

1	Introducción al desarrollo de aplicaciones Web	3
2	Requerimientos para trabajar con ASP.NET	4
3	Configurar una aplicación Web con IIS	5
4	Conceptos de Web Forms	6
5	Creando una página básica	12
6	Directivas de página	14
7	Web.config	20
8	DataList	21
9	DataGrid	29
10	Repeater	37
11	Templates	41
12	HttpRequest	49
13	HttpResponse	51
14	HttpServerUtility	53
15	HttpSessionState	54

16	HttpApplicationState	55
17	El archivo Global.asax	57
18	Aspectos avanzados de las aplicaciones Web	60
19	Manejadores y Módulos http	62
20	Seguridad en ASP.NET	63
21	Archivo de configuración	72
22	Manejo del caché	80
23	User Controls	84
24	Código del lado del cliente	88
25	XML Web Services	95
26	Introducción a Web Services	96
27	Construyendo Web Services	98
28	Probando un Web Service utilizando el Internet Explorer	100
29	Creando un cliente que utilice Web Services	103
30	Encabezados SOAP	116
31	Seguridad en Web services	120
32	Extensiones SOAP	122
33	Referencias	124

1 INTRODUCCIÓN AL DESARROLLO DE APLICACIONES WEB

Introducción a ASP.NET

Microsoft® **ASP.NET** forma parte de la plataforma de computación Microsoft .NET. ASP.NET es empleado en el desarrollo de **aplicaciones WEB** y en la implementación de **servicios WEB XML**. Las páginas ASP.NET se ejecutan en el **servidor WEB** y generan archivos en un lenguaje de **"markup"** como HTML, WML, o XML , los cuales son enviados al **cliente** (al navegador) .

Las **páginas ASP** emplean un modelo compilado de **programación orientada por eventos**. Este modelo permite mejorar el rendimiento de la aplicación y permite separar la lógica de la interfaz con el usuario de la lógica de la aplicación. Las páginas ASP.NET y los archivos de implementación de los servicios Web XML contienen la lógica ejecutada en el servidor (server-side) contrastando con las aplicaciones donde la lógica se ejecuta del lado del cliente como aplicaciones escritas en Microsoft® Visual Basic® .NET, Microsoft® Visual C#® .NET, o en cualquier lenguaje de programación compatible con la plataforma de Microsoft.Net.

Archivos HTML y programación de páginas WEB

El protocolo empleado pro Internet para el procesamiento de páginas Web es **HTML**. Los archivos HTML son archivos de texto (**hipertexto**) interpretados por los servidores de páginas Web con IIS y Apache; así como por los navegadores de Internet como Explorer o Netscape. El código programático asociado a una página Web es compilado por ASP.NET para ser transformado en HTML o alguna derivación de éste para ser desplegado finalmente en un navegador.

Al desarrollar una aplicación, el diseño de las páginas WEB puede ser realizado con el diseñador o editor de su preferencia y la programación de la misma puede ser efectuada desde el ambiente de programación Visual Studio .Net o colocando los scripts ASP.Net de manera **"embeded"**. Dentro de una herramienta como Visual Studio.NET también es posible diseñar el formato de la página Web el cual será desplegado.

Aunque en el caso especial de aplicaciones WEB en ASP.NET pareciera estar muy ligado el código de la aplicación con el de la interfaz gráfica es posible emplear técnicas de programación modular, estructurada u orientadas por objetos permitiendo el desarrollo de **aplicaciones de múltiples capas de una manera escalable**.

Todas las técnicas de programación disponibles en Visual Studio .NET para el desarrollo de aplicaciones orientadas a datos o a Windows también son aplicables para el caso de desarrollo en Web.

Diseñando aplicaciones Web

Antes de desarrollar una aplicación se debe aplicar un proceso de diseño, en el cual deben tomarse decisiones acerca de las tecnologías para desarrollo del contenido WEB (diseño gráfico y edición), la modelación de los procesos, la localización y la seguridad.

En el caso de ASP.NET existen varias pautas de diseño a considerar:

- Determinar la tecnología de **IIS** a emplear
- Diseñando aplicaciones en dos **capas** "Two-Tier"
- Diseñando aplicaciones en tres capas "Three-Tier"
- Diseñando aplicaciones de múltiples capas "Multi-Tier"
- Diseñando aplicaciones a través de **servicios web** "Web services"
- Especificando lo límites de **integración** de las capas "Application Boundaries"
- Controlando el **flujo** de la aplicación
- Diseñando una **arquitectura cliente-servidor**
- **Escalabilidad** de aplicaciones
- **Integrabilidad** de aplicaciones
- **Internacionalización** de aplicaciones

Introducción a las páginas ASP.NET

ASP.NET ofrece un "kit" para el desarrollo de aplicaciones o **SDK** "Software Development Kit" para el soporte de las tareas y la programación de las páginas web en ASP.NET.

Al crear una aplicación en ASP.Net, deben ser considerados los siguientes puntos (todos estos puntos serán considerados en detalle a lo largo del manual):

- **Ciclo de vida** del proyecto
- **Metodologías** de desarrollo y fases del proyecto
- Consideraciones de **seguridad** de la página Web
- La edición, configuración y programación de los **formularios Web**
- El desarrollo de los **controles ASP.NET** (En la terminología de microsoft, un control es un objeto empleado para el desarrollo de la interfaz gráfica).
- **Manejo de datos** en ASP.NET
- Manejo del **estado** y la **sección** en una aplicación Web
- **Servicios Web**
- Desarrollando aplicaciones para **consumir servicios Web**.

2. REQUERIMIENTOS PARA TRABAJAR CON ASP.NET

ASP.NET es soportado en **Windows 2000** (Professional, Server, y Advanced Server), **Windows XP** Professional, y la familia **Windows Server 2003** tanto para aplicaciones cliente como para aplicaciones servidor. Adicionalmente, para desarrollar aplicaciones **ASP.NET** en el servidor, el siguiente "software" es requerido:

- Windows 2000 Server o Advanced Server con Service Pack 2, Windows XP Professional o 64-Bit Edition, o uno de los productos de la familia Windows Server 2003.
- MDAC 2.7 para el manejo de los datos.
- Internet Information Services IIS (el cual va a ser utilizado como servidor de páginas WEB).

Nota Los Servicios Web XML creados utilizando ASP.NET soportan las mismas plataformas soportadas por ASP.NET. En el caso de servicios web XML clientes, éstos son soportados por todas las plataformas soportadas por la plataforma .NET.

En la familia de productos de Microsoft Windows Server 2003, ASP.NET es instalado **como un rol** del sistema de operación. Para desarrollar aplicaciones WEB ASP.NET como un servidor de producción (aplicaciones listas para ser utilizadas por usuarios finales), deben ser habilitados los roles de ASP.NET y de IIS en el servidor de producción, antes de distribuir y comenzar a trabajar con la aplicación.

En Microsoft Windows Xp y en Windows 2000 Server, ASP.NET es instalado con la plataforma .NET, bien sea en solitario o como parte del Visual Studio .NET. Para poner en funcionamiento aplicaciones Web ASP.NET en un servidor de producción, se debe garantizar que **Internet Information Services (IIS)** se encuentre instalado y ejecutándose en el servidor antes de instalar la plataforma .NET. La instalación del IIS depende del sistema de operación utilizado.

Note Si se instaló ASP.NET y la plataforma .NET en un servidor y luego se instala y reinstala IIS, ocurre un problema de **"mapping"** y ASP.NET no funcionará. En este caso, para reparar este problema se debe utilizar la herramienta de manejo de registros de ASP.NET. Detalles pueden ser encontrados en [ASP.NET IIS Registration Tool \(Aspnet_regiis.exe\)](#)

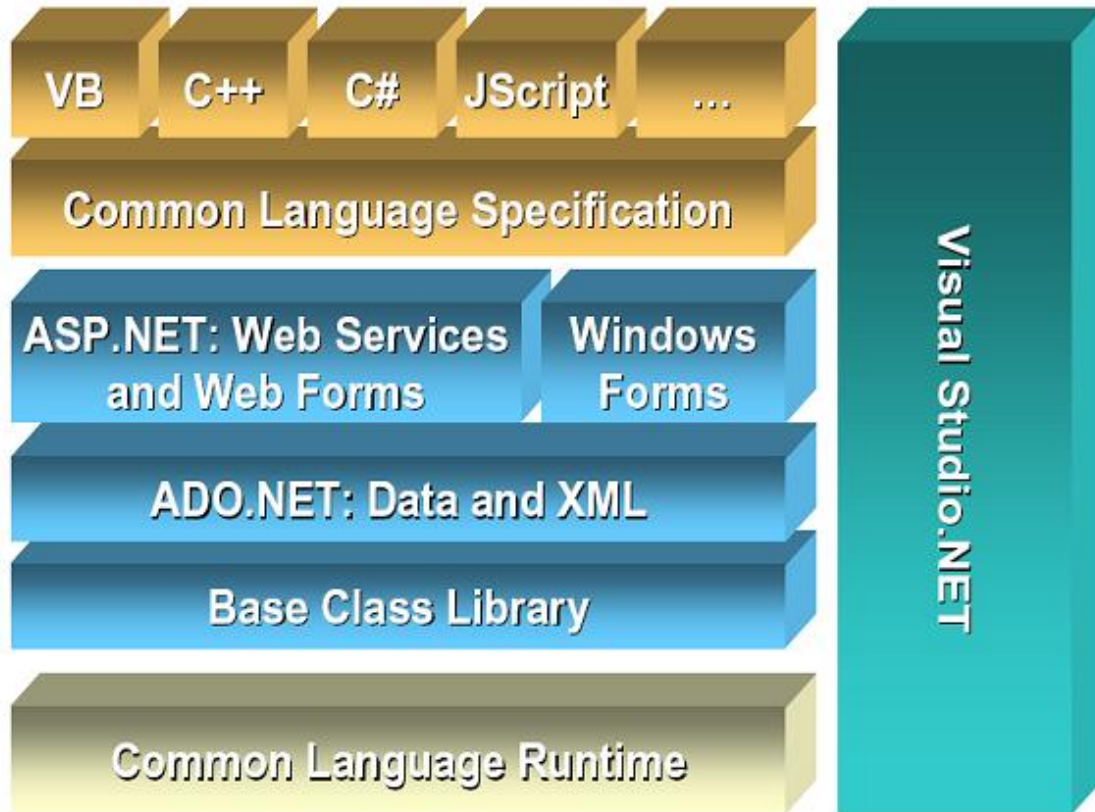
Note Si se instala la versión 1.1. de la plataforma .NET en un **controlador de dominios** "domain controller", la instalación no crea la cuenta local ASPNET. En lugar de esto, las aplicaciones ASP.NET se ejecutan sobre otras identidades. En los controladores de dominio de los servidores Windows 2000, las aplicaciones ASP.NET se ejecutan sobre la identidad **IWAM_machinename**. En los controladores de dominio de los servidores Windows 2003, las aplicaciones ASP.NET se ejecutan sobre la identidad NETWORK SERVICE (independientemente del modo de aislamiento (**isolation mode**) de IIS).

Existe casos en los cuales para la ejecución de ASP.NET sobre un controlador de dominio, requiere ejecutar pasos adicionales para que la instalación funcione adecuadamente. Información técnica sobre casos particulares de errores puede ser encontrada en los artículos publicados en <http://support.microsoft.com>.

ASP.NET es instalado en conjunto con la plataforma .NET versión 1.1 como una parte de la familia de productos de Windows Server 2003. Únicamente es requerido agregarlo como un programa nuevo desde el panel de control o utilizar el asistente “**Configurar el servidor**” para habilitarlo.

Este manual no considera el tópico de instalación de ASP.NET, información adicional puede ser encontrada en la página de Microsoft msdn.microsoft.com/library/en-us/cpguide/html/cpconaspnetplatformrequirements.asp

ASP.Net en la arquitectura Microsoft .NET



ASP.Net, en la plataforma .NET es una capa de desarrollo que permite desarrollar aplicaciones WEB ASP basadas en formularios Web Forms y Servicios Web XML. Utiliza los servicios ofrecidos en las capas anteriores y al generar las páginas dentro del ambiente de Visual Studio se produce código administrado.

La capa de ADO.NET ofrece la integración de datos y el servicio de XML. XML es parte fundamental de ASP.NET y es el corazón para desarrollar los Servicios Web, los cuales se denominan servicios Web XML. Las aplicaciones en ASP.NET, al igual que el resto de las aplicaciones en la plataforma .NET pueden ser desarrolladas en cualquier lenguaje de programación soportado por la plataforma.

3. CONFIGURAR UNA APLICACIÓN WEB CON IIS

Introducción a IIS

Internet Information Services (IIS) convierte un computador en un servidor WEB para proveer: los servicios de publicación de WWW World Wide Web, los servicios de transferencia de archivos FTP File Transfer Protocol, los servicios de correo SMTP Simple Mail Transport Protocol, y los servicios de publicación de noticias Network News Transfer Protocol. IIS puede ser empleado para realizar “host” y para administrar sitios web y otro contenido en Internet una vez obtenida una dirección IP, registrado el dominio en un servidor DNS, y configurada la red apropiadamente. IIS es un componente del sistema de operación Microsoft® Windows®.

Cuando se establece un **directorio virtual en IIS** este puede ser colocado **en cualquier ruta**, la ruta por defecto asociada a **localhost** es **C:\inetpub\wwwroot**

IIS ofrece una herramienta para desarrollar aplicaciones WEB, la sección de aplicaciones WEB del SDK (Software Development Kit). Estas utilidades se encuentran disponibles en las clases de ASP.NET.

Las aplicaciones ASP.NET requieren que la plataforma .NET se encuentre instalada en el servidor IIS.

Configuración IIS

Las versiones previas de ASP 3.0 requerían colocar la información de la configuración en los parámetros de configuración de IIS (**IIS settings**). Por ejemplo, parámetros de configuración como el estado de la sección, el "buffer", el lenguaje de "script" por defecto y el "timeout" se configuraban ejecutando los siguientes pasos:

- Click en el botón derecho de IIS Application
- Seleccionar los parámetros (settings) de application's property
- Seleccionar el tab de opciones App del dialogo "Application Configuration's"

Estos son sólo alguno de los "settings" necesarios a configurar a través de los parámetros de configuración de aplicaciones de **Microsoft Management Console (MMC)**. Todos estos parámetros o modificaciones son aplicados a "**IIS metabase**", la cual es utilizada para calcular los parámetros de aplicación en tiempo de ejecución para la aplicación web.

ASP+, la versión manejada en este manual, no requiere trabajar directamente con "**IIS metabase**". En su lugar, la **configuración en ASP.NET se basa en archivos de configuración con formato XML**, como por ejemplo **config.web**. En futuras secciones de este manual será indicado con lujo de detalles los pasos requeridos para configurar una aplicación.

4. CONCEPTOS DE WEB FORMS

Introducción

La plataforma .NET ofrece un ambiente para el desarrollo de formularios ASP.NET para WEB. Este ambiente se basa en un modelo de programación escalable soportado por el **CLR (Common Language Runtime)**, el cual es empleado en el servidor para generar dinámicamente páginas WEB.

Web Forms en ASP.NET es una Clase de la librería de clases disponibles. Por lo tanto, un **Web Form** debe ser considerado como un **objeto** y no debe confundirse con el archivo .aspx asociado el cual contiene código en HTML y código en un lenguaje de "scripting".

ASP.NET surge como la evolución lógica de **ASP (ActiveX Server Pages)**, por lo tanto ASP.NET es sintácticamente compatible con las páginas ASP existentes. El ambiente para el desarrollo de formularios WEB en ASP.NET se encuentra diseñado para resolver algunas deficiencias existentes en el modelo previo de ASP. En particular, ASP.NET provee:

- La habilidad para crear y utilizar controles de interfaz de usuario (**UI Controls**) encapsulando la funcionalidad común y por lo tanto reduciendo la cantidad de código a implementar por un desarrollador de páginas WEB.
- La habilidad otorgada a los desarrolladores de generar código de páginas WEB limpio, eficiente y estructurado.
- La habilidad ofrecida a los desarrolladores de generar interfaces **WYSIWYG** (What You See Is What You Get), típico de los ambientes de desarrollo visual.

Archivos WEB Forms

Las páginas WEB en ASP.NET son archivos de texto con la extensión **.aspx**. Estas páginas pueden ser desplegadas a través de un **directorio virtual raíz de IIS**. Cuando un "browser" (navegador) cliente solicita un **recurso .aspx**, la plataforma .NET (a través del **módulo de "runtime" de ASP.NET**) realiza la **compilación y "parsing"** del archivo .aspx (código fuente) y lo transforma en una clase de la plataforma .NET. Esta clase puede ser utilizada para procesar las solicitudes de la página WEB dinámicamente.

Observación: El archivo .aspx **solo es compilado la primera vez** que es accedido y esta instancia compilada es reutilizada cada vez que una solicitud de servicio de la página web es requerido.

Una página ASP.NET puede ser creada simplemente tomando un archivo HTML existente y cambiando su extensión a .aspx. No es requerida ninguna modificación al código de la página.

A continuación se indican los **espacios de nombres** utilizados en la plataforma .NET para desarrollar formularios Web

Entre los espacios de nombres de .NET Framework relativos a aplicaciones Web ASP.NET y servicios Web XML se incluyen los siguientes:

- **System.Web:** contiene clases e interfaces que permiten la comunicación entre explorador y servidor. Las clases de este espacio de nombres administran los resultados HTTP dirigidos al cliente (**HttpResponse**) y leen las solicitudes HTTP (**HttpRequest**). Otras clases proporcionan utilidades para herramientas y procesos de servidor, manipulación de cookies, transferencia de archivos, información sobre excepciones y control de caché de resultados.
- **System.Web.UI:** contiene clases para crear páginas de formularios web Forms, incluida la clase **Page** y otras clases estándar que se utilizan para crear interfaces de usuario Web.
- **System.Web.UI.HtmlControls:** contiene clases para controles HTML que se pueden agregar a formularios Web Forms para crear interfaces de usuario Web.
- **System.Web.UI.WebControls:** contiene clases para crear controles de servidor Web ASP.NET. Cuando se agregan a un formulario Web Forms, estos controles proporcionan código HTML específico del explorador y secuencias de comandos que permiten crear interfaces Web de usuario independientes del dispositivo.
- **System.Web.Services:** contiene las clases que permiten generar y utilizar servicios Web XML, que son entidades programables residentes en un servidor Web que se exponen por medio de protocolos estándar de Internet.

Espacio de nombres System.Web.UI

Contiene clases para crear páginas de formularios **Web Forms**, incluida la clase **Page** y otras clases estándar que se utilizan para crear interfaces de usuario Web.

El espacio de nombres System.Web.UI proporciona clases e interfaces que permiten crear los **controles y páginas** de servidor ASP.NET que aparecerán en las aplicaciones Web como elementos de interfaz de usuario. Este espacio de nombres incluye la **clase Control**, que proporciona todos los controles de servidor, ya sean **controles de servidor HTML, controles de servidor Web o controles de usuario**, con un conjunto común de funciones. Incluye también la clase Page, que se genera automáticamente si se realiza una solicitud de un archivo .aspx incluido en la aplicación Web. Se pueden crear clases heredadas de estas dos clases. Se incluyen también clases que proporcionan los controles de servidor con funciones de enlace de datos, la capacidad de guardar el estado de vista de un determinado control o página y la función de análisis para controles programables y literales.

Dentro de un **Web Form** sólo puede ser colocada **una directiva de página**, la correspondiente al objeto Page.

La jerarquía de este espacio de nombres , tiene la siguiente forma

System.Object

System.Attribute

System.Web.UI.ConstructorNeedsTagAttribute

.....

System.EventArgs

System.Web.UI.ImageClickEventArgs

System.MarshalByRefObject

System.IO.TextWriter

System.Web.UI.HtmlTextWriter

System.Web.UI.Html32TextWriter

System.ValueType

System.Enum

System.Web.UI.HtmlTextWriterAttribute

...

System.Web.UI.Control

System.Web.UI.BasePartialCachingControl

...

Espacio de nombres System.Web.UI.HtmlControls

Contiene clases para **controles HTML** que se pueden agregar a formularios **Web Forms** para crear interfaces de usuario Web.

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

Las clases disponibles en este espacio de nombres son:

Clases

Clase	Descripción
HtmlAnchor	Permite el acceso mediante programación a la etiqueta HTML <a> del servidor.
HtmlButton	Permite el acceso mediante programación a la etiqueta HTML <button> del servidor.
HtmlContainerControl	Define los métodos, las propiedades y los eventos disponibles en todos los controles de servidor HTML que deben tener una etiqueta

	de cierre.
HtmlControl	Define los métodos, las propiedades y los eventos comunes a todos los controles de servidor HTML del marco de trabajo de la página de formularios Web Forms.
HtmlForm	Proporciona acceso mediante programación al elemento HTML <form> del servidor.
HtmlGenericControl	Define los métodos, las propiedades y los eventos de todas las etiquetas de control de servidor HTML que no están representadas por una clase específica de .NET Framework.
HtmlImage	Proporciona acceso mediante programación para el elemento HTML del servidor.
HtmlInputButton	Permite el acceso mediante programación a los elementos HTML <input type= button>, <input type= submit> y <input type= reset> del servidor.
HtmlInputCheckBox	Permite el acceso mediante programación al elemento HTML <input type= checkbox> del servidor.
HtmlInputControl	Se utiliza como la clase base abstracta que define los métodos, las propiedades y los eventos comunes a todos los controles de entrada HTML como, por ejemplo, los elementos <input type=text>, <input type=submit> y <input type= file>.
HtmlInputFile	Permite el acceso mediante programación al elemento HTML <input type= file> del servidor.
HtmlInputHidden	Permite el acceso mediante programación al elemento HTML <input type=hidden> del servidor.
HtmlInputImage	Permite el acceso mediante programación al elemento HTML <input type= image> del servidor.
HtmlInputRadioButton	Permite el acceso mediante programación al elemento HTML <input type= radio> del servidor.
HtmlInputText	Permite el acceso mediante programación a los elementos HTML <input type= text> y <input type= password> del servidor.
HtmlSelect	Permite el acceso mediante programación al elemento HTML <select> del servidor.
HtmlTable	Permite obtener acceso mediante programación al elemento HTML <table> en el servidor.
HtmlTableCell	Representa los elementos HTML <td> y <th> de un objeto HtmlTableRow .
HtmlTableCellCollection	Colección de objetos HtmlTableCell que representan las celdas de una sola fila de un control HtmlTable . No se puede heredar esta clase.
HtmlTableRow	Representa el elemento HTML <tr> de un control HtmlTable .
HtmlTableRowCollection	Colección de objetos HtmlTableRow que representan las filas de un control HtmlTable . No se puede heredar esta clase.
HtmlTextArea	Permite el acceso mediante programación al elemento HTML <textarea> del servidor

Espacio de nombres System.Web.UI.WebControls

Contiene clases para crear **controles de servidor Web ASP.NET**. Cuando se agregan a un formulario Web Forms, estos controles **proporcionan código HTML específico del explorador** y secuencias de comandos que permiten crear interfaces Web de usuario **independientes del dispositivo**.

El espacio de nombres System.Web.UI.WebControls es una colección de clases que permiten crear controles de servidor Web en una página Web. Los controles de servidor Web se ejecutan en el servidor e incluye controles de formulario tales como botones y cuadros de texto. Incluyen asimismo controles para usos especiales, como por ejemplo un calendario. Como los controles de servidor Web se ejecutan en el servidor, dichos elementos se pueden controlar mediante programación. **Los controles de servidor Web son más abstractos que los controles de servidor HTML**. Su modelo de objetos no refleja necesariamente la sintaxis HTML.

Las clases asociadas con este espacio de nombres son:

Clase	Descripción
AdCreatedEventArgs	Proporciona datos para el evento AdCreated del control AdRotator . No se puede heredar esta clase.
AdRotator	Muestra un titular de anuncio en una página Web.
BaseCompareValidator	Sirve como clase base abstracta para controles de validación que realizan comparaciones con tipos.
BaseDataList	Actúa como clase base abstracta para los controles de lista de datos, como DataList y DataGrid . Esta clase proporciona los métodos y propiedades comunes para todos los controles de lista de datos.
BaseValidator	Sirve como clase base abstracta para controles de validación.
BoundColumn	Tipo de columna para el control DataGrid enlazado a un campo en un origen de datos.
Button	Muestra un control de botón de comando en la página Web.
ButtonColumn	Tipo de columna para el control DataGrid que contiene un botón de comando definido por el usuario, como Add o Remove, y que corresponde a cada fila de la columna.
Calendar	Muestra un calendario de un solo mes que permite al usuario seleccionar fechas y desplazarse al mes siguiente o al mes anterior.
CalendarDay	Representa una fecha en el control Calendar .
CheckBox	Muestra una casilla de verificación que permite al usuario seleccionar una condición true o false.
CheckBoxList	Crea un grupo de casillas de verificación de selección múltiple cuya creación podría realizarse de forma dinámica enlazando el control al origen de datos.
CommandEventArgs	Proporciona datos para el evento Command.
CompareValidator	Compara el valor especificado por el usuario en un control de entrada con el valor especificado en otro control de entrada o con un valor constante.
CustomValidator	Realiza una validación definida por el usuario en un control de entrada.
DataGrid	Control de lista enlazada a datos que muestra los elementos del origen de datos en una tabla. El control DataGrid permite seleccionar, ordenar y editar estos elementos.
DataGridColumn	Sirve como clase base de los diferentes tipos de columna del control DataGrid .
DataGridColumnCollection	Colección de objetos Column derivados de DataGridColumn que representan las columnas de un control DataGrid . No se puede heredar esta clase.
DataGridCommandEventArgs	Proporciona datos para los eventos CancelCommand , DeleteCommand , EditCommand , ItemCommand y UpdateCommand del control DataGrid . No se puede heredar esta clase.
DataGridItem	Representa un elemento (fila) del control DataGrid .
DataGridItemCollection	Representa una colección de objetos de DataGridItem en un control DataGrid .
DataGridItemEventArgs	Proporciona datos para los eventos ItemCreated y ItemDataBound del control DataGrid . No se puede heredar esta clase.
DataGridPageChangedEventArgs	Proporciona datos para el evento PageIndexChanged del control DataGrid . No se puede heredar esta clase.
DataGridPagerStyle	Especifica el estilo de los localizadores del control DataGrid . No se puede heredar esta clase.
DataGridSortCommandEventArgs	Proporciona datos para el evento SortCommand del control DataGrid . No se puede heredar esta clase.
DataKeyCollection	Representa una colección que contiene el campo clave de cada registro de un origen de datos. No se puede heredar esta clase.

DataList	Control de lista enlazada a datos que muestra los elementos mediante el uso de plantillas.
DataListCommandEventArgs	Proporciona datos para los eventos CancelCommand , DeleteCommand , EditCommand , ItemCommand y UpdateCommand del control DataList . No se puede heredar esta clase.
DataListItem	Representa un elemento de un control DataList .
DataListItemCollection	Representa la colección de objetos DataListItem en el control DataList . No se puede heredar esta clase.
DataListItemEventArgs	Proporciona datos para los eventos ItemCreated y ItemDataBound de un control DataList . No se puede heredar esta clase.
DayRenderEventArgs	Proporciona datos para el evento DayRender del control Calendar . No se puede heredar esta clase.
DropDownList	Representa un control que permite al usuario seleccionar un único elemento de una lista desplegable.
EditCommandColumn	Tipo de columna especial para el control DataGrid que contiene los botones de comando Editar para editar los elementos de datos en cada fila.
FontInfo	Encapsula las propiedades de fuente del texto. No se puede heredar esta clase.
FontNamesConverter	Convierte una cadena que contiene una lista de nombres de fuente en una matriz de cadenas que contiene los nombres individuales. También realiza la función inversa.
FontUnitConverter	Convierte un FontUnit en un objeto con un tipo de datos distinto. También convierte un objeto con un tipo de datos distinto en un FontUnit .
HyperLink	Control que muestra un vínculo a otra página Web.
HyperLinkColumn	Tipo de columna para el control DataGrid que contiene un hipervínculo para cada elemento de la columna.
HyperLinkControlBuilder	Interactúa con el analizador para generar un control HyperLink .
Image	Muestra una imagen en una página Web.
ImageButton	Control que muestra una imagen y responde a los clicks del mouse (ratón) en la imagen.
Label	Representa un control de etiqueta que muestra texto en una página Web.
LabelControlBuilder	Interactúa con el analizador para generar un control Label .
LinkButton	Muestra un control de botón de hipervínculo en una página Web.
LinkButtonControlBuilder	Interactúa con el analizador para generar un control LinkButton .
ListBox	Representa un control de cuadro de lista que permite la selección de uno o varios elementos.
ListControl	Actúa como la clase base abstracta que define las propiedades, métodos y eventos comunes a todos los controles de tipo de lista.
ListItem	Representa un elemento de datos en un control de lista con enlace de datos. No se puede heredar esta clase.
ListItemCollection	Colección de objetos ListItem de un control de lista. No se puede heredar esta clase.
ListItemControlBuilder	Interactúa con el analizador para crear un control ListItem .
Literal	Reserva una ubicación en la página Web para mostrar texto estático.
LiteralControlBuilder	Interactúa con el analizador para crear un control Literal .
Table	Muestra una tabla en una página Web.
TableCell	Representa una celda en un control Table .

TableCellCollection	Encapsula una colección de objetos TableHeaderCell y TableCell que forman una fila en un control Table . No se puede heredar esta clase.
TableCellControlBuilder	Interactúa con el analizador para crear un control TableCell .
TableHeaderCell	Representa una celda de encabezado en un control Table .
TableItemStyle	Representa las propiedades de estilo de un elemento de un control que se representa como TableRow o TableCell .
TableRow	Representa una fila de un control Table .
TableRowCollection	Encapsula una colección de objetos TableRow que representa una única fila de un control Table . No se puede heredar esta clase.
TableStyle	Representa el estilo de un control de tabla. Esta clase la usan principalmente los programadores de controles.
TargetConverter	Convierte un valor que representa la ubicación (destino) en la que se debe mostrar el contenido resultante de una exploración en el Web en una cadena. Asimismo, convierte una cadena en un valor de destino.
TemplateColumn	Representa un tipo de columna para el control DataGrid que permite personalizar el diseño de los controles de la columna.
TextBox	Muestra un control de cuadro de texto para la introducción de datos del usuario.
TextBoxControlBuilder	Interactúa con el analizador para crear un control TextBox .
UnitConverter	Convierte un Unit en un objeto con un tipo de datos distinto. También convierte un objeto con un tipo de datos distinto en un Unit.
ValidatedControlConverter	Convierte un control de la página Web Forms que puede validarse mediante un control de validación, en una cadena.
ValidationSummary	Muestra un resumen de todos los errores de validación en línea en una página Web, en un cuadro de mensaje o en ambos.
WebColorConverter	Convierte un nombre de color predefinido o un valor de color RGB en System.Drawing.Color y viceversa.
WebControl	Actúa como la clase base que define las propiedades, métodos y eventos comunes a todos los controles del espacio de nombres System.Web.UI.WebControls .
Xml	Muestra un documento XML sin formato o que utiliza Extensible Stylesheet Language Transformations (XSLT).

5 CREANDO UNA PÁGINA BÁSICA

Las páginas WEB en ASP.NET son archivos de texto con la extensión . Una página ASP.NET puede ser creada simplemente tomando un archivo HTML existente y cambiando su extensión a **.aspx**. No es requerida ninguna modificación al código de la página.

En el siguiente ejemplo se muestra el archivo .aspx con instrucciones .html para generar una página WEB con un texto de entrada conteniendo un Nombre, una lista de selección para escoger la categoría de desarrollo de aplicaciones WEB y un botón donde se acepta la selección y se envía la respuesta al servidor.

Nombre: Categoría:

```
<html>

<head>

    <link rel="stylesheet" href="pagSimple.css">

</head>

<body>

    <center>

        <form action="pagSimple.aspx" method="post">

            <h3> Nombre: <input id="Nombre" type="text">

                Categoria: <select id="Categoria" size=1>

                    <option>ASP.NET</option>

                    <option>JSP</option>

                    <option>PHP</option>

                    <option>Perl</option>

                </select>

                <input type="submit" value="Escoger">

            </h3>

        </form>

    </center>

</body>

</html>
```

Nótese que nada ocurre cuando se pulsa el botón Escoger. Esto se debe a que html sólo contiene el **código estático** de la página WEB (no contiene contenido dinámico) . Para añadir **contenido dinámico** en html se deben incluir directivas de página, código en lenguajes de script como asp, jsp, javascripts, php, perl entre otros. **En ASP.NET se soporta ASP**. En el caso de ASP los bloques de código dinámico se especifican con la sintaxis **<% código %>**.

6. DIRECTIVAS DE PÁGINA

Mostrando variables de programas en páginas WEB

Para colocar directivas de página en ASP.NET debe codificarse código en la sintaxis de ASP e incorporarlo en el archivo **.aspx**. De esta manera se añade dinamismo a la página WEB, en el caso de ASP los bloques de código dinámico se especifican con la sintaxis **<% código %>**.

En el siguiente ejemplo se muestra la generación dinámica de texto con tamaños para el font , los cuales dependen de una **variable** y se utiliza el lenguaje de programación VB.Net . En este ejemplo en particular se emplean las variables de un lenguaje de programación para mostrar contenido en una página WEB.

Nombre:	<input type="text"/>	Categoría:	ASP.NET	Escoger
Bienvenido		a		ASP.Net
Bienvenido		a		ASP.Net
Bienvenido		a		ASP.Net
Bienvenido a ASP.Net				

```
<%@ Page Language="VB" %>

<html>

  <head>

    <link rel="stylesheet" href="pagSimple.css">

  </head>

  <body>

    <center>

      <form action="pagDirect.aspx" method="post">

        <h3> Nombre: <input id="Nombre" type="text">

        Categoría: <select id="Categoría" size=1>

          <option>ASP.NET</option>

          <option>JSP</option>

          <option>PHP</option>

          <option>Perl</option>

        </select>

        <input type="submit" value="Escoger">
```

```
</h3>

<p>

<% Dim i As Integer

    For i = 2 to 5 %>

        <font size="<%=i%>"> Bienvenido a ASP.Net </font> <br>

    <% Next i%>

</form>


</center>

</body>


</html>
```

Utilizando en los programas valores de objetos HTML

En el siguiente ejemplo se utilizan los códigos de bloque para interpretar información contenida en los **componentes html**. En este caso se interpretan los resultados enviados por el cliente al seleccionar una alternativa y pulsar el botón Escoger.

Nombre: Categoría: 

Una vez presionado el botón escoger se muestra como resultado en página

Nombre: Categoría: 

Gracias GALA, usted Selecciono: ASP.NET

```
<%@ Page Language="VB" %>

<html>

<head>

    <link rel="stylesheet"href="intro.css">

</head>

<body>
```

```
<center>

<form action="pagResCli.aspx">

    <h3>        Nombre:    <input        name="Nombre"        type=text
value="<%=HttpUtility.HtmlEncode(Request.QueryString("Nombre"))%>">

    Categoria:  <select name="Categoria" size=1>

        <%

            Dim i As Integer

            Dim selTxt(4) As String

            selTxt(0) = "ASP.NET"

            selTxt(1) = "JSP"

            selTxt(2) = "PHP"

            selTxt(3) = "Perl"

        For i = 0 To 3

            %>

            <% If (Request.QueryString("Categoria") = selTxt(i)) %>

                <option selected>

            <% Else %>

                <option>

            <% End If %>

            <%=selTxt(i)%>

            </option>
```



```
<% Next %>

</select>

</h3>

<input type=submit name="Escoger" value="Escoger">

<p>

<% If (Not Request.QueryString("Escoger") = Nothing) %>

    Gracias      <%=HttpUtility.HtmlEncode(Request.QueryString("Nombre"))%>
%,              usted                               Selecciono:
<%=HttpUtility.HtmlEncode(Request.QueryString("Categoria")) %>

    <% End If %>

</form>

</center>

</body>


</html>
```

En este ejemplo se utilizan los objetos Request con el método QueryString, el cual recibe como parámetro el nombre del objeto HTML. Por ejemplo para obtener el valor de la categoría seleccionada se emplea **Request.QueryString("Categoria")**

Controles ASP ejecutándose en el servidor de WEB

Adicionalmente a la utilización de los bloques de código **<% %>** para generar y programar contenido dinámico, los programadores ASP.NET pueden utilizar **controladores del servidor** para programar las páginas WEB. Los controladores del servidor son declarados en un archivo .aspx empleando **tags personalizados ("custom tags")** o tags intrínsecos HTML conteniendo un atributo con valor **runat="server"**. Los tags intrínsecos HTML son manejados por los controles definidos en el espacio de nombres **System.Web.UI.HtmlControls**. Si un tag no tiene correspondencia explícita con uno de los controles definidos en este espacio de nombres, la plataforma .NET le asigna como tipo **System.Web.UI.HtmlControls.HtmlGenericControl**.

El siguiente ejemplo utiliza cuatro controles de servidor: **<form runat=server>**, **<asp:textbox runat=server>**, **<asp:dropdownlist runat=server>**, y **<asp:button runat=server>**. En tiempo de ejecución, estos controles de servidor automáticamente genera contenido HTML.

Nombre: Categoría: 

```
<html>

  <head>

    <link rel="stylesheet"href="intro.css">

  </head>

  <body>

    <center>

      <form action="pagConServ.aspx" method="post" runat=server>

        <h3> Nombre: <asp:textbox id="Nombre" runat="server"/>

        Categoría: <asp:dropdownlist id="Categoría" runat=server>

          <asp:listitem >ASP.NET</asp:listitem>

          <asp:listitem >JSP</asp:listitem>

          <asp:listitem >PHP</asp:listitem>

          <asp:listitem >Perl</asp:listitem>

        </asp:dropdownlist>

      </h3>

      <asp:button text="Escoger" runat="server"/>

    </form>

  </center>

</body>

</html>
```

Manejando los eventos del control del servidor

Todos los controles de servidor de ASP.NET expone un modelo orientado por objetos indicando propiedades, métodos y eventos. Los desarrolladores en ASP.NET pueden utilizar este modelo de objetos para interactuar con la página web de una manera limpia y encapsulada.

El siguiente ejemplo demuestra como manejar el evento OnClick del botón (disparado al presionar el botón) **<asp:button runat=server>** para utilizar la propiedad Text de la etiqueta **<asp:label runat=server>**.

Nombre: Categoría: ASP.NET ▼

Al presionar el botón Seleccionar se produce como resultado

Nombre: Categoría: ASP.NET ▼

Bienvenido GALA, usted seleccionó: ASP.NET

```
<html>

<head>

    <link rel="stylesheet"href="intro.css">

</head>

<script language="VB" runat=server>

    Sub bSeleccionar_Click(Sender As Object, E As EventArgs)

        Mensaje.Text = "Bienvenido " & HttpUtility.HtmlEncode(Nombre.Text)
        & ", usted seleccionó: " & Categoria.SelectedItem.Text

    End Sub

</script>

<body>

    <center>

        <form action="pagEventServ.aspx" method="post" runat="server">

            <h3> Nombre: <asp:textbox id="Nombre" runat="server"/>

            Categoría: <asp:dropdownlist id="Categoria" runat=server>

                <asp:listitem >ASP.NET</asp:listitem>

                <asp:listitem >JSP</asp:listitem>
```

```
<asp:listitem >PHP</asp:listitem>

<asp:listitem >Perl</asp:listitem>

</asp:dropdownlist>

</h3>

<asp:button      text="Seleccionar"      OnClick="bSeleccionar_Click"
runat="server"/>

<p>

<asp:label id="Mensaje" runat="server"/>

</form>

</center>

</body>

</html>
```

7 WEB.CONFIG

Formato de los archivos de configuración

Los archivos de configuración en ASP.NET son **archivos de texto basados en XML** —cada uno nombrado **web.config**--; estos archivos pueden ser ubicados en cualquier directorio del servidor de aplicaciones de ASP.NET. Cada archivo de configuración web.config permite aplicar las directivas de configuración en el directorio donde se encuentra y en todos los directorios hijos virtuales en la jerarquía de directorios. Las configuraciones realizadas en los directorios hijos pueden sobrescribir o modificar las configuraciones realizadas en los directorios padres.

El archivo de configuración raíz --WinNT\Microsoft.NET\Framework\<version>\config\machine.config— provee las directivas de configuración por defecto para toda la máquina y las aplicaciones. **ASP.NET configura IIS** para evitar el acceso de los archivos web.config desde el navegador, asegurándose que los valores de este archivo no sean públicos (cualquier intento en acceder a estos archivos producirá el error 403: Acceso prohibido).

En tiempo de ejecución, ASP.NET emplea los archivos de configuración web.config , para de una manera jerárquica, determinar una colección de parámetros de configuración para procesar un requerimiento a un URL (estos parámetros son determinados una sola vez y colocados en el “caché” para poder ser utilizados en las solicitudes subsiguientes; ASP.NET automáticamente detecta cualquier cambio en estos archivos de configuración invalidando el “caché” produciendo nuevos cálculos en los parámetros de configuración).

Por ejemplo, los parámetros de configuración para el URL para el URL <http://servidor/aplicacion/directorio/pagina.aspx> son determinados revisando los parámetros de configuración especificados en los archivos web.config en el siguiente orden:

a) Archivo base para la máquina.

<C:\WinNT\Microsoft.NET\Framework\v.1.00\config\machine.config>

b) Este archivo sobrescribe al a); y contiene los parámetros de configuración para el sitio raíz de la aplicación.

<C:\inetpub\wwwroot\web.config>

c) Este archivo sobrescribe al b); y contiene los parámetros de configuración específicos de la aplicación.

<C:\Aplicacion\web.config>

d) Este archivo sobrescribe al c); y contiene los parámetros de configuración específicos para un directorio de la aplicación.

<C:\Aplicacion\Directorio\web.config>

Si un archivo web.config se encuentra el directorio raíz del sitio, por ejemplo "inetpub\wwwroot", los parámetros de configuración de este archivo aplican a todas las aplicaciones residentes en el sitio. Es importante observar que la presencia de archivos web.config en un directorio es completamente opcional. Si un archivo web.config no se encuentra presente, todos los parámetros de configuración del directorio son heredados automáticamente del directorio padre.

El formato de los archivos de configuración y su edición será cubierto con lujo de detalles en la sección denominada "Archivo de Configuración" del presente manual.

8 DATALIST

Introducción a "Data Binding" (enlace de datos)

ASP.NET introduce una nueva sintaxis **declarativa** para efectuar "**data binding**". Esta nueva sintaxis es muy flexible y permite al desarrollador no solo realizar "binding" con fuentes de datos, sino también con propiedades, colecciones, expresiones e inclusive resultados obtenidos de llamadas a métodos. En la siguiente tabla se muestran ejemplos de esta sintaxis.

Propiedad	Cliente: <%# clienteID %>
Colección	Ordenes: <asp:ListBox id="ListaOrdenes" datasource='<%# arregloOrdenes %>' runat="server">
Expresión	Contacto: <%# (cliente.Nombre + " " + cliente.Apellido) %>
Resultado de un método	Balance: <%# obtBalance(clienteID) %>

Aunque esta sintaxis es parecida al "shortcut" para **Response.Write** -- <%= %> -- el comportamiento es un poco diferente. Mientras el "shortcut" para **Response.Write** shortcut es evaluado cuando la página es procesada, la sintaxis para ASP.NET "binding" es evaluada únicamente cuando el método **DataBind** es invocado.

DataBind es un método del objeto **Page** y de todos los controles de servidor. Cuando se efectúa una llamada a **DataBind** en un control padre este se propaga en cascada para todos los hijos del control. Por ejemplo, `DataList1.DataBind()` invoca al método **DataBind** en cada uno de los controles del template de **DataList**. Invocar **DataBind** en el objeto **Page** -- `Page.DataBind()` o simplemente `DataBind()` – produce como resultado la evaluación de todas las expresiones de data binding. **DataBind** es frecuentemente invocado desde el evento **Page_Load**, como se muestra en el siguiente ejemplo.

```
Protected Sub Page_Load(Src As Object, E As EventArgs)
```

```
    DataBind()
```

```
End Sub
```

Una expresión de binding puede ser utilizada prácticamente en cualquier lugar de la sección de declaraciones en la página **.aspx**, pero es importante considerar que el tipo de datos es provisto correctamente en tiempo de ejecución. En el ejemplo indicado en la tabla, cuando se utilizaron data binding para propiedades, expresiones y métodos se muestra texto al usuario (o aplicación cliente) cuando las expresiones son evaluadas. En estos casos, la expresión de data binding debe ser evaluada en un valor del tipo **String**. En el caso de la colección, la expresión de data binding evalúa a un valor de un tipo válido para la propiedad **DataSource** del **ListBox**. Existen casos donde es necesario realizar una coerción de tipos para producir el resultado deseado. Por ejemplo si contador es una variable del tipo integer:

Numero de Registros: `< %# contador.ToString() %>`

DataBinder.Eval

La plataforma ASP.NET ofrece un método estático (no requiere instanciación) para evaluar un data binding in **“late bound”** (a tiempo de ejecución), empleando expresiones de data binding y opcionalmente formatea el resultado como un **String**. **DataBinder.Eval** es un método muy útil y no es necesario en muchas ocasiones realizar un **“casting”** explícito para realizar la coerción entre los tipos de datos a nivel de programación. Es muy utilizado en el caso de controles con data binding (como el cubierto en esta sección y el **Repeater**) trabajando con una lista en un template, debido a que los datos contenidos en un registro y el tipo de datos deben ser “casteados”.

DataBinder.Eval es un método con tres argumentos: el contenedor de los datos (items), el nombre del campo de datos y un string de formateo. En un control como el **DataList**, el **DataGrid**, o el **Repeater**, el container es siempre **Container.DataItem**. **Page** es otro container que puede ser utilizado con **DataBinder.Eval**.

El string de formateo es opcional. Si es omitido, **DataBinder.Eval** retorna un valor del tipo **Object**.

Mostrando los datos de una tabla de base de datos en un datalist (ejemplo introductorio)

En el código del `Page_Load` se coloca el siguiente fragmento de código

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
```

```
    'Introducir aquí el código de usuario para inicializar la página
```

```
    Dim DS As DataSet
```

```
    DS = New DataSet
```

```
confDA.Fill(DS, "Empleados") 'Objeto data adapter y nombre de  
                                'Objeto  
  
'El objeto del tipo data list se llamao confDL, debe  
  
'coincidir con el id descrito en el archivo HTML  
  
confDL.DataSource = DS.Tables("Empleados") 'asociando la fuente  
  
                                ' al datalist  
  
confDL.DataBind()' invocando el método DataBind, para acceder  
  
                                ' a los datos  
  
End Sub
```

El código HTML asociado con el ejemplo se muestra a continuación

```
<asp:DataList id="confDL" style="Z-INDEX: 101; LEFT: 8px; POSITION: absolute; TOP: 24px"  
runat="server">  
  
<HeaderTemplate>  
  
    <table width="100%" style="font: 8pt verdana">  
  
        <tr style="background-color:DFA894">  
  
            <th>  
  
                Codigo  
  
            </th>  
  
            <th>  
  
                Nombre  
  
            </th>  
  
            <th>  
  
                Apellido  
  
            </th>  
  
            <th>  
  
                Direccion  
  
            </th>  
  
        </tr>
```

```
</HeaderTemplate>

<ItemTemplate>

    <tr style="background-color:FFECD8">

        <td>

            <%# DataBinder.Eval(Container.DataItem, "codigo") %>

        </td>

        <td>

            <%# DataBinder.Eval(Container.DataItem, "nombre") %>

        </td>

        <td>

            <%# DataBinder.Eval(Container.DataItem, "apellido") %>

        </td>

        <td>

            <%# DataBinder.Eval(Container.DataItem, "direccion") %>

        </td>

    </tr>

</ItemTemplate>

<FooterTemplate>

    </table>

</FooterTemplate>

</asp:DataList>
```

</ASP:DataList>El parámetro **id** contiene el nombre del objeto en la página WEB y en el código desarrollado en VB.Net. El objeto **datalist** contiene **tres templates**, el encabezado ("header"), el pie de página ("footer") y los elementos a mostrar ("item"). Las instrucciones que muestran los valores de los campos de las tablas son las siguientes, se debe colocar el nombre del campo como un String.

<%#**DataBinder.Eval(Container.DataItem, "codigo")**%>

<%#DataBinder.Eval(Container.DataItem, "nombre")%>

<%#DataBinder.Eval(Container.DataItem, "apellido")%>

<%#DataBinder.Eval(Container.DataItem, "direccion")%>

Clase DataList

Control de **lista enlazada a datos** que muestra los elementos mediante el uso de plantillas. Se encuentra en el espacio de nombres **System.Web.UI.WebControls** (Espacio de nombres)

Utilice el control **DataList** para mostrar una lista enlazada a datos definida por una plantilla. El control **DataList** admite la selección y la edición.

El contenido del control **DataList** puede manipularse mediante el uso de plantillas. En la siguiente tabla se enumeran las plantillas compatibles.

Nombre de la plantilla	Descripción
AlternatingItemTemplate	Si se define, determina el contenido y el diseño de los elementos alternos del control DataList . Si no se define, se utiliza ItemTemplate .
EditItemTemplate	Si se define, determina el contenido y el diseño del elemento que se edita en el control DataList . Si no se define, se utiliza ItemTemplate .
FooterTemplate	Si se define, determina el contenido y el diseño de la sección de pie de página del control DataList . Si no se define, no se mostrará ninguna sección de pie de página.
HeaderTemplate	Si se define, determina el contenido y el diseño de la sección de encabezado del control DataList . Si no se define, no se mostrará ninguna sección de encabezado.
ItemTemplate	Plantilla necesaria que determina el contenido y el diseño de los elementos alternos del control DataList .
SelectedItemTemplate	Si se define, determina el contenido y el diseño del elemento seleccionado actual del control DataList . Si no se define, se utiliza ItemTemplate .
SeparatorTemplate	Si se define, determina el contenido y el diseño del separador de elementos del control DataList . Si no se define, no se mostrará ningún separador.

Campos de la clase datalist

A continuación se anexan los campos para esta clase

Constructores públicos

DataList (Constructor)	Inicializa una nueva instancia de la clase DataList .
------------------------	--

Propiedades públicas

AccessKey (se hereda de WebControl)	Obtiene o establece la clave de acceso (la letra subrayada) que permite desplazarse rápidamente al control de servidor Web.
AlternatingItemStyle	Obtiene las propiedades de estilo de los elementos alternos del control DataList .
AlternatingItemTemplate	Obtiene o establece la plantilla para los elementos alternos del control DataList .
Attributes (se hereda de WebControl)	Obtiene la colección de atributos arbitrarios (sólo para procesar) que no corresponden a propiedades del control.
BackColor (se hereda de WebControl)	Obtiene o establece el color de fondo del control de servidor Web.
BorderColor (se hereda de WebControl)	Obtiene o establece el color de borde del control Web.

BorderStyle (se hereda de WebControl)	Obtiene o establece el estilo del borde del control de servidor Web.
BorderWidth (se hereda de WebControl)	Obtiene o establece el ancho del borde del control de servidor Web.
CellPadding (se hereda de BaseDataList)	Obtiene o establece la cantidad de espacio entre el contenido de una celda y el borde de la misma.
CellSpacing (se hereda de BaseDataList)	Obtiene o establece la cantidad de espacio entre las celdas.
ClientID (se hereda de Control)	Obtiene el identificador del control de servidor generado por ASP.NET.
Controls (se hereda de BaseDataList)	Reemplazado. Obtiene un System.Web.UI.ControlCollection que contiene una colección de controles secundarios en un control de lista de datos.
ControlStyle (se hereda de WebControl)	Obtiene el estilo del control de servidor Web. Esta propiedad la usan principalmente los programadores de controles.
ControlStyleCreated (se hereda de WebControl)	Obtiene un valor que indica si se ha creado un objeto Style para la propiedad ControlStyle . Esta propiedad la usan principalmente los programadores de controles.
CssClass (se hereda de WebControl)	Obtiene o establece la clase hoja de estilo en cascada (CSS) procesada por el control de servidor Web en el cliente.
DataKeyField (se hereda de BaseDataList)	Obtiene o establece el campo clave del origen de datos especificado por la propiedad DataSource .
DataKeys (se hereda de BaseDataList)	Obtiene un DataKeyCollection que almacena los valores clave de cada registro (que se muestra en forma de fila) de un control de lista de datos.
DataMember (se hereda de BaseDataList)	Obtiene o establece, en un origen de datos con varios miembros, el miembro de datos específico que se debe enlazar con un control de lista de datos.
DataSource (se hereda de BaseDataList)	Obtiene o establece el origen que contiene una lista de valores que se utiliza para rellenar los elementos contenidos en el control.
EditItemIndex	Obtiene o establece el número de índice del elemento del control DataList seleccionado para su edición.
EditItemStyle	Obtiene las propiedades de estilo del elemento del control DataList seleccionado para su edición.
EditItemTemplate	Obtiene o establece la plantilla para el elemento del control DataList seleccionado para su edición.
Enabled (se hereda de WebControl)	Obtiene o establece un valor que indica si el control de servidor Web está habilitado.
EnableViewState (se hereda de Control)	Obtiene o establece un valor que indica si el control de servidor debe mantener su estado de vista y el de los controles secundarios que contiene, en el cliente que realiza la solicitud.
ExtractTemplateRows	Obtiene o establece un valor que indica si se extraen y se muestran las filas de un control Table , definido en cada plantilla de un control DataList .
Font (se hereda de WebControl)	Obtiene las propiedades de fuente asociadas al control de servidor Web.
FooterStyle	Obtiene las propiedades de estilo de la sección de pie de página del control DataList .
FooterTemplate	Obtiene o establece la plantilla para la sección de pie de página del control DataList .
ForeColor (se hereda de WebControl)	Obtiene o establece el color de primer plano (normalmente el color del texto) del control de servidor Web.
GridLines	Reemplazado. Obtiene o establece el estilo de la línea de la cuadrícula del control DataList cuando la propiedad RepeatLayout se establece en RepeatLayout.Table .
HeaderStyle	Obtiene las propiedades de estilo de la sección de encabezado del control DataList .
HeaderTemplate	Obtiene o establece la plantilla para la sección de

	encabezado del control DataList .
Height (se hereda de WebControl)	Obtiene o establece el alto del control de servidor Web.
HorizontalAlign (se hereda de BaseDataList)	Obtiene o establece la alineación horizontal de un control de lista de datos en su contenedor.
ID (se hereda de Control)	Obtiene o establece el identificador de programación asignado al control de servidor.
Items	Obtiene una colección de objetos DataListItem que representan los elementos individuales dentro del control.
ItemStyle	Obtiene las propiedades de estilo para los elementos del control DataList .
ItemTemplate	Obtiene o establece la plantilla para los elementos del control DataList .
NamingContainer (se hereda de Control)	Obtiene una referencia al contenedor de nombres del control de servidor, que crea un espacio de nombres único para diferenciar los distintos controles de servidor que tienen el mismo valor para la propiedad Control.ID .
Page (se hereda de Control)	Obtiene una referencia a la instancia Page que contiene el control de servidor.
Parent (se hereda de Control)	Obtiene una referencia al control principal del control de servidor en la jerarquía de controles de página.
RepeatColumns	Obtiene o establece el número de columnas que se muestran en el control DataList .
RepeatDirection	Obtiene o establece si el control DataList se muestra vertical u horizontalmente.
RepeatLayout	Obtiene o establece si el control se muestra en un diseño de tabla o de flujo.
SelectedIndex	Obtiene o establece el índice del elemento seleccionado del control DataList .
SelectedItem	Obtiene el elemento seleccionado del control DataList .
SelectedItemStyle	Obtiene las propiedades de estilo del elemento seleccionado del control DataList .
SelectedItemTemplate	Obtiene o establece la plantilla para el elemento seleccionado del control DataList .
SeparatorStyle	Obtiene las propiedades de estilo del separador entre cada elemento del control DataList .
SeparatorTemplate	Obtiene o establece la plantilla para el separador de elementos del control DataList .
ShowFooter	Obtiene o establece un valor que indica si se muestra la sección de pie de página del control DataList .
ShowHeader	Obtiene o establece un valor que indica si se muestra la sección de encabezado del control DataList .
Site (se hereda de Control)	Obtiene información sobre el sitio Web al que pertenece el control de servidor.
Style (se hereda de WebControl)	Obtiene una colección de atributos de texto que se procesan como un atributo de estilo en la etiqueta externa del control de servidor Web.
TabIndex (se hereda de WebControl)	Obtiene o establece el índice de tabulación del control de servidor Web.
TemplateSourceDirectory (se hereda de Control)	Obtiene el directorio virtual de Page o UserControl que contiene el control de servidor actual.
ToolTip (se hereda de WebControl)	Obtiene o establece el texto que se muestra cuando el puntero del mouse (ratón) se desplaza sobre el control de servidor Web.
UniqueID (se hereda de Control)	Obtiene el identificador único cualificado jerárquicamente del control de servidor.
Visible (se hereda de Control)	Obtiene o establece un valor que indica si un control de servidor se debe procesar como interfaz de usuario en la página.
Width (se hereda de WebControl)	Obtiene o establece el ancho del control de servidor Web.

Métodos públicos

ApplyStyle (se hereda de WebControl)	Copia en el control Web todos los elementos no vacíos del estilo especificado, sobrescribiendo los elementos de estilo existentes del mismo. Este método lo usan principalmente los programadores de controles.
CopyBaseAttributes (se hereda de WebControl)	Copia las propiedades no encapsuladas por el objeto Style del control de servidor Web especificado al control de servidor Web desde el que se efectúa la llamada al método. Este método lo usan principalmente los programadores de controles.
DataBind (se hereda de BaseDataList)	Reemplazado. Enlaza el control y todos sus controles secundarios al origen de datos especificado por la propiedad DataSource .
Dispose (se hereda de Control)	Habilita un control de servidor para que realice la limpieza final antes de que se libere de la memoria.
Equals (se hereda de Object)	Sobrecargado. Determina si dos instancias de Object son iguales.
FindControl (se hereda de Control)	Sobrecargado. Busca el control de servidor especificado en el contenedor de nombres actual.
GetHashCode (se hereda de Object)	Sirve como función hash para un tipo concreto, apropiado para su utilización en algoritmos de hash y estructuras de datos como las tablas hash.
GetType (se hereda de Object)	Obtiene el objeto Type de la instancia actual.
HasControls (se hereda de Control)	Determina si el control de servidor contiene controles secundarios.
MergeStyle (se hereda de WebControl)	Copia en el control Web todos los elementos no vacíos del estilo especificado, pero no sobrescribe los elementos de estilo existentes en el mismo. Este método lo usan principalmente los programadores de controles.
RenderBeginTag (se hereda de WebControl)	Procesa la etiqueta HTML de apertura del control en el escritor especificado. Este método lo usan principalmente los programadores de controles.
RenderControl (se hereda de Control)	Envía el contenido del control de servidor a un objeto HtmlTextWriter que se proporciona y almacena la información de seguimiento sobre el control si dicho seguimiento está habilitado.
RenderEndTag (se hereda de WebControl)	Procesa la etiqueta HTML de cierre del control en el escritor especificado. Este método lo usan principalmente los programadores de controles.
ResolveUrl (se hereda de Control)	Convierte una dirección URL en una que el cliente solicitante pueda utilizar.
ToString (se hereda de Object)	Devuelve un objeto String que representa al objeto Object actual.

Eventos públicos

CancelCommand	Se produce cuando se hace clic en el botón Cancelar de un elemento del control DataList .
DataBinding (se hereda de Control)	Se produce cuando el control de servidor se enlaza a un origen de datos.
DeleteCommand	Se produce cuando se hace clic en el botón Eliminar de un elemento del control DataList .
Disposed (se hereda de Control)	Se produce cuando un control de servidor se libera de la memoria, lo que constituye la última fase del período de duración de un control de servidor cuando se solicita una página ASP.NET.
EditCommand	Se produce cuando se hace clic en el botón Editar de un elemento del control DataList .
Init (se hereda de Control)	Se produce cuando el control de servidor se inicializa, lo que constituye la primera fase de su período de duración.
ItemCommand	Se produce cuando se hace clic en cualquier botón del control DataList .
ItemCreated	Se produce en el servidor cuando se crea un elemento en

	el control DataList .
ItemDataBound	Se produce cuando un elemento tiene datos enlazados al control DataList .
Load (se hereda de Control)	Se produce cuando el control de servidor se carga en el objeto Page .
PreRender (se hereda de Control)	Se produce cuando el control de servidor se va a procesar en el objeto Page que lo contiene.
SelectedIndexChanged (se hereda de BaseDataList)	Tiene lugar cuando se selecciona un elemento distinto de un control de lista de datos entre cada envío al servidor.
Unload (se hereda de Control)	Se produce cuando el control de servidor se descarga de la memoria.
UpdateCommand	Se produce cuando se hace clic en el botón Actualizar de un elemento del control DataList .

9 DATAGRID

Mostrando los datos de una tabla de base de datos en un DataGrid (ejemplo introductorio)

En el código del **Page_Load** se coloca el siguiente fragmento de código

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

    Dim DS As DataSet

    DS = New DataSet

    confDA.Fill(DS, "Empleados") 'Objeto data adapter y nombre de
    'Objeto

    'El objeto del tipo Data Grid se llama confDG, debe
    'coincidir con el id descrito en el archivo HTML

    confDG.DataSource = DS.Tables("Empleados") 'asociando la fuente
    ' al datagrid

    confDG.DataBind() ' invocando el método DataBind, para acceder
    ' a los datos
```

End Sub

El código HTML asociado con el ejemplo se muestra a continuación

```
<HTML>

    <HEAD>

        <title>WebForm1 </title>

        <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.1">

        <meta name="CODE_LANGUAGE" content="Visual Basic .NET 7.1">

        <meta name="vs_defaultClientScript" content="JavaScript">

        <meta
content="http://schemas.microsoft.com/intellisense/ie5">                                name="vs_targetSchema"

    </HEAD>

    <body MS_POSITIONING="GridLayout">

        <form id="Form1" method="post" runat="server">

            <asp:DataGrid id="confDG" style="Z-INDEX: 101; LEFT: 8px;
POSITION: absolute; TOP: 8px" runat="server"></asp:DataGrid>

        </form>

    </body>

</HTML>
```

El parámetro **id** contiene el nombre del objeto en la página WEB y en el código desarrollado en VB.Net. El objeto **DataGrid** contiene tres templates, en la sección relacionada con los templates se especificará la funcionalidad asociada a cada uno de estos; pero únicamente colocando la instrucción **databind** en el método **PageLoad** es suficiente para mostrar los datos de una manera plana en pantalla, produciéndose como salida

codigo	nombre	apellido	Direccion
Emp1	N1	A1	D1
Emp2	N2	A2	D2
Emp3	N3	A3	D3

Clase DataGrid

Control de lista enlazada a datos que muestra los elementos del origen de datos en una tabla. El control **DataGrid** permite seleccionar, ordenar y editar estos elementos.

La clase DataGrid se encuentra en el **Espacio de nombres**: [System.Web.UI.WebControls](#)

[System.Object](#)

[System.Web.UI.Control](#)

[System.Web.UI.WebControls.WebControl](#)

[System.Web.UI.WebControls.BaseDataList](#)

System.Web.UI.WebControls.DataGrid

Public Class DataGrid

Inherits BaseDataList

Implements INamingContainer

Comentarios

Utilice el control **DataGrid** para mostrar los campos de un origen de datos como columnas en una tabla. Cada fila del control **DataGrid** representa un registro del origen de datos. El control **DataGrid** admite las operaciones de seleccionar, editar, eliminar, paginar y ordenar.

Diferentes tipos de columna determinan el comportamiento de las columnas del control. La siguiente tabla contiene los diferentes tipos de columna que se pueden utilizar.

Tipo de columna	Descripción
BoundColumn	Muestra una columna enlazada a un campo de un origen de datos. Muestra cada elemento del campo como texto. Éste es el tipo de columna predeterminado del control DataGrid .
ButtonColumn	Muestra un botón de comando para cada elemento de la columna. Esto permite crear una columna de controles de botón personalizados, como los botones Agregar o Quitar .
EditCommandColumn	Muestra una columna que contiene comandos de edición para cada elemento de la columna.
HyperLinkColumn	Muestra el contenido de cada elemento de la columna como un hipervínculo. El contenido de la columna puede enlazarse a un campo de un origen de datos o texto estático.
TemplateColumn	Muestra cada elemento de la columna según una plantilla especificada. Esto permite incluir controles personalizados en la columna.

De manera predeterminada, la propiedad [AutoGenerateColumns](#) está establecida en **true**, de modo que se crea un objeto **BoundColumn** para cada campo del origen de datos. A continuación, cada campo se procesa como una columna del control **DataGrid** en el orden de aparición de cada campo en el origen de datos.

También se puede controlar manualmente qué columnas aparecen en el control **DataGrid** estableciendo la propiedad **AutoGenerateColumns** en **false** y elaborando después una lista de las columnas que se desea incluir entre las etiquetas **<Columns>** de apertura y de cierre. Las columnas especificadas se agregan a la colección [Columns](#) en el orden en que aparecen en la lista. Esto permite controlar mediante programación las columnas del control **DataGrid**.

Nota El orden en que se muestran las columnas en el control **DataGrid** se rige por el orden en que aparecen las columnas en la colección **Columns**. Si bien se puede cambiar mediante programación el orden de las columnas manipulando la colección **Columns**, resulta más fácil mostrar las columnas en el orden de presentación deseado.

Se pueden mostrar columnas declaradas explícitamente junto con columnas generadas automáticamente. Cuando se utilizan ambos tipos de columnas, se procesan primero las columnas declaradas explícitamente y, a continuación, las columnas generadas automáticamente.

Nota Las columnas generadas automáticamente no se agregan a la colección **Columns**.

Se puede personalizar la apariencia del control **DataGrid** estableciendo las propiedades de estilo de las diferentes partes del control. En la siguiente tabla se muestran las diferentes propiedades de estilo.

Propiedad de estilo	Descripción
AlternatingItemStyle	Especifica el estilo de los elementos alternos del control DataGrid .

EditItemStyle	Especifica el estilo del elemento que se edita en el control DataGrid .
FooterStyle	Especifica el estilo de la sección de pie de página del control DataGrid .
HeaderStyle	Especifica el estilo de la sección de encabezado del control DataGrid .
ItemStyle	Especifica el estilo de los elementos del control DataGrid .
PagerStyle	Especifica el estilo de la sección de selección de página del control DataGrid .
SelectedItemStyle	Especifica el estilo del elemento seleccionado del control DataGrid .

También se pueden mostrar u ocultar diferentes partes del control. La siguiente tabla contiene las propiedades que controlan qué partes se muestran o se ocultan.

Propiedad	Descripción
ShowFooter	Muestra u oculta la sección de pie de página del control DataGrid .
ShowHeader	Muestra u oculta la sección de encabezado del control DataGrid .

También se puede controlar la apariencia del control **DataGrid** agregando mediante programación atributos a las etiquetas **<td>** y **<tr>** procesadas por el control en el explorador. Los atributos se pueden agregar mediante programación proporcionando código en el controlador de eventos para el evento [OnItemCreated](#) o [OnItemDataBound](#).

Para agregar un atributo a la etiqueta **<td>**, obtenga primero el objeto [TableCell](#) que representa la celda del control **DataGrid** a la que desea agregar el atributo. La colección [Control.Controls](#) de la propiedad **Item** del objeto [DataGridItemEventArgs](#) que se pasa al controlador de eventos puede utilizarse para obtener el objeto **TableCell** deseado. A continuación, se puede usar el método [AttributeCollection.Add](#) de la colección [Attributes](#) del objeto **TableCell** para agregar atributos a la etiqueta **<td>**.

Para agregar un atributo a la etiqueta **<tr>**, obtenga primero el objeto [DataGridItem](#) que representa la fila del control **DataGrid** a la que desea agregar el atributo. La propiedad **Item** del objeto [DataGridItemEventArgs](#) que se pasa al controlador de eventos puede utilizarse para obtener el objeto **DataGridItem** deseado. A continuación, se puede usar el método [AttributeCollection.Add](#) de la colección [Attributes](#) del objeto **DataGridItem** para agregar atributos a la etiqueta **<tr>**.

PRECAUCIÓN Este control se puede utilizar para mostrar los datos introducidos por el usuario, que pueden incluir secuencias de comandos de cliente maliciosas. Compruebe que cualquier información que se envíe desde un cliente no contiene secuencias de comandos ejecutables, instrucciones SQL o cualquier otro código antes de mostrarla en la aplicación. ASP .NET proporciona una función de validación de solicitud de introducción de datos para bloquear las secuencias de comandos y código HTML en los datos introducidos por el usuario. También se proporcionan controles de servidor de validación para evaluar los datos introducidos por el usuario. Para obtener más información, vea Controles de servidor de validación.

Campos de la clase DataGrid

Constructores públicos

DataGrid (Constructor)	Inicializa una nueva instancia de la clase DataGrid .
--	--

Campos públicos

CancelCommandName	Representa el nombre de comando Cancel . Este campo es de sólo lectura. (Estático, no requiere instancia)
DeleteCommandName	Representa el nombre de comando Delete . Este campo es de sólo lectura. (Estático, no requiere instancia)

EditCommandName	Representa el nombre de comando Edit . Este campo es de sólo lectura. (Estático, no requiere instancia)
NextPageCommandArgument	Representa el argumento de comando Next . Este campo es de sólo lectura. (Estático, no requiere instancia)
PageCommandName	Representa el nombre de comando Page . Este campo es de sólo lectura. (Estático, no requiere instancia)
PrevPageCommandArgument	Representa el argumento de comando Prev . Este campo es de sólo lectura. (Estático, no requiere instancia)
SelectCommandName	Representa el nombre de comando Select . Este campo es de sólo lectura. (Estático, no requiere instancia)
SortCommandName	Representa el nombre de comando Sort . Este campo es de sólo lectura. (Estático, no requiere instancia)
UpdateCommandName	Representa el nombre de comando Update . Este campo es de sólo lectura. (Estático, no requiere instancia)

Propiedades públicas

AccessKey (se hereda de WebControl)	Obtiene o establece la clave de acceso (la letra subrayada) que permite desplazarse rápidamente al control de servidor Web.
AllowCustomPaging	Obtiene o establece un valor que indica si está habilitada la paginación personalizada.
AllowPaging	Obtiene o establece un valor que indica si está habilitada la paginación.
AllowSorting	Obtiene o establece un valor que indica si está habilitada la ordenación.
AlternatingItemStyle	Obtiene las propiedades de estilo para los elementos alternos del control DataGrid .
Attributes (se hereda de WebControl)	Obtiene la colección de atributos arbitrarios (sólo para procesar) que no corresponden a propiedades del control.
AutoGenerateColumns	Obtiene o establece un valor que indica si se crean y se muestran automáticamente objetos BoundColumn del control DataGrid para cada campo del origen de datos.
BackColor (se hereda de WebControl)	Obtiene o establece el color de fondo del control de servidor Web.
BackImageUrl	Obtiene o establece la dirección URL de una imagen que se va a mostrar en el fondo del control DataGrid .
BorderColor (se hereda de WebControl)	Obtiene o establece el color de borde del control Web.
BorderStyle (se hereda de WebControl)	Obtiene o establece el estilo del borde del control de servidor Web.
BorderWidth (se hereda de WebControl)	Obtiene o establece el ancho del borde del control de servidor Web.
CellPadding (se hereda de BaseDataList)	Obtiene o establece la cantidad de espacio entre el contenido de una celda y el borde de la misma.
CellSpacing (se hereda de BaseDataList)	Obtiene o establece la cantidad de espacio entre las celdas.
ClientID (se hereda de Control)	Obtiene el identificador del control de servidor generado por ASP.NET.
Columns	Obtiene una colección de objetos que representan las columnas del control DataGrid .
Controls (se hereda de BaseDataList)	Reemplazado. Obtiene un System.Web.UI.ControlCollection que contiene una colección de controles secundarios en un control de lista de datos.
ControlStyle (se hereda de WebControl)	Obtiene el estilo del control de servidor Web. Esta propiedad la usan principalmente los programadores de controles.

ControlStyleCreated (se hereda de WebControl)	Obtiene un valor que indica si se ha creado un objeto Style para la propiedad ControlStyle . Esta propiedad la usan principalmente los programadores de controles.
CssClass (se hereda de WebControl)	Obtiene o establece la clase hoja de estilo en cascada (CSS) procesada por el control de servidor Web en el cliente.
CurrentPageIndex	Obtiene o establece el índice de la página que se muestra actualmente.
DataKeyField (se hereda de BaseDataList)	Obtiene o establece el campo clave del origen de datos especificado por la propiedad DataSource .
DataKeys (se hereda de BaseDataList)	Obtiene un DataKeyCollection que almacena los valores clave de cada registro (que se muestra en forma de fila) de un control de lista de datos.
DataMember (se hereda de BaseDataList)	Obtiene o establece, en un origen de datos con varios miembros, el miembro de datos específico que se debe enlazar con un control de lista de datos.
DataSource (se hereda de BaseDataList)	Obtiene o establece el origen que contiene una lista de valores que se utiliza para rellenar los elementos contenidos en el control.
EditItemIndex	Obtiene o establece el índice de un elemento del control DataGrid para su edición.
EditItemStyle	Obtiene las propiedades de estilo del elemento seleccionado para su edición en el control DataGrid .
Enabled (se hereda de WebControl)	Obtiene o establece un valor que indica si el control de servidor Web está habilitado.
EnableViewState (se hereda de Control)	Obtiene o establece un valor que indica si el control de servidor debe mantener su estado de vista y el de los controles secundarios que contiene, en el cliente que realiza la solicitud.
Font (se hereda de WebControl)	Obtiene las propiedades de fuente asociadas al control de servidor Web.
FooterStyle	Obtiene las propiedades de estilo de la sección de pie de página del control DataGrid .
ForeColor (se hereda de WebControl)	Obtiene o establece el color de primer plano (normalmente el color del texto) del control de servidor Web.
GridLines (se hereda de BaseDataList)	Obtiene o establece un valor que especifica si se debe mostrar el borde entre las celdas de un control de lista de datos.
HeaderStyle	Obtiene las propiedades de estilo de la sección de encabezado del control DataGrid .
Height (se hereda de WebControl)	Obtiene o establece el alto del control de servidor Web.
HorizontalAlign (se hereda de BaseDataList)	Obtiene o establece la alineación horizontal de un control de lista de datos en su contenedor.
ID (se hereda de Control)	Obtiene o establece el identificador de programación asignado al control de servidor.
Items	Obtiene una colección de objetos DataGridItem que representan los elementos individuales del control DataGrid .
ItemStyle	Obtiene las propiedades de estilo de los elementos del control DataGrid .
NamingContainer (se hereda de Control)	Obtiene una referencia al contenedor de nombres del control de servidor, que crea un espacio de nombres único para diferenciar los distintos controles de servidor que tienen el mismo valor para la propiedad Control.ID .
Page (se hereda de Control)	Obtiene una referencia a la instancia Page que contiene el control de servidor.
PageCount	Obtiene el número total de páginas necesario para mostrar los elementos del control DataGrid .

PagerStyle	Obtiene las propiedades de estilo de la sección de paginación del control DataGrid .
PageSize	Obtiene o establece el número de elementos que se van a mostrar en una sola página del control DataGrid .
Parent (se hereda de Control)	Obtiene una referencia al control principal del control de servidor en la jerarquía de controles de página.
SelectedIndex	Obtiene o establece el índice del elemento seleccionado del control DataGrid .
SelectedItem	Obtiene un objeto DataGridItem que representa el elemento seleccionado del control DataGrid .
SelectedItemStyle	Obtiene las propiedades de estilo del elemento seleccionado actualmente en el control DataGrid .
ShowFooter	Obtiene o establece un valor que indica si se muestra el pie de página en el control DataGrid .
ShowHeader	Obtiene o establece un valor que indica si se muestra el encabezado del control DataGrid .
Site (se hereda de Control)	Obtiene información sobre el sitio Web al que pertenece el control de servidor.
Style (se hereda de WebControl)	Obtiene una colección de atributos de texto que se procesan como un atributo de estilo en la etiqueta externa del control de servidor Web.
TabIndex (se hereda de WebControl)	Obtiene o establece el índice de tabulación del control de servidor Web.
TemplateSourceDirectory (se hereda de control)	Obtiene el directorio virtual de Page o UserControl que contiene el control de servidor actual.
ToolTip (se hereda de WebControl)	Obtiene o establece el texto que se muestra cuando el puntero del mouse (ratón) se desplaza sobre el control de servidor Web.
UniqueID (se hereda de Control)	Obtiene el identificador único cualificado jerárquicamente del control de servidor.
VirtualItemCount	Obtiene o establece el número virtual de elementos del control DataGrid cuando se utiliza la paginación personalizada.
Visible (se hereda de Control)	Obtiene o establece un valor que indica si un control de servidor se debe procesar como interfaz de usuario en la página.
Width (se hereda de WebControl)	Obtiene o establece el ancho del control de servidor Web.

Métodos públicos

ApplyStyle (se hereda de WebControl)	Copia en el control Web todos los elementos no vacíos del estilo especificado, sobrescribiendo los elementos de estilo existentes del mismo. Este método lo usan principalmente los programadores de controles.
CopyBaseAttributes (se hereda de WebControl)	Copia las propiedades no encapsuladas por el objeto Style del control de servidor Web especificado al control de servidor Web desde el que se efectúa la llamada al método. Este método lo usan principalmente los programadores de controles.
DataBind (se hereda de BaseDataList)	Reemplazado. Enlaza el control y todos sus controles secundarios al origen de datos especificado por la propiedad DataSource .
Dispose (se hereda de Control)	Habilita un control de servidor para que realice la limpieza final antes de que se libere de la memoria.
Equals (se hereda de Object)	Sobrecargado. Determina si dos instancias de Object son iguales.
FindControl (se hereda de Control)	Sobrecargado. Busca el control de servidor especificado en el contenedor de nombres actual.
GetHashCode (se hereda de Object)	Sirve como función hash para un tipo concreto, apropiado para su utilización en algoritmos de hash y estructuras de datos como las tablas hash.
GetType (se hereda de Object)	Obtiene el objeto Type de la instancia actual.

HasControls (se hereda de Control)	Determina si el control de servidor contiene controles secundarios.
MergeStyle (se hereda de WebControl)	Copia en el control Web todos los elementos no vacíos del estilo especificado, pero no sobrescribe los elementos de estilo existentes en el mismo. Este método lo usan principalmente los programadores de controles.
RenderBeginTag (se hereda de WebControl)	Procesa la etiqueta HTML de apertura del control en el escritor especificado. Este método lo usan principalmente los programadores de controles.
RenderControl (se hereda de Control)	Envía el contenido del control de servidor a un objeto HtmlTextWriter que se proporciona y almacena la información de seguimiento sobre el control si dicho seguimiento está habilitado.
RenderEndTag (se hereda de WebControl)	Procesa la etiqueta HTML de cierre del control en el escritor especificado. Este método lo usan principalmente los programadores de controles.
ResolveUrl (se hereda de Control)	Convierte una dirección URL en una que el cliente solicitante pueda utilizar.
ToString (se hereda de Object)	Devuelve un objeto String que representa al objeto Object actual.

Eventos públicos

CancelCommand	Se produce cuando se hace clic en el botón Cancelar de un elemento del control DataGrid .
DataBinding (se hereda de Control)	Se produce cuando el control de servidor se enlaza a un origen de datos.
DeleteCommand	Se produce cuando se hace clic en el botón Eliminar de un elemento del control DataGrid .
Disposed (se hereda de Control)	Se produce cuando un control de servidor se libera de la memoria, lo que constituye la última fase del período de duración de un control de servidor cuando se solicita una página ASP.NET.
EditCommand	Se produce cuando se hace clic en el botón Editar de un elemento del control DataGrid .
Init (se hereda de Control)	Se produce cuando el control de servidor se inicializa, lo que constituye la primera fase de su período de duración.
ItemCommand	Se produce cuando se hace clic en cualquier botón del control DataGrid .
ItemCreated	Se produce en el servidor cuando se crea un elemento del control DataGrid .
ItemDataBound	Se produce después del enlace de datos de un elemento al control DataGrid .
Load (se hereda de Control)	Se produce cuando el control de servidor se carga en el objeto Page .
PageIndexChanged	Se produce cuando se hace clic en uno de los elementos de selección de página.
PreRender (se hereda de Control)	Se produce cuando el control de servidor se va a procesar en el objeto Page que lo contiene.
SelectedIndexChanged (se hereda de BaseDataList)	Tiene lugar cuando se selecciona un elemento distinto de un control de lista de datos entre cada envío al servidor.
SortCommand	Se produce cuando se ordena una columna.
Unload (se hereda de Control)	Se produce cuando el control de servidor se descarga de la memoria.
UpdateCommand	Se produce cuando se hace clic en el botón Actualizar de un elemento del control DataGrid .

10 REPEATER

Mostrando los datos de una tabla de base de datos en un repeater (ejemplo introductorio)

En el código del Page_Load se coloca el siguiente fragmento de código

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

    'Introducir aquí el código de usuario para inicializar la página

    Dim DS As DataSet

    DS = New DataSet

    confDA.Fill(DS, "Empleados") 'Objeto data adapter y nombre de
                                'Objeto

    'El objeto del tipo Repeater se llama confRep, debe
    'coincidir con el id descrito en el archivo HTML

    confRep.DataSource = DS.Tables("Empleados") 'asociando la fuente
        ' al datalist

    confRep.DataBind() ' invocando el método DataBind, para acceder
        ' a los datos

End Sub
```

El código HTML asociado con el ejemplo se muestra a continuación

```
<asp:Repeater id="confRep" runat="server">

<HeaderTemplate>

    <table width="100%" style="font: 8pt verdana">

    <tr style="background-color:DFA894">

    <th>

        Codigo
```

```
</th>

<th>

    Nombre

</th>

<th>

    Apellido

</th>

<th>

    Direccion

</th>

</tr>

</HeaderTemplate>

<ItemTemplate>

    <tr style="background-color:FFECD8">

        <td>

            <%# DataBinder.Eval(Container.DataItem, "codigo") %>

        </td>

        <td>

            <%# DataBinder.Eval(Container.DataItem, "nombre") %>

        </td>

        <td>

            <%# DataBinder.Eval(Container.DataItem, "apellido") %>

        </td>

        <td>

            <%# DataBinder.Eval(Container.DataItem, "direccion") %>

        </td>

    </tr>

</ItemTemplate>

<FooterTemplate>
```

```
</table>

</FooterTemplate>

</asp:Repeater>
```

El parámetro id contiene el nombre del objeto en la página WEB y en el código desarrollado en VB.Net. El objeto Repeater contiene tres templates, el encabezado ("header"), el pie de página ("footer") y los elementos a mostrar ("item"). Las instrucciones que muestran los valores de los campos de las tablas son las siguientes, se debe colocar el nombre del campo como un String.

```
<%#DataBinder.Eval(Container.DataItem, "codigo")%>

<%#DataBinder.Eval(Container.DataItem, "nombre")%>

<%#DataBinder.Eval(Container.DataItem, "apellido")%>

<%#DataBinder.Eval(Container.DataItem, "direccion")%>
```

Clase Repeater

Control de lista con enlace a datos que permite especificar un diseño personalizado aplicando la misma plantilla a cada uno de los elementos mostrados en la lista. Se encuentra en el espacio de nombres **System.Web.UI.WebControls** (Espacio de nombres), es muy inferior al **DataList** debido a que mucha de la funcionalidad debe ser añadida en código HTML.

Repeater es una lista básica con enlace a datos basada en plantilla. No dispone de estilos ni de diseños integrados, por lo que deberá declarar explícitamente todas las etiquetas de diseño, formato y estilo HTML en las plantillas del control.

El control Repeater es el único que permite a los programadores dividir etiquetas HTML entre las plantillas. Para crear una tabla mediante plantillas, hay que incluir una etiqueta de inicio de tabla (<table>) en HeaderTemplate, una etiqueta de fila de tabla única (<tr>) en ItemTemplate y la etiqueta de final de tabla (</table>) en FooterTemplate.

Repeater no admite funciones de selección ni edición integradas. El usuario puede utilizar el evento ItemCommand para procesar los eventos de control que se provocan desde las plantillas al control. Repeater enlaza su ItemTemplate y AlternatingItemTemplate a un modelo de datos declarado y referenciado por su propiedad **DataSource**. HeaderTemplate, FooterTemplate y SeparatorTemplate no están enlazados a datos.

Si se establece el origen de los datos de Repeater, pero no se devuelve ningún dato, el control procesa HeaderTemplate y FooterTemplate sin elementos. Si el origen de los datos es una referencia nula (**Nothing** en Visual Basic), Repeater no se procesa.

Como mínimo, cada Repeater debe definir un ItemTemplate. Sin embargo, se pueden utilizar otras plantillas opcionales, descritas en la tabla siguiente, para personalizar el aspecto de la lista.

La clase **Repeater** maneja **eventos**, en el caso de los **Items** existen por ejemplo los eventos **ItemCreated** e **ItemCommand**.

Nombre de la plantilla	Descripción
ItemTemplate	Define el contenido y el diseño de los elementos incluidos en la lista. Esta plantilla es necesaria.
AlternatingItemTemplate	Si se define, determina el contenido y el diseño de elementos alternos (con índice impar y base cero). Si no se define, se utiliza ItemTemplate .
SeparatorTemplate	Si se define, se procesa entre elementos (y elementos alternos). Si no se define, no se procesa ningún separador.
HeaderTemplate	Si se define, determina el contenido y el diseño del encabezado de la lista. Si no se define, no se

	procesa ningún encabezado.
FooterTemplate	Si se define, determina el contenido y el diseño del pie de página de la lista. Si no se define, no se procesa ningún pie de página.

PRECAUCIÓN: Este control se puede utilizar para mostrar los datos introducidos por el usuario, que pueden incluir secuencias de comandos de cliente maliciosas. Compruebe que cualquier información que se envíe desde un cliente no contiene secuencias de comandos ejecutables, instrucciones SQL o cualquier otro código antes de mostrarla en la aplicación. ASP .NET proporciona una función de validación de solicitud de introducción de datos para bloquear las secuencias de comandos y código HTML en los datos introducidos por el usuario. También se proporcionan controles de servidor de validación para evaluar los datos introducidos por el usuario. Para obtener más información, vea Controles de servidor de validación.

Campos de la clase Repeater

Constructores públicos

Repeater (Constructor)	Inicializa una nueva instancia de la clase Repeater.
--	--

Propiedades públicas

AlternatingItemTemplate	Obtiene o establece System.Web.UI.ITemplate , que define cómo se muestran los elementos alternos del control.
ClientID (se hereda de Control)	Obtiene el identificador del control de servidor generado por ASP.NET.
Controls	Reemplazado. Obtiene un System.Web.UI.ControlCollection que contiene los controles secundarios de Repeater.
DataMember	Obtiene o establece la tabla específica de DataSource para enlazarla al control.
DataSource	Obtiene o establece el origen de datos que proporciona datos para llenar la lista.
EnableViewState (se hereda de Control)	Obtiene o establece un valor que indica si el control de servidor debe mantener su estado de vista y el de los controles secundarios que contiene, en el cliente que realiza la solicitud.
FooterTemplate	Obtiene o establece System.Web.UI.ITemplate , que define cómo se muestra la sección de pie de página del control Repeater.
HeaderTemplate	Obtiene o establece System.Web.UI.ITemplate , que define cómo se muestra la sección de encabezado del control Repeater.
ID (se hereda de Control)	Obtiene o establece el identificador de programación asignado al control de servidor.
Items	Obtiene una colección de objetos RepeaterItem de Repeater.
ItemTemplate	Obtiene o establece System.Web.UI.ITemplate , que define cómo se muestran los elementos del control Repeater.
NamingContainer (se hereda de Control)	Obtiene una referencia al contenedor de nombres del control de servidor, que crea un espacio de nombres único para diferenciar los distintos controles de servidor que tienen el mismo valor para la propiedad Control.ID .

Page (se hereda de Control)	Obtiene una referencia a la instancia Page que contiene el control de servidor.
Parent (se hereda de Control)	Obtiene una referencia al control principal del control de servidor en la jerarquía de controles de página.
SeparatorTemplate	Obtiene o establece System.Web.UI.ITemplate , que define cómo se muestra el separador de elementos.
Site (se hereda de Control)	Obtiene información sobre el sitio Web al que pertenece el control de servidor.
TemplateSourceDirectory (se hereda de Control)	Obtiene el directorio virtual de Page o UserControl que contiene el control de servidor actual.
UniqueID (se hereda de Control)	Obtiene el identificador único cualificado jerárquicamente del control de servidor.
Visible (se hereda de Control)	Obtiene o establece un valor que indica si un control de servidor se debe procesar como interfaz de usuario en la página.

Métodos públicos

DataBind	Reemplazado. Vea Control.DataBind .
Dispose (se hereda de Control)	Habilita un control de servidor para que realice la limpieza final antes de que se libere de la memoria.
Equals (se hereda de Object)	Sobrecargado. Determina si dos instancias de Object son iguales.
FindControl (se hereda de Control)	Sobrecargado. Busca el control de servidor especificado en el contenedor de nombres actual.
GetHashCode (se hereda de Object)	Sirve como función hash para un tipo concreto, apropiado para su utilización en algoritmos de hash y estructuras de datos como las tablas hash.
GetType (se hereda de Object)	Obtiene el objeto Type de la instancia actual.
HasControls (se hereda de Control)	Determina si el control de servidor contiene controles secundarios.
RenderControl (se hereda de Control)	Envía el contenido del control de servidor a un objeto HtmlTextWriter que se proporciona y almacena la información de seguimiento sobre el control si dicho seguimiento está habilitado.
ResolveUrl (se hereda de Control)	Convierte una dirección URL en una que el cliente solicitante pueda utilizar.
ToString (se hereda de Object)	Devuelve un objeto String que representa al objeto Object actual.

11 TEMPLATES

Los **controles del servidor web** pueden ser personalizados colocando valores en las propiedades o utilizando los **estilos CSS**. Algunos de los controles permiten personalizar su apariencia ("look and feel") empleando los **templates ASP.NET**. Un template ASP.NET es una mezcla de elementos HTML con controles ASP.NET para personalizar (darle un aspecto deseado y cambiar un "layout") un área particular del control.

Un template no es lo mismo que un estilo. Los estilos se refieren principalmente a los CSS stylesheets y afectan las propiedades gráficas como los colores, las fuentes, el estilo del border, el espaciamiento entre celdas, ente otros. Con los estilos, el "layout" del control permanece inalterado en su estructura pero puede ser modificado en su apariencia. En contraste, los templates involucran cambios más profundos que modifican porciones del control de interfaz con el usuario. Por ejemplo, el control Repeater permite utilizar una combinación de elementos HTML con controles ASP.NET para definir el

“layout” de cada fila en la fuente de datos. De la misma manera, el control DataGrid permite formatear las celdas de una columna entera empleando cualquier combinación de los controles ASP.NET.

Adicionalmente, el control DataGrid soporta la personalización de las filas aplicando diferentes templates a cada fila individual, alternando las filas, las filas seleccionadas, entre otras operaciones.

El diseño de los templates en ASP.NET no es muy diferente de los estilos visuales de Windows® XP o de los controles comunes de Windows. El punto clave a considerar es que ciertas partes de un elemento de interfaz con el usuario puede ser personalizado. Los controles comunes Win32® tienen partes clientes y no clientes, los estilos Windows XP tienen partes específicas de control y los servidores de controles en algunos controles de ASP.NET poseen templates específicos de los controles.

En ASP.NET las **clases de controles que heredan la interfaz Itemplate** pueden dinámicamente construir “pagelets” e insertar código en la página principal como texto HTML.

Estilos y templates no son mutuamente exclusivos. Se pueden emplear estilos y templates en conjunto, o usarlos separadamente para controlar la apariencia de los elementos definidos dentro de los templates. Cuando un control que dispone de templates como un DataGrid es procesado, los templates son instanciados en la página del contenido y desplegado en HTML.

En esta sección se consideran los templates en ASP.NET realizando ejemplos para tres controles iterativos: Repeater, DataList y DataGrid.

¿ Qué es un template ?

En términos generales, un template es la descripción indicando como un elemento es desplegado “rendered” en tiempo de ejecución. En ASP.NET, un template es una **propiedad** de un control del servidor que describe HTML estático, controles y código en script para mostrar un control. Por ejemplo, un control Repeater tiene una propiedad HeaderTemplate, la cual define el contenido de la región de “header”. Normalmente, se define un template dentro del body de la página ASP.NET empleando una sintaxis declarativa. Por ejemplo, el siguiente código muestra como especificar un template para dibujar el header de cada fila del control Repeater.

```
<asp:repeater runat="server">

  <HeaderTemplate>

    <h2>Empleados</h2>

    <table border="0">

  </HeaderTemplate>

  <ItemTemplate>

    <tr>...</tr>

  </ItemTemplate>

</asp:repeater>
```

Cuando se realiza el “rendering” de los contenidos del control Repeater, el motor de ejecución de ASP.NET utiliza el contenido definido en los templates y lo procesa – frecuentemente con los datos- para crear una representación HTML de la región. Todos los controles del lado del servidor en conjunto con el template realizan un rendering individual como un HTML..

La plataforma Microsoft® .NET utiliza la interfaz Itemplate en tiempo de ejecución para procesar los templates en una jerarquía del control que puede tomar los datos de una fuente externa como parte de un control ASP.Net. Para definir los templates de una manera declarativa – empleando tags en páginas ASPX – no es necesario conocer los detalles de la interfaz Itemplate.

Columnas con template en un dataGrid

Las columnas con template en un DataGrid de WEB ofrece una funcionalidad importante, permitiendo agregar un **formato** personalizado para las columnas del DataGrid. Normalmente, el control dataGrid despliega el texto de su contenido a través de strings con texto plano ("BoundColumns") o a través de uno de los tipos predefinidos de columna. Sin embargo, en ocasiones los tipos predefinidos de las columnas no proveen la representación deseada por el desarrollador de la página. Una columna de template permite definir hasta cuatro tipos diferentes de templates.

Nombre	Descripción
ItemTemplate	<p>Contiene los templates para los items (elementos) en una columna de un DataGrid.</p> <pre><ItemTemplate> <asp:label runat="server" text= '<%# ... %>'...> </ItemTemplate></pre> <p>Se puede utilizar cualquier combinación de texto HTML y controles ASP.NET para llenar los datos de la columna.</p>
EditItemTemplate	<p>Controla los contenidos de un elemento seleccionado para la edición en una columna de un control DataGrid. Coloque los controles necesarios para editar una celda entre los tags de apertura y cierre EditItemTemplate.</p> <pre><EditItemTemplate> <asp:textbox runat="server" text= '<%# ... %>'...> </EditItemTemplate></pre>
HeaderTemplate	<p>Contiene le template de la sección de encabezado.</p> <pre><HeaderTemplate> <asp:label runat="server" text= "Header"...> </HeaderTemplate></pre> <p>Si este template es omitido, el encabezado de la columna es rendered con una etiqueta o con un enlace si ordenamiento es permitido. Cuando se especifica un template personalizado, el programador deber proveer la interfaz de usuario para poder realizar un ordenamiento de los datos.</p>
FooterTemplate	<p>Contiene el template para la sección de pie de página de la columna. El valor por defecto es una referencia null.</p> <pre><FooterTemplate> <asp:label runat="server" text= "..."...> </FooterTemplate></pre> <p>El pie de página es mostrado, sólo si la propiedad ShowFooter del DataGrid se encuentra con el valor True.</p>

Por lo general, el template más utilizado es el ItemTemplate. Este define como será mostrada la celda n de la columna y cual será su contenido. Los templates de encabezado y pie de página se explican por sí solos. El template EditItem permite especificar como la celda cambiara cuando la fila es colocada en modo de edición.

Se emplean templates cuando se desea realizar una operación no estándar para la columna. En caso de requerir mostrar los datos en un tipo distinto a los provistos por los tipos de columnas (botones, textos, enlaces), entonces se debe utilizar una columna con template. El siguiente código muestra como definir una **templatedcolumn** y como asociarla a un DataGrid.

```
<asp:TemplateColumn runat="server"

HeaderText="encabezado">

    <itemtemplate>

        ...ASP.NET layout se coloca aquí...

    </itemtemplate>

</asp:TemplateColumn>
```

Una **templated column**, como cualquier otra columna, puede tener un texto de encabezado, así como una expresión de ordenamiento. Por otro lado, templated columns no tienen un campo explícito de fuente de datos para realizar un enlace. Entre los miembros de la clase TemplateColumn no se encuentran las propiedades DataField y DataTextField y como asociarla a un DataGrid.

La falta de una propiedad explícita para realizar enlace de datos es justificada por la gran flexibilidad para realizar el "layout" de la columna. Para realizar el render de una columna, se puede utilizar una etiqueta con la propiedad texto, pero también una lista dropdown o una imagen. Como resultado, el programador debe emplear expresiones de enlaces de datos para enlazar los datos.

El siguiente ejemplo muestra código válido para un item template

```
<asp:label runat="server"

Text='<%# DataBinder.Eval(Container.DataItem,

    "apellido") %>'

/>
```

Empleando **DataBinder.Eval**, es posible acceder cualquier campo en la fuente de datos actual asociada con el control. Adicionalmente, pueden ser combinados en cualquier orden para formar una expresión de ordenamiento la cual hubiera sido imposible obtener con una columna más simple.

Por ejemplo, si se desea colocar en una columna la información de dos campos de una base de datos se debería utilizar una columna de template como se muestra a continuación.

```
<itemtemplate>

<%#

    "<b>" +

    DataBinder.Eval(Container.DataItem, "apellido") +

    "</b>," +

    DataBinder.Eval(Container.DataItem, "nombre")

%>

</itemtemplate>
```

Si se desean **combinar varios campos** en un sólo formato, la única manera de realizarlo es a través de templated columns.

Es también posible manejar los eventos ocurridos en el grid y cambiar la apariencia del control cuando esos ocurren. Estos eventos son el ItemCreated y el ItemDataBound. Por ejemplo, si se desea cambiar el color de fondo de una celda, se escribe un manejador para el ItemCreated. Cuando se produce el evento ItemCreated, no existe garantía que existe un enlace de datos asociado. Los datos normalmente se toman de la propiedad DataItem de la estructura de datos asociada con el evento. Para el evento ItemCreated del DataGrid la "signature" del evento es como se indica a continuación.

```
Sub ItemCreated(sender as Object, e as DataGridItemEventArgs)
```

La expresión e.Item.DataItem evalúa los datos asociados con el ítem cuando este es creado. Si el grid se encuentra asociado con un DataTable, entonces DataItem es un objeto del tipo DataRow.

Se debe tomar en cuenta, que el enlace de datos (binding) ocurre cuando se dispara el evento ItemDataBound.

Encabezados Templated

La clase **TemplateColumn** ofrece la **propiedad HeaderTemplate** , permitiendo personalizar el encabezado (y el pie de página) de una columna. Templates de encabezado o de pie de página aplican únicamente e instancias de la clase TemplateColumn. Por ejemplo, si se desean editar los contenidos de una columna de una manera personalizada (por ejemplo se quieren incorporar validaciones especiales) , se deben utilizar templates así cuando en las columnas se muestren valores de campos simples.

El cambiar el "layout" de un encabezado, puede ser problemático si se desea ordenar una columna empleando una expresión. El mecanismo de ordenamiento es disparado por un control de enlace que el DataGrid automáticamente coloca en el encabezado de la columna. La propiedad href del enlace genera un evento postback cuando el usuario realiza clic sobre el elemento.

El código mostrado a continuación agrega una lista dropdown al encabezado de una columna template para permitir seleccionar la expresión de ordenamiento .

```
public Sub ItemCreated(sender as Object, e as DataGridItemEventArgs)

    lit as ListItemType = e.Item.ItemType

    if (lit = ListItemType.Header) then

        ' Crea y llena el control DropDownList

        dd as DropDownList = new DropDownList()

        dd.ID = "ddOrdenar"

        li1, li2, li3 as ListItem

        ' constructor ListItem toma texto y valores para el item

        li1 = new ListItem("Nombre del empleado", "nombreEmp")

        dd.Items.Add(li1)

        li2 = new ListItem("Apellido del empleado", "apellidoEmp")

        dd.Items.Add(li2)

        li3 = new ListItem("Posicion del empleado", "posicionEmp")

        dd.Items.Add(li3)
```

```
' Selecciona el indice en caso de seleccion
dd.SelectedIndex = Convert.ToInt32(grid.Attributes("FieldIndex"))

' Agrega el dropdown list al encabezado de la segunda columna
TableCell as cell = e.Item.Controls(1)
cell.Controls.Add(dd)
end if
end sub

public sub SortCommand(sender as Object, e as DataGridSortCommandEventArgs)

'Codigo que devuelve la fuente de datos del data grid se coloca aquí
' ...

' expression para realizar el ordenamiento
if (e.SortExpression <> "") then
    dv.Sort = e.SortExpression;
else

' toma el dropdown por su id
dgi as DataGridItem = e.CommandSource
dd DropDownList = dgi.FindControl("ddOrdenar")

' obtiene el resultado del sorting
dv.Sort = dd.SelectedItem.Value

' toma los valores y los realiza parsistntes
grid.Attributes("FieldIndex") = dd.SelectedIndex.ToString();
end if

// Refreshes the grid
grid.DataBind()
end sub
<asp:TemplateColumn runat="server"
    HeaderText="Ordenar Por"
```

```
SortExpression="*">
```

Adaptando la interface del usuario a los datos

Columnas basadas en templates permiten personalizar la interfaz del usuario dependiendo del tipo de datos a desplegar. No todos los datos en un DataGrid pueden ser fácilmente leídos o desplegados si son tratados con texto. Esto es particularmente cierto en el caso de los valores Booleans, arreglos o imágenes. Por ejemplo en el caso de valores lógicos se pueden emplear imágenes conteniendo una marca en el caso de trae y sin marca en el caso de false. Utilizar imágenes en lugar de un control checkbox es más efectivo porque la imagen no va a procesar eventos ni tomar el foco, produciéndose una interfaz con el usuario más natural.

La propiedad ImageUrl del elemento <asp:image> es un elemento de enlace de datos y puede asociarse con una expresión de enlace.

```
<itemtemplate>

    <asp:image runat="server"

        imageurl='<%# ObtImagenAdecuada(

            DataBinder.Eval(Container.DataItem,

                "cancelado")) %>'

    />

</itemtemplate>
```

En este código, la expresión de enlace consiste de una función retornando una imagen y tomando como argumento un valor Boolean. Dependiendo del argumento (variable cancelado) se utiliza la imagen con marca o sin marca.

El siguiente código muestra un ejemplo interesante cuando la información a desplegar en una columna consiste de imágenes. Si la imagen se encuentra disponible la muestra y en caso contrario muestra un texto.

```
<asp:DataGrid id="grid" runat="server"

    AutoGenerateColumns="false"

    CssClass="Shadow" BackColor="white"

    CellPadding="2" CellSpacing="0"

    BorderStyle="solid" BorderColor="black" BorderWidth="1"

    font-size="x-small" font-names="verdana"

    AllowPaging="true"

    PageSize="4"

    DataKeyField="idEmpleado"

    OnPageIndexChanged="PageIndexChanged">
<AlternatingItemStyle BackColor="palegoldenrod" />
<ItemStyle BackColor="beige" VerticalAlign="top" />
<EditItemStyle BackColor="yellow" Font-Bold="true" />
<PagerStyle Mode="NumericPages" />
```

```
<HeaderStyle ForeColor="white" BackColor="brown" HorizontalAlign="center"
    Font-Bold="true" />

<columns>

<asp:BoundColumn runat="server" DataField="idEmpleado"
    HeaderText="ID">

    <itemstyle bgcolor="lightblue" font-bold="true"
        HorizontalAlign="right" />
</asp:BoundColumn>

<asp:TemplateColumn runat="server" HeaderText="Nombre Empleado">
    <itemtemplate>
        <asp:label runat="server"
            Text='<%# DataBinder.Eval(Container.DataItem,
                "Nombre") + " <b> " +
                DataBinder.Eval(Container.DataItem, "Apellido") +
                "</b>" + ", " +
                DataBinder.Eval(Container.DataItem, "Posicion") %>' />
    </itemtemplate>
</asp:TemplateColumn>

    <asp:TemplateColumn runat="server" HeaderText="Fotografia">
    <itemtemplate>
        <img runat="server" width="50"
            visible='<%# ImagenDisponible(DataBinder.Eval
                (Container.DataItem, "apellido")
                ToString()) %>'
            src='<%# "imagenes\\" + DataBinder.Eval(Container.DataItem,
                "apellido") + ".bmp" %>' />
        <asp:label runat="server" text="<i><small>Imagen no disponible.
            </small></i>"
            visible='<%# NOT ImagenDisponible(DataBinder.Eval
                (Container.DataItem,
                "apellido").ToString()) %>' />
    </itemtemplate>
```



```
</asp:TemplateColumn>

</columns>

</asp:DataGrid>
```

La idea es definir dos "layouts" dentro de la misma columna de template, pero solo se despliega uno a la vez. El template tiene un **elemento ** y un **elemento <asp:label>**. Ambos asignan su propiedad visible dependiendo del resultado de la evaluación de una función llamada `ImagenDisponible`.

```
FUNCTION ImagenDisponible(apellido as string) as boolean

    nombreImagen as String = "imagenes\\";

    nombreImagen = nombreImagen + apellido + ".bmp";

    return File.Exists(Server.MapPath(nombreImagen));
```

Esta función asume que las imágenes se encuentran en archivos .bmp debajo del directorio `imagenes` del path de la aplicación y tienen como nombre el apellido del empleado.

12. HTTP REQUEST

Introducción

La clase **HttpRequest** permite leer los **http valores enviados por un cliente** durante un requerimiento Web (web Request) en ASP.NET. Se encuentra dentro del **espacio de nombres System.Web** y se trata de una clase sellada (no puede ser heredada).

System.Web.HttpRequest

[NotInheritable Public Class HttpRequest

Los métodos y propiedades de la clase **HttpRequest** son accedidos empleando la propiedad **Request** de las clases `HttpApplication`, `HttpContext`, `Page`, y `UserControl`.

Constructor

El constructor de esta clase soporta la infraestructura de la plataforma .NET y no está diseñada para ser utilizada directamente desde el código de la aplicación.

```
Public Sub New( _
    ByVal filename As String, _
    ByVal url As String, _
    ByVal queryString As String _
)
```

Propiedades

En la siguiente tabla se encuentran las propiedades de la clase **http request**.

Propiedades Públicas

ApplicationPath	Obtiene el camino de la aplicación en el servidor (ASP.NET application's virtual application root path)
Browser	Obtiene información acerca de las capacidades del navegador ejecutándose en el cliente.
ClientCertificate	Obtiene el certificado de seguridad del cliente.
ContentEncoding	
ContentLength	Especifica la longitud, en bytes, del contenido enviado por el cliente.
ContentType	
Cookies	Obtiene la colección de "cookies" enviadas por el cliente.
CurrentExecutionFilePath	Obtiene el camino virtual del requerimiento actual.
FilePath	Obtiene el camino virtual del requerimiento actual.
Files	
Filter	
Form	Obtiene la colección de las variables del formulario.
Headers	Obtiene la colección de headers HTTP.
HttpMethod	Obtiene el método de transferencia de datos (GET, POST o HEAD) utilizado por el cliente.
InputStream	Obtiene los contenidos del BODY del HTTP.
IsAuthenticated	
IsSecureConnection	
Item	Obtiene un objeto específico en las colecciones Cookies , Form , QueryString o ServerVariables
Params	Obtiene una colección combinada de items QueryString , Form , ServerVariables , y Cookies
Path	Obtiene el camino virtual del requerimiento.
PathInfo	
PhysicalApplicationPath	
PhysicalPath	
QueryString	Obtiene la colección de las variables de la consulta en "query string" realizada en un cliente.
RawUrl	
RequestType	
ServerVariables	Obtiene la colección de las variables del servidor de web.
TotalBytes	Obtiene el número de bytes del stream de entrada del request.
Url	
UrlReferrer	
UserAgent	
UserHostAddress	
UserHostName	
UserLanguages	

Métodos

En la siguiente tabla se encuentran los métodos de la clase httpRequest.

Métodos Públicos

BinaryRead	Performs a binary read of a specified number of bytes from the current input stream.
Equals (hereda de Object)	Sobrecargado.
GetHashCode (hereda	

de Object)	
GetType (hereda de Object)	
MapImageCoordinates	
MapPath	Sobrecargado. Mapea el camino virtual en el URL requerido a un camino físico del servidor del requerimiento actual.
SaveAs	Almacena en disco un requerimiento http.
ToString (hereda de Object)	
ValidateInput	Valida los datos enviados por el navegador de un cliente y dispara una excepción si se sospecha de datos alterados.

13. HTTP RESPONSE

Introducción

La clase **HttpResponse** permite encapsular la información de respuesta http en una operación de ASP.NET. Se encuentra dentro del espacio de nombres **System.Web** y se trata de una clase sellada (no puede ser heredada).

NotInheritable Public Class HttpResponse

Los métodos y propiedades de la clase **HttpResponse** son expuestos a través de la propiedad **Response** de las clases [HttpApplication](#), [HttpContext](#), [Page](#), y [UserControl](#).

Constructor

El constructor de esta clase soporta la infraestructura de la plataforma .NET y no está diseñada para ser utilizada directamente desde el código de la aplicación.

```
Public Sub New( _
    ByVal writer As TextWriter _
)
```

Propiedades

En la siguiente tabla se encuentran las propiedades de la clase **HttpResponse**.

Propiedades Públicas

Buffer	Obtiene o actualiza un valor indicando cuando colocar la salida en un "buffer" y enviarlo después que el "response" entero ha sido finalizado.
BufferOutput	Obtiene o actualiza un valor indicando cuando colocar la salida en un "buffer" y enviarlo después que el "page" entero ha sido finalizado.
Cache	Obtiene la política de "cache" de una página Web (tiempo de expiración, privacidad, cambió contenido)
CacheControl	Actualiza el Cache-Control del header HTTP a Public o Private .
Charset	
ContentEncoding	
ContentType	
Cookies	Obtiene la colección "cookies" del response.
Expires	Obtiene o actualiza el número de minutos antes de que el caché de una página en un servidor expira.
ExpiresAbsolute	
Filter	
IsClientConnected	Obtiene un valor indicando cuando el cliente se encuentra aún conectado a un servidor.

Output	Habilita la salida de texto del stream de la respuesta de salida de http.
OutputStream	Habilita la salida de texto del stream del body de salida de http.
RedirectLocation	Obtiene o actualiza el valor del "header" de la locación del http.
Status	Actualiza la línea de Status que es retornada al cliente.
StatusCode	
StatusDescription	
SuppressContent	Obtiene o actualiza un valor indicando cuando enviar el contenido de un HTTP a un cliente.

Métodos

En la siguiente tabla se encuentran los métodos de la clase HttpResponse.

Métodos Públicos

AddCacheItemDependencies	Fuerza a que la validez de un ítem en el caché sea dependiente de otros ítems en el caché.
AddCacheItemDependency	Fuerza a que la validez de un ítem en el caché sea dependiente de otro ítem en el caché.
AddFileDependencies	Agrega un grupo de nombres de archivo a la colección de nombres de archivos en el cual el "response" es dependiente.
AddFileDependency	Agrega un sólo nombre de archivo a la colección de nombres de archivos en el cual el "response" es dependiente.
AddHeader	Agrega un "header" HTTP al "stream" de salida.
AppendHeader	Agrega un "header" HTTP al "stream" de salida.
AppendToLog	Agrega información al "custom log" del archivo log de IIS.
ApplyAppPathModifier	
BinaryWrite	Escribe un "string" de caracteres binarios al "stream" de salida de HTTP.
Clear	Limpia toda la salida del "response" del "buffer".
ClearContent	Limpia toda la salida del "content" del "buffer"

ClearHeaders	Limpia toda la salida del "header" del "buffer"
Close	Cierra la conexión del "socket" de un cliente
End	Envía toda la salida almacenada en el buffer al cliente, termina la ejecución de la página y levanta el evento Application_EndRequest .
Equals (hereda de Object)	Sobrecargado.
Flush	Envía toda la salida almacenada en el "buffer" al cliente.
GetHashCode (hereda de Object)	
GetType (hereda Object)	
Pics	
Redirect	Sobrecargado. Redirecciona el cliente a un Nuevo URL.
RemoveOutputCacheItem	Método estático. Remueve del cache todos los items almacenados encontrados en el camino especificado.
ToString (hereda de Object)	
Write	Sobrecargado. Escribe información en el contenido de salida de un stream http.
WriteFile	Sobrecargado. Escribe información en el contenido de salida de un stream http en un archivo especificado

Métodos Protegidos

Finalize (hereda de Object)	Sobreescrito.
MemberwiseClone (hereda de Object)	

14. HTTPSERVERUTILITY

Introducción

Provee un conjunto de métodos para facilitar el procesamiento de requerimientos Web en ASP.NET. Se encuentra dentro del espacio de nombres **System.Web** y se trata de una clase sellada (no puede ser heredada).

System.Web.HttpServerUtility
NotInheritable Public Class HttpServerUtility

Los métodos y propiedades de la clase **HttpServerUtility** son expuestos a través del objeto intrínseco a ASP.NET's llamado [Server](#).

Propiedades

En la siguiente tabla se encuentran las propiedades de la clase HttpServerUtility.

Propiedades Públicas

MachineName	Obtiene el nombre del computador servidor.
ScriptTimeout	Actualiza o retorna el "timeout" del requerimiento en segundos.

Métodos

En la siguiente tabla se encuentran los métodos de la clase HttpServerUtility.

Métodos Públicos

ClearError	Limpia la información asociada con la última excepción
CreateObject	Sobrecargado. Crea una instancia del servidor como un objeto COM.
CreateObjectFromClsid	Sobrecargado. Crea una instancia del servidor como un objeto COM, empleando el identificador de clase del objeto (CLSID).
Equals (hereda de Object)	Sobrecargado.
Execute	Sobrecargado. Ejecuta el requerimiento actual utilizando otra página.
GetHashCode (hereda de Object)	
GetLastError	Devuelve la excepción ocurrida anteriormente.
GetType (hereda de Object)	
HtmlDecode	Sobrecargado. Decodifica un string que ha sido codificado para eliminar los caracteres HTML inválidos.
HtmlEncode	Sobrecargado. Codifica un string para ser desplegado en un manejador.
MapPath	Devuelve el camino físico de un archivo correspondiente con el camino virtual en el servidor.
ToString (hereda de Object)	
Transfer	Sobrecargado. Finaliza la ejecución de la página actual y comienza la ejecución de una nueva página para el requerimiento actual.
UriDecode	Sobrecargado. Decodifica un string, el cual fue codificado para ser transmitido por y enviado al servidor en un URL.
UriEncode	Sobrecargado. Codifica un string garantizando la transmisión HTTP segura del servidor de Web a un cliente vía un URL.
UriPathEncode	Codifica en formato URL la porción del "path" de un string URL y devuelve el string

15. HTTPSESSIONSTATE

Introducción

Esta clase provee acceso a los valores del **estado de una sección** así como los "settings" a nivel de sección y los métodos para controlar el ciclo de vida de la sección. Se encuentra dentro del **espacio de nombres System.Web.SessionState** y se trata de una clase sellada (no puede ser heredada).

[System.Object](#)

System.Web.SessionState.HttpSessionState
NotInheritable Public Class HttpSessionState

Implements ICollection, IEnumerable

Propiedades

En la siguiente tabla se encuentran las propiedades de la clase HttpSessionState.

Propiedades públicas

CodePage	Actualiza o devuelve el identificador del código de la página para la sección actual
Contents	Obtiene una referencia del objeto conteniendo el estado de la sección actual.
Count	Devuelve el número de items almacenado en la colección del estado de la sección.
IsCookieless	Devuelve un valor indicando cuando el ID de la sección esta embebido en el URL o almacenado en un "cookie".
IsNewSession	Devuelve un valor indicando cuando la sección fue creada con el requerimiento actual.
IsReadOnly	Devuelve un valor indicando cuando la sección es de sólo lectura
IsSynchronized	Devuelve un valor indicando cuando el acceso a los valores de la colección del estado de la sección es sincronizado.
Item	Sobrecargado. Obtiene o modifica los valores de los elementos de la sección.

Keys	Devuelve en una colección, las claves de todos los valores almacenados en una sección.
LCID	Obtiene o actualiza el identificador local (LCID) de la sección actual.
Mode	Devuelve el modo del estado actual de la sección.
SessionID	Devuelve un identificador único de una sección.
StaticObjects	Devuelve la colección de objetos declarados con los tags <object Runat="Server" Scope="Session"/> dentro del archivo global.asax en una aplicación ASP.NET
SyncRoot	Devuelve un objeto para ser utilizado en la sincronización de los acceso a los valores de la colección de los estados de la sección.
Timeout	Obtiene y actualiza el período de "time-out" (en minutos) permitido entre los requerimientos antes de que el proveedor del estado de la sección finalice la sección.

Métodos

En la siguiente tabla se encuentran los métodos de la clase **HttpSessionState**.

Métodos Públicos

Abandon	Cancela la sección actual.
Add	Agrega un Nuevo elemento al estado de la sección.
Clear	Limpia todos los valores del estado de la sección.
CopyTo	Copia los valores de la colección estado de la sección en un arreglo de una dimensión, comenzando en el índice especificado.
Equals (hereda de Object)	Sobrecargado.
GetEnumerator	Devuelve en un enumerador todos los valores del estado de la sección.
GetHashCode (hereda de Object)	
GetType (hereda de Object)	
Remove	Elimina un elemento de la colección session-state .
RemoveAll	Limpia todos los valores del session-state .
RemoveAt	Elimina un elemento en un índice especificado en la colección session-state.
ToString (hereda de Object)	

16. HTTPAPPLICATIONSTATE

Introducción

Permite compartir información global mediante **múltiples sesiones y peticiones en una aplicación ASP.NET**.

Se encuentra dentro del espacio de nombres **System.Web** y se trata de una clase sellada (no puede ser heredada).

[System.Object](#)
[System.Collections.Specialized.NameObjectCollectionBase](#)
System.Web.HttpApplicationState

NotInheritable Public Class HttpApplicationState

Inherits NameObjectCollectionBase

Una aplicación ASP.NET es la suma de todos los archivos, páginas, controladores, módulos y código que se encuentran en el ámbito de un directorio virtual y sus subdirectorios en un único servidor Web.

La primera vez que un cliente solicita un recurso URL de un directorio virtual de una aplicación ASP.NET se crea una única instancia de una clase **HttpApplicationState**. Para cada aplicación ASP.NET de un servidor Web se crea una única instancia independiente. Después, mediante el objeto intrínseco [Application](#), se expone una referencia a cada instancia.

HttpContext.Application (Propiedad)

Obtiene el objeto [HttpApplicationState](#) de la actual solicitud HTTP.

El estado de la aplicación no se comparte a través de una batería Web (donde múltiples servidores alojan la aplicación) ni de una unidad Web (donde múltiples procesos del mismo equipo alojan la aplicación).

Propiedades

En la siguiente tabla se encuentran las propiedades de la clase [HttpApplicationState](#).

Propiedades públicas

AllKeys	Obtiene las claves de acceso de la colección HttpApplicationState .
Contents	Obtiene una referencia al objeto HttpApplicationState .
Count	Reemplazado. Obtiene el número de objetos de la colección HttpApplicationState .
Item	Sobrecargado. Obtiene acceso a un objeto de una colección HttpApplicationState . Esta propiedad se sobrecarga para permitir el acceso a un objeto por nombre o por índice numérico. En C#, esta propiedad es el indizador de la clase HttpApplicationState .
Keys (se hereda de NameObjectCollectionBase)	Obtiene una instancia de NameObjectCollectionBase.KeysCollection que contiene todas las claves de la instancia de NameObjectCollectionBase .
StaticObjects	Obtiene todos los objetos declarados por una etiqueta <object> donde el ámbito se establece en "Application" en la aplicación ASP.NET.

Métodos

En la siguiente tabla se encuentran los métodos de la clase [HttpApplicationState](#).

Métodos públicos

Add	Agrega un nuevo objeto a la colección HttpApplicationState .
Clear	Quita todos los objetos de una colección HttpApplicationState .
Equals (se hereda de Object)	Sobrecargado. Determina si dos instancias de Object son iguales.
Get	Sobrecargado. Obtiene un objeto HttpApplicationState por nombre o índice.
GetEnumerator (se hereda de NameObjectCollectionBase)	Devuelve un enumerador que puede iterar a través de NameObjectCollectionBase .
GetHashCode (se hereda de Object)	Sirve como función hash para un tipo concreto, apropiado para su utilización en algoritmos de hash y estructuras de datos como las tablas hash.
GetKey	Obtiene un nombre de objeto HttpApplicationState por índice.
GetObjectData (se hereda de NameObjectCollectionBase)	Implementa la interfaz de ISerializable y devuelve los datos necesarios para serializar la instancia de NameObjectCollectionBase .
GetType (se hereda de Object)	Obtiene el objeto Type de la instancia actual.
Lock	Bloquea el acceso a una variable HttpApplicationState para que la sincronización del acceso sea más sencilla.
OnDeserialization (se hereda de NameObjectCollectionBase)	Implementa la interfaz de ISerializable y provoca el evento de deserialización cuando ésta ha finalizado.
Remove	Quita el objeto con nombre de una colección HttpApplicationState .
RemoveAll	Quita todos los objetos de una colección HttpApplicationState .
RemoveAt	Quita un objeto de una colección HttpApplicationState atendiendo al índice.
Set	Actualiza el valor de un objeto en una colección HttpApplicationState .
ToString (se hereda de Object)	Devuelve un objeto String que representa al objeto Object actual.
Unlock	Desbloquea el acceso a una variable HttpApplicationState para que la sincronización del acceso sea más sencilla.

17. EL ARCHIVO GLOBAL.ASAX

Introducción

En una aplicación ASP.NET con múltiples capas es posible separar la lógica del negocio. Un programador, además de escribir código asociado con la interfaz gráfica con el usuario, puede codificar la lógica asociada con la aplicación y el manejador de los eventos dentro de la aplicación WEB. Este código no manipula las interfaces gráficas y no es típicamente invocado en respuesta a "requests" de las páginas. En lugar de esto, este código es el encargado de manejar eventos de alto nivel en la aplicación tales como: **Application_Start**, **Application_End**, **Session_Start**, **Session_End**, entre otros.

Los desarrolladores codifican esta lógica de alto nivel empleando el archivo **Global.asax** file localizado en la raíz del árbol de directorio virtual asociado con la aplicación Web. ASP.NET automáticamente realiza el "parking" de este archivo y genera dinámicamente una clase en la plataforma .NET, la cual extiende la clase base **HttpApplication**, la primera vez que cualquier recurso o URL dentro del espacio de nombres de la aplicación es activado o requerido. El archivo Global.asax es configurado para rechazar automáticamente cualquier requerimiento directo a un URL, de modo que los usuarios externos no pueden descargar visualizar el código asociado con los recursos.

Eventos con alcance de session o aplicación (Application – Session Scoped)

Los desarrolladores pueden codificar manejadores a eventos para la clase base **HttpApplication** , codificando métodos en el archivo Global.asax siguiendo el patrón de nomenclatura "Application_NombreEvento(SignaturaEvento)". Por ejemplo:

```
<script language="VB" runat="server">  
  
Sub Application_Start(Sender As Object, E As EventArgs)  
  
    ' Código del arranque se coloca aquí  
  
End Sub  
  
</script>
```

Si el código necesario para manejar el evento requiere utilizar otros espacios de nombres, la directiva **@import** puede ser utilizada en una página .aspx de la siguiente manera:

```
<%@ Import Namespace="System.Data.SqlClient" %>
```

El siguiente ejemplo ilustra la utilización de los ciclos de vida para los objetos **Application**, **Session**, y **Request**.

En esta aplicación se emplean cuatro archivos. En primer lugar se tiene el archivo Global.asax con el siguiente código:

```
<script language="VB" runat="server">  
  
Sub Application_Start(Sender As Object, E As EventArgs)  
  
    ' Colocar código de inicio de la aplicación aquí
```

```
End Sub

Sub Application_End(Sender As Object, E As EventArgs)

    ' Colocar código liberación de recursos de la aplicación aquí

End Sub

Sub Session_Start(Sender As Object, E As EventArgs)

    Response.Write("Comenzando Session...<br>")

End Sub

Sub Session_End(Sender As Object, E As EventArgs)

    ' Colocar código liberación de recursos de la session aquí

End Sub

Sub Application_BeginRequest(Sender As Object, E As EventArgs)

    Response.Write("<h3><font    face='Verdana'>Utilizando    el    archivo  
Global.asax </font></h3>")

    Response.Write("Iniciando Requerimiento...<br>")

End Sub

Sub Application_EndRequest(Sender As Object, E As EventArgs)

    Response.Write("Finalizando Requerimiento...<br>")

End Sub
```

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

    ' Los errores se re-direccionan en otra página
    Context.ClearError()
    Response.Redirect("PaginaError.htm")
End Sub
</script>
```

La primera vez que la página es abierta el evento **Start** es disparado para los objetos Application y Session y se ejecuta el siguiente código:

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

    ' Colocar código de inicio de la aplicación aquí

End Sub

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

```
Response.Write("Comenzando Session...<br>")
```

```
End Sub
```

Los eventos **BeginRequest** y **EndRequest** son disparados cada vez que se produce un requerimiento "request". Cuando la página es refrescada, solo los mensajes de los métodos **BeginRequest**, **EndRequest**, **Page_Load** aparecerán en la pantalla.

El segundo archivo en este ejemplo es la página WEB conteniendo la aplicación. En este caso se llama Aplicacion1.aspx y contiene el siguiente código:

```
<html>

<script language="VB" runat="server">

    Sub Page_Load(Sender As Object, E As EventArgs)

        Response.Write("Cargando Pagina con Page.Load()...<br>")

    End Sub

    Sub Session_Click(Sender As Object, E As EventArgs)

        Session.Abandon()

        Response.Redirect("Aplicacion1.aspx")

    End Sub

    Sub Error_Click(Sender As Object, E As EventArgs)

        throw New Exception()

    End Sub

</script>

<body>

    <form runat="server">

        <input type="submit" Value="Actualizar Esta Pagina" runat="server"/>

        <input type="submit" OnServerClick="Session_Click" Value="Finalizar esta Session" runat="server"/>

        <input type="submit" OnServerClick="Error_Click" Value="Generar un Error" runat="server"/><p>

        <hr>

    </form>

</body>

</html>
```

El tercer archivo en este ejemplo es la página mostrada al producirse un error. En este caso se llama PaginaError.aspx y contiene el siguiente código:

```
<html>

<body>

    Este es un ejemplo de una página de error personalizada, la cual

    sera mostrada a los clientes cada vez que una excepción no manejada se
    produzca en

    la aplicación Web en ejecución. Este archivo puede ser configurado

    a través del archivo web.config .

</body>

</html>
```

Finalmente, el archivo de configuración de la aplicación web.config

```
<configuration>

<system.web>

    <globalization requestEncoding="UTF-8" responseEncoding="UTF-8" />

</system.web>

</configuration>
```

Al ejecutar la aplicación desde un navegador <http://localhost/Aplicacion1.aspx> se produce el siguiente resultado

Utilizando el archivo Global.asax

Iniciando			Requerimiento...
Comenzando			Session...
Cargando	Página	con	Page.Load()...

Enviar	Enviar	Enviar
--------	--------	--------

Finalizando Requerimiento...

18 ASPECTOS AVANZADOS DE LAS APLICACIONES WEB

El desarrollo de aplicaciones web debe considerar varios aspectos: un buen diseño de interfaz gráfica, criterios de seguridad y acceso a la información, estrategias de optimización del rendimiento (como por ejemplo empleo de la memoria caché), programación de controles específicos a un dominio y su reutilización, manejo en bajo nivel del protocolo HTML, desarrollando aplicaciones cliente-servidor, configuración y personalización de las aplicaciones a través de archivos XML.

Todos estos tópicos se consideran como aspectos avanzados en el desarrollo de las aplicaciones web y los mismos serán considerados en detalle en las siguientes secciones de este manual.

M019 CURSO ASP .NET Y SERVICIOS WEB , 24 Horas

Copyright, Instituto GALA de Venezuela 2004, 2005, 2006

Antes de desarrollar una aplicación se debe aplicar un proceso de diseño, en el cual deben tomarse decisiones acerca de las tecnologías para desarrollo del contenido WEB (diseño gráfico y edición), la modelación de los procesos, la localización y la seguridad.

En el caso de ASP.NET existen varias pautas de diseño a considerar, al desarrollar aplicaciones avanzadas en ASP.NET:

- Determinando la tecnología de IIS a emplear.
-
- Diseñando aplicaciones en dos capas "Two-Tier".
-
- Diseñando aplicaciones en tres capas "Three-Tier".
-
- Diseñando aplicaciones de múltiples capas "Multi-Tier".
-
- Diseñando aplicaciones a través de servicios web "Web services"
-
- Especificando los límites de integración de las capas "Application Boundaries"
-
- Controlando el flujo de la aplicación
-
- Diseñando una arquitectura cliente-servidor
-
- Escalabilidad de aplicaciones
-
- Integrabilidad de aplicaciones
-
- Internacionalización de aplicaciones
-
- Seguridad en las aplicaciones (mecanismos de autenticación y autorización)
-
- Manejo interno de las características de HTML
-
- Diseñando la configuración de la aplicación, así como la estructura de los documentos XML conformando la configuración
-
- Estableciendo criterios de optimización del rendimiento de la aplicación como la utilización de la memoria caché para almacenar la información de las páginas web o recursos más accedidos.

Al crear una aplicación en ASP.Net, también deben ser considerados los siguientes puntos:

- Ciclo de vida del proyecto
-
- Metodologías de desarrollo y fases del proyecto
-
- Consideraciones de seguridad de la página Web
-
- La edición, configuración y programación de los formularios Web
-
- El desarrollo de los controles ASP.NET (En la terminología de microsoft, un control es un objeto empleado para el desarrollo de la interfaz gráfica).
-
- Manejo de datos en ASP.NET
-
- Manejo del estado y la sección en una aplicación Web
-
- Servicios Web
-
- Desarrollando aplicaciones para consumir servicios Web.

19 MANEJADORES Y MODULOS HTTP

Introducción

ASP.NET provee un **API** de bajo nivel para manejar el protocolo **request/response** permitiendo más control a los programadores. La plataforma .NET dispone de un conjunto de clases para servir los requerimientos recibidos por el protocolo http conocidos como HTTP requests.

Este soporte se obtiene a través de la interfaz **System.Web.IHTTPHandler**, por lo tanto los programadores deben codificar las clases que implementan dicha interfaz. Dentro de esta interfaz el método más importante es el método **ProcessRequest()**. Los manejadores ("Handlers") son útiles cuando los servicios provistos por la abstracción de la plataforma de páginas de alto nivel no son requeridos para procesar el request HTTP. Usos comunes de los manejadores incluyen filtros y aplicaciones de estilo CGI, especialmente aquellos que devuelven datos binarios.

Cada request HTTP recibido por ASP.NET es procesado a bajo nivel por una instancia de una clase que implementa la interfaz **IHTTPHandler**. **IHttpHandlerFactory** provee la infraestructura para manejar las respuestas a los requerimientos URL asignándoselos a instancias de **IHttpHandler**. Adicionalmente a las clases **IHttpHandlerFactory** provistas por defecto por ASP.NET, los programadores pueden opcionalmente crear y registrar fabricas de manejadores para darle una mayor versatilidad al manejo de los request y producir diferentes escenarios de activación.

Configurando Manejadores HTTP y Fábricas

Los manejadores HTTP y las fábricas son declarados en la configuración de ASP.NET como parte de un archivo web.config. ASP.NET define una sección de configuración **<httphandlers>** donde los manejadores y las fábricas pueden ser agregados y removidos. La configuración especificada para **HttpHandlerFactory** y **HttpHandler** son heredadas por los subdirectorios.

Por ejemplo, ASP.NET asocia todos los requests de los archivos .aspx a la clase **PageHandlerFactory** en el archivo global machine.config:

```
<httphandlers>

...

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

...

</httphandlers>
```

Creando un manejador HTTP personalizado

A continuación se muestra un fragmento de código para crear un manejador HTTP personalizado.

```
Imports System.Web

Namespace manHTTP

    Public Class manejadorHTTP : Implements IHttpHandler

        Public Sub ProcessRequest(contexto As HttpContext) Implements
IHttpHandler.ProcessRequest

            contexto.Response.Write("Manejador Personalizado !")
        End Sub

        Public ReadOnly Property IsReusable As Boolean Implements
```

```
IHttpHandler.IsReusable
```

```
    Get
```

```
        Return true
```

```
    End Get
```

```
End Property
```

```
End Class
```

```
End Namespace
```

Un manejador HTTP personalizado puede ser creado implementando la interfaz **IHttpHandler**, la cual solo contiene dos métodos. Al invocar el método **IsReusable**, una fábrica HTTP puede consultar un manejador para determinar cuando la misma instancia puede ser utilizada para servir múltiples requerimientos. El método **ProcessRequest** toma una instancia de **HttpContext** como un parámetro, el cual permite acceder internamente a los detalles internos de **Request** y **Response**. En el ejemplo mostrado anteriormente, los datos del request son ignorados y un mensaje fijo es enviado en respuesta al cliente.

Después de haber compilado el código, la clase de manejador de HTTP debe ser especificado como el manejador de los requests. En este caso, todos los request para la página "manejadorHTTP.aspx" serán enrutados a una instancia de la clase **SimpleHandler**, la cual reside en el espacio de nombres `manHTTP.manejadorHTTP`

```
<httphandlers>
```

```
  <add verb="*" path="manejadorHTTP.aspx" type="manHTTP.manejadorHTTP, manejadorHTTP" />
```

```
</httphandlers>
```

20 SEGURIDAD EN ASP.NET

Introducción

Una parte importante de cualquier aplicación WEB es la habilidad de identificar los usuarios y **controlar el acceso a los recursos**. El acto de determinar la identidad de una entidad que efectúa el requerimiento se llama **autenticación** (determinar que el usuario dice ser quien es). Generalmente, el usuario debe presentar credenciales como el nombre y el password para ser autenticado. Una vez que la identidad de autenticación es válida, debe ser determinada la lista de recursos a los que ese usuario tiene acceso. Este proceso es conocido como **autorización**. ASP.NET trabaja en conjunto con **IIS** para proveer los servicios de autenticación y autorización en las aplicaciones.

Una característica importante de los objetos **COM** es la habilidad de controlar la identidad sobre la cual el objeto COM es ejecutado. Cuando un objeto COM ejecuta el código con la identidad de la entidad que lo requiere (el cliente) se produce lo que se denomina impersonalización. Las aplicaciones en la plataforma ASP.NET pueden opcionalmente escoger la impersonalización de los requerimientos.

En algunas aplicaciones, puede ser necesario ajustar el contenido mostrado o accesible dependiendo de valores determinados dinámicamente, empleando la identidad de la solicitud del requerimiento o empleando un conjunto de roles asociados con la identidad de la entidad asociada. Por ejemplo, una aplicación podría necesitar chequear cuando un usuario en particular posee el rol de gerente, en orden de mostrar contenido asociado con los gerentes.

Autenticación y Autorización

La plataforma ASP.NET trabaja en conjunto con IIS para soportar autenticación, empleando **Basic, Digest, y autenticación Windows**. ASP.NET soporta el servicio de **autenticación Microsoft Passport**, el cual provee un conjunto de servicios sencillos de sign-on y soporta los servicios para manejar perfiles de usuario.

ASP.NET también provee un servicio robusto para aplicaciones que requieren emplear autenticaciones basadas en formularios **"forms-based"**. Autenticación "forms-based" utiliza **"cookies"** para autenticar usuarios.

Es importante tener en cuenta que los servicios de autenticación de ASP.NET se basan en los servicios de autenticación provistos por IIS. Por ejemplo, para utilizar autenticación básica en una aplicación IIS, se debe configurar el uso de autenticación básica para la aplicación empleando el herramienta de Internet Service Manager.

ASP.NET soporta dos tipos de servicios de autorización:

- Revisa los **ACLs** o la permisología de un recurso para determinar cuando una cuenta de usuario (previamente autenticada) puede acceder a los recursos.
- Autorización URL, la cual autoriza a una identidad utilizar las piezas de un espacio web.

Para ilustrar la diferencia, si se considera un escenario en el cual una aplicación es configurada para permitir acceso anónimo empleando la cuenta *IUSR_MYMACHINE*. Cuando un requerimiento para una página ASP.NET (por ejemplo, */default.aspx*) es autorizada, el chequeo es realizado contra el archivo (por ejemplo, *"c:\inetpub\wwwroot\default.aspx"*) para determinar cuando la cuenta *IUSR_MYMACHINE* posee permiso para leer un archivo. Si se cumple la condición, entonces el acceso es autorizado. Si el contenido WEB reside en un volumen NTFS, y la autenticación Windows es configurada para el directorio virtual, la autorización del archivo es realizada automáticamente.

Para la autorización URL, el usuario anónimo es chequeado en contra de los datos de configuración manejados por la aplicación ASP.NET. Si el acceso es permitido para el URL requerido , el requerimiento es autorizado. En este caso, ASP.NET chequea para determinar cuando el usuario anónimo puede acceder al archivo */Default.aspx* (en otras palabras, es chequeado contra el URL en sí mismo y no en contra del archivo servido por el URL).

Esto podría parecer una distinción sutil, pero permite a las aplicaciones emplear esquemas de autenticación como basados en formularios o con un "Passport", en el cual los usuarios no se asocian a una máquina o a un dominio. Esto también permite la autorización contra recursos virtuales, para los cuales no existe un archivo físico asociado con el recurso. Por ejemplo, una aplicación podría escoger mapear todos los requerimientos para archivos terminando en *.stk* a un manejador prestando servicios de "stock quotes" basado en variables presentes en el String de consulta. En este caso, no existe archivo físico *.stk* para chequear (operación realizada por el ACL), entonces la autorización URL es empleada para controlar el control de acceso al recurso virtual.

La autorización de archivos es siempre realizada en contra de la cuenta de autenticación provista por IIS. Si acceso anónimo es permitido, ésta es la cuenta anónima configurada. De otra manera, se emplea la cuenta NT. Esto trabaja del mismo modo que ASP.

Los archivos ACLs son configurados para un directorio o archivo indicado empleando el tab **Security** en la página de propiedades del Explorer. Autorización URL es configurada como parte de una aplicación de la plataforma ASP.NET.

Para activar un servicio de autenticación en ASP.NET, se debe configurar el elemento **<authentication>** en el archivo de configuración de la aplicación. Este elemento puede tener cualquiera de los siguientes valores indicados en esta tabla.

Valor	Descripción
NONE	Los servicios de autenticación de ASP.NET no se activan. Es importante destacar que los servicios de autenticación de IIS pueden estar aún presentes.
Windows	Los servicios de autenticación de ASP.NET le "atachan" una ventana WindowsPrincipal (System.Security.Principal.WindowsPrincipal) al requerimiento actual para habilitar la autorización en contra de usuarios NT o grupos.

Forms (Formularios)	Los servicios de autenticación de ASP.NET administran "cookies" y redireccionan los usuarios no autenticados a una página de "logon". Esto es frecuentemente utilizado en conjunción con la opción IIS para permitir acceso anónimo a una aplicación.
Passport	Los servicios de autenticación de ASP.NET proveen encapsulamiento ("wrapper") a los servicios provistos por el Passport SDK, el cual debe estar instalado en la máquina.

Por ejemplo, el siguiente archivo de configuración habilita la autenticación basada en formularios forms-based (cookie) para una aplicación:

```
<configuration>

  <system.web>

    <authentication mode="Forms"/>

  </system.web>

</configuration>
```

Autenticación Windows-Based

Cuando se emplea la autenticación basada en ASP.NET Windows, ASP.NET anexa "attaches" un objeto de la clase **WindowsPrincipal** al requerimiento actual. Este objeto es utilizado por la autorización URL. Una aplicación puede a través de las propiedades del objeto determinar cuando una identidad efectuando un requerimiento posee un rol específico.

```
If User.IsInRole("Administradores") Then

  DisplayPrivilegedContent()

End If
```

La clase **WindowsPrincipal** determina los roles por la membresía de grupos definida en NT. Una aplicación puede determinar sus roles propio manejando el evento **WindowsAuthentication_OnAuthenticate** en el archivo Global.asax y programando una clase implementando la interfaz **System.Security.Principal.IPrincipal**, a continuación se muestra un ejemplo:

```
' Crear una clase implementando la interfaz IPrincipal

Public Class MyPrincipal : Inherits IPrincipal

  ' Implementa los roles asociados a la aplicación ( application-defined )

End Class

' Esto se coloca en un archivo Global.asax

Public Sub WindowsAuthentication_OnAuthenticate(Source As Object, e As WindowsAuthenticationEventArgs)

  ' Agrega una clase específica a la aplicación implementando la interfaz

  ' IPrincipal.

  ' Se asume que IIS realiza previamente una autenticación
```

' se puede utilizar la atributo identidad (Identity)

e.User = New MyPrincipal(e.Identity)

End Sub

El ejemplo mostrado a continuación muestra como acceder al nombre de un usuario autenticado, el cual es disponible a través de **User.Identity.Name**. En el caso de ASP en versiones anteriores, este valor se encuentra disponible como *AUTH_USER* Server.

En el ejemplo este código html se almacena en el archivo WindowsAuten.aspx

```
<html>

<script language="VB" runat=server>

Sub Page_Load(Src As Object, E As EventArgs)

    usuarioAut.Text = User.Identity.Name

    tipoAut.Text = User.Identity.AuthenticationType

End Sub

</script>

<body>

<h3><font face="Verdana">Ejemplo de Autenticación Windows </font></h3>

<table Width="700" rules="all" bordercolor="Black" style="background-
color: #ccccff;bordercolor: black;font-family: Verdana;font-size: 8pt;border-
collapse: collapse;">

<tr>

<td>Usuario: </td>

<td><asp:label id=usuarioAut runat=server/>

</tr>

<tr>

<td>Tipo de Autenticación: </td>

<td><asp:label id=tipoAut runat=server/>

</tr>

</table>

</body>

</html>
```

Autenticación basada en formularios Form-Based

Autenticación basada en formularios es un servicio de autenticación disponible en ASP.NET permitiendo a las aplicaciones utilizar unas cuentas de "login" definidas específicamente para la aplicación. La aplicación emplearía su propio mecanismo de verificación de credenciales. En este

caso, ASP.NET autentica a los usuarios, redirecciona a la página de "login" a los usuarios no autenticados, y realiza la administración de los "cookies". Este mecanismo de autenticación es una técnica muy popular empleada en la mayoría de los sitios Web.

Una aplicación para poder utilizar esta forma de autenticación, debe ser configurada colando el parámetro **<authentication>** en **Forms**, y denegando el acceso a usuarios anónimos. El siguiente ejemplo muestra como debe ser modificado el archivo Web.config file para lograr este fin:

```
<configuration>

  <system.web>

    <authentication mode="Forms"/>

    <authorization>

      <deny users="?" />

    </authorization>

  </system.web>

</configuration>
```

Los administradores emplean la autenticación basada en formularios para configurar el nombre del "cookie" a utilizar, el tipo de protección, el URL de la página de "login", el tiempo de vida del "cookie" y el "path" utilizado para el "cookie". La siguiente tabla muestra los valores válidos para el elemento **<Forms>** element, el cual es un sub-elemento del elemento **<authentication>**:

```
<authentication mode="Forms"/> <forms name=".ASPXCOOKIEDEMO"
loginUrl="login.aspx" protection="All" timeout="30" path="/"> <!--
protection="[All|None|Encryption|Validation]" --> </forms> </authentication>
```

Atributo	Descripción
loginUrl	Especifica el URL al cual son redireccionados los usuarios no autenticados. Puede ser el mismo computador o un computador remoto. En el caso de ser un computador remoto, ambos computadores necesitan utilizar el mismo valor para el atributo decryptionkey .
name	Nombre del "cookie" HTTP utilizado para propósitos de autenticación. Si se ejecutan varias aplicaciones en la misma máquina empleando este mecanismo de autenticación, estas deben ser configuradas para utilizar el mismo "cookie". Para evitar dependencias en los URLs, ASP.NET emplea "/" como el valor del "Path" al momento de setear las "cookies" de autenticación, de modo que estas son enviadas a cada aplicación.
timeout	Cantidad de tiempo en minutos (valor entero), después del cual el "cookie" expira. El valor defecto es 30. Este atributo es un valor corridizo "sliding", venciendo n minutos después de que el ultimo requerimiento fue realizado.
path	Path empleado por el "cookie". El valor por defecto es "/"
protection	Método empleado para proteger los datos de los "cookies". Los posibles valores son los siguientes: <ul style="list-style-type: none">• All: Emplea validación de datos y encriptación para proteger el "cookie". All es el valor por defecto (y el sugerido).• None: Empleado en sites empleando únicamente "cookies" para fines de personalización y no requieren altos mecanismos de seguridad.

	<ul style="list-style-type: none">• Encryption: Encripta el "cookie" empleando TripleDES o DES.• Validation: No encripta los contenidos del "cookie", pero valida que el "cookie" no ha sido alterado en el tránsito..
--	---

Después de configurar una aplicación para soportar este mecanismo de autenticación , se debe proveer una página de "login". En el siguiente ejemplo, se muestra una página de "logon" . Los usuarios no autenticados son enviados a la página de "login" en este caso (Login.aspx), la cual presenta un formulario que pregunta por Cuenta y Clave.

Una vez validada las credenciales, la aplicación realiza la siguiente llamada

FormsAuthentication.RedirectFromLoginPage(Cuenta.Value, PersistCookie.Checked)

Esta instrucción permite regresar al URL original donde se realizó el requerimiento. Si no se desea realizar el redireccionamiento se puede emplear **FormsAuthentication.GetAuthCookie** para obtener el valor del "cookie" o **FormsAuthentication.SetAuthCookie** para realizar encriptación a la respuesta en tránsito.

Las "cookies" de autenticación empleadas con formularios son una versión lineal de la clase **System.Web.Security.FormsAuthenticationTicket**. Estos "cookies" contienen un nombre de usuario , pero no password, la versión de la autenticación empleada, la fecha cuando el "cookie" fue emitido y un campo opcional para almacenar datos específicos a la aplicación.

En el código de la aplicación se puede recuperar o eliminar los "cookies" de autenticación empleando el método **FormsAuthentication.SignOut**. Este método elimina el "cookie" independientemente si este es temporal o permanente.

A continuación se muestra un ejemplo del uso de autenticación.

Cuando se carga el ASP

<http://localhost/WindowsAuten.aspx>

Se direcciona a la página de login

<http://localhost/login.aspx?ReturnUrl=%2fWindowsAuten.aspx>

y aparece la siguiente ventana de login

Página de Login

Cuenta:	<input type="text"/>	*
Clave:	<input type="password"/>	*
Cookie Persistente:	<input type="checkbox"/>	
<input type="button" value="Enviar"/>		

Al colocar la Cuenta y la Clave indicada entonces si se muestra el requerimiento original:

Ejemplo de Autenticación Windows

Usuario:	GALA
Tipo de Autenticación:	Forms

El archivo login.aspx contiene el siguiente código:

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

<html>

<script language="VB" runat=server> Sub Login_Click(Src As Object, E As EventArgs)

' Autenticación de usuario: este ejemplo solo acepta un usuario ' llamado GALA con password
GALA1

If Cuenta.Value = "GALA" And Clave.Value = "GALA1" then

                                FormsAuthentication.RedirectFromLoginPage(Cuenta.Value,
CookiePersistente.Checked)

Else

    Mensaje.Text = "Clave Inválida: Trate de nuevo" End If End Sub </script>

<body>

<form runat=server>

<h3><font face="Verdana">Página de Login</font></h3>

<table>

<tr>

<td>Cuenta: </td>

<td><input id="Cuenta" type="text" runat=server/></td>

<td><ASP:RequiredFieldValidator          ControlToValidate="Cuenta"          Display="Static"
ErrorMessage="*" runat=server/></td>

</tr>

<tr>

<td>Clave: </td>

<td><input id="Clave" type=password runat=server/></td>

<td><ASP:RequiredFieldValidator          ControlToValidate="Clave"          Display="Static"
ErrorMessage="*" runat=server/></td>

</tr>

<tr>

<td>Cookie Persistente: </td>

<td><ASP:CheckBox id=CookiePersistente runat="server" /> </td>

<td></td>

</tr>
```

```
</table>

<asp:button text="Login" OnClick="Login_Click" runat=server/>

<p>

<asp:Label id="Mensaje" ForeColor="red" Font-Name="Verdana" Font-Size="10"
runat=server />

</form>

</body>

</html>
```

El archivo de configuración contiene el siguiente contenido

```
<configuration>

<system.web>

<authentication mode="Forms">

  <forms name=".ASPXUSERDEMO" loginUrl="login.aspx" protection="All" timeout="60"
/>

</authentication>

<authorization> <deny users="?" />

</authorization>

<globalization requestEncoding="UTF-8" responseEncoding="UTF-8" />

</system.web>

</configuration>
```

Es también posible implementar servicios de autenticación con una lista de credenciales válidas en el archivo de configuración, como se muestra en este ejemplo:

```
<authentication>

<credentials passwordFormat="SHA1" >

  <user name="Pedro"
password="94F85995C7492EEC546C321821AA4BECA9A3E2B1"/>

  <user name="Pablo"
password="5753A498F025464D72E088A9D5D6E872592D5F91"/>

</credentials>

</authentication>
```

La aplicación invoca **FormsAuthentication.Authenticate**, proveyendo el nombre del usuario y el password, y ASP.NET verificará las credenciales. Las credenciales pueden ser almacenadas en tres tipos (Hash Type) cleartext, SHA1 o MD5, dependiendo del valor del atributo **passwordFormat** :

Hash Type	Descripción
Clear	Passwords son almacenados en cleartext
SHA1	Passwords son almacenados en SHA1
MD5	Passwords son almacenados en MD5

Autorizando Usuarios y Roles

ASP.NET es utilizado para control el acceso de los clientes a los accesos URL. El método HTTP realizado para procesar el requerimiento (**GET** o **POST**) puede ser configurado para autorizar o negar el acceso a grupos de usuarios o roles. El siguiente ejemplo garantiza el acceso a un usuario llamado GALA y a un rol llamado ADMIN. Todos los otros usuarios son rechazados.

```
<authorization>

  <allow users="GALA " />

  <allow roles="ADMIN" />

  <deny users="*" />

</authorization>
```

Los elementos permisibles (tags xml) para la autorización son allow o deny. Cada elemento **allow** o **deny** debe contener como atributos users o roles. Múltiples usuarios o roles pueden ser especificados en un solo elemento indicando una lista separada por comas.

```
<allow users="GALA1,GALA2" />
```

El método HTTP puede ser indicado empleando el atributo **Verb**:

```
<allow VERB="POST" users=" GALA1,GALA2" />

<deny VERB="POST" users="*" />

<allow VERB="GET" users="*" />
```

En el ejemplo anterior GALA1 y GALA2 pueden realizar POST a los recursos protegidos, mientras todos los usuarios pueden realizar únicamente GET.

Existen dos nombres de usuarios especiales:

- *: Todos los usuarios
- ?: Usuarios Anónimos (no autenticados)

Estos nombres especiales son comúnmente empleados en aplicaciones empleando autenticación forms-based para rechazar el acceso a usuarios no autenticados, como se indica en el ejemplo:

```
<authorization>

  <deny users="?" />

</authorization>
```

La autorización a los URL se determina jerárquicamente y las reglas empleadas para determinar el acceso son las siguientes:

- Las reglas relevantes al URL son revisadas de forma jerárquica y se van almacenando en una lista.
- Las reglas definidas en los directorios hijos sobrescriben (tienen prioridad) sobre las reglas colocadas en los directorios padres.

- Las reglas son revisadas hasta que se consiga un “match” o hasta que no existan más. Si es encontrado un “match” el acceso es permitido, en caso contrario el acceso es deshabilitado.

Si una aplicación no desea heredar ninguna característica de configuración, debe configurar todas las posibilidades relevantes a la aplicación.

21. ARCHIVO DE CONFIGURACIÓN

Introducción a la Configuración en ASP.NET

Un requerimiento importante para un servidor de aplicaciones WEB es un sistema de configuración flexible y poderoso. Un sistema que permita a los desarrolladores asociar “**settings**” de una manera sencilla a través de una aplicación con características de instalación, sin necesidad de colocar estos valores dentro del código (“bake values into code”)., y a los administradores de la aplicación poder cambiar estos valores durante el ciclo de vida de la aplicación. El sistema de configuración de ASP.NET ha sido diseñado para satisfacer a estas dos audiencias; este sistema provee una infraestructura de configuración jerárquica permitiendo la extensión de los datos de configuración a través de una aplicación, un “site” o una máquina.

El sistema de configuración de ASP.NET posee las siguientes características, las cuales hacen muy flexible la construcción y el mantenimiento de las aplicaciones Web:

- ASP.NET permite que los settings de configuración sean almacenados en conjunto con contenido estático, páginas dinámicas y lógica del negocio todos dentro de una jerarquía de directorios para la aplicación. Un usuario o administrador, simplemente necesita una copia del árbol de directorios para configurar una aplicación de ASP.NET en una máquina.
- Los datos de configuración son almacenados en archivos de texto legibles. Administradores y desarrolladores pueden emplear cualquier editor de texto, parser de XML o lenguaje de scripts para interpretar o actualizar los settings de configuración.
- ASP.NET soporta una infraestructura de configuración extensible, la cual habilita los desarrolladores de aplicaciones comerciales almacenar las características particulares de configuración de sus aplicaciones. Es posible definir el formato de persistencia y participar inteligentemente en el procesamiento y el control de las características de configuración. El objetivo es minimizar el trabajo de los usuarios finales. Esta ventaja es muy similar a la provista por los instaladores de aplicaciones.
- Cambios en los archivos de configuración de ASP.NET son automáticamente detectados por el sistema y son aplicados sin requerir la intervención del usuario (No es necesario reiniciar la aplicación, ni el servidor de web).
- Las secciones de configuración pueden ser bloqueadas hacia abajo en la jerarquía empleando el tag **<location>** y el atributo **allowOverride**.

Los archivos de configuración de ASP.NET son archivos de texto con formato XML “XML-based” y deben ser llamados **web.config**—son almacenados en cualquier directorio de un servidor de aplicaciones WEB ASP.

Jerarquía de los archivo de configuración

La configuración contenida en un archivo **web.config** abarca el directorio donde se encuentra alojado y todos los directorios hijos que se encuentran bajo este directorio. Si se colocan archivos de configuración en los directorios hijos, estos sobrescriben al archivo alojado en el directorio padre.

El	archivo	de	configuración	“root”	--
----	---------	----	---------------	--------	----

WinNT\Microsoft.NET\Framework\<version>\config\machine.config—provee la configuración por defecto para toda la máquina. **ASP.NET configura IIS** para evitar el acceso directo a través de un navegador de los archivos web.config, evitando que sus valores sean de dominio público. Un intento de acceder a estos archivos produce a ASP.NET retornar un error 403: Access Forbidden.

En tiempo de ejecución, ASP.NET emplea los archivos de configuración web.config para determinar **de una manera jerárquica los "settings"** (valores asignados a los parámetros de configuración) cada vez que se accede a un requerimiento de un URL (estos "settings" son calculados una sola vez y son almacenados en el caché para atender a los siguientes requerimientos; ASP.NET determina automáticamente cuando se produce un cambio en los archivos URL invalidando lo almacenado en el caché si algún parámetro de configuración es modificado.

Por ejemplo, los "settings" de configuración para el URL `http://servidorGala/Curso019/Ejemplo/pag.aspx` son determinados aplicando los "settings" del archivo web.config en el siguiente orden:

"Settings" de la máquina

`C:\WinNT\Microsoft.NET\Framework\v.1.00\config\machine.config`

Sobrescritos por los "settings" del "site" (aplicación raíz).

`C:\inetpub\wwwroot\web.config`

Sobrescritos por los "settings" de la aplicación.

`D:\Curso019\web.config`

Sobrescritos por los "settings" de un subdirectorio de la aplicación.

`D:\Curso019\Ejemplo\web.config`

Si existe un archivo **web.config** en el directorio raíz de un "site", por ejemplo "inetpub\wwwroot", los "settings" de configuración aplican a todas las aplicaciones ejecutadas en el "site". Es importante destacar que la presencia de un archivo web.config en un directorio o en la raíz es opcional. Si no se encuentra presente el archivo web.config, todos los "settings" de configuración son **automáticamente heredados** del directorio padre.

Configurando Section Handlers y Sections

Un archivo web.config es un archivo de texto en formato XML , conteniendo elementos de documentos XML standard, como tags bien formados, comentarios, texto, cdata, etc. El formato de codificación del archivo puede ser ANSI, UTF-8, o Unicode; el sistema detecta automáticamente la codificación. El nodo raíz de un archivo web.config es siempre el tag **<configuration>**. Los "settings" de ASP.NET y los "settings" del usuario final son encapsulados en este tag como se muestra a continuación:

```
<configuration>

  <!-- Aquí se colocan los "settings" de configuración. -->

</configuration>
```

Toda la información de configuración reside entre los tags raíz de XML **<configuration>** y **</configuration>**. La información de configuración contenida entre estos tags es agrupada en dos áreas principales: el área de declaración section handler "settings" y el área section "settings".

- **Section handlers** – La infraestructura de configuración de ASP.NET no realiza "assumptions" sobre el formato o el tipo de "settings" en un archivo web.config. En cambio, ASP.NET delega el procesamiento de los datos del archivo web.config a los handlers, las clases de la plataforma .NET implementan la interfaz **IConfigurationSectionHandler**. Una declaración individual de un **IConfigurationSectionHandler** aparece una sola vez, por lo general en el archivo

machine.config. Los archivos web.config ubicados en directorios hijos, automáticamente heredan esta declaración. Los handlers de configuración de la sección son declarados dentro del archivo web.config empleando el tag **<configSections>**. Tags de sección pueden ser "qualified" aún más empleando grupos de tags de sección de modo de organizarlos en grupos lógicos, en las siguientes secciones se mostrará un ejemplo. Cada tag de sección identifica un nombre de tag denotando una sección específica de datos de configuración y una clase **ConfigurationSectionHandler** asociada la cual lo procesa.

- **Section groups** – La configuración en ASP.NET permite la agrupación de secciones de una manera jerárquica facilitando el proceso de organización de los tags. Un tag **<sectionGroup>** puede aparecer dentro de un tag **<configSections>** o dentro de otro tag **<sectionGroup>**. Por ejemplo, todos los handlers de sección de ASP.NET aparecen con el grupo de sección **<system.web>**.

Configuration sections – Los settings de configuración de ASP.NET son representados con configuration tag sections, éstos también son anidados dentro de un tag **<configuration>** (y opcionalmente con tags de grupos de sección). Por cada sección de configuración, un section handler apropiado debe ser definido en la jerarquía. Por ejemplo, en el archivo mostrado a continuación, el tag **<httpModules>** es la sección de configuración que define los datos de configuración para los módulos HTTP modules. La clase **System.Configuration.HttpModulesConfigurationHandler** es la responsable de interpretar el contenido asociado con el tag **<httpModules>** durante el tiempo de ejecución. Se debe observar que ambos, la definición del section handler y la sección en sí deben tener el mismo "qualifier" para el grupo de sección (in este caso, **<system.web>**).

Observación: Los nombres de los tags son sensitivos a mayúsculas-minúsculas y deben ser escritos idénticamente a como son mostrados en este ejemplo. A continuación se ejemplifica lo antes expuesto, a través de un archivo **web.config**.

```
<configuration>

<configSections>

<sectionGroup name="system.web">

<section
                                name="httpModules"

type="System.Web.Configuration.HttpModulesConfigurationHandler, System.Web,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
/>

</sectionGroup>
</configSections>
```

```
<system.web>

<httpModules>

<add
name="CookielessSession"
type="System.Web.SessionState.CookielessSessionModule, System.Web,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
/>

<add
name="OutputCache"
type="System.Web.Caching.OutputCacheModule, System.Web,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
/>

<add
name="Session"
```

```
type="System.Web.SessionState.SessionStateModule,System.Web,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
/>

<add
name="WindowsAuthentication"
type="System.Web.Security.WindowsAuthenticationModule,System.Web,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
/>

<add
name="FormsAuthentication"
type="System.Web.Security.FormsAuthenticationModule,System.Web,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
/>

<add
name="PassportAuthentication"
type="System.Web.Security.PassportAuthenticationModule,System.Web,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
/>

<add
name="UrlAuthorization"
type="System.Web.Security.UrlAuthorizationModule,System.Web,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
/>

<add
name="FileAuthorization"
type="System.Web.Security.FileAuthorizationModule,System.Web,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
/>

</httpModules>

</system.web>

</configuration>
```

Utilizando Location y Path

Por defecto todos los settings de configuración definidos dentro del tag de alto nivel **<configuration>** aplican a la locación del directorio donde se almacena el archive web.config y a todos los "paths" hijos dentro de éste. Opcionalmente, se pueden aplicar settings de configuración solo a "paths" hijos específicos empleando el tag **<location>** restringiendo el atributo **path** apropiadamente. Si el archivo de configuración es machine.config, se pueden aplicar "settings" a directorios virtuales específicos o a aplicaciones específicas. Si el archivo de configuración es el archive web.config, los "settings" pueden ser aplicados a un archive específico, a un directorio hijo, a un directorio virtual o a una aplicación. A continuación se muestra un ejemplo, donde la codificación de los request y response cambian dependiendo de los "locations" de las aplicaciones:

```
<configuration>

  <location path="EnglishPages">

    <system.web>

      <globalization

        requestEncoding="iso-8859-1"
```

```
        responseEncoding="iso-8859-1"

    />

</system.web>

</location>

<location path="Default Web Site/EnglishPages/OneJapanesePage.aspx">

    <system.web>

        <globalization

            requestEncoding="Shift-JIS"

            responseEncoding="Shift-JIS"

        />

    </system.web>

</location>

</configuration>
```

Bloqueando los settings de configuración

Adicionalmente a la especificación de la información del path empleando el tag **<location>** , se puede especificar seguridad en las configuraciones de modo que el archive de configuración no pueda ser sobrescrito al descender en la jerarquía de configuración. Para bloquear hacia abajo "lock down" un grupo de settings, se emplea el atributo **allowOverride** del tag **<location>** colocándolo en false. El siguiente código bloquea hacia abajo los settings de "impersonation" en dos aplicaciones diferentes.

```
<configuration>

    <location path="app1" allowOverride="false">

        <system.web>

            <identity impersonate="false" userName="app1" password="app1pw"
        />

        </system.web>

    </location>

    <location path="app2" allowOverride="false">

        <system.web>

            <identity impersonate="false" userName="app2" password="app2pw"
        />

        </system.web>

    </location>
```

```
</configuration>
```

Si un usuario, trata de realizar sobreescritura sobre este grupo de "settings" en otro archivo de configuración, se producirá un error.

```
<configuration>

  <system.web>

    <identity userName="developer" password="loginpw" />

  </system.web>

</configuration>
```

Configuración de Sección Standard ASP.NET

ASP.NET se encuentra provisto de un número de manejadores para la configuración de sección standard utilizados para procesar los settings de configuración dentro de los archivos web.config. La siguiente tabla describe brevemente cada una de estas secciones.

Nombre de Sección	Descripción
<httpModules>	Responsable de la configuración de los módulos de HTTP en una aplicación. Los módulos de HTTP participan en el procesamiento de todos los "request" en una aplicación. Los usos más comunes incluyen la seguridad y la autenticación.
<httpHandlers>	Responsable por realizar el mapeo entre los URLs requeridos y las clases IHttpHandler . Los subdirectorios no heredan estos settings. También es responsable de realizar el mapeo entre los URLs requeridos y las clases IHttpHandlerFactory . Los datos representados en las secciones <httpHandlers> son heredadas jerárquicamente por los subdirectorios.
<sessionState>	Responsable de la configuración del session state de un modulo HTTP.
<globalization>	Responsable de la configuración de los "settings" de globalización de una aplicación.
<compilation>	Responsable de todos los settings de compilación empleados por ASP.NET.
<trace>	Responsable por la configuración del servicio de "trace" de ASP.NET.
<processModel>	Responsable por la configuración de los settings de los modelos de procesos en los servidores de WEB IIS.
<browserCaps>	Responsable del control de los settings asociados con las capacidades de navegación.

Atributos Opcionales – Los siguientes atributos son validos solo en las aplicaciones ASP.NET. El sistema de configuración ignora estos atributos cuando se ejecutan aplicaciones de otro tipo.

Atributo	Descripción
----------	-------------

allowDefinition	<p>Especifica en cual archivo de configuración puede ser la sección utilizada. Se debe usar uno de los siguientes valores:</p> <p>Everywhere</p> <p>Permite que la sección sea utilizada en cualquier archive de configuración. Esta es la opción defecto.</p> <p>MachineOnly</p> <p>Permite que la sección solo sea utilizada en el archive de configuración de la máquina (Machine.config).</p> <p>MachineToApplication</p> <p>Permite que la sección solo sea utilizada en el archive de configuración de la máquina o en el archive de configuración de la aplicación.</p>
allowLocation	<p>Determina cuando la sección puede ser utilizada con el elemento <location>. Se debe especificar uno de los valores siguientes:</p> <p>true</p> <p>Permite que la sección sea utilizada con el elemento <location>. Esta es la opción por defecto.</p> <p>false</p> <p>No permite que la sección sea utilizada con el elemento <location>.</p>

Obteniendo los parámetros de Configuración

ASP.NET permite a los desarrolladores acceder a los "settings" de configuración desde una aplicación a través de la exposición de los "settings" de configuración directamente (como propiedades fuertemente tipadas) o empleando APIs de configuración. El siguiente ejemplo muestra una página que accede a la sección de configuración **<browserCaps>** empleando la propiedad **Browser** de la clase **System.Web.HttpRequest**.

A continuación se muestra el archivo capacNav.aspx conteniendo el código para realizar lo indicado:

```
<%@ Page Language="VB" %>

<html>

<body style="font: 10pt verdana">

    <h3>Obteniendo las capacidades del Navegador </h3>

    Boolean                                ActiveXControls                                =
    <%=Request.Browser.ActiveXControls.ToString()%> <br>
```

```

Boolean AOL = <%=Request.Browser.AOL.ToString()%><br>

Boolean                                BackgroundSounds                =
<%=Request.Browser.BackgroundSounds.ToString()%><br>


Boolean Beta = <%=Request.Browser.Beta.ToString()%><br>

String Browser = <%=Request.Browser.Browser%><br>

Boolean CDF = <%=Request.Browser.CDF.ToString()%><br>

Boolean Cookies = <%=Request.Browser.Cookies.ToString()%><br>

Boolean Crawler = <%=Request.Browser.Crawler.ToString()%><br>

Boolean Frames = <%=Request.Browser.Frames.ToString()%><br>

Boolean                                JavaApplets                    =
<%=Request.Browser.JavaApplets.ToString()%><br>

Boolean JavaScript = <%=Request.Browser.JavaScript.ToString()%><br>

Int32                                MajorVersion                    =
<%=Request.Browser.MajorVersion.ToString()%><br>

Double                                MinorVersion                    =
<%=Request.Browser.MinorVersion.ToString()%><br>

String Platform = <%=Request.Browser.Platform%><br>

Boolean Tables = <%=Request.Browser.Tables.ToString()%><br>

String Type = <%=Request.Browser.Type%><br>

Boolean VBScript = <%=Request.Browser.VBScript.ToString()%><br>

String Version = <%=Request.Browser.Version%><br>

Boolean Win16 = <%=Request.Browser.Win16.ToString()%><br>

Boolean Win32 = <%=Request.Browser.Win32.ToString()%><br>


</body>

</html>

```

A continuación se muestra la salida producida:
 Obteniendo las capacidades del Navegador

Boolean	ActiveXControls	=	True
Boolean	AOL	=	False
Boolean	BackgroundSounds	=	True
Boolean	Beta	=	False
String	Browser	=	IE
Boolean	CDF	=	False
Boolean	Cookies	=	True

Boolean	Crawler	=	False
Boolean	Frames	=	True
Boolean	JavaApplets	=	True
Boolean	JavaScript	=	True
Int32	MajorVersion	=	6
Double	MinorVersion	=	0
String	Platform	=	WinXP
Boolean	Tables	=	True
String	Type	=	IE6
Boolean	VBScript	=	True
String	Version	=	6.0
Boolean	Win16	=	False
Boolean Win32 = True			

Adicionalmente a acceder los settings de configuración como se mostró anteriormente, los desarrolladores también pueden emplear la clase **System.Configuration.ConfigurationSettings** para obtener los datos de cualquier sección de configuración. Es importante destacar que el objeto particular retornado por **ConfigurationSettings** depende del handler de sección mapeado con la sección de configuración (en método Create de **IConfigurationSectionHandler.Create** ofrece esta funcionalidad).

El siguiente código demuestra como acceder a la data de configuración expuesta por la sección **<cualquiera>**. En este ejemplo, se asume que el handler de la configuración de la sección retorna un objeto del tipo **SettingsCualquiera** con la propiedad **Enabled**.

```
Dim conf As SettingsCualquiera =
CType(ConfigurationSettings.GetConfig("cualquiera"), SettingsCualquiera)

If conf.Enabled = True Then
    ' Código a ejecutar.
End If
```

Los archivos de configuración son perfectamente adecuados para almacenar settings de un aplicación, tales como las cadenas de conexión con la base de datos, los "paths" de los archivos o direcciones de URL. Las secciones de configuración por defecto ofrecen una sección **<appSettings>** la cual puede ser utilizada para almacenar estos settings en pares nombre/valor. El ejemplo siguiente muestra como configurar un string de conexión de una base de datos para una aplicación.

```
<configuration>
<appSettings>
<add key="sistConf" value="server=(local);database=sistConf;Integrated
Security=SSPI" />
</appSettings>
</configuration>
```

El objeto **ConfigurationSettings** expone una propiedad especial llamada **AppSettings** la cual puede ser utilizada para obtener estos "settings"

```
Dim sConex As String = ConfigurationSettings.AppSettings("sistConf")
```

Dentro del archivo de configuración web.config también se puede crear una sección para **manejar los errores** de una **manera genérica** creando una sección llamada **<customErrors>** **</customErrors>**, permitiendo administrar un error el cual es **independiente de la página** donde se produjo el mismo.

22 MANEJO DEL CACHE

Introducción

El caché es una técnica ampliamente empleada en computación para incrementar el rendimiento de una aplicación. La técnica consiste en almacenar en memoria los datos más utilizados o los más costosos de acceder. En el contexto de una aplicación WEB, el caché es utilizado para almacenar

páginas o datos utilizados en los requerimientos HTTP y reutilizarlos sin pagar el precio de volverlos a cargar y reconstruir.

ASP.NET posee tres tipos de caché para poder ser utilizados en las aplicaciones Web:

- **Output caching**, para almacenar la respuesta dinámica generada por un requerimiento.
- **Fragment caching**, para almacenar parte de la respuesta dinámica generada por un requerimiento.
- **Data caching**, para almacenar los objetos deseados de una manera programática. ASP.NET soporta un mecanismo de caché permitiendo a los programadores almacenar datos transmitidos entre los requerimientos

Output caching es útil cuando es necesario capturar el contenido de una página completa. En un site muy accedido , almacenar en el caché las páginas más frecuentemente utilizadas puede sustancialmente mejorar la capacidad de atención "throughput" de la página.

En ocasiones, no es práctico realizar el caché de una página completa, debido al dinamismo o la personalización involucrada en la misma. En este caso, es mejor identificar los objetos candidatos a ser almacenados en el caché. Alternativamente se pueden almacenar porciones de la página.

Es también importante decidir los intervalos de tiempos para realizar el caché. Existen elementos que requieren actualizar los datos en un periodo de tiempo, en esta caso el elemento en el caché deber trabajar con una política de expiración de modo de ser refrescado cuando ya no es válido. En ASP.NET cuando un elemento cumple su lapso de expiración es eliminado del caché, cuando el elemento requiere ser utilizado como no se consigue en el caché tiene que ser recreado.

El caché de ASP.NET soporta dependencias de archivos y dependencias de claves, permitiendo a los programadores asociar el caché de un elemento a un archive externo o a otro elemento con caché. Esta técnica puede ser utilizada para invalidar elementos cuando la fuente de datos asociada es modificada.

Page Output Caching

Output caching se encuentra habilitado por defecto, pero el resultado de una respuesta no es almacenada en el caché hasta que se produzca implícitamente una acción.

Para que una respuesta pueda ser almacenada en un caché, ésta debe tener una política de expiración/validación válida y la visibilidad de caché pública. Estas operaciones pueden ser efectuadas a través del API en bajo nivel **OutputCache** API o la directiva de alto nivel **@OutputCache**. Cuando output caching se encuentra activo, el resultado del requerimiento es almacenado cuando se realiza el **GET** de la página por primera vez. Los siguientes requerimientos **GET** o **HEAD** son servidos del caché hasta que este expire.

Output caché respeta las políticas de expiración y validación asociadas con las páginas. Si una página se encuentra en el caché y posee una política de expiración indicando que la página expira en 60 minutos, la página es eliminada del caché una vez pasados los 60 minutos. Si la página recibe otro requerimiento, se ejecuta nuevamente el código de la página y esta es almacenada en el caché nuevamente. Este tipo de política de expiración es llamado expiración absoluta

El siguiente ejemplo muestra como utilizar la directiva **OutputCache**. El ejemplo simplemente muestra el tiempo cuando fue generada la respuesta. Para probar el código, se debe invocar la página y tomar en cuenta el tiempo de creación. Al refrescar la página se observa que el tiempo de actualización no cambió.

```
<%@ OutputCache Duration="60" VaryByParam="none" %>
```

```
<html>
```

```
<script language="VB" runat="server">
```

```
Sub Page_Load(Src As Object, E As EventArgs)

    mensajeTiempo.Text = DateTime.Now.ToString("G")

End Sub

</script>

<body>

    <h3><font face="Verdana"> Ejemplo Output Cache</font></h3>

    <p><i>Ultima vez cargado :</i> <asp:label id=" mensajeTiempo "
runat="server"/>

</body>

</html>
```

La siguiente directiva activa el output caching en la respuesta:

```
<%@ OutputCache Duration="60" VaryByParam="none"%>
```

Esta directiva simplemente indica que la página debería ser almacenada en el caché durante 60 segundos y la página no modifica ninguno de los parámetros **GET** o **POST**. Los requerimientos recibidos son atendidos desde el caché. Después de transcurridos los 60 segundos, la página es removida del caché y el requerimiento siguiente es manejado explícitamente y la página nuevamente almacenada en el caché.

Desde luego, en el ejemplo anterior, muy pocos elementos son almacenados en el cache. Pero esta técnica de output caché podría ser utilizada en cualquier otro contexto, como por ejemplo una consulta en una base de datos la cual despliega los resultados en un data grid.

Las aplicaciones que requieren un control mayor sobre los headers HTTP relacionados con el caché pueden emplear la funcionalidad provista por la clase **System.Web.HttpCachePolicy**. El siguiente ejemplo muestra en código utilizando esta clase la misma funcionalidad de caché mostrada en los ejemplos anteriores.

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(60))

Response.Cache.SetCacheability(HttpCacheability.Public)
```

Para permitir una política de expiración corrediza "sliding" , donde el tiempo de expiración se actualice cada vez que la página es requerida, se debe actualizar la propiedad **SlidingExpiration** como es mostrado en el siguiente código

```
Response.Cache.SetSlidingExpiration(True)
```

Page Fragment Caching

ASP.Net también ofrece la posibilidad de almacenar en el caché regiones del contenido de una página. Las regiones de la página se determinan por medio de controles de usuario y se marcan

empleando la directiva **@ OutputCache**. Por ejemplo la siguiente directiva le indica a ASP.NET realizar el cache para el control del usuario por 120 segundos, y variar el caché empleando los parámetros de consulta o parámetros del post "categoriaID" y "seleccionID".

```
<%@ OutputCache Duration="120"  
VaryByParam="categoriaID;seleccionID"%>
```

El atributo **VaryByParam** es muy poderoso y permite al control de usuario indicar a ASP.NET el almacenamiento en el caché de múltiples instancias de una región en el servidor. Por ejemplo, los siguientes URLs permiten separar las instancias de los controles de usuario cada quién con su respectivo caché.

<http://localhost/mypage.aspx?categoriaId=cursos&seleccionId=0>
<http://localhost/mypage.aspx?categoriaId=cursos&seleccionId=1>

La lógica dentro de un control de usuario puede entonces dinámicamente generar diferente contenido (y estos son almacenados en diferentes cachés) dependiendo de los parámetros provistos.

ASP.Net también soporta el atributo **VaryByControl** para realizar el caché. En el caso del atributo **VaryByControl** los caché varían dependiendo de los valores contenidos en los controles dentro del control de usuario. Por ejemplo:

```
<%@ OutputCache Duration="120" VaryByParam="none" VaryByControl="categoria" %>
```

Si el control de usuario contiene una caja de selección drop-down llamado categoría, la salida producida por el control de usuario será diferente (y por ende la información almacenada en el caché) dependiendo del valor seleccionado dentro del control.

Page Data Caching

ASP.NET soporta un servicio de caché poderoso permitiendo almacenar y obtener del caché cualquier objeto solicitado a través de requerimientos de HTTP. El caché de ASP.NET es privado para cada aplicación y almacena los objetos en memoria. El tiempo de vida del caché es equivalente al tiempo de vida de la aplicación. Por lo tanto, cuando la aplicación es reiniciada, el caché es recreado.

El caché provee una interfaz del tipo "diccionario" permitiendo a los programadores almacenar y obtener objetos del caché de una manera cómoda. De una manera sencilla, colocar un elemento en el caché es idéntico a agregar un elemento en el diccionario:

```
Cache("clave") = valor
```

Obtener los elementos del caché es también simple. La existencia del objeto debe ser chequeada antes de obtener los datos del caché como se muestra en el ejemplo siguiente:

```
valor = Cache("clave")  
  
If Not (valor Is Nothing) Then  
    DisplayData(valor)  
  
EndIf
```

En aplicaciones donde se requiere una funcionalidad más sofisticada, el caché de ASP.NET soporta **"scavenging"**, **expiración**, y **dependencias de archivos y de claves**.

- **"Scavenging"** significa que el caché intenta eliminar los elementos menos utilizados o los menos significativos si la memoria se torna escasa. Los programadores que deseen controlar como el "scavenging" funciona, pueden proveer pistas al "scavenger" indicándole el costo relativo de almacenar un elemento en el caché y el factor de utilización relativo para indicar que el elemento sigue siendo útil.
- La **expiración** permite a los programadores expresar tiempos de vida para los elementos, los cuales pueden ser explícitos (por ejemplo, expira a las 8:00) o pueden ser relativos al tiempo de última utilización del elemento (por ejemplo, expira 10 minutos después de su último acceso).

- Las **dependencias de archivos y de claves** permiten expresar la validez de un caché en términos de un archivo externo o de otro elemento almacenado en el caché. Si una dependencia cambia, el elemento del caché es invalidado y removido del caché. Por ejemplo, una aplicación obtiene información financiera de un archivo XML, el cual es periódicamente actualizado. La aplicación procesa los datos en el archivo y crea un gráfico. La aplicación almacena los datos en el caché e inserta una dependencia en el archivo a partir del cual los datos fueron leídos. Cuando el archivo es actualizado, los datos son eliminados del caché y la aplicación vuelve a leer el archivo y coloca otra vez la referencia.

23 USER CONTROLS

Introducción

Adicionalmente a los controles "built-in" del lado del servidor provistos por ASP.NET, el programador puede definir de una manera sencilla nuevos controles (en la terminología .NET se denominan user controls) aplicando las mismas técnicas de programación empleadas para escribir páginas basadas en formularios WEB. En efecto, con solo realizar unas pequeñas modificaciones, cualquier página web form puede ser reutilizada en otra página como un control de servidor (La clase para de los controles de usuario **System.Web.UI.UserControl**, hereda directamente de **System.Web.UI.Control**). Una manera directa de crear un User Control es a través de la combinación de controles existentes, estos controles que agrupan estos controles ya existentes se denominan Composite Controls.

Una página web form usada como un control de servidor es llamada user control. Para respetar una convención de nombres, la extensión .ascx es utilizada para identificar este tipo de controles. Esta convención, también garantiza que un archivo .ascx no puede ser ejecutado como un página web form "standalone". Los User controls son incluidos en una página Web Form empleando la directiva **Register**:

```
<%@ Register TagPrefix="GALA" TagName="Mensaje" Src="UCGala.ascx" %>
```

TagPrefix determina un espacio de nombres único para el user control (de esta manera múltiples user controls con el mismo nombre pueden ser diferenciados de los otros). **TagName** es el nombre único para el user control (cualquier nombre puede ser utilizado). El atributo **Src** es el camino virtual para el user control – por ejemplo "UCGala.ascx" r "/application/incluye/UCGala.ascx". Después de registrar el user control, el tag del user control puede ser colocado en la página Web Forms como si se empleara in control de servidor (colocando el atributo **runat="server"**):

```
<GALA:Mensaje runat="server"/>
```

A continuación se muestra un ejemplo, en el archivo UC.aspx conteniendo una página de web form se indica el uso de un user control definido en el archivo UC.ascx. El archivo UC.aspx contiene

```
<%@ Register TagPrefix="GALA" TagName="Mensaje" Src="UC.ascx" %>
```

```
<html>
```

```
<body style="font: 10pt verdana">
```

```
<h3>Ejemplo de User Control mostrando un mensaje </h3>
```

```
<GALA:Mensaje runat="server"/>

</body>

</html>
```

El archivo UC.ascx contiene únicamente texto estático

Ejemplo de un mensaje empleando USER CONTROL GALA Institute

Exponiendo las propiedades de un User Control

Cuando una página de web form es tratada como un control, los campos públicos y métodos de la web form son promocionados como propiedades públicas (como atributos de tag) y métodos del control. En este ejemplo se muestra una extensión del anterior donde se agregan dos campos públicos del tipo String. Estos campos pueden ser especificados en la página de una manera programática o de una manera declarativa.

A continuación se muestra un ejemplo de un User Control conteniendo dos propiedades Color y textoMensaje.

El archivo UC2.aspx contiene

```
<%@ Register TagPrefix="GALA" TagName="Mensaje" Src="UC2.ascx" %>

<html>

<script language="VB" runat="server">

    Sub bSummit_Click(Sender As Object, E As EventArgs)

        bMensaje.textoMensaje = "El texto del mensaje cambio!"

        bMensaje.Color = "red"

    End Sub

</script>

<body style="font: 10pt verdana">

<h3>Ejemplo User Control con Propiedades</h3>

<form runat="server">

    <GALA:Mensaje id="bMensaje" textoMensaje="Mensaje personalizado !"
    Color="blue" runat="server"/>

    <p>

        <asp:button text="Cambiar Propiedades" OnClick="bSummit_Click"
```

```
runat=server/>

</form>

</body>

</html>
```

El archivo UC2.ascx contiene

```
<script language="VB" runat="server">

    Public Color As String = "blue"

    Public textoMensaje As String = "Ejemplo de un UC con propiedades publicas
    !"

</script>

<span
    style="color: <%=Color%>"><%=textoMensaje%></span>
    id="Mensaje"
```

Adicionalmente a mostrar campos colocando campos públicos se puede utilizar la sintaxis Property (con Get y Set) . Este código se colocaría en el archivo .ascx.

Encapsulando Eventos en un User Control

Un User Control participan completamente en la ejecución de un ciclo de vida de un **"request"**, prácticamente de la misma manera como participan los controles del lado del servidor. Por lo tanto, un User Control puede manejar sus propios eventos, encapsulando la lógica requerida en el User Control permitiendo obtener una funcionalidad mayor. El siguiente ejemplo demuestra el uso de manejo de eventos al producirse una selección de DropDownList .

El archivo UC3.aspx contiene

```
<%@ Register TagPrefix="GALA" TagName="ListaLibros" Src="UC3.ascx" %>

<html>

<body style="font: 10pt verdana">

    <h3>Ejemplo de User Control con evento de Selección de un dropdown
    </h3>

    <form runat="server">

        <GALA:ListaLibros runat="server"/>

    </form>

</body>

</html>
```

El archivo UC3.ascx contiene (aquí es donde se manejan los eventos)

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

<%@ Import Namespace="System.Data.SqlClient" %>

<script language="VB" runat="server">

    Sub Page_Load(Src As Object, E As EventArgs)

        If Not (Page.IsPostBack)

            ' Colocar código de acceso a Base de Datos

            ' Donde se puede tener una consulta donde la parte

            ' del where tiene Categoria.SelectedItem.Value

            ' para ser colocados en un data list o un repeater

            ' o un data grid

        End If

    End Sub

    Sub SeleccionCategoria(Sender As Object, E As EventArgs)

        ' Colocar código de acceso a Base de Datos

        ' Donde se puede tener una consulta donde la parte

        ' del where tiene Categoria.SelectedItem.Value

        ' para ser colocados en un data list o un repeater

        ' o un data grid

    End Sub

</script>

<table style="font: 10pt verdana">

    <tr>

        <td><b>Select a Category:</b></td>

        <td style="padding-left: 15">

    <table style="font: 10pt verdana">

        <tr>

            <td><b>Select a Category:</b></td>

            <td style="padding-left: 15">
```

```
<ASP:DropDownList      AutoPostBack="true"      id="Categoria"
OnSelectedIndexChanged="SeleccionCategoria" runat="server">

    <ASP:ListItem value="Deportes">Deportes</ASP:ListItem>

    <ASP:ListItem value="Finanzas">Finanzas</ASP:ListItem>

    <ASP:ListItem value="Economia">Economia</ASP:ListItem>

</ASP:DropDownList>

</td>

</tr>

</table>
```

Creando User Controls dinámicos (de manera programática)

Al igual que los controles de servidor, los User Controls pueden ser creados dinámicamente. El método **LoadControl** de la página es utilizado para cargar el User Control y requiere como parámetro el camino virtual del archivo que implementa el User Control.

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

    Page.Controls.Add(New HtmlGenericControl("hr"))

    Dim c1 As Control = LoadControl("UC3din.ascx")

    CType(c1, UC3).Categoria = "Deportes"

    Page.Controls.Add(c1)

End Sub
```

24 CÓDIGO DEL LADO DEL CLIENTE

Funcionalidad de "Callback" en los clientes en ASP.NET 2.0

Una de las funcionalidades más poderosas de ASP.NET es la disponibilidad de realizar **"callback"** del lado del cliente. Esta funcionalidad permite de una manera programática invocar métodos del lado del servidor a través de código del lado del cliente, el cual es **código en JavaScript**, sin necesidad de realizar **"post Back"** de la página. El código de las validaciones implementadas en los objetos **Validators**, es también de este tipo (**JavaScript**).

La necesidad de realizar llamadas del lado del cliente

Debido a que el protocolo HTTP no almacena el estado del requerimiento **"stateless"**, cada vez que un cliente Web requiere obtener datos de un servidor o quiere invocar alguna porción de código ejecutándose del lado del servidor el servidor debe enviar la página. En el caso de realizarse un "post-back", los manejadores de los eventos ejecutarán el código del lado del servidor y enviarán los datos de respuesta al navegador cliente, el cual debe mostrar los datos nuevamente. Este modelo de eventos trabaja apropiadamente en la mayoría de los casos, pero impone unas restricciones con las cuales los desarrolladores deben programar.

Para mantener el estado de los controles de entrada en los formularios del cliente, el programador o trabaja con la información del **ViewState** (lo cual afecta en el rendimiento del procesamiento en el servidor) o debe codificar lógica complicada del lado del servidor. En segundo lugar, cada vez que se requiere cargar la página del lado del cliente, tiempo de computación es consumido.

Por estas razones, la necesidad de realizar llamadas del lado del cliente siempre ha estado latente desde el punto de vista de los desarrolladores. En ASP.NET 2.0 esta funcionalidad se encuentra disponible.

Implementación a llamadas del lado del cliente

Antes de describir esta funcionalidad en ASP.NET 2.0 se explicará como se puede realizar en versiones anteriores. Llamadas a métodos del lado del cliente desde **JavaScript** puede realizarse con un objeto **ActiveX XMLHttpRequest** de Microsoft.

Este objeto **ActiveX** permite manipular archivos **XML** en Internet empleando el protocolo **HTTP**. Sin embargo, como su nombre lo indica, este objeto puede ser utilizado para realizar un requerimiento **HTTP** en cualquier servidor, incluyendo versiones previas de ASP, HTML estándar e inclusive archivos PHP.

Debido a que este objeto **XMLHTTP** es un objeto **ActiveX** estándar, este puede ser instanciado empleando un **JavaScript**. En el siguiente ejemplo se muestra como obtener el código html de la página de GALA , a través de código **JavaScript**:

```
function obtenerGala() {  
  
    var XmlHttp = new ActiveXObject("Msxml2.XMLHTTP.4.0");  
  
    XmlHttp.Open("GET", "http://www.galainstitute.com", false);  
  
    XmlHttp.Send();  
  
    return XmlHttp.responseText;  
  
}
```

En este ejemplo se demuestra como se utiliza el objeto **XmlHttp** donde únicamente es necesario pasar como parámetro el URL a obtener y se obtiene en texto el contenido completo de la página. Todo el código se ejecuta en el **JavaScript** del lado del cliente, por lo tanto no se realiza postback.

Llamadas del lado del cliente en ASP.NET 2.0's

La nueva versión de ASP.NET encapsula (abstrae) para los programadores el objeto **XMLHTTP** . Internamente la función de "callback" en los clientes, emplea el objeto **XMLHTTP** , pero para el programador de la aplicación Web esto es transparente.

La funcionalidad de "**callback**" en los clientes consiste de dos partes: la **interfaz **ICallbackEventHandler**** y el **método **Page.GetCallbackEventReference****.

El método **Page.GetCallbackEventReference** y cualquier sobrescritura de este creará el código **JavaScript** que se necesita colocar del lado del cliente. Este código contiene las llamadas necesarias para enviar un requerimiento de **HTTP** a la página realizando el requerimiento (empleando en bajo nivel un objeto **XMLHTTP**). El requerimiento es luego manejado del lado del servidor por un control Web implementando la interfaz **ICallbackEventHandler** . En la mayoría de los casos, el control Web es la misma página, pero deben ser especificados controles de usuario específicos o controles Web que responden a los eventos generados por los requerimientos. Una vez manejado el requerimiento, el resultado es enviado al cliente a través de otra función **JavaScript** la cual únicamente tiene como propósito reaccionar al resultado del requerimiento.

M019 CURSO ASP .NET Y SERVICIOS WEB , 24 Horas
Copyright, Instituto GALA de Venezuela 2004, 2005, 2006

A continuación se muestra un ejemplo el cual muestra la hora del servidor y la muestra utilizando el JavaScript alert:

```
<%@ page language="VB" compilewith="TiempoServ.aspx.vb"
classname="ASP.TiempoServ_aspx" %>

<html>

<head>

    <title>Server Time</title>

    <script language="javascript">

function obtTiempo() {

    var mensaje = "";

    var contexto = "";

    <%=funcioninvocada%>

}

function mostrarTiempo(mensaje, contexto) {

    alert('El tiempo en el servidor es :\n' + mensaje);

}

function manError(mensaje, contexto) {

    alert('Un error ha ocurrido :\n' + mensaje);

}

</script>

</head>

<body>

    <form id="MainForm" runat="server">

        <input type="button" value="obtenerTiempoServidor" onclick="obtenerTiempo();" />
    </form>

</body>

</html>

imports System

imports System.Web.UI;
```

```
namespace ASP

public class TiempoServ implements ICallbackEventHandler

    public string funcioninvocada;

    Sub Page_Load(sender as object, e System.EventArgs)

        funcioninvocada = "this.GetCallbackEventReference
(this,"mensaje","mostrarTiempo","contexto","manError")";

    End Sub

    Function RaiseCallbackEvent(argumento as String) as string

        ' Eliminar la linea de comentarios para probar el manejador de errores

        ' throw new ApplicationException("Excepción no manejada");

        return DateTime.Now.ToString();

    End Function

End Class

End Namespace
```

En este ejemplo se muestra que **Page** implementa la **interfaz ICallbackEventHandler**. Esta interfaz sólo posee un método llamado **RaiseCallbackEvent**.

Este método es ejecutado cuando un requerimiento es manejado, en este caso simplemente devuelve el tiempo actual en el servidor.

Para crear el código del cliente, el método **Page.GetCallbackEventReference** es invocado en el page load. Este método creará un código JavaScript cuando es invocado, el cual iniciará el "callback" del cliente.

El método **GetCallbackEventReference** puede ser sobrecargado de varias maneras , pero en todas las operaciones de sobrecarga se debe indicar cual control Web reaccionará al requerimiento (en este caso es la misma página).

Los nombres de las variables JavaScript contiene los parámetros específicos para este requerimiento y las funciones JavaScript que son llamadas cuando el requerimiento se sirve o se produce el error.

Como este método devuelve el código JavaScript para iniciar el "callback", este es colocado en una función JavaScript invocada cuando se presiona Click. En tiempo de ejecución, la expresión `<%=funcioninvocada%>` evaluará:
__doCallback('__Page',mensaje,mostrarTiempo,contexto,manError)

__doCallback es una función interna de JavaScript en ASP.NET 2.0 la cual ejecutar el requerimiento de HTTP realizando el "callback" .

El proceso realizado por ASP traduce los nombres de las variables y de las funciones de javascript en el método. El código de javascript reside del lado del cliente, el código de visual basic.net reside en el lado del servidor.

Cuando se ejecuta el "callback", la variable mensaje es el primer parámetro pasado al método **RaiseCallbackEvent** (el cual se ejecuta del lado del servidor) . Es importante comprender que el valor

String del JavaScript es pasado a un método .NET. La manera de pasar parámetros **es a través de strings**, de requerirse pasar una estructura de datos más compleja debe ser construido un string conteniendo los datos. En este ejemplo, no se requiere pasar ninguna información al servidor, por lo tanto se inicializa la variable con un string vacío. El mismo caso aplica para el valor devuelto por este método.

En este caso, el string de .NET es retornado al método javascript como un valor de string.

Las dos funciones de JavaScript mostrarTiempo y manError se explican por si solas. Estas funciones serán invocadas por el servidor (se ejecutan en el cliente) cuando se finaliza el requerimiento. El primer parámetro de mostrarTiempo contiene el valor retornado por el método RaiseCallbackEvent (el cual es el tiempo del servidor).

El primer parámetro de manError contiene el valor del tipo de la excepción ocurrida en el servidor.

La variable contexto es una variable interesante, debido a que aunque esta variable es pasada a la función esta no es pasada al método del lado del servidor (debido a que el método RaiseCallbackEvent sólo tiene un parámetro). En lugar de esto, la variable contexto es almacenada en el caché del navegador durante el "callback" y luego pasada como el segundo a las funciones Javascript. Esto nos permite identificar el contexto de la llamada entera.

Ejemplo de validación del lado del Cliente

ASP.NET soporta varios controles de validación como RequiredFieldValidator, CompareValidator, y otros - los cuales son útiles para asegurar que los usuarios introducen valores válidos en los campos de los formularios.

En el caso de Internet Explorer, estos controles generan JavaScript a ejecutar del lado del cliente. Par el caso de navegadores non-Microsoft sólo validación del lado del servidor es posible.

Como se muestran los controles de validación del lado del cliente
Todos los controles de validación de ASP.NET derivan (heredan) de la clase System.Web.UI.WebControls.BaseValidator. La clase BaseValidator define los métodos y las propiedades comunes a todos los controles de validación. Una de las funciones de esta clase es determinar cuando un control de validación puede o no ser ejecutado con un script del lado del cliente. Esto es realizado por el método DetermineRenderUplevel().

Agregando Message Boxes del lado del cliente en ASP.NET

Una de las mayores ventajas de una aplicación Windows es la utilización de los message boxes, como las alertas y las confirmaciones.

El problema de los message boxes es que estos son creados en el cliente y no pueden ser creados como código del lado del servidor. En este caso el código del lado del servidor necesita producir el código JavaScript apropiado, el cual se ejecuta del lado del cliente, con el HTML enviado al cliente para que éste pueda desplegar el message Box.

Para desplegar un messagebox, una manera de realizarlo ASP.NET se puede crear un botón HTML <input> y agregar un evento JavaScript onclick al botón el cual invoca el método confirm de JavaScript **confirm()**, para desplegar el message box del lado del cliente cuando el botón es presionado.

Sin embargo en ASP.NET, existe otra manera más eficiente de realizarlo y de una manera programática. El primer paso es añadir un control web button y colocare como id BBorrar, esto produce el código :

```
<input type="submit" name="BBorrar" value="Borrar" id="BBorrar" />
```

Luego se le puede añadir el onclick event el cual contiene código JavaScript, de la siguiente manera

```
<input type="submit" name="BBorrar" value="Borrar" id="BBorrar"
onclick="return confirm('Seguro desea borrar?');" />
```

De una manera programática, todos los controles Web de ASP.NET soportan la **propiedad Attributes** conteniendo una colección de atributos del elemento HTML mostrados (y se almacenan de pares atributo, valor). La propiedad attributes posee un **método Add(key, value)**, el cual puede ser utilizado para agregar un atributo nuevo para el control web. De una manera programática, se puede realizar:

```
Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    If (Not Page.IsPostBack) Then

        Me.BBorrar.Attributes.Add("onclick", _

            "return confirm('Seguro desea borrar?');")

    End If

End Sub
```

Debido a que este código sólo necesita ser añadido una vez (y no cada vez que se carga la página), este código es colocado en el bloque **Not Page.IsPostBack** block.

La función confirm en JavaScript despliega un message conx de confirmación, con los botones OK y Cancel. Si se presiona el botón Ok esta función retorna true, en caso contrario esta función retorna false. Por lo tanto, es necesario colocar un return a la función confirm cuando se agrega la propiedad.

Si se requiere agregar un message box de confirmación para varias páginas ASP.NET, se podría crear una clase llamada Utilitarios con un método estático que reciba como parámetro un botón (de control Web) y un mensaje que actualiza las propiedades del Botón apropiadamente.

```
Public Class Utilitarios

    Public Shared Sub crearCajaConfir(

        ByRef bot As WebControls.Button, _

        ByVal mens As String)

        bot.Attributes.Add("onclick", "return confirm("'" & mens & "'");")    End Sub

End Class
```

Entonces este método puede ser invocado en el Page_load:

```
Utilities.CreateConfirmBox(Me.BBorrar, "Seguro desea borrar?")
```

De esta manera podemos reutilizar este código.

Agregando Alert Message Boxes del lado del cliente en ASP.NET

ASP.NET provee un conjunto de métodos para manejar los JavaScript de html sin necesidad de modificar los HTML de manera manual.

M019 CURSO ASP .NET Y SERVICIOS WEB , 24 Horas

Copyright, Instituto GALA de Venezuela 2004, 2005, 2006

En estos casos se requiere registrar un bloque de script que se ejecuta en el cliente dentro de la página web asp.net y esto debe ser ejecutado cuando el cliente ejecuta la página.

La clase Page de ASP.NET provee un método llamado RegisterStartupScript

En el siguiente ejemplo se muestra el código para el botón de Guardar.

```
Private Sub BSalvar_Click(ByVal sender As System.Object, _           ByVal e As
System.EventArgs) Handles BSalvar.Click

    'Generar el mensaje

    Dim mensaje As String

    If (Me.CheckBox1.Checked()) Then

        Mensaje = "Los datos fueron salvados"

    Else

        Mensaje = " Los datos no fueron salvados "

    End If

    'Finaliza el procedimiento y retorna el script.

    Dim scriptM As String = "<script language=JavaScript>"

    scriptM += "alert('"' & Mensaje & "');"

    scriptM += "</script>"

    If (Not Page.IsStartupScriptRegistered("scriptCliente")) Then
        Page.RegisterStartupScript("scriptCliente", scriptM)
    End If

End Sub
```

El método RegisterStartupScript() toma dos argumentos : el primero una clave que identifica al script registrado y el código del script. El Segundo parámetro incluye el tag <script> y el código necesario para ejecutar (alert('...'));

Antes de ejecutar RegisterStartupScript() es importante revisar si el script fue previamente registrado.

25 XML WEB SERVICES

Introducción

La programación en Internet evoluciona rápidamente desde las **aplicaciones WEB** más comunes que permiten a través de un navegador mostrar interfaces de usuario con el contenido deseado hasta la próxima generación de aplicaciones que permiten la integración de negocios, organizaciones, aplicaciones, servicios, y dispositivos entre otros. La tecnología que permite realizar esta integración, se llama **servicios WEB XML inteligentes**, permitiendo acceder a los recursos WEB como servicios de programación.

El **CLR (Common Language Runtime)** de la plataforma .NET ofrece un soporte para la creación y publicación de los servicios WEB, empleando una abstracción de programación la cual es consiste con los desarrolladores de ASP.NET o los desarrolladores en otro lenguaje orientado por objetos. El modelo resultante es escalable y extensible y se basa en estándares de Internet abiertos y públicos (open source) como **HTTP, XML, SOAP, WSDL**. Por lo tanto un servicio WEB puede ser utilizado y accedido (consumido en la terminología de servicios WEB) en cualquier plataforma donde Internet se encuentre disponible.

Servicios Web XML empleando ASP.NET

ASP.NET ofrece soporte total para la creación de servicios WEB XML con los **archivos .asmx**. Un archivo .asmx es un archivo de texto similar a un **archivo .aspx**. Estos archivos (.asmx) también pueden ser parte de una aplicación ASP.NET que incluye archivos .aspx files. Los archivos .asmx, al igual que los archivos .aspx son direccionables a través de un URI.

A continuación se muestra un ejemplo de un archivo .asmx sencillo

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

```
Imports System
```

```
Imports System.Web.Services
```

```
Public Class ejemploSW : Inherits WebService
```

```
    <WebMethod()> Public Function mensaje() As String
```

```
        Return("Ejemplo Servicio WEB")
```

```
    End Function
```

```
End Class
```

El ejemplo anterior comienza con la directiva de ASP.NET **WebService**, y permite especificar el lenguaje de programación a C#, Visual Basic, o JScript. Para poder utilizar los servicios se requiere utilizar el espacio de nombres **System.Web.Services**. Es necesario también crear una clase derivada de **WebService**; no es obligatorio heredar de **WebService**, pero es útil para tener disponibles los métodos y atributos. Los métodos que serán accesibles como parte del servicio deben tener el atributo **[WebMethod]** en C#, **<WebMethod()>** en Visual Basic, o **WebMethodAttribute** en JScript, antes de especificar la "signatura" del método.

Para colocar este servicio disponible, el archivo .asmx debe ser colocado en un directorio virtual de un dominio y el nombre del archivo .asmx puede ser invocado como un URL. Ejemplo, si el archivo del servicio web se llama SW1.asmx y se coloca en el directorio virtual local host, al escribir en el navegador **http://localhost/SW1.asmx** mostrará como página resultante la información conteniendo los métodos públicos de este servicio WEB (aquellos marcados con el atributo **WebMethod**), e igualmente indicará con cuales protocolos (SOAP o HTTP GET) se pueden invocar los métodos.

Especificando la dirección **http://localhost/SW1.asmx?WSDL** retornará un documento en formato WSDL (Web Service Description Language). Este documento WSDL es muy importante y será utilizado por los clientes consumiendo el servicio.

Utilizando o Accediendo (consumiendo) servicios WEB

Adicionalmente a la tecnología ASP.NET del lado del servidor que permite a los desarrolladores crear servicios WEB XML, la plataforma .NET provee un conjunto de herramientas y clases sofisticadas para consumir los servicios web XML. Debido a que los servicios web se basan en protocolos públicos (open) como el **SOAP (Simple Object Access Protocol)**, esta tecnología del lado del cliente también puede ser utilizada para consumir servicios WEB no creados en la plataforma ASP.NET.

Dentro del **Software Development Kit (SDK)**, existe una herramienta llamada **Web Services Description Language tool (WSDL.exe)** – Lenguaje de Descripción de los Servicios Web). Esta herramienta es utilizada para crear clases con interfaces ("proxy classes") partiendo de una especificación WSDL).

Por ejemplo, si en una línea de comandos se coloca:

WSDL http://localhost /SW1.asmx?WSDL

Se crea una clase interfaz llamada ejemploSW.vb.

Desde el punto de vista del cliente, el código luciría como el siguiente.

```
Dim esw As New ejemploSW()  
  
Dim sdev As String = esw.mensaje()
```

26 INTRODUCCIÓN A WEB SERVICES (SERVICIOS WEB)

¿Qué es un servicio Web ?

Es un servicio ofrecido a través de la **Web**.

En un escenario típico se servicios Web se siguen los siguientes pasos:

- Una aplicación **cliente** (de negocio) solicita un **requerimiento** a un servicio a través de un URL empleando el protocolo **SOAP** sobre **HTTP**
- El servicio recibe el **requerimiento**, lo procesa y envía la respuesta

Los servicios web pueden verse como los habilitadores **"building blocks"** para implementar una arquitectura de computación distribuida en Internet. Adicionalmente los servicios Web trabajan bajo estándares abiertos como **XML**, de allí su gran potencial en las áreas de integración de aplicaciones.

Las aplicaciones son construidas invocando varios servicios web, servidos desde varias fuentes, para trabajar de una manera **colaborativa** independientemente de donde residan o de cómo fueron implementados.

Los servicios web exportan su funcionalidad a través de un protocolo web estándar, por lo general emplean **SOAP (Simple Object Access Protocol)**. Los servicios web proveen un mecanismo para describir sus interfaces lo suficientemente detallado y preciso de modo que los clientes pueden comunicarse efectivamente con ellos para solicitar un servicio.

La **descripción de las interfaces** de los servicios Web se realiza a través de un **documento XML** llamado documento **WSDL (Web Services Description Language)**. Adicionalmente, los servicios Web son **registrados** de modo que los usuarios puedan fácilmente encontrarlos, esto se realiza a través de un **UDDI Universal Discovery Description and Integration** o empleando **ebXML**.

Componentes de un servicio web

Un servicio web es un servicio de “software”, el cual es:

- publicado en la web a través de **SOAP**
- descrito con un documento XML llamado **WSDL**
- registrado en un **UDDI**

En un nivel de detalle de implementación, un servicio web contiene un conjunto de:

- Especificaciones de **puertos** (Port Types)
- Definiciones de **esquemas XML**
- Definiciones de **mensajes**
- **Operaciones**
- **“Bindings”** (enlaces) que proveen los detalles de comunicación y protocolos.

Servicios Web vs. Aplicaciones Web

Los servicios web son aplicaciones **distribuidas**, construidos con componentes ejecutándose del lado del **servidor** similar a los sitios web comunes. Sin embargo, se diferencian de las aplicaciones web en el hecho de **no poseer interfaces con el usuario GUI y no ser diseñadas para mostrar directamente en los navegadores**.

Los servicios web es mejor interpretarlos como **componentes de “software” reutilizables**. Los servicios web se encuentran diseñados **para ser consumidos** por otras aplicaciones como:

- aplicaciones clientes tradicionales
- aplicaciones web
- otros servicios web

Arquitectura de los servicios Web

La arquitectura de los servicios web posee tres características principales:

- Una capa de **invocación del servicio** ("consumer")
- Una capa para **el manejo del flujo** entre el servicio y el proceso
- Una capa de **funcionalidad del servicio**

Beneficios de los servicios Web

El desarrollo de "software" industrial se encuentra migrando a una arquitectura basada en servicios web debido a las múltiples ventajas que ofrece sobre la arquitectura tradicional de desarrollo. Una de estas ventajas es que cualquiera de las capas **puede ser cambiada independientemente** de las otras y el grupo de desarrollo responsable de desarrollar cada una de las capas **no requiere conocer los detalles** de las otras capas.

En los servicios web se separa la funcionalidad del servicio de la funcionalidad del cliente:

- Permitiendo **compartir y reutilizar** la funcionalidad entre varias aplicaciones
- Ofreciendo **transparencia en las aplicaciones cliente**, no requieren conocer cómo fue implementado el servicio . Por ejemplo Microsoft .Net o Java 2 Enterprise Edition (J2EE) o Enterprise Java Beans (EJB)
- Permite la migración de la funcionalidad del servicio sin necesidad de cambiar o realizar pruebas en las aplicaciones clientes.

Otro de los beneficios de adoptar una arquitectura basada en servicios web es la portabilidad de la plataforma tanto en localidad como el tipo de invocación de los servicios.

27 CONSTRUYENDO WEB SERVICES

Codificando un ejemplo sencillo de Web Service

El desarrollo de un servicio web XML sencillo puede ser realizado en cuestión de minutos empleando cualquier editor de texto (o en la plataforma de Visual Studio). En esta sección se muestra un ejemplo del manejo de operaciones matemáticas, donde se tienen los métodos de sumar, restar, multiplicar y dividir un par de números del tipo Single. En el principio del **archivo .asmx** se coloca la **directiva para especificar la clase** implementando el servicio **y el lenguaje de programación** .

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

En el archivo **.asmx**, se define una clase para encapsular la funcionalidad del servicio. Esta clase debe ser pública, y puede opcionalmente heredar de la clase base **XML Web service**. Se recomienda heredar la clase base para poder reutilizar las propiedades y métodos allí definidos. Cada **método a ser expuesto (publicado)** por el servicio debe ser cualificado con el atributo **[WebMethod]** como parte de su **"signature"**. Si este atributo no es colocado el método no estará

M019 CURSO ASP .NET Y SERVICIOS WEB , 24 Horas
Copyright, Instituto GALA de Venezuela 2004, 2005, 2006

disponible como parte del servicio. Por lo tanto, en caso de requerirse implementar estrategias de ocultamiento de información no se colocaría la cualificación de **[WebMethod]**.

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

Imports System

Imports System.Web.Services

Public Class SWOperMat : Inherits WebService

    <WebMethod()> Public Function Sumar(A As System.Single, B As
System.Single) As System.Single

        Return A + B

    End Function

    <WebMethod()> Public Function Restar(A As System.Single, B As
System.Single) As System.Single

        Return A - B

    End Function

    <WebMethod()> Public Function Multiplicar(A As System.Single, B As
System.Single) As System.Single

        Return A * B

    End Function

    <WebMethod()> Public Function Dividir(A As System.Single, B As
System.Single) As System.Single

        If B = 0
```

```
Return -1

End If

Return Convert.ToSingle(A / B)

End Function

End Class
```

Al igual que los archivos **.aspx**, los archivos implementando servicios web XML **.asmx**, son **compilados automáticamente por el motor en tiempo de ejecución de ASP.NET** cuando el requerimiento al servicio es realizado por primera vez. En los siguientes requerimientos, éstos son servidos empleando la política de caché de ASP.NET donde se almacenan objetos de las páginas.

Cuando se solicita desde un navegador un archivo **.asmx**, **el motor de ejecución de ASP.NET retorna** una página de ayuda asociada con el servicio Web XML describiendo **el contrato del servicio web** (los métodos disponibles)

Servicios Web XML Precompilados

En caso de requerir emplear una clase precompilada para exponerla como un servicio web XML (exponiendo los métodos marcados con el atributo **[WebMethod]**). En este caso el archivo **.asmx** puede contener únicamente la línea.

```
<%@ WebService Class="Aplicacion1.ServicioWeb1" %>
```

Aplicacion1.ServicioWeb1 define la **clase del servicio web XML** y debería ser colocada en el **subdirectorio \bin** de la aplicación ASP.NET.

28 PROBANDO UN WEB SERVICE UTILIZANDO EL INTERNET EXPLORER

Para probar un servicio web desde Internet Explorer se coloca el nombre de la página **.asmx** igual que un URL.

En nuestro ejemplo al colocarlo aparece la siguiente información.

<http://localhost/SW2.asmx>

Al invocar el URL aparece la siguiente información.

Se coloca un enlace para cada uno de los **métodos publicados**

SWOperMat

Las operaciones siguientes son compatibles. Para una definición formal, revise la [descripción de servicios](#).

- [Restar](#)
 - [Sumar](#)
 - [Dividir](#)
 - [Multiplicar](#)
-

Luego se especifica información sobre los **espacios de nombres**

Este servicio Web utiliza <http://tempuri.org/> como espacio de nombres predeterminado.

Recomendación: cambiar el espacio de nombres predeterminado antes de hacer público el servicio Web XML.

Cada servicio Web XML **necesita un espacio de nombres único** para que las aplicaciones de cliente puedan distinguir este servicio de otros servicios del Web. <http://tempuri.org/> está disponible para servicios Web XML que están en desarrollo, pero los servicios Web XML publicados deberían utilizar un espacio de nombres más permanente.

Debe identificar su servicio Web XML con un espacio de nombres que controle. Por ejemplo, puede utilizar el nombre de dominio de Internet de su compañía como parte del espacio de nombres. Aunque muchos espacios de nombres de servicios Web XML parecen direcciones URL, éstos no pueden señalar a recursos reales en el Web. (Los espacios de nombres de los servicios Web XML son los URI.)

En los servicios Web XML que se crean con ASP.NET, se puede cambiar el espacio de nombres predeterminado utilizando la propiedad Namespace del atributo WebService. Este atributo es un atributo aplicado a la clase que contiene los métodos del servicio Web XML. A continuación se muestra un ejemplo de código que establece el espacio de nombres en "<http://microsoft.com/webservices/>":

Una vez seleccionado uno de los enlaces se muestra una página donde se introducen los argumentos de los métodos y además indica la especificación del servicio en el formato SOAP.

Por ejemplo al pulsar el enlace Sumar, la siguiente ventana aparece

SWOperMat

Haga clic [aquí](#) para obtener una lista completa de operaciones.

Sumar

Prueba

Haga clic en el botón 'Invocar', para probar la operación utilizando el protocolo HTTP POST.

Parámetro	Valor
A:	<input type="text"/>
B:	<input type="text"/>
<input type="button" value="Invocar"/>	

SOAP

A continuación se muestra un ejemplo de solicitud y respuesta para SOAP. Es necesario reemplazar los marcadores de posición que aparecen con valores reales.

POST /SW2.asmx HTTP/1.1

Host: localhost

Content-Type: text/xml; charset=utf-8

Content-Length: **length**

SOAPAction: "http://tempuri.org/Sumar"

<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>

<Sumar xmlns="http://tempuri.org/">

<A>**float**

float

</Sumar>

</soap:Envelope>

HTTP/1.1 200 OK

Content-Type: text/xml; charset=utf-8

Content-Length: **length**

<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

```
<soap:Body>

  <SumarResponse xmlns="http://tempuri.org/">

    <SumarResult>float</SumarResult>

  </SumarResponse>

</soap:Body>
</soap:Envelope>
</soap:Envelope>

HTTP/1.1 200 OK

Content-Type: text/xml; charset=utf-8

Content-Length: length

<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

  <soap:Body>

    <SumarResponse xmlns="http://tempuri.org/">

      <SumarResult>float</SumarResult>

    </SumarResponse>

  </soap:Body>

</soap:Envelope>
```

HTTP POST

A continuación se muestra un ejemplo de solicitud y respuesta para HTTP POST. Es necesario reemplazar los marcadores de posición que aparecen con valores reales.

```
POST /SW2.asmx/Sumar HTTP/1.1

Host: localhost

Content-Type: application/x-www-form-urlencoded

Content-Length: length

A=string&B=string

HTTP/1.1 200 OK
```

Content-Type: text/xml; charset=utf-8

Content-Length: **length**

```
<?xml version="1.0" encoding="utf-8"?>

<float xmlns="http://tempuri.org/">float</float>
```

Al colocar los valores 8 y 3, por ejemplo, y pulsar invocar aparece una ventana con el siguiente archivo XML conteniendo la respuesta.

```
<?xml version="1.0" encoding="utf-8" ?>
<float xmlns="http://tempuri.org/">11</float>
```

Entre los tags de float aparece el número 11, el cual es la respuesta de la utilización del servicio.

29 CREANDO UN CLIENTE QUE UTILICE WEB SERVICES

En esta sección se indican los pasos para **consumir un servicio WEB** en un cliente. Se utiliza el mismo ejemplo de operaciones matemáticas descrito en las secciones anteriores.

Para poder consumir un servicio web (utilizar las llamadas a los métodos publicados) se requiere emplear **el utilitario WSDL.exe** desde la línea de comandos. WSDL es la abreviatura de Web Services Description Language , Lenguaje de Descripción de Servicios Web. Este utilitario crea una clase con **interfaz para poder ser utilizada (proxy class)** con el mismo contrato definido en el archivo .asmx (pero solo contiene los métodos públicos, los **WebMethod**)
Luego, se requiere compilar el código con esta clase incluida.

WSDL.exe soporta una variada de opciones desde la línea de comandos, sin embargo para crear un proxy solo una opción es requerida: el URI para el WSDL.

El archivo WSDL el cual se puede generar desde el navegador con el siguiente comando:

```
http://localhost/ SWOperMat.asmx?WSDL
```

Produciendo un archivo .wsdl con el siguiente formato

```
<definitions          xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://tempuri.org/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://tempuri.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>

<s:schema              elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">
```



```
<s:element name="Sumar">

  <s:complexType>

    <s:sequence>

      <s:element minOccurs="1" maxOccurs="1" name="A" type="s:float" />

      <s:element minOccurs="1" maxOccurs="1" name="B" type="s:float" />

    </s:sequence>

  </s:complexType>

</s:element>

<s:element name="SumarResponse">

  <s:complexType>

    <s:sequence>

      <s:element minOccurs="1" maxOccurs="1" name="SumarResult"
type="s:float" />

    </s:sequence>

  </s:complexType>

</s:element>

<s:element name="Restar">

  <s:complexType>

    <s:sequence>

      <s:element minOccurs="1" maxOccurs="1" name="A" type="s:float" />

      <s:element minOccurs="1" maxOccurs="1" name="B" type="s:float" />

    </s:sequence>

  </s:complexType>

</s:element>

<s:element name="RestarResponse">

  <s:complexType>

    <s:sequence>

      <s:element minOccurs="1" maxOccurs="1" name="RestarResult"
type="s:float" />

    </s:sequence>
```

```
</s:complexType>

</s:element>

<s:element name="Multiplicar">

  <s:complexType>

    <s:sequence>

      <s:element minOccurs="1" maxOccurs="1" name="A" type="s:float" />

      <s:element minOccurs="1" maxOccurs="1" name="B" type="s:float" />

    </s:sequence>

  </s:complexType>

</s:element>

<s:element name="MultiplicarResponse">

  <s:complexType>

    <s:sequence>

      <s:element minOccurs="1" maxOccurs="1" name="MultiplicarResult"
type="s:float" />

    </s:sequence>

  </s:complexType>

</s:element>

<s:element name="Dividir">

  <s:complexType>

    <s:sequence>

      <s:element minOccurs="1" maxOccurs="1" name="A" type="s:float" />

      <s:element minOccurs="1" maxOccurs="1" name="B" type="s:float" />

    </s:sequence>

  </s:complexType>

</s:element>

<s:element name="DividirResponse">

  <s:complexType>

    <s:sequence>

      <s:element minOccurs="1" maxOccurs="1" name="DividirResult"
```

```
type="s:float" />

</s:sequence>

</s:complexType>

</s:element>

</s:schema>

</types>

<message name="SumarSoapIn">

  <part name="parameters" element="s0:Sumar" />

</message>

<message name="SumarSoapOut">

  <part name="parameters" element="s0:SumarResponse" />

</message>

<message name="RestarSoapIn">

  <part name="parameters" element="s0:Restar" />

</message>

<message name="RestarSoapOut">

  <part name="parameters" element="s0:RestarResponse" />

</message>

<message name="MultiplicarSoapIn">

  <part name="parameters" element="s0:Multiplicar" />

</message>

<message name="MultiplicarSoapOut">

  <part name="parameters" element="s0:MultiplicarResponse" />

</message>

<message name="DividirSoapIn">

  <part name="parameters" element="s0:Dividir" />

</message>

<message name="DividirSoapOut">

  <part name="parameters" element="s0:DividirResponse" />
```

```
</message>

<portType name="SWOperMatSoap">

  <operation name="Sumar">

    <input message="s0: SumarSoapIn" />

    <output message="s0: SumarSoapOut" />

  </operation>

  <operation name="Restar">

    <input message="s0: RestarSoapIn" />

    <output message="s0: RestarSoapOut" />

  </operation>

  <operation name="Multiplicar">

    <input message="s0: MultiplicarSoapIn" />

    <output message="s0: MultiplicarSoapOut" />

  </operation>

  <operation name="Dividir">

    <input message="s0: DividirSoapIn" />

    <output message="s0: DividirSoapOut" />

  </operation>

</portType>

<binding name="SWOperMatSoap" type="s0: SWOperMatSoap">

  <soap:binding
    transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />

  <operation name="Sumar">

    <soap:operation soapAction="http://tempuri.org/Sumar" style="document"
    />

    <input>

      <soap:body use="literal" />

    </input>

    <output>

      <soap:body use="literal" />

    </output>

  </operation>

</binding>

</service>
```

```
</output>

</operation>

<operation name="Restar">

  <soap:operation soapAction="http://tempuri.org/Restar" style="document"
/>

  <input>

    <soap:body use="literal" />

  </input>

  <output>

    <soap:body use="literal" />

  </output>

</operation>

<operation name="Multiplicar">

  <soap:operation soapAction="http://tempuri.org/Multiplicar" style="document"
/>

  <input>

    <soap:body use="literal" />

  </input>

  <output>

    <soap:body use="literal" />

  </output>

</operation>

<operation name="Dividir">

  <soap:operation soapAction="http://tempuri.org/Dividir" style="document"
/>

  <input>

    <soap:body use="literal" />

  </input>

  <output>

    <soap:body use="literal" />
```

```
</output>

</operation>

</binding>

<service name="SWOperMat">

  <port name="SWOperMatSoap" binding="s0:SWOperMatSoap">

    <soap:address location="http://localhost/SW2.asmx" />

  </port>

</service>

</definitions>
```

En este ejemplo, se pasan varias opciones el lenguaje a generar, el espacio de nombres y el directorio donde colocará el Proxy.

```
wsdl.exe /l:VB /n:SWOperMat /out:C:\Inetput\wwwroot\SWOperMat.vb
C:\Inetpub\wwwroot\SWOperMat.wsdl
```

Una vez creada la clase proxy se puede utilizar como cualquier clase en el lenguaje de programación, permitiendo la creación de objetos. Cada llamada a un método realizada con el objeto es interpretada como una salida del URI del servicio web XML (generalmente como un requerimiento SOAP).

El utilitario wsdl genera el siguiente código en Visual Basic .NET

```
'-----
' <autogenerated>
'   This code was generated by a tool.
'
'   Runtime Version: 1.1.4322.573
'
'   Changes to this file may cause incorrect behavior and will be lost if
'   the code is regenerated.
' </autogenerated>
'-----

Option Strict Off

Option Explicit On
```

Imports System

Imports System.ComponentModel

Imports System.Diagnostics

Imports System.Web.Services

Imports System.Web.Services.Protocols

Imports System.Xml.Serialization

,

'wsdl generó automáticamente este código fuente, versión=1.1.4322.573.

,

Namespace SWOperMatVB

'<remarks/>

<System.Diagnostics.DebuggerStepThroughAttribute(), _

System.ComponentModel.DesignerCategoryAttribute("code"), _

System.Web.Services.WebServiceBindingAttribute(Name: = "SWOperMatSoap",
[Namespace]: = "http://tempuri.org/")> _

Public Class SWOperMat

Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

'<remarks/>

Public Sub New()

MyBase.New

Me.Url = "http://localhost/SW2.asmx"

End Sub

'<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/Sumar",
RequestNamespace: = "http://tempuri.org/", ResponseNamespace: = "http://tempuri.org/",
Use: = System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle: = System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _

```
Public Function Sumar(ByVal A As Single, ByVal B As Single) As Single

    Dim results() As Object = Me.Invoke("Sumar", New Object() {A, B})

    Return CType(results(0),Single)

End Function

'<remarks/>

Public Function BeginSumar(ByVal A As Single, ByVal B As Single, ByVal callback As
System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult

    Return Me.BeginInvoke("Sumar", New Object() {A, B}, callback, asyncState)

End Function

'<remarks/>

Public Function EndSumar(ByVal asyncResult As System.IAsyncResult) As Single

    Dim results() As Object = Me.EndInvoke(asyncResult)

    Return CType(results(0),Single)

End Function

'<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/Restar",
RequestNamespace:="http://tempuri.org/", ResponseNamespace:="http://tempuri.org/",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _

Public Function Restar(ByVal A As Single, ByVal B As Single) As Single

    Dim results() As Object = Me.Invoke("Restar", New Object() {A, B})

    Return CType(results(0),Single)

End Function

'<remarks/>

Public Function BeginRestar(ByVal A As Single, ByVal B As Single, ByVal callback As
System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult

    Return Me.BeginInvoke("Restar", New Object() {A, B}, callback, asyncState)
```



```
End Function

'<remarks/>

Public Function EndRestar(ByVal asyncResult As System.IAsyncResult) As Single

    Dim results() As Object = Me.EndInvoke(asyncResult)

    Return CType(results(0),Single)

End Function

'<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/Multiplicar",
RequestNamespace: ="http://tempuri.org/", ResponseNamespace: ="http://tempuri.org/",
Use: =System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle: =System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _

Public Function Multiplicar(ByVal A As Single, ByVal B As Single) As Single

    Dim results() As Object = Me.Invoke("Multiplicar", New Object() {A, B})

    Return CType(results(0),Single)

End Function

'<remarks/>

Public Function BeginMultiplicar(ByVal A As Single, ByVal B As Single, ByVal callback As
System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult

    Return Me.BeginInvoke("Multiplicar", New Object() {A, B}, callback, asyncState)

End Function

'<remarks/>

Public Function EndMultiplicar(ByVal asyncResult As System.IAsyncResult) As Single

    Dim results() As Object = Me.EndInvoke(asyncResult)

    Return CType(results(0),Single)

End Function

<remarks/>

<System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/Dividir",
```

```
RequestNamespace: ="http://tempuri.org/", ResponseNamespace: ="http://tempuri.org/",  
Use: =System.Web.Services.Description.SoapBindingUse.Literal,  
ParameterStyle: =System.Web.Services.Protocols.SoapParameterStyle.Wrapped)> _  
  
Public Function Dividir(ByVal A As Single, ByVal B As Single) As Single  
  
    Dim results() As Object = Me.Invoke("Dividir", New Object() {A, B})  
  
    Return CType(results(0),Single)  
  
End Function  
  
'<remarks/>  
  
Public Function BeginDividir(ByVal A As Single, ByVal B As Single, ByVal callback As  
System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult  
  
    Return Me.BeginInvoke("Dividir", New Object() {A, B}, callback, asyncState)  
  
End Function  
  
'<remarks/>  
  
Public Function EndDividir(ByVal asyncResult As System.IAsyncResult) As Single  
  
    Dim results() As Object = Me.EndInvoke(asyncResult)  
  
    Return CType(results(0),Single)  
  
End Function  
  
End Class  
  
End Namespace
```

Ahora es posible crear un cliente para **consumir el servicio web** en una página .aspx

```
<%@ Import Namespace="SWOperMatVB" %>  
  
<html>  
  
<script language="VB" runat="server">  
  
    Dim Op1 As Single = 0  
  
    Dim Op2 As Single = 0  
  
    Public Sub Submit_Click(Sender As Object, E As EventArgs)  
  
        Try  
  
            Op1 = Single.Parse(Operando1.Text)
```

```
Op2 = Single.Parse(Operando2.Text)

Catch Exp As Exception

    ' Ignored

End Try

Dim Servicio As SWOperMatVB.SWOperMat = New
SWOperMatVB.SWOperMat()

Select (CType(sender,Control).ID)

    Case "Sumar" :

        Resultado.Text = "<b>Resultado</b> = " & Servicio.Sumar(Op1,
Op2).ToString()

    Case "Restar" :

        Resultado.Text = "<b>Resultado</b> = " & Servicio.Restar(Op1,
Op2).ToString()

    Case "Multiplicar" :

        Resultado.Text = "<b>Resultado</b> = " & Servicio.Multiplicar(Op1,
Op2).ToString()

    Case "Divide" :

        Resultado.Text = "<b>Resultado</b> = " & Servicio.Dividir(Op1,
Op2).ToString()

End Select

End Sub

</script>

<body style="font: 10pt verdana">

    <h4>Consumiendo un servicio web de operaciones matemáticas </h4>

    <form runat="server">

        <div style="padding: 15,15,15,15;background-color: beige;width: 300;border-
color: black;border-width: 1;border-style: solid">

            Operando 1: <br><asp:TextBox id="Operando1" Text="15"
runat="server"/><br>

            Operando 2: <br><asp:TextBox id="Operando2" Text="5"
runat="server"/><p>

            <input type="submit" id="Sumar" value="Sumar"
OnServerClick="Submit_Click" runat="server">

            <input type="submit" id="Restar" value="Restar"
OnServerClick="Submit_Click" runat="server">
```

```
<input type="submit" id="Multiplicar" value="Multiplicar"
OnServerClick="Submit_Click" runat="server">

<input type="submit" id="Dividir" value="Dividir"
OnServerClick="Submit_Click" runat="server">

<p>

<asp:Label id="Resultado" runat="server"/>

</div>

</form>

</body>

</html>
```

30 ENCABEZADOS SOAP

Introducción a SOAP

SOAP es la abreviatura, acrónimo o “stand” para Simple Object Access Protocol.

Técnicamente, los servicios web XML no se refieren a ninguna forma particular de mensaje siendo enviado en Internet, pero uno de los tipos más comunes (y por ahora el más estandarizado) de mensajes son los mensajes SOAP.

En el siguiente ejemplo se muestra un archivo SOAP con Ordenes.

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

<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-
envelope">

<env:Header>

<itemOrden:cliente xmlns: itemOrden ="http://localhost /clientes">

< itemOrden:idCliente>CL423A</ itemOrden: idCliente >
```

```
< itemOrden:metodoPago>Cheque</ itemOrden: metodoPago >

< itemOrden:referencia>9993er4</ itemOrden: referencia >

</ itemOrden: cliente >

</env:Header>

<env:Body>

<orden:action type="agregar" xmlns:orden="http:// localhost /ordenes">

<orden:id>CC21</ orden:id >

< orden:precio>123</ orden:precio >

< orden:cantidad>3</ orden:cantidad>

</orden:action>

</env:Body>

</env:Envelope>
```

Secciones de un archivo SOAP

Un mensaje soap posee las secciones **Envelope**, **Body** y el **Header**. El **Header es opcional**. La estructura del mensaje en sí se compone del envelope y el body. La estructura del mensaje la define el diseñador del servicio WEB y será determinada por la aplicación a procesar. El elemento Header contiene información acerca del requerimiento, mientras el body contiene el mensaje en sí.

La especificación de SOAP no obliga a utilizar una estructura en particular para los elementos de Header y Body. Estos elementos permiten controlar el modo como los mensajes son tratados e interpretados por los sistemas que consumen los servicios web.

Utilización del Header

Como ejemplo, se puede configurar el mensaje de modo que el departamento de Mercadeo obtenga la notificación cuando una de las ventas actuales haya producido una venta. Esto se realiza agregando un elemento en el Header:

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

<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">

    <env:Header>

        <mercadeo:activity xmlns:mercadeo="http://localhost/mercadeo"

            env:role="http://www.w3.org/2002/12/soap-envelope/role/next"

            env:mustUnderstand="true">

                <mercadeo:source>Ventas de Hoy </mercadeo:source>

            </marketing:activity>

        </mercadeo:activity>

    </env:Header>

    <itemOrden:cliente xmlns: itemOrden ="http://localhost /clientes">
```

```
< itemOrden:idCliente>CL423A</ itemOrden: idCliente >

< itemOrden:metodoPago>Cheque</ itemOrden: metodoPago >

< itemOrden:referencia>9993er4</ itemOrden: referencia >

</ itemOrden: cliente >

</env:Header>

<env:Body>

<orden:action type="agregar" xmlns:orden="http:// localhost /ordenes">

<orden:id>CC21</ orden:id >

< orden:precio>123</ orden:precio >

< orden:cantidad>3</ orden:cantidad>

</orden:action>

</env:Body>

</env:Envelope>
```

31 SEGURIDAD EN SERVICIOS WEB (WS SECURITY)

Introducción

La **seguridad en servicios Web** involucra modificaciones al **servicio de mensajes SOAP** para ofrecer un mecanismo de protección el cual incluya integridad de los mensajes, confidencialidad en los mensajes y autenticación en los mensajes. Estos mecanismos pueden ser empleados para adaptar SOAP a una variedad de modelos de seguridad y de **encriptación** de datos.

La seguridad en servicios Web también provee de un mecanismo de propósito general para asociar **"tokens" de seguridad** con los mensajes. El manejo de los "tokens" de seguridad es extensible, i.e. cada aplicación puede crear sus propios "tokens" .

Adicionalmente, la seguridad en servicios Web describe como codificar y decodificar "tokens" de seguridad en formato binario y encriptados.

Empleando SOAP como un modelo extensible, las especificaciones de SOAP pueden ser compuestas para lograr una infraestructura de mensajes poderoso.

La seguridad en servicios Web debe ser codificada en conjunto con otros servicios web y protocolos específicos de las aplicaciones de modo de utilizar servicios de encriptación y modelos de seguridad disponibles.

La seguridad en servicios Web es flexible y es diseñada para ser utilizada como base para la construcción de una amplia variedad de modelos de seguridad incluyendo PKI, Kerberos, y SSL. Específicamente, WS-Security ofrece soporte para manejar múltiples "tokens" de seguridad, múltiples dominios "trust", múltiples formatos de firmas y múltiples tecnologías de encriptación.

WS-Security es una especificación técnica y la empleada en este manual contempla **tres mecanismos principales de seguridad**: propagación segura de "tokens", integridad en los mensajes y confidencialidad en los mensajes.

Estos mecanismos pueden ser utilizados de manera independiente o integrados unos con otros.

Ejemplo

A continuación, se muestra un ejemplo de un documento escrito en el protocolo SOAP conteniendo mecanismos de seguridad implementados

```
(
(001) <?xml version="1.0" encoding="utf-8"?>
(002) <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
      xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
(003)   <S:Header>
(004)     <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
(005)       <m:action>http://fabrikam123.com/getQuote</m:action>
(006)       <m:to>http://fabrikam123.com/stocks</m:to>
(007)       <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
(008)     </m:path>
(009)     <wsse:Security
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
```



```
(010)    wsse:UsernameToken Id="MyID">
(011)        <wsse:Username>Zoe</wsse:Username>
(012)    </wsse:UsernameToken>
(013)    <ds:Signature>
(014)        <ds:SignedInfo>
(015)            <ds:CanonicalizationMethod
(016)                Algorithm=
(017)                "http://www.w3.org/2001/10/xml-exc-c14n#" />
(018)            <ds:Reference URI="#MsgBody">
(019)                <ds:DigestMethod
(020)                    Algorithm=
(021)                    "http://www.w3.org/2000/09/xmldsig#sha1" />
(022)                <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
(023)            </ds:Reference>
(024)        </ds:SignedInfo>
(025)        <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
(026)        <ds:KeyInfo>
(027)            <wsse:SecurityTokenReference>
(028)                <wsse:Reference URI="#MyID" />
(029)            </wsse:SecurityTokenReference>
(030)        </ds:KeyInfo>
(031)    </ds:Signature>
(032) </wsse:Security>
(033) </S:Header>
(034) <S:Body Id="MsgBody">
(035)    <tru:StockSymbol xmlns:tru="http://fabrikam123.com/payloads">
(036)        QQQ
(037)    </tru:StockSymbol>
(038) </S:Body>
(039) </S:Envelope>
```

Las primeras dos líneas contienen el "SOAP Envelope".

La línea (003) comienza con los "headers" asociados con el con el mensaje.

Las líneas (004) a la (008) especifican como enrutar el mensaje.

La línea (009) comienza con el tag <Security> , en este tag se coloca la información de seguridad del elemento a recibir. Este tag finaliza en la línea (029).

Las líneas (010) a (012) especifican un token de seguridad asociado con el mensaje. En este caso el nombre del usuario es identificado con el token <UsernameToken>. En este ejemplo, se asume que el servicio conoce el password.

Las líneas (013) a (028) especifican una firma digital. Esta firma garantiza la integridad de los elementos firmado. La firma utiliza la especificación "XML Signature Specification" . En este ejemplo, la firma se basa en una clave generada por el password de los usuarios.

Las líneas (014) a (021) describen la firma digital. La línea (015) especifica como normalizar los datos que están siendo firmados.

Las líneas (017) a (020) seleccionan los elementos que se encuentran firmados.

Específicamente la línea (017) indica que el elemento <S:Body> se encuentra firmado. En este ejemplo únicamente el cuerpo (body) del mensaje es firmado

La línea (022) especifica el valor de la firma de los datos normalizados.

Las líneas (023) a (027) provee una ayuda para indicar donde encontrar el token de seguridad asociado con la firma.

En las líneas (024) y (025) se especifica que el token puede ser extraído de un URL.

Las líneas (031) a (033) contienen el "body" del mensaje SOAP.

La descripción de la especificación de seguridad de los servicios Web se encuentra fuera del alcance de este documento. Para mayor información se debe ubicar la especificación de WSSecurity en la información disponible de Microsoft.

32 EXTENSIONES SOAP

Uno de los aspectos más interesantes a manejar de los servicios Web en la plataforma de desarrollo .NET es la creación de **extensiones SOAP**. Estas extensiones **permiten acceder el "stream" transmitido por la red** antes de que el mismo sea des-serializado en objetos dentro de la plataforma y viceversa.

Las extensiones SOAP permiten a los desarrolladores crear aplicaciones interesantes montadas sobre la arquitectura SOAP. Por ejemplo, se puede implementar un algoritmo para encriptar datos y el cual es utilizado cuando se realiza la llamada a un servicio Web. De manera alternativa, también se puede implementar una rutina de compresión o crear una extensión de SOAP la cual acepta otros documentos SOAP anexados (attached).

Para implementar una extensión SOAP se **requieren** dos cosas:

- Crear una clase que derive de

System.Web.Services.Protocols.SoapExtension .
- Crear una clase que derive de

System.Web.Services.Protocols.SoapExtensionAttribute .

Luego se requiere colocar la implementación de la extensión dentro de los métodos creados para estas clase.

La pieza fundamental de código necesaria para realizar la extensión SOAP es el método **ProcessMessage**. Este método es invocado varias veces, y cada vez que se invoca éste envía el objeto **SoapMessage**, el cual incluye la información del estado "**stage**" del mensaje. A manera de ejemplo a continuación se detalla el flujo de un mensaje SOAP dentro de un servicio Web. Los conceptos aquí ilustrados aplican tanto al cliente como al servidor, pero este ejemplo explica el caso del lado del servidor.

Las cambios provistos por Visual Basic .NET permiten:

- Un mensaje SOAP es recibido por el servidor y este determina cuál método va a emplear para enrutarlo.
- Seguidamente, el servidor revisa si existe alguna extensión SOAP a ser invocada, de ser así, éste se invoca con el estado **BeforeDeserialize** (event stage).
- Finalmente, el servidor deserializa el "stream" e invoca todas las extensiones con el estado **AfterDeserialize** (event stage).

A continuación se coloca el código de la extension, lo que se va a ejecutar cuando se revisan los estados.

```
public Sub WriteInput(message as SoapMessage)

    ' Colocar código para procesar message

    ' propiedad message.Stream (devuelve el mensaje como

    ' un stream)

end sub

public sub SetStream(message as SoapMessage )

    ' colocar codigo

end sub
```

En este código se pueden realizar las pruebas de las extensiones:

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

imports System
imports System.IO
imports System.Text
imports System.Web.Services
imports System.Web.Services.Protocols

public class Prueba{

    <WebMethod>

    ' colocar código

public class ExtensionAtrib inherits SoapExtensionAttribute
```

```
' colocar código  
end class  
  
public class Extension inherits SoapExtension  
  
' colocar código  
  
' realizar override de process message  
' ProcessMessage(SoapMessage message)  
  
' aquí se realice un Select Case de message.Stage  
  
' y se comparan con los case  
  
'case SoapMessageStage.BeforeSerialize:  
  
' case SoapMessageStage.AfterSerialize:  
  
' case SoapMessageStage.BeforeDeserialize:  
  
' case SoapMessageStage.AfterDeserialize:  
  
' case default:  
  
' throw new Exception("estado invalido")  
  
end class
```

33 REFERENCIAS

MSDN : Microsoft Software Development Network Library. On – line Help

<http://msdn.microsoft.com/library>

<http://www.ondotnet.com>

<http://www.vbip.com>

<http://www.dotnetbips.com>

<http://www.c-sharpcorner.com>

<http://www.vbdotnetheaven.com>

<http://www.asp.net>

<http://www.sitepoint.com>

<http://www.pcquest.com>

<http://www.xmlforasp.net>

<http://www.dotnet247.com>

<http://www.codeproject.com>

<http://www.informit.com>

<http://www.expresscomputeronline.com>