EventArgs e)
{
        int valor = int.Parse(val.Text);
        valor++;
        val.Text = "" + valor;
        Label1.Text = "Es la " + valor + " que ves esta página";

}
private void Reload_Click(object sender, System.EventArgs e)
{
        ActiveForm = Form1;
}
```

El resultado en ejecución es el siguiente.

Como hemos visto, el estado de la aplicación se mantiene en el servidor. Esto presenta un problema en el caso de que el usuario seleccione la funcionalidad de atrás o *back* del navegador. En este caso, es posible que se pierda la sincronización entre el servidor y el cliente.

Para solucionar este problema en la medida de lo posible, los formularios móviles mantienen una pequeña historia del estado, dentro de la sesión del usuario, tal y como ya hemos dicho. Cada identificador enviado al cliente, se corresponde con una determinada posición dentro de dicha historia. Así, con el identificador de la página, el servidor es capaz de sincronizar la historia con la página activa en cada momento. El tamaño de esta historia es configurable por el desarrollador. El tamaño por

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

defecto es seis y este valor se puede modificar en el fichero *web.config*, modificando el valor del atributo *sessionStateHistorySize*.
Debido a que el estado de la aplicación es salvado dentro de la sesión, es posible que dicho estado expire o caduque. Cuando hay problemas para cargar la información correspondiente al estado de la sesión, se llama al evento *OnViewStateExpire*. En este lugar podemos reconstruir el estado o dejar la implementación por defecto, que lanzará una excepción. Por ejemplo, imaginemos que tenemos un control de tipo lista en el que los elementos son creados al comienzo. En este caso, tendríamos el contenido de la lista gestionado por el entorno de ejecución y en el caso de que se llamara a *OnViewStateExpire*, podríamos volver a crear dinámicamente los elementos de la lista.
El método de gestión del estado de la aplicación que acabamos de ver, es muy sencillo, pero puede causar una cierta merma en el rendimiento del servidor, si el número de peticiones es elevado y la cantidad de información que contienen los controles también es muy alta. Para desactivar la gestión del estado de la aplicación en un control y en todos sus hijos, simplemente tenemos que establecer la variable *EnableViewState* de dicho control a *false*.
Por defecto, la gestión de la sesión en ASP.NET necesita la escritura por parte del servidor de una *cookie* en el cliente. El problema se presenta cuando nos enfrentamos a algunos dispositivos móviles, que no tiene soporte para *cookies*. En este caso, para que la aplicación sea capaz de trabajar correctamente con estos terminales, deberemos configurar la aplicación para que mantenga el estado de la sesión sin utilizar *cookies*. Cuando una aplicación trabaja sin *cookies*, automáticamente inserta una clave o identificador de sesión dentro de las diferentes URLs de la aplicación.

11 GESTIÓN DE LAS CAPACIDADES DEL DISPOSITIVO

A estas alturas no debe haber dudas sobre la orientación del MIT a cubrir las necesidades de un abanico muy amplio de terminales. Dentro de este amplio abanico, tenemos terminales con capacidades muy diversas en lo que se refiere a la resolución de pantalla, el número de colores o el tipo de elementos de interacción con el usuario disponibles.
Para permitir al desarrollador el trabajo con estas diferentes capacidades, el MIT incluye un componente que nos permite acceder y gestionar dichas capacidades. Este componente identifica el dispositivo desde el que se hace la petición, recupera toda la información posible sobre dicho dispositivo y pone a disposición del desarrollador dicha información. Esto se hace a través del objeto *MobileCapabilities*.
Entre otras informaciones, el objeto *MobileCapabilities* nos proporciona información sobre el modelo del terminal que hace la petición. El siguiente ejemplo muestra cómo podríamos recuperar dicho modelo.

```
MobileCapabilities capabilities =
   (MobileCapabilities)Request.Browser;
String deviceModel = capabilities.MobileDeviceModel;
```

12 ANEXO A – REFERENCIA DE LOS CONTROLES DEL MIT

12.1 CONTROL Form

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
Method	Método usado para enviar el formulario al servidor.	Post Get
PagerStyle	El estilo usado para las propiedades de paginación del control	
Paginate	Indica si los contenidos deben ser divididos en páginas.	True False
StyleReference	Estilo que usa el control.	error subcommand title
Títle	Título usado para identificar el formulario	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

12.2 CONTROL Panel

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Paginate	Indica si los contenidos deben ser divididos en páginas.	True False
StyleReference	Estilo que usa el control.	error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.3 CONTROL Label

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
StyleReference	Estilo que usa el control.	error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.4 CONTROL TextBox

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
MaxLength	El máximo número de caracteres que se puede introducir	True False
Numeric	Indica si la entrada se restringe a datos numéricos	

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Password	Indica si el texto debe ser ocultado	True False
Size	El tamaño esperado de texto introducido	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.5 CONTROL TextView

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.6 CONTROL Command

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
CausesValidation	Indica si se debe activar la validación en el formulario cuando se active	True False
CommandArgument	El argumento asociado con el comando	
CommandName	El nombre asociado con el comando	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
Format	Indica la apariencia visual del control	Button Link
ImageUrl	La url de la imagen a mostrar	
SoftKeyLabel	Etiqueta usada para el comando cuando se muestra como un *tecla*. Sólo se aplica a dispositivos que usen teclas para los comandos	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.7 CONTROL Link

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
NavigateUrl	El URL o el formulario al que enlaza	
SoftKeyLabel	Etiqueta usada para el comando cuando se muestra como un *tecla*. Sólo se aplica a dispositivos que usen teclas para los comandos	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.8 CONTROL PhoneCall

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
AlternateFormat	Formato del texto que aparecerá para los dispositivos que no soportan llamadas	
AlternateURL	La URL o formulario de destino para dispositivos que no soportan llamadas	
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
PhoneNumber	Número de teléfono a llamar para teléfonos que soporten las llamadas	
SoftKeyLabel	Etiqueta usada para el comando cuando se muestra como un *tecla*. Sólo se aplica a dispositivos que usen teclas para los comandos	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.9 CONTROL Image

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
AlternateText	El texto alternativa a mostrar en el caso de que no se pueda mostrar la imagen	
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ImageURL	La URL de la imagen	
NavigateURL	La URL o el form al que navegar	
SoftKeyLabel	Etiqueta usada para el comando cuando se muestra como un *tecla*. Sólo se aplica a dispositivos que usen teclas para los comandos	
StyleReference	Estilo que usa el control.	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.10 CONTROL List

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
DataMember	Tabla usada como fuente cuando un DataSet es usado como fuente de datos	
DataSource	DataSource del que se obtienen los datos de la lista	
DataTextField	Especifica qué propiedad del item de datos se usa para determinar el texto del item en la lista	
DataValueField	Especifica qué propiedad del item de datos se usa para determinar el valor del item en la lista	
Decoration	Formato de la lista	Bulleted Numbered
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ItemCount	Número de items a mostrar en el control. El valor 0 indica *paginación automática*	
Items	Colección de elementos de la lista	
ItemAsLink	Indica si los elementos de la lista deben ser interpretados como hiperenlaces	True False
ItemsPerPage	Indica el número de items a mostrar en cada página del control. El valor 0 indica *paginación automática*	
StyleReference	Estilo que usa el control.	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.11 CONTROL SelectionList

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
DataMember	Tabla usada como fuente cuando un DataSet es usado como fuente de datos	
DataSource	DataSource del que se obtienen los datos de la lista	
DataTextField	Especifica qué propiedad del item de datos se usa para determinar el texto del item en la lista	
DataValueField	Especifica qué propiedad del item de datos se usa para determinar el valor del item en la lista	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ItemCount	Número de items a mostrar en el control. El valor 0 indica *paginación automática*	
Items	Colección de elementos de la lista	
Rows	El número de líneas visibles a mostrar	
SelectType	El tipo de selección de la lista	DropDown ListBox Radio MultiSelectListBox Checkbox
StyleReference	Estilo que usa el control.	Error subcommand title
Title	El título mostrado sobre el área de entrada de datos en los dispositivos que lo soporten	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.12 CONTROL ObjectList

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
AutoGenerateFields	Indica si los campos son generados automáticamente en tiempo de ejecución, en bsae a una fuente de datos asocidad	True False
BackColor	Color de fondo del control	
BackCommandText	Texto para el enlace de retorno en la vista de detalles	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
Commands	Colección de comandos para cada elemento de la listas	
DataMember	Tabla usada como fuente cuando un DataSet es usado como fuente de datos	
DataSource	DataSource del que se obtienen los datos de la lista	
DefaultCommand	Comando lanzado cuando un elemento es seleccionado	
DetailsCommandText	Texto para el enlace a la vista de detalles	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Fields	Colección de campos que componen la vista de detalles	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ItemCount	Número de items a mostrar en el control. El valor 0 indica *paginación automática*	
ItemsPerPage	Número de items a mostrar en cada página del control. El valor 0 indica *paginación por defecto*	
LabelField	Campo de la fuente de datos a usar como representación compacta de los elementos de la lista	
LabelStyle	El estilo aplicado a la etiqueta de cabecera	
MoreText	Texto para el enlace a *más detalles y comandos*	

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

StyleReference	Estilo que usa el control.	Error subcommand title
TableFields	Campos de la fuente de datos a usar en una representación extensa de los elementos de la lista	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.13 CONTROL StyleSheet

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
ReferencePath	Camino relativo al fichero de usuario que contiene el control StyleSheet	
TemplateStyle	El estilo usado como plantilla	

12.14 CONTROL Calendar

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
CalendarEntryText	Texto para el enlace que permite la selección de una fecha. El enlace es sólo usado cuando se necesitan varios pasos	
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
FirstDayOfWeek	El día de la semana que aparece primero	Sunday Monday...
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
SelectedDate	La fecha seleccionada en cada momento	
SelectionMode	Indica si se seleccionan dias, semanas o meses	Day DayWeek DayWeekMonth

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

ShowDayHeader	Será *true* si muestra la cabecera de días de la semana	True False
StyleReference	Estilo que usa el control.	Error subcommand title
TableFields	Campos de la fuente de datos a usar en una representación extensa de los elementos de la lista	
Visible	Indica si el control está visible y se ha procesado.	True False
VisibleDate	El mes que aparece	
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.15 CONTROL AdRotator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
AdvertisementFile	Fichero XML que contiene los anuncios	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
ImageKey	Nombre del elemento XML que especifica la URL de la imagen a descargar	
KeywordFilter	Palabra clave para limitar la selección de anuncios	
NavigateURLKey	Nombre del elemento XML que indica la URL de la página web que recuperar	
StyleReference	Estilo que usa el control.	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

12.16 CONTROL RequiredFieldValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
InitialValue	Valor inicial del campo a validar	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.17 CONTROL CompareValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToCompare	Id del control con el que comparar	
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
Operator	Operación de comparación a aplicar	Equal Not equal GreaterThan GreaterThanEqual LessThan LessThanEqual DataTypeCheck
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Type	Tipo de datos de los valores a comparar	String Integer Double Date Currency
ValueToCompare	Valor con el que comparar	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.18 CONTROL RangeValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
MaximumValue	Valor máximo del control a validar	
MinimumValue	Valor mínimo del control a validar	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Type	Tipo de datos de los valores a comparar	String Integer Double Date Currency
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.19 CONTROL RegularExpressionValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
ValidationExpression	Expresión regular que determinad la validez	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la “ruptura en líneas”	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

12.20 CONTROL CustomValidator

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
ControlToValidate	Id del control a validar	
Display	Apariencia del validador	Static Dynamic
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
ErrorMessage	Mensaje a mostrar cuando falla la validación	
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
StyleReference	Estilo que usa el control.	Error subcommand title
Text	Texto que aparece en el control	
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

12.21 CONTROL ValidationSummary

Propiedad	Descripción	Valores posibles
ID	Identificador del control	
Aligment	Alineamiento horizontal del control.	Not Set Left Center Rigth
BackColor	Color de fondo del control	
BackLabel	Etiqueta del enlace al formulario que causo el error	
BreakAfter	Indica si se debe añadir un salto de línea después del control en tiempo de ejecución	True False
EnableViewState	Indica si el control guarda automáticamente su estado para utilizarlo en acciones de ida y vuelta.	True False
Font	Fuente que usa el control	
ForeColor	Color del texto dentro del control	
FormToValidate	Identificador del formulario a validar	
HeaderText	Texto que aparece en la cabecera	
StyleReference	Estilo que usa el control.	Error subcommand title
Visible	Indica si el control está visible y se ha procesado.	True False
Wrapping	El comportamiento con respecto a la "ruptura en líneas"	Not Set Wrap No Wrap

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

References

1. Bogdan Okresa Durik. (2017) Organisational Metamodel for Large-Scale Multi-Agent Systems: First Steps Towards Modelling Organisation Dynamics. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
2. Jörg Bremer, Sebastian Lehnhoff. (2017) Decentralized Coalition Formation with Agent-based Combinatorial Heuristics. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
3. Rafael Cauê Cardoso, Rafael Heitor Bordini. (2017) A Multi-Agent Extension of a Hierarchical Task Network Planning Formalism. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
4. Enyo Gonçalves, Mariela Cortés, Marcos De Oliveira, Nécio Veras, Mário Falcão, Jaelson Castro (2017). An Analysis of Software Agents, Environments and Applications School: Retrospective, Relevance, and Trends. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
5. Eduardo Porto Teixeira, Eder M. N. Goncalves, Diana F. Adamatti (2017). Ulises: A Agent-Based System For Timbre Classification. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
6. Casado-Vara, R., Chamoso, P., De la Prieta, F., Prieto J., & Corchado J.M. (2019). Non-linear adaptive closed-loop control system for improved efficiency in IoT-blockchain management. Information Fusion.
7. González-Briones, A., Chamoso, P., Yoe, H., & Corchado, J. M. (2018). GreenVMAS: virtual organization-based platform for heating greenhouses using waste energy from power plants. Sensors, 18(3), 861.
8. Casado-Vara, R., Novais, P., Gil, A. B., Prieto, J., & Corchado, J. M. (2019). Distributed continuous-time fault estimation control for multiple devices in IoT networks. *IEEE Access*.
9. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado J.M. (2019). Social computing in currency exchange. Knowledge and Information Systems.
10. Casado-Vara, R., Prieto-Castrillo, F., & Corchado, J. M. (2018). A game theory approach for cooperative control to improve data quality and false data detection in WSN. *International Journal of Robust and Nonlinear Control*, *28*(16), 5087-5102.
11. Chamoso, P., Rodríguez, S., de la Prieta, F., & Bajo, J. (2018). Classification of retinal vessels using a collaborative agent-based architecture. AI Communications, (Preprint), 1-18.
12. Chamoso, P., González-Briones, A., Rodríguez, S., & Corchado, J. M. (2018). Tendencies of technologies and platforms in smart cities: A state-of-the-art review. Wireless Communications and Mobile Computing, 2018.
13. Gonzalez-Briones, A., Prieto, J., De La Prieta, F., Herrera-Viedma, E., & Corchado, J. M. (2018). Energy Optimization Using a Case-Based Reasoning Strategy. Sensors (Basel), 18(3), 865-865. doi:10.3390/s18030865
14. Gonzalez-Briones, A., Chamoso, P., De La Prieta, F., Demazeau, Y., & Corchado, J. M. (2018). Agreement Technologies for Energy Optimization at Home. Sensors (Basel), 18(5), 1633-1633. doi:10.3390/s18051633
15. Chamoso, P., González-Briones, A., Rivas, A., De La Prieta, F., & Corchado, J. M. (2019). Social computing in currency exchange. Knowledge and Information Systems, 1-21.
16. Casado-Vara, R., Vale, Z., Prieto, J., & Corchado, J. (2018). Fault-tolerant temperature control algorithm for IoT networks in smart buildings. Energies, 11(12), 3430.
17. Choon, Y. W., Mohamad, M. S., Deris, S., Illias, R. M., Chong, C. K., Chai, L. E., … Corchado, J. M. (2014). Differential bees flux balance analysis with OptKnock for in silico microbial strains optimization. *PLoS ONE*, *9*(7). https://doi.org/10.1371/journal.pone.0102744
18. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). A particle dyeing approach for track continuity for the SMC-PHD filter. In *FUSION 2014 - 17th International Conference on Information Fusion*. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637583&partnerID=40&md5=709eb4815eaf544ce01a2c21aa749d8f
19. García Coria, J. A., Castellanos-Garzón, J. A., & Corchado, J. M. (2014). Intelligent business processes composition based on multi-agent systems. *Expert Systems with Applications*, *41*(4 PART 1), 1189–1205. https://doi.org/10.1016/j.eswa.2013.08.003
20. Lima, A. C. E. S., De Castro, L. N., & Corchado, J. M. (2015). A polarity analysis framework for Twitter messages. *Applied Mathematics and Computation*, *270*, 756–767. https://doi.org/10.1016/j.amc.2015.08.059

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241

21. Li, T., Sun, S., Corchado, J. M., & Siyau, M. F. (2014). Random finite set-based Bayesian filters using magnitude-adaptive target birth intensity. In FUSION 2014 - 17th International Conference on Information Fusion. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-84910637788&partnerID=40&md5=bd8602d6146b014266cf07dc35a681e0
22. Casado-Vara, R., de la Prieta, F., Prieto, J., & Corchado, J. M. (2018, November). Blockchain framework for IoT data quality via edge computing. In Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems (pp. 19-24). ACM.
23. Costa, Â., Novais, P., Corchado, J. M., & Neves, J. (2012). Increased performance and better patient attendance in an hospital with the use of smart agendas. *Logic Journal of the IGPL*, *20*(4), 689–698. https://doi.org/10.1093/jigpal/jzr021
24. Rodríguez, S., De La Prieta, F., Tapia, D. I., & Corchado, J. M. (2010). Agents and computer vision for processing stereoscopic images. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 6077 LNAI). https://doi.org/10.1007/978-3-642-13803-4_12
25. Rodríguez, S., Gil, O., De La Prieta, F., Zato, C., Corchado, J. M., Vega, P., & Francisco, M. (2010). People detection and stereoscopic analysis using MAS. In *INES 2010 - 14th International Conference on Intelligent Engineering Systems, Proceedings*. https://doi.org/10.1109/INES.2010.5483855
26. Baruque, B., Corchado, E., Mata, A., & Corchado, J. M. (2010). A forecasting solution to the oil spill problem based on a hybrid intelligent system. *Information Sciences*, *180*(10), 2029–2043. https://doi.org/10.1016/j.ins.2009.12.032
27. Tapia, D. I., & Corchado, J. M. (2009). An ambient intelligence based multi-agent system for alzheimer health care. *International Journal of Ambient Computing and Intelligence*, *v 1*, *n 1*(1), 15–26. https://doi.org/10.4018/jaci.2009010102
28. Mata, A., & Corchado, J. M. (2009). Forecasting the probability of finding oil slicks using a CBR system. *Expert Systems with Applications*, *36*(4), 8239–8246. https://doi.org/10.1016/j.eswa.2008.10.003
29. Glez-Peña, D., Díaz, F., Hernández, J. M., Corchado, J. M., & Fdez-Riverola, F. (2009). geneCBR: A translational tool for multiple-microarray analysis and integrative information retrieval for aiding diagnosis in cancer research. *BMC Bioinformatics*, *10*. https://doi.org/10.1186/1471-2105-10-187
30. Fernández-Riverola, F., Díaz, F., & Corchado, J. M. (2007). Reducing the memory size of a Fuzzy case-based reasoning system applying rough set techniques. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *37*(1), 138–146. https://doi.org/10.1109/TSMCC.2006.876058
31. Méndez, J. R., Fdez-Riverola, F., Díaz, F., Iglesias, E. L., & Corchado, J. M. (2006). A comparative performance study of feature selection methods for the anti-spam filtering domain. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4065 LNAI*, 106–120. Retrieved from https://www.scopus.com/inward/record.uri?eid=2-s2.0-33746435792&partnerID=40&md5=25345ac884f61c182680241828d448c5
32. Di Mascio, T., Vittorini, P., Gennari, R., Melonio, A., De La Prieta, F., & Alrifai, M. (2012, July). The Learners' User Classes in the TERENCE Adaptive Learning System. In 2012 IEEE 12th International Conference on Advanced Learning Technologies (pp. 572-576). IEEE.
33. Chamoso, P., Rivas, A., Martín-Limorti, J. J., & Rodríguez, S. (2018). A Hash Based Image Matching Algorithm for Social Networks. In *Advances in Intelligent Systems and Computing* (Vol. 619, pp. 183–190). https://doi.org/10.1007/978-3-319-61578-3_18
34. Sittón, I., & Rodríguez, S. (2017). Pattern Extraction for the Design of Predictive Models in Industry 4.0. In *International Conference on Practical Applications of Agents and Multi-Agent Systems* (pp. 258–261).
35. García, O., Chamoso, P., Prieto, J., Rodríguez, S., & De La Prieta, F. (2017). A serious game to reduce consumption in smart buildings. In *Communications in Computer and Information Science* (Vol. 722, pp. 481–493). https://doi.org/10.1007/978-3-319-60285-1_41
36. Palomino, C. G., Nunes, C. S., Silveira, R. A., González, S. R., & Nakayama, M. K. (2017). Adaptive agent-based environment model to enable the teacher to create an adaptive class. Advances in Intelligent Systems and Computing (Vol. 617). https://doi.org/10.1007/978-3-319-60819-8_3

ISBN: 978-84-9012-861-9
Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc

37. Canizes, B., Pinto, T., Soares, J., Vale, Z., Chamoso, P., & Santos, D. (2017). Smart City: A GECAD-BISITE Energy Management Case Study. In *15th International Conference on Practical Applications of Agents and Multi-Agent Systems PAAMS 2017, Trends in Cyber-Physical Multi-Agent Systems* (Vol. 2, pp. 92–100). https://doi.org/10.1007/978-3-319-61578-3_9
38. Chamoso, P., de La Prieta, F., Eibenstein, A., Santos-Santos, D., Tizio, A., & Vittorini, P. (2017). A device supporting the self-management of tinnitus. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 10209 LNCS, pp. 399–410). https://doi.org/10.1007/978-3-319-56154-7_36
39. Tiago Pinto, Luis Marques, Tiago M Sousa, Isabel Praça, Zita Vale, Samuel L Abreu. (2017) Data-Mining-based filtering to support Solar Forecasting Methodologies. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
40. Casado-Vara, R., & Corchado, J. (2019). Distributed e-health wide-world accounting ledger via blockchain. *Journal of Intelligent & Fuzzy Systems*, *36*(3), 2381-2386.
41. M.ª Belén Aige (2017). The online tourist fraud: the new measures of technological investigation in Spain. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 2
42. Silvia Susana Toscano (2017). Freedom of Expression, Right to Information, Personal Data and the Internet in the view of the Inter-American System of Human Rights. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1
43. Víctor Corcoba Magaña, Mario Muñoz Organero, Juan Antonio Álvarez-García, Jorge Yago Fernández Rodríguez. (2017) Design of a Speed Assistant to Minimize the Driver Stress. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
44. Miguel Oliver, José Pascual Molina, Antonio Fernández-Caballero, Pascual González. (2017) Collaborative Computer-Assisted Cognitive Rehabilitation System. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 3
45. Miki Ueno, Toshinori Suenaga, Hitoshi Isahara (2017). Classification of Two Comic Books based on Convolutional Neural Networks. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1
46. Eduardo Munera, Jose-Luis Poza-Lujan, Juan-Luis Posadas-Yagüe, Jose-Enrique Simó-Ten, Francisco Blanes (2017). Integrating Smart Resources in ROS-based systems to distribute services. ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal (ISSN: 2255-2863), Salamanca, v. 6, n. 1

ISBN: 978-84-9012-861-9

Colección Aquilafuente: AQ-241
Ediciones Universidad de Salamanca - cc by nc dc