Seminario de titulación

Modulo:

Aplicaciones para Internet en .NET

Instructor:

Ing. Joel A. González Estrada
Maestría en Tecnologías de la Información

Temario

Introducción al ASP

Microsoft Active Server Pages (ASP) es un lenguaje para entornos de servidor con el que puede crear páginas dinámicas e interactivas.

Con el ASP puedes combinar paginas HTML, Script y componentes COM para crear páginas web dinámicas y aplicaciones web de forma rápida.

Un fichero asp puede contener etiquetas HTML y Scripts ASP. El Script de Asp comienza con <% y termina con %>. Todo lo que va entre medias se interpreta en el servidor.

¿Que es ASP .NET ?

ASP.NET es un ambiente de programación construido sobre el entorno NGWS (New Generation Windows Services, o sea, "Servicios de la Nueva Generación de Windows"), que permite crear poderosas aplicaciones de Internet.

ASP.NET ofrece varias ventajas importantes sobre los modelos previos de desarrollo para Internet :

- **Mejor Eficiencia**

  ASP.NET corre código *compilado* sobre el entorno NGWS en el servidor. Distinto a sus predecesores interpretados, ASP.NET usa amarres tempranos ("early binding"), así como compilación justo a tiempo ("just-in-time compilation"), optimización nativa, y servicios de caché, sin configuración adicional. Para los desarrolladores, esto significa eficiencia dramáticamente superior *antes de escribir la primera línea de código*.

- **Herramientas superiores de desarrollo**

  ASP.NET tiene una "caja de herramientas" rica : el ambiente de desarrollo integrado de Visual Studio.NET. La edición WYSIWYG, la creación de controles mediante "drag-and-drop", y la publicación automática son varias ventajas.

- **Poder y Flexibilidad**

  Porque ASP.NET está basado en el Entorno Común de Ejecución de Lenguajes (Common Language Runtime, o "CLR"), el poder y la flexibilidad de la plataforma completa está disponible para los desarrolladores. Las librerías de Clases del CLR, la Mensajería, y las soluciones de Acceso a Datos, son accesibles al través del Internet. ASP.NET permite el uso de una gran variedad de lenguajes de programación y, por tanto, usted puede escoger el mejor lenguaje

para su aplicación, o particionar su aplicación en varios lenguajes. Mas aún, la interoperabilidad del CLR garantiza que su inversión en el desarrollo de aplicaciones COM es preservada cuando se migra a ASP.NET.

- **Simplicidad**

ASP.NET hace fácil el ejecutar tareas comunes, desde el simple envío de un formulario o la autenticación de un cliente, hasta el despliegue y la configuración de un Web. Por ejemplo, el entorno de paginado de ASP.NET le permite construir interfases de usuario que separan limpiamente la lógica de su aplicación del código de su presentación, y maneja eventos con un modelo sencillo de procesamiento de formularios al estilo de Visual Basic. Adicionalmente, el CLR simplifica el desarrollo con servicios de código gerenciado, como el conteo automático de referencias y la limpieza automática de la memoria utilizada por su aplicación.

- **Gerenciabilidad**

ASP.NET usa un sistema jerárquico de configuración, basado en archivos de texto, que simplifica la aplicación de parámetros de configuración al servidor y sus aplicaciones. Porque la información de configuración es almacenada como texto, nuevos parámetros pueden ser configurados sin recurrir a herramientas de administración locales. Esta filosofía de "cero administración local" también se extiende al despliegue de aplicaciones de ASP.NET. Una aplicación de ASP.NET se despliega a un servidor simplemente copiando los archivos necesarios al servidor. No hay que reiniciar el servidor, ni siquiera para reemplazar código compilado que ya está en servicio.

- **Escalabilidad y Disponibilidad**

ASP.NET ha sido diseñado para la escalabilidad con características específicamente dirigidas a mejorar el funcionamiento de servidores racimados (clustered) y de servidores con procesadores múltiples. Los procesos del servidor son vigilados y gerenciados por el entorno del ambiente de ejecución de ASP.NET, así que si algun proceso se entorpece o se detiene, un nuevo proceso puede ser creado para reemplazarlo, lo cual ayuda a mantener la disponibilidad de su aplicación para manejar solicitudes de servicio.

- **Personalización y Extensibilidad**

ASP.NET entrega una arquitectura bien formada que permite que los desarrolladores "enchufen" su código al nivel apropiado. De hecho, es posible el extender o reemplazar cualquier sub-componente del ambiente de ejecución de ASP.NET con un componente personalizado. La implementación de autenticación personalizada o de servicios de mantenimiento de estado nunca ha sido tan sencillo.

- **Seguridad**

Con autenticación nativa de Windows y configuración individual por aplicación, usted puede estar tranquilo: sus aplicaciones están seguras.

Para poder crear y probar ficheros ASP necesitaras dos programas: un editor y un servidor.

Servidores ASP

El ASP es una tecnología de MicroSoft y para poder trabajar un hacer ejemplos necesitaras tener instalado uno de estos dos programas:

- IIS (Internet Information Server)
- PWS (Personal Web Server)

Editores ASP

Además necesitarás algún editor, te recomendamos el Dreamweaver de Macromedia, aunque también te valdrá un simple Notepad, Med, Kawa, etc.

- Dreamweaver MX
- Notepad de Windows

Elección de Servidores Web ASP

Es posible desarrollar aplicaciones ASP en casi cualquier entorno o sistema operativo, sin embargo nuestra recomendación es hacerlo sobre un sistema operativo de MicroSoft.

Windows 95, 98

Si tienes alguno de los sistemas operativos anteriores Windows 95 o 98 puedes instalarte el PWS (personal web server) que encontrarás en la copia del Sistema Operativo.

El PSW es una buena opción para hacer prácticas y crear tus primeros sitios web.

Windows 2000 Professional

El sistema windows 2000 Professional incorpora una versión reducida del Internet Information Server que encontrarás en tu copia del sistema operativo.

Si ya estas familiarizado con el PWS tambien puedes utilizarlo aunque la seguridad y estabilidad de IIS es mucho mejor.

Windows XP Home Edition

Windows XP Home Edition no incluye ni soporta las versiones (1.0, 2.0, 4.0) de Microsoft Personal Web Server (PWS).

Los usuarios que necesiten servidor web en su ordenador deberá utilizar Windows XP Professional.

Windows XP Professional

Windows XP Professional ha sido diseñado para usuarios de negocio e incluye Internet Information Services (IIS) version 5.1. IIS 5.1 incluye servidores web and FTP server, también incluye Microsoft FrontPage transactions, Active Server Pages, and conexiones a bases de datos.

Todo ello esta incluido en un componente opcional que puede instalarse desde el CD de Windows XP Professional.

El IIS 5.1 se instalará automáticamente si esta actualizando un ordenador de una versión anterior de windows que tenga instalador el PWS.

IIS (Internet Information Server) para Windows 2000 Professional

IIS es un programa servidor de páginas web, el IIS soporta ficheros ASP y lo necesitarás para visualizar las páginas ASP que crees.

Encontrarás el programa IIS en tu copia del sistema operativo de Windows 2000, si ya lo tienes instalado deberia poder ver el icono de "Servicios de Internet IIS" al entrar en Panel de Control > Herramientas administrativas y no necesitarás realizar ninguna instalación.

Instalación de IIS (Internet Information Server)

Introduce tu CD de Windows 2000 Professional. **Con Windows Home Edition No Funcionará.**

Desde Panel de Control > Agregar y quitar Programas y ahora selecciones Agregar componentes de Windows, selecciones IIS e instálelo.

Probar IIS para Windows 2000

Para probar el correcto funcionamiento de tu servidor web, abre un navegador y teclea en la dirección "http://localhost", si visualizas una pagina web entonces ya lo tienes funcionando.

Otros enlaces sobre IIS para Windows 2000

Si quieres información paso a paso para la instalación puedes consultar uno de estos dos artículos que detallan como instarlo.

IIS (Internet Information Server) para Windows XP Professional

IIS es un programa servidor de páginas web, el IIS soporta ficheros ASP y lo necesitarás para visualizar las páginas ASP que crees.

Encontrarás el programa IIS en tu copia del sistema operativo de Windows XP, si ya lo tienes instalado deberia poder ver el icono de "Servicios de Internet IIS" al entrar en Panel de Control > Herramientas administrativas y no necesitarás realizar ninguna instalación.

Instalación de XP (Internet Information Server)

Introduce tu CD de Windows XP Professional. **Con Windows XP Home Edition No Funcionará.**

Desde Panel de Control > Agregar y quitar Programas y ahora selecciones Agregar componentes de Windows, selecciones IIS e instalelo.

Probar IIS para Windows XP

Para probar el correcto funcionamiento de tu servidor web, abre un navegador y teclea en la dirección "http://localhost", si visualizas una pagina web entonces ya lo tienes funcionando.

Otros enlaces sobre IIS para Windows XP

Si quieres información paso a paso para la instalación puedes consultar uno de estos dos artículos que detallan como instarlo.

Ejemplo 1: Hola Mundo en ASP NET

Escribe un texto dinámicamente con ASP Net

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="utf-8"
%>
<script runat="server">
Sub Page_Load
p1.InnerHtml = "<b>Hola Mundo</b> "
End Sub
</script>
<html>
<body>
```

```
<form runat="server">
<p id="p1" runat="server" />
</form>
</body>
</html>
```

Ejemplo 2: Leer Formulario y escribir respuesta

Lee un texto escrito en un formulario y escribe la respuesta en la página web.

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="utf-8"
%>
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
p1.InnerHtml = "<b>Usted escribio:</b> " & textarea1.Value
End Sub
</script>
<html>
<body>
<form runat="server">
Introduzca un texto:<br />
<textarea id="textarea1" cols="35" rows="6" runat="server" />
<input type="submit" value="Submit" OnServerClick="submit" runat="server" />
<p id="p1" runat="server" />       
</form>
</body>
</html>
```

Ejemplo 3: Concatenar cadenas de texto

Combinar dos cadenas de texto, nombre y apellido y escribirlos en una página web.

Código: Concatenar cadenas de texto en ASP NET

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="utf-8"
%>
<script runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
Dim nombre, apellido as String
nombre = "Felipe"
apellido ="Borbon"
t1.Text= nombre & apellido
```

```
End Sub
</script><html>
<head>
<title>Documento sin t&iacute;tulo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<asp:label ID="t1" runat="server"></asp:label>
</body>
</html>
```

Ejemplo 4: Sumar, Restar, Multiplicar y Dividir Números

Aprende a manejar los números y a operar con numeros en ASP.

Código: Sumar, Restar, Multiplicar y Dividir en ASP NET

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="utf-8"
%>
<script runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
Dim numeroA as Double = 90.90
Dim numeroB as Double = 3.30
Dim suma, resta, multiplicar, dividir as Double
Dim sumaS, restaS, multiplicarS, dividirS as String

suma = numeroA + numeroB
sumaS = String.Format("{0:c}", suma)
sumaId.Text = sumaS

resta = numeroA - numeroB
restaS = String.Format("{0:c}", resta)
restaId.Text = restaS

multiplicar = numeroA * numeroB
multiplicarS = String.Format("{0:c}", multiplicar)
multiplicarId.Text = multiplicarS

dividir = numeroA / numeroB
dividirS = String.Format("{0:c}", dividir)
dividirId.Text = dividirS

End Sub
</script><html>
<head>
<title>Operar y Formatear Numeros en ASP Net</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

```
</head>
<body>
Suma: <asp:label ID="sumaId" runat="server"></asp:label><br>
Resta: <asp:label ID="restaId" runat="server"></asp:label><br>
Multiplicar: <asp:label ID="multiplicarId" runat="server"></asp:label><br>
Dividir: <asp:label ID="dividirId" runat="server"></asp:label><br>
</body>
</html>
```

Comentarios sobre Manipulación de números

Los operadores para la suma, resta, multiplicación y división son +, -, * y / respectivamente.

Puedes tratar de hacer tus propias formulas como 2 veces el numero A menos una vez el B que sería

Observa lo sencillo que es operar con cifras en asp, prueba a cambiar los valores de los números. ¿ te sale algún decimal? ¿Necesitas formatear los números para limitar el número de decimales? ¿tienes problemas con la división por 0?

Orden de preferencia para operadores numéricos y uso de paréntesis.

También puedes utilizar paréntesis para aislar o crear formulas matemáticas.

```
<% resultado1 = 2*numeroA - numeroB %>

<% resultado2 = (2*numeroA) - numeroB %>

<% resultado3= 2* (numeroA - numeroB) %>
```

¿Sabes cual de estos dos son iguales? haz la prueba y descúbrelo por ti mismo. Un dato teórico que te puede ayudar es que el orden de preferencia del operador * es mayor que el -

Números y operadores Mayor, Menor e Igual

Otros operadores que puedes utilizar con los números son > < e = que al comparar números entre si te generan valores de verdadero o falso.

Formatear Numeros en ASP Net con String.Format

El método String.Format permite el formateo de números, al transformar un número a cadenas de caracteres podemos definir el símbolo decimal, el numero de decimales y si deseamos que algún símbolo de moneda sea introducido.

Así pues si queremos conseguir 95.34 pondremos #.## y si queremos que solo sea un decimal 95.3 pondremos #.# en el parámetro de String.Format.

En el código puedes ver diferentes usos para **dar formato a números en ASP.NET**

Ejemplo 5: Formatos de Fechas en ASP NET

Aprende a manejar las fechas y a darlas cualquier formato.

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="utf-8"
Debug="true" %>
<%@ Import Namespace="System.Web"%>
<%@ Import namespace="System.Globalization" %>
<script runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
Dim fechaS as String
Dim miFecha as DateTime = DateTime.Now()
fechaS = String.Format("{0:dd/MM/yyyy}", miFecha)
fechaId.Text = fechaS
End Sub
</script><html>
<head>
<title>Formato de Fechas en Asp NET</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
Fecha con formato de hoy en dia, mes y año: <asp:label ID="fechaId"
runat="server"></asp:label><br>
</body>
</html>
```

Ejemplo 6: Sumar dos números introducidos en un formulario

La página creada para explicar esta técnica solicita dos números al visitante. cuando se reenvía la pagina se suman estos numeros.La pagina incluye dos procedimientos definidos por el usuario a los que se llaman desde dos rutinas de tratamientos de eventos.Uno de los procedimientos es una Function (Función), dado que devuelve un valor. El otro procedimientos no devuelve ningún valor dado que es una subrutina (Sub)

Así es como se quedaría montada la estructura de la pagina una vez seguido los pasos y comprendiendo cada símbolo.

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="iso-
8859-1" %>
<script runat="server">
Sub MostrarInstrucciones (Mode as String)
If Mode ="Inicial" Then
alvi.Text = "introduza los dos numeros que" & "desee sumar."
Else
alvi.Text = "El resultado se muestra" & "debajo"
End If
End Sub
Sub Page_Load(ByVal Sender as Object , Byval E as EventArgs)
If IsPostBack then
MostrarInstrucciones("Final")
else
MostrarInstrucciones("Inicial")
End If
End Sub

Function AddNums (Num1 as Single, Num2 as Single) as Single
AddNums=Num1+Num2
End Function

sub ApretarelBoton(Sender As Object, E As EventArgs)
alvi.text="Resultado: " & AddNumS (numeros1.Text, numeros2.Text)
End Sub
</script>


<html>
<head>
<title>Documento sin t&iacute;tulo</title>
</head>
<body>
<form runat="server">
<asp:label ID="alvi" runat="server"/>
<asp:textbox ID="numeros2" runat="server"/>
<asp:textbox ID="numeros1" runat="server" />
<asp:button ID="Botonok" Text="sumar" OnClick="ApretarelBoton"
runat="server"/>
</form>
</body>
</html>
```

Ejemplo 7: Ejemplo de un control Span basico

Si utiliza los controles del espacio de nombres HTMLControls, problamente se encuentre con la necesidad de mostrar texto en la página Asp.Net que se pueda

manipular a través de codigo. Esto se puede conseguir definiendo controles Span. Esta tecnica muestra como hacerlo.

Código: Span Basico

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="iso-
8859-1" %>
<script runat="server">
Sub Page_Load(ByVal Sender as Object, Byval E as EventArgs)
span2.InnerText = "<B> Este texto no aparece & _ en Negrita.</B>"
End sub
</script>
<html>
<head>
<title>Documento sin t&iacute;tulo</title>

</head>
<body>

<span id="Span1" runat="server"><b>Este texto aparece en negrita</b>
</span>
<br>
<br>
<span id="Span2" runat="server">
</span>
<br>
<br>
<span id="Span3" runat="server" disabled="true">Este texto aparece inhabilitado
</span>
<br>
<br>
<span id="Spand4" runat="Server" visible="false" > este texto no aparece
</span>
</body>
</html>
```

Ejemplo 8: Presentación de una imagen a través del control HTMLImage

Un control HTMLImage proporciona el mecanismo necesario para mostrar una etiqueta HTMLImage simple que se puede manipular mediante código. Esto resulta útil cuando se debe controlar mediante código la imagen mostrada a los visitantes.

Código: HTMLimage

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="iso-
8859-1" %>
<script runat="server">
Sub Page_Load(ByVal Sender as Object, Byval E as EventArgs)
End sub
</script>
<html>
<head>
<title>Documento sin t&iacute;tulo</title>

</head>
<body>

<img id="image1" runat="server" src="../../NBASALTO.2jpg.jpg">
<br>
<img id="image" runat="server" src="../../NBASALTO.2jpg.jpg" alt="Imagen de
un boton" border="4">
<br>
<img id="image3" runat="server" src="../../NBASALTO.2jpg.jpg" width="300"
height="300">
</body>
</html>
```

Ejemplo 9: Utilización del control HTMLinputCheckBox

La página creada para describir esta técnica salicita a los visitantes que respondan dos preguntas de tipo sí/no a través de controlesHTMLInputCheckBox.cuando los visitantes envíen la página, verán un texto basado en sus respuestas.

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="iso-
8859-1" %>
<script runat="server">


Sub SubmitBtn_Click(Source as Object, E as EventArgs)


mimensage.InnerHTML = ""
If (chekeoeldibujo.Checked = True ) Then
mimensage.InnerHTML = " Ha seleccionado el " & "dibujo. <br>"

End If
```

```
If chekeodecontacto.Checked =True Then
mimensage1.InnerHTML = " mimensage1 " & "Esta en nuestra lista" & "de
contactos. <br>"

end If
end sub

</script>
<html>
<head>
<title>Documento sin t&iacute;tulo</title>
</head>
<body bgcolor="#FFFFFF">
<form runat="server">
<input type="checkbox" id="chekeoeldibujo" runat="server">

¿obtener un dibujo gratuito?
<input id="chekeodecontacto" runat="server" type="checkbox" checkbox>
¿ desea q se añadan sus datos a la lista de contactos?
<button id="Button1" runat="server"
onserverClick="SubmitBtn_Click">Enviar</button>
<span ID="mimensage" runat="server"></span>
<span ID="mimensage1" runat="server"></span>

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

Ejemplo 10: Utilización del control HTMLSelect

El control HTML Select proporciona un método para ofrecer un elemento a los visitantes en forma de lista.

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="iso-
8859-1" %>
<script runat="server">
Sub SubmitBtn_Click(Source as Object, E as EventArgs)
MiMensage.InnerHTML = "Ha seleccionado como coches Favoritos" &
coches.Value
End sub
</script>
<html>
<head>
<title>Documento sin t&iacute;tulo</title>
</head>
<body bgcolor="#FFFFFF">
<form runat="server">
```

```
<span id="MiMensage" runat="server"></span>
<select id="coches" runat="server" multiple="False" size="1">
<option>Familiares</option>
<option>Deportivos</option>
<option>todoTerrenos</option></select>
<button id="boton1" runat="server"
onserverclick="SubmitBtn_click">Enviar</button>

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

Ejemplo 11: Inserción de filas y celdas en un control HTMLTable a través de código

Además de definir las filas y celdas de un control HTMLTable directamente obre el formulario de la página, se puede añadir filas y celdas directamente desde este código.

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="iso-
8859-1" %>
<script runat="server">
Sub Page_Load(ByVal Sender as Object, E as EventArgs)
Dim MIFila as new HTMLTableRow
Dim MiCelda as new HTMLTableCell
Dim i as integer
Dim j as integer
Tabla1.BGcolor="Ivory"
Tabla1.Border=2
Tabla1.BorderColor="LawGreen"
Tabla1.CellPadding=4
Tabla1.CellSpacing=3
Tabla1.Align="Center"
MiCelda.InnerText = "Columna 1"
MiFila.Cells.Add(MiCelda)
MiCelda =New HTMLTableCell
MiCelda.InnerText =" Columna 2"
MiFila.Cells.Add(MiCelda)
Tabla1.Rows.Add(MiFila)
For i =2 to 6
```

```
MiFila = New HTMLTableRow
For j=1 to 2
Next
MiCelda = New HTMLTableCell
MiCelda.InnerText="Celda "& i & " ," & j
MiFila.Cells.Add(MiCelda)
Next
End Sub
</script>
<html>
<head>
<title>Documento sin t&iacute;tulo</title>
</head>
<body bgcolor="#FFFFFF">
<form runat="server">
<table id="Tabla1" runat="server"></table>
<table id="TableCell" runat="server"></table>
</form>
</body>
</html>
```

Ejemplo 12: Visualialización de un control CheckBox básico

Proporciona un mecanismo para recibir respuestas de tipo verdadero y falso o si o no de los visitantes.

Código: Visualización de un control CheckBox básico

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="iso-
8859-1" %>
<script runat="server">
Sub SubmitBtn_Click(Sender As Object, E As Eventargs)
If chkMailingList.Checked = "True" Then
Response.Write("<script Language=""JavaScript!"" >" &Chr(13) & "<!-" & chr(13)
& "alert(´se han añadido sus datos a nuestra " & "-><" &"/" & "SCRIPT>")
End If
End Sub
</script>
<html>
<head>
<title>Documento sin t&iacute;tulo</title>
</head>
<body>
<form runat="server">
<br><br>
<asp:checkbox ID="chkMailingList" Text="¿Le gustaria que se las llevasemosa
casa la comida que compre?" Checked="true" BackColor="#FFFF00"
```

```
BorderColor="#FF0000" BorderWidth="3" Font-Size="12pt" Font-Name="Comic
Sans MS" TextAlign="Left" runat="server"/>
<br><br>
<asp:checkbox ID="chkContact" Text="¿ Le gusta la comida del super?"
Checked="false" BackColor="#FFFF00" BorderWidth="3" Font-Size="12pt" Font-
Name="Comic Sans MS" TextAlign="Right" runat="server"/>
<br><br>
<asp:button ID="butOK" Text="Aceptar" type="Submit"
OnClick="SubmitBtn_Click" runat="server"/>
</form>
</body>
</html>
```

Ejemplo 13 Enviar un Email con ASP NET

A continuación presentamos el ejemplo más sencillo de como enviar un email con asp net.

Debes cambiar lo que esta en rojo para que funcione. Pon la IP del servidor sustituyendo a localhost.

```
<%@ Import Namespace="System.Web.Mail"%>
<script runat="server">
Sub enviaEmail(Src As Object, E As EventArgs)
Dim Mensaje as New MailMessage
Dim MailConexion as SmtpMail
Mensaje.From = "emaildesde@email.com"
Mensaje.To = "emailhacia@email.com"
Mensaje.Subject = "TituloDelMensaje"
Mensaje.Body = "Textos del mesaje:" & nombre.Text & Chr(13) & Chr(10)
Mensaje.Priority = MailPriority.High
SmtpMail.SmtpServer = "localhost"
Try
MailConexion.Send(Mensaje)
Response.Write("Tu mensaje ha sido enviado con exito")
```

```
Catch exc as Exception
Response.Write("Send failure: " + exc.ToString())
End Try

End Sub
</script>
<html>
<head>
<title>Formato de Fechas en Asp NET</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<form runat="server">
<asp:textbox ID="nombre" MaxLength="100" runat="server" />
<asp:button ID="boton" Text="Enviar" runat="server" OnClick="enviaEmail"/>
</form>
</body>
</html>
```

Código para el Head

```
<%@ Import Namespace="System.Web.Mail"%>

<script runat="server">
Sub enviaEmail(Src As Object, E As EventArgs)
Dim Mensaje as New MailMessage
Dim MailConexion as SmtpMail
Mensaje.From = "aquitu@email.com"
Mensaje.To = "aquieldesdedestino@email.com"
Mensaje.Subject = "TituloDelMensaje"
Mensaje.Body = "Textos del mesaje:" & nombre.Text & Chr(13) & Chr(10)
Mensaje.Priority = MailPriority.High
SmtpMail.SmtpServer = "ip.del.servidor.de.email"
Try
MailConexion.Send(Mensaje)
Response.Write("Tu mensaje ha sido enviado con exito")

Catch exc as Exception
Response.Write("Send failure: " + exc.ToString())
End Try

End Sub
</script>
```

Codigo para el body

```
<form runat="server">
<asp:textbox ID="nombre" MaxLength="100" runat="server" />
<asp:button ID="boton" Text="Enviar" runat="server" OnClick="enviaEmail"/>
</form>
```

Comentarios sobre configuraciones

No es necesario a veces poner IP puedes eliminar la línea SmtpMail.SmtpServer = "ip.del.servidor.de.email" y normalmente seguirá funcionando aunque es posible que el mensaje pueda retrasarse.

Si deseas ampliar el texto del email puedes concatenar otras líneas poniendo &_ al final de cada linea de este modo.

```
"Este es linea 1 del mensaje" &_
"Esta es la linea 2" &_
"Observa que la ultima linea no lleva"
```

Utilización del objeto Webresponse y Webrequest

Lee el código de una página web y lo presenta en pantalla, puede también modificarse para que presente la información de la pagina web en cuestión.

```
<%@ Page Language="VB" ContentType="text/html" ResponseEncoding="utf-8"
%>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
Try
Dim objResponse As System.Net.WebResponse
Dim objRequest As System.Net.WebRequest
DIM theurlresult as string
DIM theurl as string = "http://www.terra.es/"
objRequest = System.Net.HttpWebRequest.Create(theurl)
objResponse = objRequest.GetResponse()
Dim sr As new system.io.StreamReader(objResponse.GetResponseStream())
theURLresult=server.HTMLencode(sr.ReadToEnd())
```

```
Page.Controls.Add(new LiteralControl(theURLresult))
Catch ex As Exception
Page.Controls.Add(new LiteralControl(ex.Message))
End Try
End Sub
</script>
<html><head>
<title>Webresponse y Webrequest</title>
</head>
<body bgcolor="#FFFFFF">
<h3><font face="Verdana">Codigo de la pagina www.terra.es</font></h3>

</body></html>
```

Acceso a datos

Otra área importante en los preparativos para llevar a cabo una migración es el acceso a datos. La aparición de ADO .NET proporciona una forma nueva y eficaz de obtener acceso a los datos. Dado que se podrían escribir muchas páginas sobre el acceso a datos, este tema se escapa de los objetivos de este artículo. En la mayoría de los casos, se puede continuar usando ADO como en el pasado, pero es muy recomendable echar un vistazo a ADO .NET a fin de mejorar los métodos de acceso a datos en la aplicación ASP .NET.

Preparativos para ASP .NET

Después de repasar los problemas que pueden aparecer con más frecuencia, quizá se pregunte qué soluciones se pueden aplicar hoy mismo para estar mejor preparado cuando llegue el momento de migrar a ASP .NET. Para facilitar este proceso, se pueden seguir una serie de acciones. Muchas de estas sugerencias mejorarán el código ASP incluso si piensa migrar a ASP. NET dentro de bastante tiempo.

Utilice Option Explicit

A pesar de que usar esta opción siempre ha sido una buena idea, no todo el mundo lo hace. La obligación de declarar las variables en ASP mediante **Option Explicit** permite al menos tener una idea precisa sobre dónde se ubican y cómo se utilizan las variables. Una vez que migre a ASP .NET, sugiero que utilice **Option Strict**. **Option Explicit** será la opción predeterminada en Visual Basic .NET, pero al utilizar la opción más estricta **Option Strict**, se asegurará de que todas las variables se declaran como el tipo de datos correcto. A pesar de que esta metodología requiere más trabajo, a largo plazo descubrirá que los resultados compensan el esfuerzo.

Evite utilizar propiedades predeterminadas

Tal y como se ha explicado, ya no se admiten las propiedades predeterminadas. De cualquier manera, obtener acceso a las propiedades de forma explícita tampoco resulta una tarea complicada. Así su código no sólo será más legible, sino que también ahorrará tiempo en futuras migraciones.

Utilice paréntesis y la palabra clave Call

Utilice paréntesis e instrucciones **Call** siempre que sea posible, tal y como se explicó anteriormente. En ASP .NET, es obligatorio utilizar paréntesis. Si comienza a utilizar la instrucción **Call** a partir de hoy, estará mejor preparado para el futuro.

Evite la anidación de archivos de inclusión

Aunque parezca una sugerencia difícil de seguir, evite en la medida de lo posible la anidación de sus archivos de inclusión. Lo que quiero decir con esto es que debería eliminar cualquier área en la que existan archivos de inclusión que contengan a su vez otros archivos de inclusión. Con el tiempo, lo que sucede es que su código termina dependiendo de una variable global que está definida en un archivo de inclusión en cualquier otra ubicación y sólo tiene acceso a él porque ha incluido otro archivo que contiene el que realmente necesita.

Al migrar a ASP .NET, probablemente traslade sus variables y rutinas globales a bibliotecas de clase, en cuyo caso resulta mucho más sencillo si tiene una idea clara de las ubicaciones que va a utilizar para tener acceso a cualquier elemento. Puede que se vea obligado a mover objetos y cambiar los nombres de algunas rutinas duplicadas en varios archivos.

Organice las funciones de la utilidad en archivos individuales

Una estrategia utilizada en el proceso de migración es migrar todas las funciones y código de la utilidad contenido en los archivos de inclusión del servidor a bibliotecas de Visual Basic o C#. Esto permite colocar todo el código en el sitio adecuado, es decir, en objetos, en contraposición a los archivos ASP de múltiples interpretaciones. Si prepara su código con antelación, ahorrará tiempo en el futuro. Lo ideal sería agrupar las subrutinas en archivos lógicos, lo que permitiría crear fácilmente un conjunto de clase de VB o C#. En realidad, éstas funciones son las que probablemente deberían haber estado en objetos COM desde el primer momento.

Si dispone de unas cuantas variables globales y constantes mezcladas en archivos de inclusión del lado del servidor, considere la posibilidad de colocarlos todos en

un único archivo. Una vez que haya migrado a ASP .NET, se puede crear fácilmente una clase que albergará todos los datos globales o constantes. De esta manera, el sistema estará mucho más limpio y será más fácil de mantener.

Elimine todo el código que pueda del contenido

Se trata también de un consejo difícil de seguir. No obstante, intente separar siempre que pueda el código del contenido HTML. Depure las funciones que mezclen código y archivos de comandos dentro del cuerpo de una función. De este modo, estará mejor preparado para volver a utilizar código oculto que, a fin de cuentas, es el modelo ideal en ASP .NET.

No declare funciones dentro de bloques <% %>

Esta posibilidad no se admite en ASP .NET. Por tanto, debe declarar las funciones dentro de los bloques <script>. Consulte la sección Cambios estructurales para ver un ejemplo de esta técnica.

Evite las funciones de procesamiento

Tal y como se ha expuesto anteriormente, debe evitar el uso de "funciones de procesamiento". Si puede modificar o preparar el código ahora, debería utilizar bloques **Response.Write** a la hora de construir este tipo de funciones.

Libere recursos de forma explícita (métodos de cierre de llamadas)

Asegúrese de llamar explícitamente cualquier método **close()** o de limpieza que exista en los objetos o recursos que esté utilizando. Todos sabemos lo descuidados que son Visual Basic y VBScript en los asuntos de limpieza. Normalmente son muy buenos en la limpieza inmediata de elementos, pero tras la migración a .NET y el mundo de la recolección de "basura", resulta difícil saber a ciencia cierta cuándo se limpiarán los objetos. Si puede limpiar y liberar recursos de forma explícita, debería hacerlo.

Evite mezclar lenguajes

En la medida de lo posible, evite mezclar JScript y VBScript del lado del servidor en la misma página. Por lo general, esta mezcla es característica de la programación de baja calidad. Además, plantea un problema de migración para ASP .NET, ya que requiere un solo lenguaje en línea <% %> por página debido al nuevo modelo de compilación. Se pueden seguir emitiendo secuencias de comandos del lado del cliente del mismo modo en que se hace ahora.

Resumen

Como se ha podido comprobar, hay que tener en cuenta varios puntos antes de efectuar la migración de una aplicación a ASP .NET. Sin embargo, casi todos los cambios que se han explicado en este artículo deberían implementarse con relativa facilidad.

Si tiene un sitio muy grande, al final del proceso se sorprenderá de la cantidad de errores y código inútil e ineficaz que ha descubierto y corregido. Además, será capaz de aprovechar las numerosas y eficaces características que se han integrado en ASP .NET y en la plataforma .NET en general.

¿En qué consiste una aplicación ASP.NET Framework?

ASP.NET define una aplicación como el conjunto de todos los archivos, páginas, controladores, módulos y código ejecutable que se pueden invocar o ejecutar dentro del ámbito de un determinado directorio virtual (y sus subdirectorios) en un servidor de aplicaciones Web. Por ejemplo, una aplicación de "pedido" podría publicarse dentro del directorio virtual "/pedido" de un servidor Web. Para IIS, el directorio virtual se puede configurar en el Administrador de servicios de Internet y contiene todos los subdirectorios, a menos que los propios subdirectorios sean directorios virtuales.

Cada aplicación ASP.NET Framework de un servidor Web se ejecuta dentro de un dominio único de aplicaciones ejecutables de .NET Framework, lo que garantiza el aislamiento de clases (no se producen conflictos de nombres o versiones), el uso seguro de recursos (se impide el acceso a determinados equipos o recursos de red) y el aislamiento de variables estáticas.

ASP.NET mantiene una agrupación de instancias **HttpApplication** durante el período de duración de una aplicación Web. ASP.NET asigna automáticamente una de estas instancias para procesar cada solicitud HTTP entrante recibida por la aplicación. La instancia **HttpApplication** asignada en particular es responsable del proceso de la solicitud a lo largo de todo su período de duración y sólo se puede volver a utilizar después de que la solicitud se haya completado. Esto significa que el código de usuario incluido en la instancia **HttpApplication** no necesita ser reentrante.

Crear una aplicación

Para crear una aplicación ASP.NET Framework, se puede utilizar un directorio virtual existente o crear uno nuevo. Por ejemplo, si Windows 2000 Server se instaló con IIS, probablemente existirá un directorio C:\\InetPub\\WWWRoot. IIS

se puede configurar mediante el Administrador de servicios de Internet, que se encuentra en Inicio -> Programas -> Herramientas administrativas. Haga clic con el botón secundario del *mouse* (ratón) en un directorio existente y elija Nuevo (para crear un nuevo directorio virtual) o Propiedades (para convertir un directorio normal existente).

Cuando se coloca una simple página .aspx, como la siguiente, en el directorio virtual, y se obtiene acceso a ella con el explorador, se desencadena el proceso de creación de la aplicación ASP.NET.

```
    <%@Page Language="VB"%>
<html>
<body>
<h1>hello world, <% Response.Write(DateTime.Now.ToString()) %></h1>
</body>
</html>
```

Ahora, ya se puede agregar el código apropiado para utilizar el objeto Application (por ejemplo, para almacenar objetos con ámbito de aplicación). Mediante la creación un archivo global.asax, también se pueden definir diversos controladores de eventos (por ejemplo, para el evento **Application_Start**).

Duración de una aplicación

Una aplicación ASP.NET Framework se crea la primera vez que se realiza una solicitud al servidor; antes de ello, no se ejecuta ningún código ASP.NET. Cuando se realiza la primera solicitud, se crea una agrupación de instancias **HttpApplication** y se provoca el evento **Application_Start**. Las instancias **HttpApplication** procesan esta solicitud y las siguientes hasta que la última instancia termina y se provoca el evento **Application_End**.

Observe que los métodos **Init** y **Dispose** de **HttpApplication** se invocan por cada instancia y, de este modo, pueden utilizarse varias veces entre **Application_Start** y **Application_End**. Sólo se comparten estos eventos entre todas las instancias de **HttpApplication** en una aplicación ASP.NET.

Nota sobre subprocesos múltiples

Si se utilizan objetos con ámbito de aplicación, se debería tener en cuenta que ASP.NET procesa las solicitudes de forma concurrente y que pueden ser varios los subprocesos que obtengan acceso al objeto **Application**. Por lo tanto, el siguiente código es peligroso y podría no producir el resultado esperado si diferentes clientes solicitan la página reiteradamente al mismo tiempo.

```
<%
```

```
Application("counter") = CType(Application("counter") + 1, Int32)
%>
```

Para conseguir que este código sea seguro durante la ejecución de subprocesos, hay que serializar el acceso al objeto **Application** mediante los métodos **Lock** y **UnLock**. Sin embargo, al hacerlo se produce una efecto de merma considerable en el rendimiento:

```
<%
Application.Lock()
Application("counter") = CType(Application("counter") + 1, Int32)
Application.UnLock()
%>
```

Otra solución consiste en hacer que el objeto almacenado con un ámbito de aplicación sea seguro para la ejecución de subprocesos. Por ejemplo, observe que las clases de colección del espacio de nombres **System.Collections** no son seguras para la ejecución de subprocesos por razones de rendimiento.

Resumen de la sección

1. Las aplicaciones ASP.NET Framework se componen de todo aquello que se encuentra bajo un directorio virtual del servidor Web.
2. Una aplicación ASP.NET Framework se crea agregando archivos a un directorio virtual del servidor Web.
3. La duración de una aplicación ASP.NET Framework viene marcada por los eventos **Application_Start** y **Application_End**.
4. El acceso a los objetos con ámbito de aplicación debe ser seguro para el acceso a múltiples subprocesos.

Archivo Global.asax

Además de escribir código para interfaces de usuario, los programadores también pueden agregar lógica del nivel de aplicación y código de control de eventos a sus aplicaciones Web. Este código no se encarga de generar interfaces de usuario y no se invoca normalmente en respuesta a solicitudes de páginas individuales. En vez de ello, se encarga de procesar eventos de la aplicación de nivel superior, tales como **Application_Start**, **Application_End**, **Session_Start**, **Session_End**, etc. Los programadores crean esta lógica mediante un archivo **Global.asax** ubicado en la raíz del árbol de directorios virtuales de una aplicación Web. ASP.NET analiza y compila automáticamente este archivo para producir una clase dinámica de .NET Framework, la cual extiende la clase base **HttpApplication** (la

primera vez que se activa o se solicita cualquier recurso o URL dentro del espacio de nombres de la aplicación).

ASP.NET analiza y compila dinámicamente el archivo Global.asax para producir una clase de .NET Framework la primera vez que se activa o se solicita cualquier recurso o URL dentro del espacio de nombres de la aplicación. El archivo Global.asax está configurado para rechazar automáticamente cualquier solicitud de URL directa de modo que los usuarios externos no puedan descargar o ver el código interno.

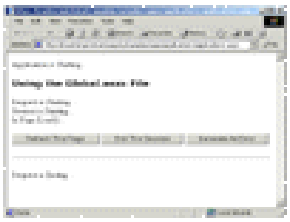
Eventos cuyo ámbito es una sesión o una aplicación Los programadores pueden definir controladores para eventos de la clase base **HttpApplication** creando métodos en el archivo Global.asax que se ajusten al modelo de nomenclatura "NombreDeEventoDeLaAplicación(FirmaDeArgumentosDelEvento)".
Por ejemplo:

```
<script language="VB" runat="server">Sub Application_Start(Sender As Object, E As EventArgs)
 ' Application startup code goes here
End Sub
</script>
```

Si el código de control de eventos necesita importar espacios de nombres adicionales, se puede utilizar la directiva **@ import** en una página .aspx, como se indica a continuación:

```
<%@ Import Namespace="System.Text" %>
```

El siguiente ejemplo ilustra el período de vida de **Application**, **Session** y **Request**.



VB Application1.aspx

[Ejecutar ejemplo] | [Ver código fuente]

La primera vez que se abre la página, se provoca el evento **Start** para la aplicación y la sesión:

```
Sub Application_Start(Sender As Object, E As EventArgs)
```

```
  ' Application startup code goes here
End Sub

Sub Session_Start(Sender As Object, E As EventArgs)
  Response.Write("Session is Starting...<br>")
  Session.Timeout = 1
End Sub
```

Los eventos **BeginRequest** y **EndRequest** se provocan en cada solicitud. Cuando la página se actualiza, sólo aparecen mensajes de **BeginRequest**, **EndRequest** y el método **Page_Load**. Observe que, al abandonar la sesión actual (hacer clic en el botón "Finalizar esta sesión"), se crea una nueva sesión y se provoca de nuevo el evento **Session_Start**.

Objetos cuyo ámbito es una sesión o una aplicación

Los objetos estáticos, las clases de .NET Framework y los componentes COM se pueden definir en el archivo Global.asax mediante la etiqueta object. El ámbito puede ser **appinstance**, **session** o **application**. El ámbito **appinstance** denota que el objeto es específico para una instancia de **HttpApplication** y no está compartido.

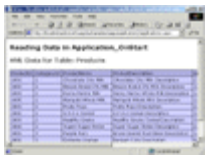
```
<object id="id" runat="server" class=".NET Framework class Name"
scope="appinstance"/>
<object id="id" runat="server" progid="COM ProgID" scope="session"/>
<object id="id" runat="server" classid="COM ClassID" scope="application"/>
```

Resumen de la sección

1. En las aplicaciones ASP.NET Framework se pueden definir controladores de eventos cuyo ámbito sea toda la aplicación o toda la sesión, en el archivo Global.asax.
2. En las aplicaciones ASP.NET Framework se pueden definir objetos cuyo ámbito sea toda la aplicación o toda la sesión, en el archivo Global.asax.

Utilizar el estado de las aplicaciones

En este ejemplo se ilustra el uso del estado de una aplicación para leer un conjunto de datos en **Application_Start**.



VB Application2.aspx

[Ejecutar ejemplo] | [Ver código fuente]

Ya que distintos subprocesos pueden tener acceso a una aplicación y a todos los objetos que contiene dicha aplicación de forma simultánea, es mejor almacenar con ámbito de aplicación únicamente los datos que se modifican con poca frecuencia. Idealmente, los objetos se inicializan en el evento **Application_Start** y los accesos posteriores son de sólo lectura.

En el siguiente ejemplo se lee un archivo en **Application_Start** (definido en el archivo Global.asax) y el contenido se almacena en un objeto **DataView** en el estado de la aplicación.

```
 Sub Application_Start()
  Dim ds As New DataSet()

  Dim fs As New
FileStream(Server.MapPath("schemadata.xml"),FileMode.Open,FileAccess.Read)
  Dim reader As New StreamReader(fs)
  ds.ReadXml(reader)
  fs.Close()

  Dim view As New DataView (ds.Tables(0))
  Application("Source") = view
End Sub
```

En el método **Page_Load** se recupera el objeto **DataView** y después se utiliza para llenar un objeto **DataGrid**:

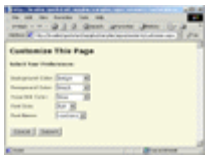
```
Sub Page_Load(sender As Object, e As EventArgs)
  Dim Source As New DataView  = CType(Application("Source"), DataView)
  ...
  MyDataGrid.DataSource = Source
  ...
End Sub
```

La ventaja de esta solución es que el precio de obtener los datos sólo lo paga la primera solicitud. Las siguientes solicitudes utilizarán el objeto **DataView**

existente. Como los datos no se modifican nunca después de la inicialización, no es necesaria la serialización del acceso.

Utilizar el estado de las sesiones

En el ejemplo siguiente se ilustra el uso del estado de una sesión para almacenar preferencias de usuario volátiles.



VB Session1.aspx

[Ejecutar ejemplo] | [Ver código fuente]

Es posible almacenar los datos con ámbito de sesión a fin de proporcionar datos individuales a un usuario durante una sesión. En el ejemplo siguiente se inicializan los valores de preferencias de usuario en el evento **Session_Start** del archivo Global.asax.

```
Sub Session_Start()
   Session("BackColor") = "beige"
   ...
End Sub
```

En la siguiente página de personalización, se modifican los valores de las preferencias de usuario en el controlador de eventos **Submit_Click** con la información aportada por el usuario.

```
Protected Sub Submit_Click(sender As Object, e As EventArgs)
   Session("BackColor") = BackColor.Value
   ...

   Response.Redirect(State("Referer").ToString())
End Sub
```

Los valores individuales se obtienen mediante el método **GetStyle**:

```
Protected GetStyle(key As String) As String
```

```
  Return(Session(key).ToString())
End Sub
```

El **método GetStyle** se utiliza para crear estilos específicos de una sesión:

```
<style>
  body
  {
   font: <%=GetStyle("FontSize")%> <%=GetStyle("FontName")%>;
   background-color: <%=GetStyle("BackColor")%>;
  }
  a
  {
    color: <%=GetStyle("LinkColor")%>
  }
</style>
```

Para comprobar que los valores se almacenan realmente con ámbito de sesión, hay que abrir dos veces la página del ejemplo, cambiar un valor en la primera ventana del explorador y actualizarlo en la segunda. La segunda ventana reflejará los cambios, ya que las dos instancias de explorador comparten un objeto **Session** común.

Configurar el estado de una sesión: es posible configurar las características del estado de una sesión en la sección **<sessionState>** de un archivo web.config. Para doblar el tiempo de espera predeterminado de 20 minutos, se puede agregar el código siguiente al archivo web.config de una aplicación:

De forma predeterminada, ASP.NET almacenará el estado de la sesión en el mismo proceso que atiende la solicitud, de la misma manera que ASP. Si no hay cookies disponibles, se puede hacer un seguimiento de una sesión agregando un identificador de sesión a la dirección URL. Se puede habilitar de la manera siguiente:

De forma predeterminada, ASP.NET almacenará el estado de la sesión en el mismo proceso que atiende la solicitud, de la misma manera que ASP. Además, ASP.NET puede almacenar datos de sesión en un proceso externo, que podría residir en otro equipo. Para habilitar esta función:

- Inicie el servicio de estado de ASP.NET mediante el complemento Servicios o ejecutando la instrucción "net start aspnet_state" desde la línea de comandos. De manera predeterminada, el servicio de estado escuchará en el puerto 42424. Para cambiar el puerto, modifique la clave del Registro para el servicio:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\aspnet_state\Parameters\Port

- Establezca el atributo **mode** de la sección **<sessionState>** en "StateServer".
- Configure el atributo **stateConnectionString** con los valores del equipo en que se inició aspnet_state.

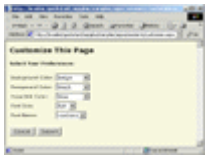
En el siguiente ejemplo se da por hecho que el servicio de estado se ejecuta en el mismo equipo que el servidor Web ("localhost") y utiliza el puerto predeterminado (42424):

```
<sessionState
  mode="StateServer"
  stateConnectionString="tcpip=localhost:42424"
/>
```

Hay que tener en cuenta que si se prueba el ejemplo anterior con este valor, es posible restablecer el servidor Web (escriba iisreset en la línea de comandos) y así se conservará el valor de estado de la sesión.

Utilizar las cookies del cliente

En el ejemplo siguiente se ilustra el uso del estado de las cookies del cliente para almacenar preferencias de usuario volátiles.



VB Cookies1.aspx

[Ejecutar ejemplo] | [Ver código fuente]

Almacenar cookies en el cliente es uno de los métodos que utiliza el estado de las sesiones de ASP.NET para asociar solicitudes a sesiones. También se pueden utilizar cookies directamente para conservar datos entre solicitudes, pero después se almacenan los datos en el cliente y se envían al servidor con cada solicitud. Los exploradores limitan el tamaño de las cookies; por tanto, sólo se garantiza la aceptación de un número máximo de 4096.

Cuando se almacenan los datos en el cliente, el método **Page_Load** del archivo cookies1.aspx comprueba si el cliente ha enviado una cookie. Si no se ha enviado ninguna, se crea una cookie nueva y después se inicializa y almacena en el cliente:

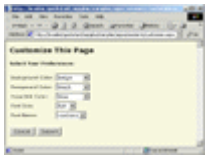
```
Protected Sub Page_Load(sender As Object, e As EventArgs)
```

```
    If Request.Cookies("preferences1") = Null Then
      Dim cookie As New HttpCookie("preferences1")
      cookie.Values.Add("ForeColor", "black")
      ...
      Response.AppendCookie(cookie)
    End If
End Sub
```

En la misma página, se utiliza de nuevo un método **GetStyle** con el fin de proporcionar los valores individuales almacenados en la cookie:

```
Protected Function GetStyle(key As String) As String
  Dim cookie As HttpCookie = Request.Cookies("preferences1")
  If cookie <> Null Then
    Select Case key
      Case "ForeColor"
        Return(cookie.Values("ForeColor"))
      Case ...
    End Select
  End If
  Return("")
End Function
```

Compruebe que el ejemplo funciona correctamente; para ello, abra la página cookies1.aspx y modifique las preferencias. Abra la página en otra ventana; esta ventana debería reflejar las nuevas preferencias. Cierre todas las ventanas del explorador y abra de nuevo la página cookies1.aspx. Esto debería eliminar la cookie temporal y restaurar los valores de preferencias predeterminados.



VB Cookies2.aspx

[Ejecutar ejemplo] | [Ver código fuente]

Para conservar una cookie entre sesiones es necesario establecer el valor de la propiedad **Expires** de la clase **HttpCookie** en una fecha futura. El siguiente fragmento de código de la página customization.aspx es idéntico al ejemplo anterior, con la excepción de la asignación a **Cookie.Expires**:

```
Protected Sub Submit_Click(sender As Object, e As EventArgs)
   Dim cookie As New HttpCookie("preferences2")
   cookie.Values.Add("ForeColor",ForeColor.Value)
   ...
   cookie.Expires = DateTime.MaxValue ' Never Expires

   Response.AppendCookie(cookie)

   Response.Redirect(State("Referer").ToString())
End Sub
```

Compruebe que el ejemplo funciona; para ello, modifique un valor, cierre todas las ventanas del explorador y abra de nuevo el archivo cookies2.aspx. La ventana debería mostrar aún el valor personalizado.

Utilizar ViewState

En este ejemplo se ilustra el uso de la propiedad **ViewState** para almacenar valores específicos de una solicitud.

VB PageState1.aspx

[Ejecutar ejemplo] | [Ver código fuente]

ASP.NET proporciona la noción de estado de la presentación de cada control en el servidor. Un control puede guardar su estado interno entre solicitudes mediante la propiedad **ViewState** de una instancia de la clase **StateBag**. La clase **StateBag** proporciona una interfaz de tipo diccionario para almacenar objetos asociados a una clave de tipo cadena.

El archivo pagestate1.aspx muestra un panel visible y almacena el índice del panel en el estado de la presentación de la página con la clave **PanelIndex**:

```
Protected Sub Next_Click(sender As Object, e As EventArgs)
   Dim PrevPanelId As String = "Panel" + ViewState("PanelIndex").ToString()
   ViewState("PanelIndex") = CType(ViewState("PanelIndex") + 1, Integer)
   Dim PanelId As String = "Panel" + ViewState("PanelIndex").ToString()
   ...
End Sub
```

Hay que tener en cuenta que si se abre la página en varias ventanas del explorador, cada ventana del explorador mostrará inicialmente el panel del nombre. Todas las ventanas permiten el desplazamiento entre paneles de forma independiente.

Resumen de la sección

1. Utilice variables de estado de la aplicación para almacenar datos que se modifican con poca frecuencia pero que se utilizan a menudo.
2. Utilice variables de estado de la sesión para almacenar datos específicos de una sesión o de un usuario. Los datos se almacenan completamente en el servidor. Hay que utilizar estas variables para datos de poca duración, datos importantes o grandes cantidades de datos.
3. Almacene pequeñas cantidades de datos volátiles en una cookie no persistente. Los datos se almacenan en el cliente, se envían al servidor en cada solicitud y caducan cuando finalice la ejecución en el cliente.
4. Almacene pequeñas cantidades de datos no volátiles en una cookie persistente. Los datos se almacenarán en el cliente hasta que caduquen y se enviarán al servidor en cada solicitud.
5. Almacene en el estado de la presentación pequeñas cantidades de datos específicos de una solicitud. Los datos se envían desde el servidor al cliente y viceversa.

Información general

ASP.NET proporciona una API de bajo nivel para solicitudes y respuestas que permite a los programadores utilizar clases de .NET Framework para ocuparse de las solicitudes HTTP entrantes. Para ello, los programadores crean clases que admiten la interfaz **System.Web.IHTTPHandler** e implementan el método **ProcessRequest()**. Los controladores suelen ser útiles cuando los servicios suministrados por la abstracción del marco de trabajo de página de alto nivel no son necesarios para procesar la solicitud HTTP. Entre los usos habituales de los controladores, se incluyen los filtros y las aplicaciones al estilo CGI, especialmente aquéllas que devuelven datos binarios.

Cada solicitud HTTP recibida por ASP.NET se procesa en último término mediante una instancia específica de una clase que implementa **IHTTPHandler**. **IHttpHandlerFactory** proporciona la infraestructura que controla la resolución real de las solicitudes URL en instancias **IHttpHandler**. Además de las clases **IHttpHandlerFactory** predeterminadas suministradas por ASP.NET, los programadores pueden opcionalmente crear y registrar generadores que admitan escenarios complejos de activación y resolución de solicitudes.

Configurar generadores y controladores HTTP

Los generadores y controladores HTTP se declaran en la configuración de ASP.NET como parte de un archivo web.config. ASP.NET define una sección de configuración **<httphandlers>** en la que se pueden agregar y quitar

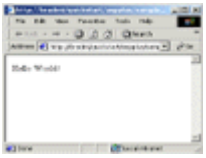
controladores y generadores. Los valores de configuración para **HttpHandlerFactory** y **HttpHandler** son heredados por los subdirectorios.

Por ejemplo, ASP.NET asigna todas las solicitudes de archivos .aspx a la clase **PageHandlerFactory** del archivo global machine.config:

```
<httphandlers>
  ...
  <add verb="*" path="*.aspx"
type="System.Web.UI.PageHandlerFactory,System.Web" />
  ...
</httphandlers>
```

Crear un controlador HTTP personalizado

El siguiente ejemplo crea un controlador **HttpHandler** personalizado que controla todas las solicitudes que se realizan a "SimpleHandler.aspx".



SimpleHandler de VB

[Ejecutar ejemplo] | [Ver código fuente]

Un controlador HTTP personalizado se puede crear implementando la interfaz **IHttpHandler**, la cual contiene sólo dos métodos. Mediante una llamada a **IsReusable**, un generador HTTP puede consultar a un controlador para determinar si se puede utilizar la misma instancia con el f in de atender a varias solicitudes. El método **ProcessRequest** toma una instancia **HttpContext** como parámetro y le proporciona acceso a los intrínsecos **Request** y **Response**. En el siguiente ejemplo, los datos de solicitud no se tienen en cuenta y se envía una constante de tipo cadena como respuesta al cliente.

```
   Public Class SimpleHandler : Inherits IHttpHandler
  Public Sub ProcessRequest(context As HttpContext)
    context.Response.Write("Hello World!")
  End Sub

  Public Function IsReusable() As Boolean
    Return(True)
  End Function
End Class
```

Después de incluir el ensamblado del controlador compilado en el directorio \bin de la aplicación, la clase del controlador se puede especificar como objetivo de las solicitudes. En este caso, todas las solicitudes dirigidas a "SimpleHandler.aspx" se desvían a una instancia de la clase **SimpleHandler**, la cual reside en el espacio de nombres **Acme.SimpleHandler**.

Resumen de la sección

1. Los controladores y generadores HTTP constituyen el eje del marco de trabajo de las páginas ASP.NET.
2. Los generadores asignan cada solicitud a un controlador, que se encarga de procesarlas.
3. Los generadores y controladores se definen en el archivo web.config. Los subdirectorios heredan la configuración de los generadores..
4. Para crear un controlador personalizado, hay que implementar **IHttpHandler** y agregar la clase a la sección **<httphandlers>** del archivo web.config del directorio.

Presentación de servicios Web de XML

Internet está evolucionando rápidamente desde los sitios Web actuales, que simplemente proporcionan páginas de interfaz de usuario a través de exploradores, a una futura generación de sitios Web programables que establecen vínculos directamente con organizaciones, aplicaciones servicios y dispositivos entre sí. Estos sitios Web programables adquieren un valor adicional al de aquellos sitios a los que se obtiene acceso de forma pasiva, convirtiéndose en servicios Web reutilizables y programables.

Common Language Runtime proporciona soporte integrado para crear y exponer servicios Web, utilizando una abstracción de programación coherente con programadores de Web Forms ASP.NET y con usuarios existentes de Visual Basic que resulta familiar para ambos. El modelo resultante es escalable y ampliable y comprende estándares abiertos de Internet (HTTP, XML, SOAP, WSDL) de forma que cualquier cliente o dispositivo que cuente con servicios de Internet puede obtener acceso al modelo y lo puede consumir.

Servicios Web de ASP.NET

ASP.NET proporciona soporte para servicios Web con el archivo .asmx. Un archivo .asmx es un archivo de texto similar a un archivo .aspx. Estos archivos pueden formar parte de una aplicación ASP.NET que incluya archivos .aspx. De esta forma, se puede asignar una dirección URI a los archivos .asmx, como se hace con los archivos .aspx.

En el ejemplo siguiente se muestra un archivo .asmx muy sencillo.

```
<%@ WebService Language="VB" Class="HelloWorld" %>
```

```
Imports System
Imports System.Web.Services

Public Class HelloWorld :Inherits WebService

    <WebMethod()> Public Function SayHelloWorld() As String
        Return("Hello World")
    End Function

End Class
```

Este archivo comienza con la directiva **WebService** ASP.NET y establece el lenguaje en C#, Visual Basic o JScript. A continuación, importa el espacio de nombres **System.Web.Services**. Debe incluir este espacio de nombres. Seguidamente, se declara la clase HelloWorld. Esta clase se deriva de la clase base **WebService**; tenga en cuenta que la derivación de la clase base **WebService** es opcional. Por último, cualquier método que sea accesible como parte del servicio tiene el atributo **[WebMethod]** en C#, **<WebMethod()>** en Visual Basic o **WebMethodAttribute** en JScript, delante de su firma.

Para que este servicio esté disponible, podemos asignar al archivo el nombre **HelloWorld.asmx** y colocarlo en un servidor denominado **SomeDomain.com** dentro de un directorio virtual cuyo nombre sea **someFolder**. Mediante un explorador Web, se podría insertar la dirección URL **http://SomeDomain.com/someFolder/HelloWorld.asmx** y la página resultante mostraría los métodos públicos para este servicio Web (aquéllos marcados con el atributo **WebMethod**), así como los protocolos (SOAP o HTTP GET) que se pueden utilizar para invocar dichos métodos.

Al insertar la dirección **http://SomeDomain.com/someFolder/HelloWorld.asmx?WSDL** en el explorador, se devuelve un documento de Lenguaje de descripción del servicio Web (WSDL). Este documento WSDL es muy importante y lo utilizarán los clientes que obtengan acceso al servicio.

Obtener acceso a servicios Web

Además de la tecnología de servidor ASP.NET que permite a los programadores crear servicios Web, .NET Framework proporciona conjuntos de herramientas y código sofisticados para consumir servicios Web. Como los servicios Web se basan en protocolos abiertos como SOAP (Simple Object Access Protocol), esta tecnología para cliente también se puede utilizar para consumir servicios Web que no estén basados en ASP.NET.

El SDK incluye una herramienta denominada WSDL.exe (Web Services Description Language). Esta herramienta basada en la línea de comandos se utiliza para crear clases proxy a partir de WSDL. Por ejemplo, podría escribir:

```
WSDL http://someDomain.com/someFolder/HelloWorld.asmx?WSDL
```

Para crear una clase proxy denominada HelloWorld.cs.

Esta clase se parecerá mucho a la clase creada en la sección anterior. Contendrá un método denominado SayHelloWorld que devuelve una cadena. Si compila una clase proxy en una aplicación y después llama a su método, la clase proxy empaqueta una solicitud SOAP a través de HTTP y recibe la respuesta codificada por SOAP, que se resolverá como una cadena.

Desde la perspectiva del cliente, el código es sencillo, tal y como se muestra en el siguiente ejemplo.

```
Dim myHelloWorld As New HelloWorld()
Dim sReturn As String = myHelloWorld.SayHelloWorld()
```

El resultado devuelto sería "Hello World".

El resto de esta sección contiene temas más avanzados de los servicios Web, como el envío y recepción de tipos de datos complejos. También puede encontrar una sección sobre coincidencias de patrones de texto, una tecnología que trata cualquier identificador URI que devuelve texto como si fuera un servicio Web. También puede realizar operaciones de enlace de datos con servicio Web; este tema se amplía en la sección de datos.

Escribir un servicio Web sencillo

Es posible escribir un servicio Web de XML sencillo en pocos minutos mediante el uso de cualquier editor de texto. El servicio que se creará en esta sección, MathService, presenta métodos para sumar, restar, dividir y multiplicar dos números. En la parte superior de la página, la directiva siguiente identifica el archivo como un servicio Web de XML además de especificar el lenguaje para el servicio (C#, en este caso).

```
<%@ WebService Language="C#" Class="MathService" %>
```

En este mismo archivo, se define una clase que encapsula la funcionalidad del servicio. Esta clase debe ser pública y opcionalmente puede heredar de la clase base **WebService**. Cada método que se va a exponer del servicio presenta un indicador con un atributo **[WebMethod]** delante de él. Sin este atributo, no se expondrá el método del servicio. A veces, esto resulta útil para ocultar los detalles de implementación llamados por métodos **Web Service** públicos o en el caso en que la clase **WebService** se utilice también en aplicaciones locales (una aplicación local puede utilizar cualquier clase pública, pero sólo

las clases **WebMethod** se encuentran accesibles de forma remota como servicios Web de XML).

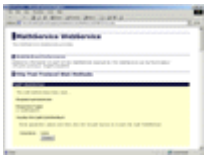
```
Imports System
Imports System.Web.Services

Public Class MathService : Inherits WebService

  <WebMethod()> Public Function Add(a As Integer, b As Integer) As Integer
    Return(a + b)
  End Function

End Class
```

Los archivos de servicios Web de XML se guardan con la extensión de archivo .asmx. Al igual que los archivos .aspx, estos archivos se compilan de forma automática mediante el motor de tiempo de ejecución de ASP.NET cuando se realiza una solicitud al servicio (las solicitudes posteriores se atienden mediante un objeto de tipo precompilado almacenado en caché). En el caso de MathService, se ha definido la clase **WebService** en el propio archivo .asmx. Hay que tener en cuenta que si un explorador solicita un archivo .asmx, el motor de tiempo de ejecución de ASP.NET devuelve una página de ayuda del servicio Web de XML que describe el servicio Web en cuestión.



VB MathService.asmx

[Ejecutar ejemplo] | [Ver código fuente]

VB MathService.asmx?wsdl

[Ver ejemplo]

Servicios Web de XML precompilados

Si se tiene una clase precompilada que se desea exponer como un servicio Web de XML (y esta clase expone métodos marcados con el atributo **[WebMethod]**), es posible crear un archivo .asmx con sólo la siguiente línea.
<%@ WebService Class="MyWebApplication.MyWebService" %>

MyWebApplication.MyWebService define la clase **WebService** y está incluido en el subdirectorio \bin de la aplicación ASP.NET.

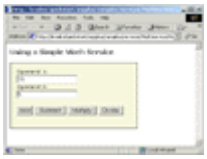
Consumir un servicio Web de XML desde una aplicación cliente

Para utilizar este servicio, hay que hacer uso de la herramienta de línea de comandos del lenguaje de descripción de servicios Web (Web Services Description Language), WSDL.exe, incluida en el kit de desarrollo de software (SDK) con el fin de crear una clase proxy que sea similar a la clase definida en el archivo .asmx. (Solamente contendrá los métodos **WebMethod**). A continuación, hay que compilar el código con esta clase proxy incluida.

WSDL.exe acepta diversas opciones de línea de comandos; sin embargo, para crear un proxy sólo se requiere una opción: el identificador URI al WSDL. En este ejemplo, se pasan algunas opciones adicionales que especifican el lenguaje preferido, el espacio de nombres y la ubicación de salida para el proxy. También es posible compilar con un archivo WSDL guardado anteriormente en lugar del identificador URI al propio servicio:

wsdl.exe /l:CS /n:MathService /out:MathService.cs MathService.wsdl

Una vez que existe la clase proxy, es posible crear objetos basados en ella. Cada llamada de método realizada con el objeto pasa seguidamente al identificador URI del servicio Web de XML (normalmente como una solicitud SOAP).



VB MathServiceClient.aspx

[Ejecutar ejemplo] | [Ver código fuente]

Cálculo de referencias de servicios Web de XML

En esta sección se muestra que se pueden pasar varios tipos de datos a métodos de **servicios Web** y devolverlos desde los mismos. Dado que la implementación de servicios Web de XML se genera en la parte superior de la arquitectura de serialización de XML, admite un importante número de tipos de datos. En la siguiente tabla se enumeran los tipos de datos admitidos para métodos de

servicios Web cuando se utiliza el protocolo SOAP (por ejemplo, al utilizar el proxy generado por la herramienta del Lenguaje de descripción de servicios Web, WSDL.exe).

Tipo	Descripción
Tipos primitivos	Tipos primitivos estándar. En la lista completa de tipos primitivos admitidos se encuentran String, Char, Byte, Boolean, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Guid, Decimal, DateTime (como timeInstant de XML), DateTime (como fecha de XML), DateTime (como hora de XML) y XmlQualifiedName (como QName de XML).
Tipos Enum	Tipos de enumeración, como por ejemplo, "public enum color { red=1, blue=2 }"
Matrices de primitivos, Enums	Matrices de los tipos primitivos anteriores, como string[] e int[]
Clases y estructuras	Tipos de clases y estructuras con campos o propiedades públicas. Se pueden serializar los campos y propiedades públicos.
Matrices de clases (estructuras)	Matrices de los tipos anteriores.
DataSet	Tipos de DataSet de ADO.NET (vea la siguiente sección para obtener un ejemplo). También pueden aparecer DataSet como campos en estructuras y clases. **Nota:** Microsoft Visual Studio.NET y la utilidad del SDK XSD.EXE son compatibles para "establecer tipos inflexiblemente" en DataSet. Estas herramientas generan una clase que se hereda de DataSet para producir DataSet1 al agregar varios métodos, propiedades, etc. que son específicos para un esquema XML concreto. Si se pasa DataSet, los servicios Web de XML siempre transmiten el esquema junto con los datos (de

	forma que sabe qué tablas y columnas se están pasando) y los tipos correspondientes (por ejemplo, int, string). Si se pasa una subclase de DataSet (por ejemplo, DataSet1), los servicios Web de XML suponen que se agregan tablas o columnas al constructor, así como que esas tablas o columnas representan el esquema.
Matrices de DataSet	Matrices del tipo anterior.
XmlNode	XmlNode consiste en una representación en memoria de un fragmento de XML (como un modelo ligero de objeto de documento de XML). Por ejemplo, "<comment>This ispretty neat</comment>" podría almacenarse en un tipo XmlNode. Se pueden pasar tipos XmlNode como parámetros y se agregan al resto de XML que se está pasando al servicio Web de XML (los otros parámetros) de forma compatible con SOAP. Lo mismo sirve para los valores devueltos. Esto permite pasar o devolver XML cuya estructura cambia de llamada en llamada o donde no se pueden conocer todos los tipos que se están pasando. También pueden aparecer tipos XmlNode como campos en estructuras y clases.
Matrices de XmlNode	Matrices del tipo anterior.
Valores devueltos:	

Tanto si se llama a un servicio Web de XML mediante SOAP o HTTP GET/POST, todos los tipos anteriores son compatibles con los valores devueltos.

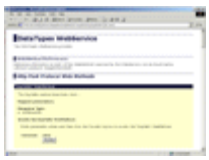
Parámetros:

Tanto los parámetros por valor, como por referencia (in/out) son compatibles si se utiliza el protocolo SOAP. Los parámetros por referencia pueden enviar el valor en dos direcciones: hasta el servidor y de vuelta al cliente. Cuando se analizan parámetros de entrada a un servicio Web de XML mediante HTTP GET/POST, sólo se admite un conjunto limitado de tipos de datos que deben ser parámetros por valor. A continuación se enumeran los tipos compatibles con los parámetros GET/POST de HTTP:

Tipo	Descripción
Tipos primitivos (limitados)	La mayoría de los tipos primitivos estándar. La lista completa de tipos primitivos admitidos comprende los tipos Int32, String, Int16, Int64, Boolean, Single, Double, Decimal, DateTime, UInt16, UInt32, UInt64 y Currency. Desde el punto de vista del cliente, todos estos tipos se transforman en cadenas.
Tipos Enum	Tipos de enumeración, como por ejemplo, "public enum color { red=1, blue=2 }". Desde el punto de vista del cliente, los tipos enum se convierten en clases con una cadena const estática para cada valor.
Matrices de primitivos, Enums	Matrices de los tipos primitivos anteriores, como string[] e int[]

En el siguiente ejemplo se muestra el uso de los tipos que se acaban de enumerar, mediante un proxy SOAP generado desde WSDL.exe. Debe tenerse en cuenta que, como hay más de una clase pública definida en el archivo .asmx file, se debe especificar cuál se tratará como clase **WebService** mediante el atributo "Class" de la directiva **WebService**:

```
<%@ WebService Language="C#" Class="DataTypes" %>
```



VB DataTypes.asmx

VB DataTypes.asmx?wsdl

[Ejecutar ejemplo] | [Ver código fuente] [Ver ejemplo]

- El método **SayHello** muestra cómo devolver una cadena desde un servicio.
- El método **SayHelloName** devuelve una cadena y también toma una cadena como parámetro.
- El método **GetIntArray** muestra cómo devolver una matriz de enteros.
- El método **GetMode** devuelve un valor enum.
- El método **GetOrder** devuelve una clase (que, en este caso, es casi lo mismo que una estructura).
- El método **GetOrders** devuelve una matriz de objetos **Order**.

Mediante la herramienta de generación del proxy de la línea de comandos WSDL.exe, el cálculo de referencias de estos tipos de datos resulta transparente para la aplicación del cliente consumidor. A continuación, se incluye una aplicación del cliente de muestra para el anterior servicio Web de XML:

VB DataTypesClient.aspx

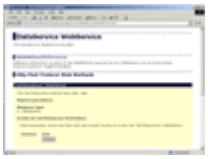
[Ejecutar ejemplo] | [Ver código fuente

Utilizar datos en servicios Web de XML

En este ejemplo se muestra la forma de devolver conjuntos de datos (objetos DataSet), que representan una nueva y eficaz manera de representar datos desconectados basada en XML, desde un método de **servicio Web**. Esto constituye un uso muy eficaz de los servicios Web de XML, ya que los objetos DataSet pueden almacenar información y relaciones complejas en una estructura inteligente. Al exponer conjuntos de datos a través de un servicio, es posible limitar las conexiones de base de datos que se establecen con el servidor de datos.

El método **GetTitleAuthors** establece una conexión con una base de datos y emite dos instrucciones SQL: una que devuelve una lista de autores y otra que devuelve una lista de títulos de libros. Coloca los dos conjuntos de resultados en un único conjunto de datos denominado ds y después devuelve el objeto DataSet en cuestión.

El método **PutTitleAuthors** ilustra un método de **servicio Web** que utiliza un conjunto de datos como parámetro y devuelve un número entero que representa el número de filas recibidas en la tabla "Authors" del conjunto de datos. Aunque la implementación de este método es un poco simplista, también se podrían combinar de forma inteligente los datos transferidos con el servidor de la base de datos.



VB DataService.asmx

[Ejecutar ejemplo] | [Ver código fuente]

VB DataService.asmx?wsdl

[Ver ejemplo]

La aplicación cliente para este servicio Web de XML llama a GetTitleAuthors y enlaza la tabla Authors con un control DataGrid, de la manera ilustrada en los ejemplos anteriores. Para ilustrar el método **PutTitleAuthors**, el cliente quita tres filas de datos del conjunto de datos antes de llamar a este método y muestra el número de filas recibidas por el servicio.

VB DataServiceClient.aspx

[Ejecutar ejemplo] | [Ver código fuente]

Utilizar objetos y elementos intrínsecos

En este ejemplo se ilustra la forma de tener acceso a los elementos intrínsecos de ASP.NET, como **Session** y **Application**. También se muestra la forma de desactivar el objeto **Session** mediante WebMethod.

El primer método del archivo .asmx de ejemplo, **UpdateHitCounter**, tiene acceso al objeto **Session** y agrega e incrementa en una unidad el valor de "HitCounter". Después devuelve este valor en forma de cadena. El segundo método, **UpdateAppCounter**, hace lo mismo, pero con el objeto **Application**. Hay que tener en cuenta lo siguiente:

```
<WebMethod(EnableSession:=true)>
```

El estado de sesión de los servicios Web de XML está deshabilitado de forma predeterminada y hay que utilizar una propiedad de atributo especial para habilitar objetos **Session**. Sin embargo, este objeto no necesita objetos **Session**, ya que sólo utiliza el objeto **Application**.

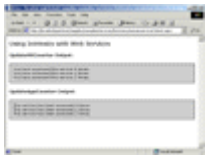
VB SessionService.asmx

[Ejecutar ejemplo] | [Ver código fuente]

VB SessionService.asmx?wsdl

[Ver ejemplo]

Cuando se tiene acceso al cliente proxy, éste contiene una colección de cookies. Esta colección se utiliza para aceptar y devolver la cookie APSESSIONID que utiliza ASP.NET para hacer un seguimiento de las sesiones. Esto es lo que permite al cliente recibir distintas respuestas al método Hit de **Session**.



VB SessionServiceClient.aspx

[Ejecutar ejemplo] | [Ver código fuente]

Comportamiento WebService

Microsoft lanzó recientemente un comportamiento DHTML con habilitación SOAP nuevo para Microsoft Internet Explorer 5.0 y versiones posteriores. El nuevo comportamiento **WebService** permite a la secuencia de comandos en el cliente invocar métodos remotos expuestos por servicios Web .NET XML de Microsoft u otros servidores Web que admiten el protocolo SOAP (Simple Object Access Protocol). El comportamiento **WebService** se implementa con un archivo HTML Components (HTC) como un comportamiento asociado, por lo que puede utilizarse en Internet Explorer.

La finalidad del comportamiento **WebService** es proporcionar una manera sencilla de utilizar y aplicar SOAP, sin requerir un conocimiento profundo de su implementación. El comportamiento **WebService** admite el uso de una amplia variedad de tipos de datos, incluidos tipos de datos SOAP intrínsecos, matrices y datos XML (Lenguaje de marcado extensible, Extensible Markup Language). Este componente flexible permite a Internet Explorer recuperar información de servicios Web de XML y actualizar una página de forma dinámica mediante el uso de DHTML y secuencias de comandos, sin requerir el desplazamiento ni una actualización de página completa.

La siguiente generación de la infraestructura y las herramientas de programación .NET, incluido Visual Studio.NET, .NET Framework y servidores .NET Enterprise, está diseñada para la programación de aplicaciones según el modelo de servicios Web de XML. El comportamiento **WebService** es especialmente significativo ya que permite a Internet Explorer utilizar estos servicios Web de XML de próxima generación.

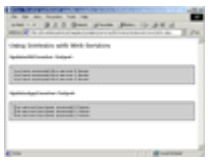
El sitio de Microsoft Developer Network (MSDN) proporciona la documentación siguiente.

Hacer coincidir modelos de texto HTML

Este ejemplo muestra cómo crear un proxy cliente para cualquier URL que proporcione texto. En vez de crear el archivo .asmx, se puede crear un archivo WSDL que describa la página HTML (o XML o cualquier otro formato no binario) actual. El archivo WSDL puede utilizarse para generar un proxy cliente mediante la herramienta de línea de comandos WSDL.exe que utilizará RegEx para analizar la página HTML especificada y extraer los valores.

Esto se puede realizar agregando etiquetas <Match> a la sección Response del archivo WSDL. Estas etiquetas utilizan un atributo denominado **pattern**, que es la expresión regular que se corresponde con la parte de texto de la página que constituye el valor de la propiedad. (Nota: la propiedad de la clase proxy es de sólo lectura).

El código consumidor puede entonces crear el objeto, obtener acceso al objeto **Matches** devuelto por el nombre convertido en función que aparece en el archivo WSDL y conseguir acceso a cualquier parte del código HTML como una propiedad. No es necesario ningún conocimiento de WSDL, expresiones regulares, ni HTML para poder utilizar la clase proxy. Ésta se comporta como cualquier otra clase de .NET Framework.



VB MatchClient.aspx

[Ejecutar ejemplo] | [Ver código fuente]

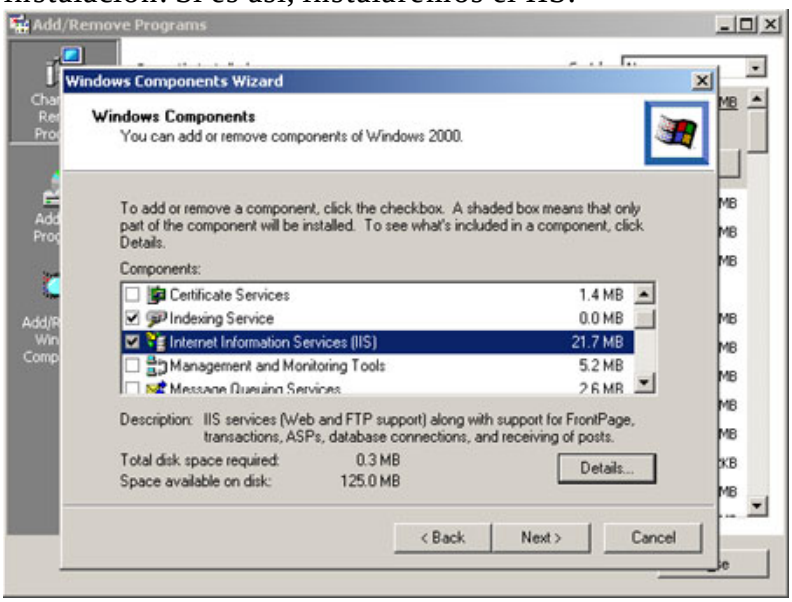
ASP.NET con Visual Studio

Antes de empezar a crear la aplicación vamos a ver si tenemos los requisitos mínimos para empezar.

Pre-requisitos

Empezaremos por el sistema operativo, necesitaremos W2000 Profesional SP3, W2000 Server SP3, Windows XP Profesional o W2003 Server. No sirven W95, W98 o XP Home, porque no llevan IIS (Internet Information Services).

Puede ser que aún teniendo uno de los sistemas operativos necesarios no tengamos instalado el IIS, que por defecto no viene en la configuración por defecto de instalación. Si es así, instalaremos el IIS.



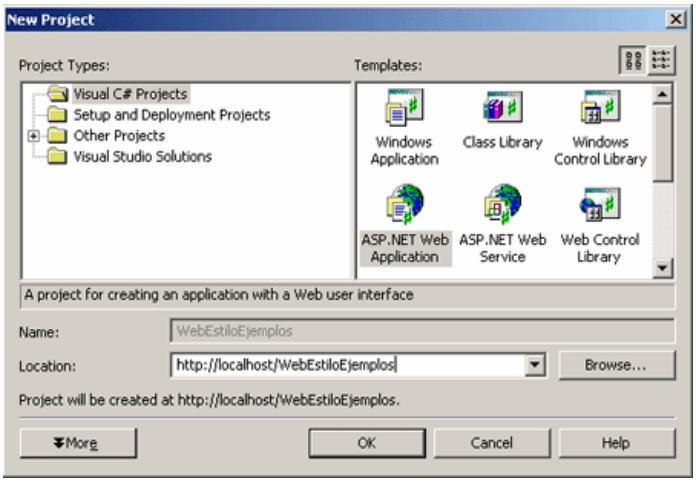
Si hemos instalado el IIS después de haber instalado el .NET Framework, deberemos activar el Framework en el IIS para que funcionen las páginas ASP.NET, para ello abriremos una ventana de comandos y teclearemos lo siguiente:

```
C:\> C:
C:\> cd C:\WINNT\Microsoft.NET\Framework\v1.0.3705
C:\WINNT\Microsoft.NET\Framework\v1.0.3705> aspnet_regiis -i
```

Dependiendo de la de versión del .NET Framework el directorio (v1.0.3705) puede variar, en este caso es la versión 1.0 de .NET.
Si instalamos el .NET Framework después del IIS no tendremos que hacer esto puesto que la instalación de .NET lo hace automáticamente.
Crear un proyecto ASP.NET
Es muy importante antes de ejecutar el siguiente paso tener el IIS instalado y corriendo. Lo podemos poner en marcha desde "Panel de Control" -> "Herramientas Administrativas" -> "Internet Information Services".

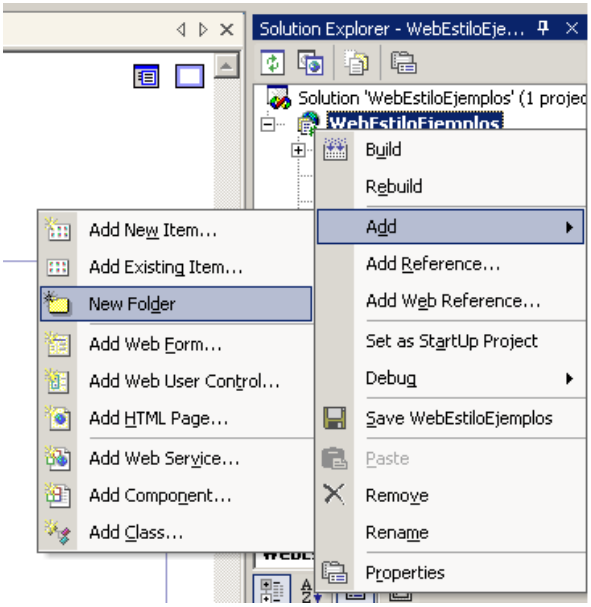
Suponiendo que hemos instalado el Visual Studio .NET, lo abrimos y pulsamos sobre el botón de nuevo proyecto.

Una vez con el IIS corriendo, seleccionamos "Visual C# Projects" -> ASP.NET Web Application y ponemos el nombre del proyecto que vamos a crear, en este ejemplo "WebEstiloEjemplos".

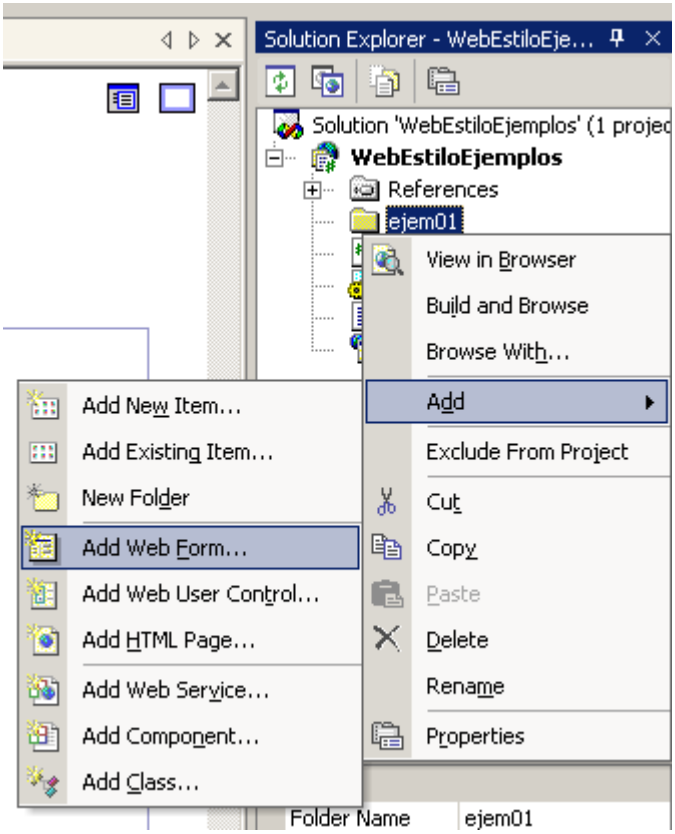


Pulsamos el botón "Ok". Si todo va bien se creará el web.

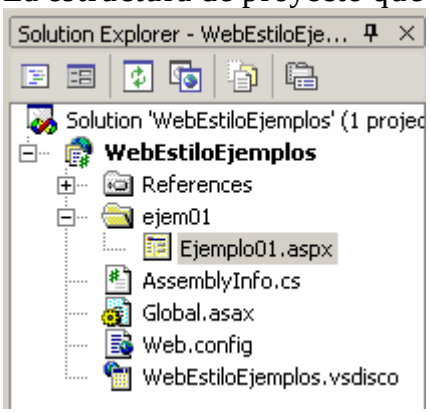
Creando un Web Form
Una vez que tenemos generada la aplicación ASP.NET, vamos a crear un Web Form que es la base de cualquier página web .NET.
Primero crearemos un directorio dentro del proyecto para tener los ejemplos ordenados, el directorio lo llamaremos "ejem01". Para ello pulsaremos el botón derecho sobre el nombre de proyecto y seleccionaremos "Add" -> "New Folder"



Una vez creado el directorio "ejem01", pasamos a añadir nuestro primer Web Form. Seleccionamos con el ratón el directorio "ejem01", pulsamos el botón derecho y seleccionaremos "Add" -> "Add Web Form...", como se puede ver en la imagen.



Nos aparecerá un cuadro de diálogo en el que se nos preguntará el nombre del web form (página web), en este ejemplo le daremos "Ejemplo01.aspx".
La estructura de proyecto que nos tiene que quedar ha de ser esta.



Añadiendo Controles de Servidor

Una vez creado nuestro Web Form, ya podemos añadir controles de servidor, primero hacemos doble clic sobre el fichero "Ejemplo01.aspx". Nos aparecerá una página en blanco con una cuadrícula en la que añadiremos los controles.
Para añadir controles tan solo hay que arrastrarlos desde la ventana "Toolbox" a al Web Form en blanco.

Añadiremos un "TextBox", una "Label" y un "Button". Y cambiaremos los nombres (Propiedades -> Atributo (ID) en negrita) por "txtNombre", "lblNombre" y "btnEnviar", respectivamente. Así mismo cambiaremos la propiedad "Text" de "lblNombre" por "Su nombre es: ".

Si estamos acostumbrados a manejarnos con HTML es muy posible que queramos retocar un poco la página añadiendo algún que otro elemento, para ello hacemos clic sobre la vista HTML del documento.

En este caso vamos a añadir un título <H1> y algunos retornos de carro
 y obtenemos algo como esto:

Ejemplo01.aspx

```
<?@ Page language="c#" Codebehind="Ejemplo01.aspx.cs"
AutoEventWireup="false" Inherits="WebEstiloEjemplos.ejem01.Ejemplo01" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>Ejemplo01</title>
```

```
    <meta content="Microsoft Visual Studio 7.0" name="GENERATOR">
    <meta content="C#" name="CODE_LANGUAGE">
    <meta content="JavaScript" name="vs_defaultClientScript">
    <meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <form id="Ejemplo01" method="post" runat="server">
      <h1>Ejemplo 01</h1>
      <asp:textbox id="txtNombre" runat="server"></asp:textbox><br>
      <asp:label id="lblNombre" runat="server">Su nombre es: </asp:label><br>
      <br>
      <asp:button id="btnEnviar" runat="server"
Text="Button"></asp:button></form>
  </body>
</HTML>
```

Usando el Code-Behind

En el apartado anterior hemos visto cómo añadir controles de servidor a un Web Form y el código HTML que se genera, ahora vamos a ver el código C# que hay "detrás" de esos controles.
Para ello pulsaremos F7 y veremos el código C# que lleva esta página, que es el siguiente:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace WebEstiloEjemplos.ejem01
{
  /// <summary>
  /// Summary description for Ejemplo01.
  /// </summary>
  public class Ejemplo01 : System.Web.UI.Page
  {
    protected System.Web.UI.WebControls.TextBox txtNombre;
    protected System.Web.UI.WebControls.Label lblNombre;
    protected System.Web.UI.WebControls.Button btnEnviar;
```

```
    private void Page_Load(object sender, System.EventArgs e)
    {
      // Put user code to initialize the page here
    }

    #region Web Form Designer generated code
    override protected void OnInit(EventArgs e)
    {
      //
      // CODEGEN: This call is required by the ASP.NET Web Form Designer.
      //
      InitializeComponent();
      base.OnInit(e);
    }

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
      this.Load += new System.EventHandler(this.Page_Load);
    }
    #endregion
  }
}
```

Vamos a explicar un poco la estructura de este fichero de código C#.

Primero vemos que hay una sección "using" en la que se especifican qué espacios de nombre vamos a usar en este fichero. Los espacios de nombre sirven para agrupar clases, las cuales tiene normalmente funcionalidades comunes. Así por ejemplo, hemos incluido el espacio de nombres System.Web.UI.WebControls para trabajar con controles web de servidor. Este espacio de nombres (namespace) contiene todas las clases relacionadas con los controles de servidor.
Seguidamente pasamos a especificar en qué espacio de nombres estará la clase que vamos a definir, todas las clases deben pertenecer a un espacio de nombres. En este caso indicamos que el espacio de nombres (namespace) en el que se incluirá nuestra clase es WebEstiloEjemplos.ejem01.
A continuación definimos la clase que se asociará con la página web, la clase que manejará todo el funcionamiento de la página web Ejem01.apsx.
La clase se llama Ejemplo01 y como pertenece al espacio de nombres WebEstiloEjemplos.ejem01, su nombre real es WebEstiloEjemplos.ejem01.Ejemplo01, si os dais cuenta coincide con el atributo Inherits de la página Ejemplo01.aspx que se encuentra en la cabecera de dicha

página, de esta forma es cómo se enlaza la página aspx con la clase de código C# que se ejecutará.
Una vez dentro de la clase, definimos los atributos que contiene, en este caso son 3 correspondientes a los 3 controles de servidor que hemos arrastrado antes desde la barra de herramientas. Una caja de texto (TextBox) llamada txtNombre, una etiqueta (Label) llamada lblNombre y un botón (Button) llamado btnEnviar.
Luego hay un método que se llama Page_Load() que se ejecuta cuando se carga la página y que por ahora está vacío.
Y para finalizar hay una región de código que la genera el diseñador de Visual Studio .NET que de momento no vamos a entrar a comentar y que es mejor, como advierte, que no toquemos.
Añadiendo eventos a los controles
Ahora vamos a añadir un evento a un control de servidor, dicho evento ejecutará código C# para realizar una acción en concreto.
Primero vamos a la vista de diseño, en la que podremos ver la caja de texto, la etiqueta y el botón. Seguidamente añadiremos un evento al botón para que cuando sea pulsado se ejecute una acción en el servidor. Para añadir el evento de clic al botón pulsaremos doble clic sobre el botón en la ventana de diseño del formulario. Esto nos llevará a la ventana de edición de código C#, en la que se nos habrá creado un método btnEnviar_Click() que se habrá asociado con el evento clic del botón.
En este método escribiremos el código que queremos que se ejecute cuando se pulsa clic sobre el botón "Enviar". Sencillamente vamos a agregar a la etiqueta el valor de la caja de texto.

```
private void btnEnviar_Click(object sender, System.EventArgs e)
{
  lblNombre.Text+= txtNombre.Text;
}
```

Con lo que tendremos el siguiente código:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace WebEstiloEjemplos.ejem01
{
  /// <summary>
  /// Summary description for Ejemplo01.
  /// </summary>
```

```
  public class Ejemplo01 : System.Web.UI.Page
  {
    protected System.Web.UI.WebControls.TextBox txtNombre;
    protected System.Web.UI.WebControls.Label lblNombre;
    protected System.Web.UI.WebControls.Button btnEnviar;

    private void Page_Load(object sender, System.EventArgs e)
    {
      // Put user code to initialize the page here
    }

    #region Web Form Designer generated code
    override protected void OnInit(EventArgs e)
    {
      //
      // CODEGEN: This call is required by the ASP.NET Web Form Designer.
      //
      InitializeComponent();
      base.OnInit(e);
    }

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
      this.btnEnviar.Click += new System.EventHandler(this.btnEnviar_Click);
      this.Load += new System.EventHandler(this.Page_Load);

    }
    #endregion

    private void btnEnviar_Click(object sender, System.EventArgs e)
    {
      lblNombre.Text+= txtNombre.Text;
    }
  }
}
```

Eventos de Página

Así como los controles de servidor pueden tener eventos, también las páginas disparan eventos. El más habitual de los eventos es el que se produce cuando la página se carga, es el evento "Load". Este evento es disparado siempre que se carga la página.

En el siguiente ejemplo tenemos una etiqueta que se actualiza cada vez que se carga la página con la hora actual del servidor.

Ejemplo02.aspx

```
<?@ Page language="c#" Codebehind="Ejemplo02.aspx.cs"
AutoEventWireup="false" Inherits="WebEstiloEjemplos.ejem02.Ejemplo02" ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
  <HEAD>
    <title>Ejemplo02</title>
    <meta name="GENERATOR" Content="Microsoft Visual Studio 7.0">
    <meta name="CODE_LANGUAGE" Content="C#">
    <meta name="vs_defaultClientScript" content="JavaScript">
    <meta name="vs_targetSchema"
content="http://schemas.microsoft.com/intellisense/ie5">
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <form id="Ejemplo02" method="post" runat="server">
      <h1>WebEstilo - Ejemplo 02</h1>
      <asp:Label id="lblHora" runat="server"></asp:Label>
    </form>
  </body>
</HTML>
```

El correspondiente código C# asociado a la página.
Ejemplo02.aspx.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace WebEstiloEjemplos.ejem02
{
  /// <summary>
  /// WebEstilo.com
  /// Clase que muestra el funcinamiento del evento Load.
  /// </summary>
  public class Ejemplo02 : System.Web.UI.Page
  {
    protected System.Web.UI.WebControls.Label lblHora;
```

```
private void Page_Load(object sender, System.EventArgs e)
{
  // Código que se ejecuta cada vez que se carga la página
  lblHora.Text = "Hora actual: "+ DateTime.Now.ToShortTimeString();
}

#region Web Form Designer generated code
override protected void OnInit(EventArgs e)
{
  //
  // CODEGEN: This call is required by the ASP.NET Web Form Designer.
  //
  InitializeComponent();
  base.OnInit(e);
}
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
  this.Load += new System.EventHandler(this.Page_Load);
}
#endregion   } }
```

Firma de término de modulo.

Alumno	Instructor
_____	_____
	Ing. Joel A. González Estrada