# Módulo I

# Introducción a Aplicaciones Web

### 1- Introducción a Aplicaciones Web

#### ¿Qué es ASP.NET?

ASP.NET es un modelo de desarrollo Web (framework) unificado que incluye los servicios necesarios para crear aplicaciones Web empresariales con el código mínimo. ASP.NET forma parte de .NET Framework y al codificar las aplicaciones ASP.NET tiene acceso a las clases en .NET Framework .El código de las aplicaciones puede escribirse en cualquier lenguaje compatible con el Common Language Runtime (CLR), entre ellos Microsoft Visual Basic y C#. Estos lenguajes permiten desarrollar aplicaciones ASP.NET que se benefician del Common Language Runtime, seguridad de tipos, herencia, etc.

#### Tipos de Aplicaciones

En Visual Studio se pueden crear **proyectos de aplicación web** o **proyectos de sitio web**. Cada tipo de proyecto tiene ventajas y desventajas. Por eso, es conveniente conocer las diferencias entre ellos, para poder seleccionar el tipo de proyecto que mejor se ajuste a sus necesidades.

#### Proyecto de Aplicación Web

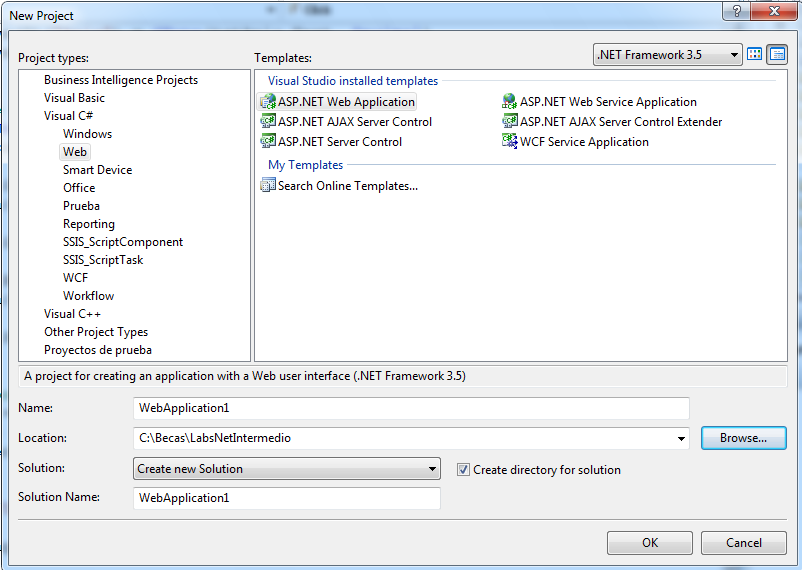
Los proyectos de aplicación web usan los archivos de proyecto de Visual Studio (.**csproj** o .**vbproj**) para realizar un seguimiento de la información sobre el proyecto, como la lista de archivos incluidos en este y cualquier referencia entre proyectos.

El código fuente se compila explícitamente en el equipo que se usa para el desarrollo o el control de código fuente. De forma predeterminada, la compilación de los archivos de código (a excepción de los archivos .aspx y .ascx) genera un ensamblado único.

Para la implementación se puede copiar el ensamblado generado durante la compilación en un servidor. Visual Studio proporciona herramientas que se integran con la herramienta de implementación web de IIS para automatizar muchas de las tareas de implementación.

Es similar a los proyectos que existían en la versión 2003.

Para crearlos seleccione en menú **File** – **New Project** – Opción **Web** – **ASP.NET Web Application**.



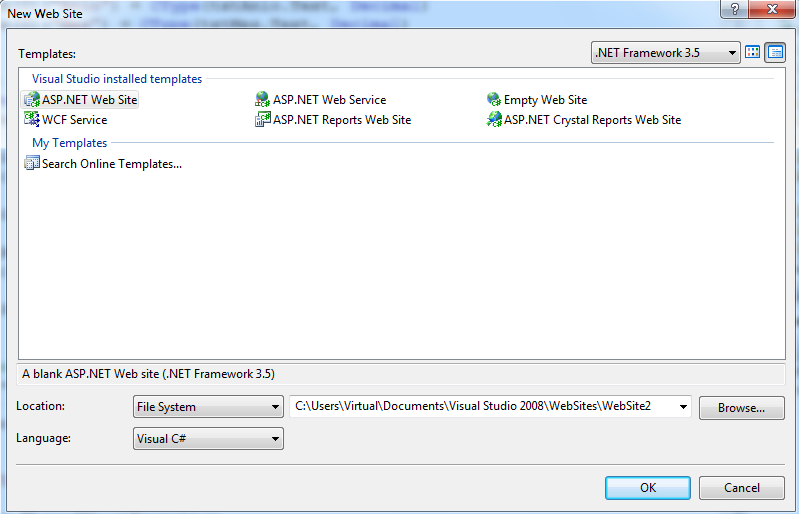
#### Proyectos de Sitio Web

No hay ningún archivo de proyecto (.csproj o .vbproj).Todos los archivos de una estructura de carpetas se incluyen automáticamente en el sitio.

Permite compilar el código fuente de forma dinámica (automáticamente) en el servidor la primera vez que se recibe una solicitud después de haberse instalado o actualizado el sitio, o sea permite instalar los fuentes en el servidor y generar la compilación dinámicamente. También se puede precompilar el sitio y generar los ensamblados. De forma predeterminada, la compilación genera múltiples ensamblados.

Con respecto a la implementación Visual Studio proporciona herramientas para la implementación, pero estas no automatizan tantas tareas de implementación como las herramientas disponibles para los proyectos de aplicación web.

Para crearlos seleccione en menú **File** – **New Web Site – ASP.NET Web Site**



#### Carpetas de Aplicación

ASP.NET reconoce ciertos nombres de carpeta que se pueden utilizar para determinados tipos de contenido:

* App\_Browsers: Contiene definiciones del explorador (archivos .browser) que ASP.NET utiliza para identificar los exploradores individuales y determinar sus funciones.
* App\_Code: Contiene código fuente para clases de utilidad y objetos comerciales (por ejemplo, archivos .cs y .vb) que debe compilar como parte de su aplicación. En una aplicación compilada de forma dinámica, ASP.NET compila el código en la carpeta App\_Code en la solicitud inicial de su aplicación. Los elementos de esta carpeta se vuelven a compilar cuando se detecta cualquier cambio.
* App\_Data: Contiene los archivos de datos de aplicación incluso los archivos MDF, archivos XML, así como otros archivos de almacén de datos.
* App\_GlobalResources: Contiene recursos (archivos .resx y .resources) que se compilan en los ensamblados con ámbito global.
* App\_LocalResources: Contiene recursos (archivos .resx y .resources) que están asociados con una página específica, control de usuario o página principal en una aplicación.
* App\_Themes: Contiene una colección de archivos (archivos .skin y .css, así como archivos de imagen y recursos genéricos) que definen el aspecto de las páginas Web y controles ASP.NET.
* App\_WebReferences: Contiene archivos de contrato de referencia (archivos .wsdl), esquemas (archivos .xsd) y archivos de documentos de descubrimiento (archivos .disco y .discomap) que definen una referencia Web para utilizarla en una aplicación
* Bin: Contiene ensamblados compilados (archivos .dll) para los controles, componentes u otro código al que desea hacer referencia en su aplicación.

#### Archivos

Las aplicaciones de sitios Web pueden contener distintos tipos de archivos, algunos admitidos y administrados por ASP.NET, y otros admitidos y administrados por el servidor IIS.

La mayoría de los tipos de archivo de ASP.NET se puede generar utilizando automáticamente el elemento de menú **Agregar nuevo elemento:**

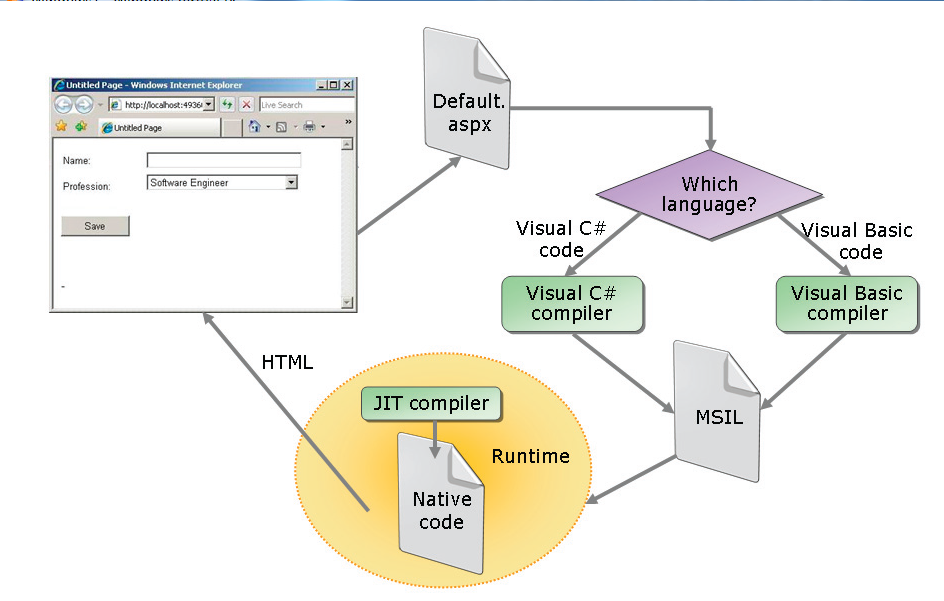
* .asax: Normalmente, un archivo Global.asax. Este archivo representa la aplicación y contiene métodos opcionales que se ejecutan al principio o al final de la duración de la aplicación.
* .ascx: Archivo de control de usuario Web que define un control reutilizable y personalizado.
* .asmx: Archivo de servicios Web XML que contiene las clases y métodos que están disponibles para otras aplicaciones Web mediante SOAP.
* .aspx: Archivo (página) de formularios Web Forms de ASP.NET que puede contener controles y presentación Web, y lógica empresarial.
* .config: Archivo de configuración (normalmente Web.config) que contiene elementos XML que representan la configuración de características de ASP.NET.
* .cs, .vb: Archivo de código fuente de clase que se compila en tiempo de ejecución.
* .master: Página principal que define el diseño de otras páginas Web de la aplicación.
* .resources, .resx: Un archivo de recursos que contiene cadenas de recursos que hacen referencia a imágenes, texto localizable u otros datos.

#### Compilación Dinámica y Ejecución

Para que la aplicación Web atienda las solicitudes, ASP.NET debe analizar y compilar primero el código de la aplicación Web en uno o varios ensamblados. Cuando se compila el código, se traduce en una representación independiente del lenguaje y de la CPU llamado **Lenguaje intermedio de Microsoft (MSIL)**. En tiempo de ejecución, MSIL se ejecuta en el contexto de .NET Framework, que traduce MSIL en instrucciones específicas de la CPU para el procesador en el equipo que ejecuta la aplicación (el CLR convierte el MSIL a código nativo usando lo que se llama compilación **JIT**).

De forma predeterminada, las páginas Web ASP.NET y los archivos de código se compilan de forma dinámica la primera vez que los usuarios solicitan un recurso, como una página de ASP.NET (archivo. aspx), de un sitio Web. Una vez compiladas las páginas y los archivos de código por primera vez, los recursos compilados se almacenan en la caché para que las siguientes veces que solicite la misma página sea lo más eficaz posible.

Cualquiera cambio efectuado en un archivo compilado dinámicamente invalidará de forma automática el ensamblado compilado almacenado en caché del archivo y activará la recompilación de todos los recursos afectados. La próxima vez que se realice una solicitud en el código, ASP.NET reconocerá que el código ha cambiado y volverá a compilar los recursos afectados de la aplicación Web. Este sistema le permite desarrollar rápidamente las aplicaciones con una carga de procesamiento de compilación mínima.



#### Espacios de Nombre (namespaces)

Los espacios de nombres organizan jerárquicamente los objetos definidos en un ensamblado. Un ensamblado puede contener varios espacios de nombres; éstos, a su vez, pueden contener otros espacios de nombres. Los espacios de nombres evitan las ambigüedades y simplifican las referencias cuando se utilizan grupos grandes de objetos, como bibliotecas de clases.

Dentro de un espacio de nombres pueden definirse elementos como los módulos, interfaces, clases, delegados, enumeraciones, estructuras y otros espacios de nombres. Los elementos como propiedades, procedimientos, variables y eventos no se pueden definir en el nivel de espacio de nombres sino que deben declararse dentro de los contenedores como módulos, estructuras o clases.

Los nombres completos son referencias de objetos que llevan como prefijo el nombre del espacio de nombres en el que está definido el objeto. Los nombres completos evitan los conflictos de nombres porque permiten al compilador determinar qué objeto se utiliza. No obstante, los nombres pueden llegar a ser largos y complicados. Para solucionar este problema, puede utilizar la instrucción **using** para definir un alias, es decir, un nombre abreviado que sustituya al nombre completo.

Ejemplo

namespace EspacioEjemplo

{

namespace EspacioEjemplo2

{

public partial class ClaseEjemplo

{

}

}

}

El nombre completo de la clase del ejemplo anterior sería:

EspacioEjemplo.EspacioEjemplo2.ClaseEjemplo

Definiendo:

using EspacioEjemplo.EspacioEjemplo2;

Podemos entonces utilizar a la clase ClaseEjemplo sin la necesidad de escribir el nombre completo.

### 2- Formularios Web

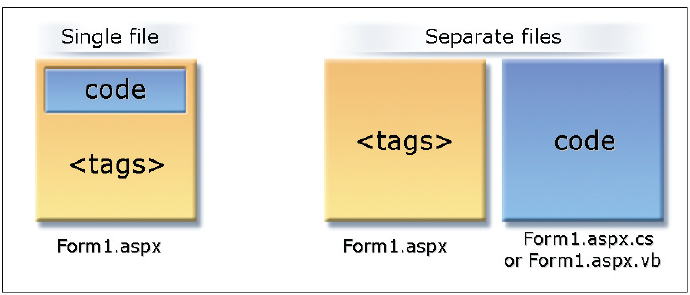
#### Formularios Web

Los formularios Web Forms permiten crear páginas Web. Al crear estas páginas, se pueden usar controles de servidor ASP.NET para crear elementos comunes de la interfaz de usuario y programarlos para que realicen tareas. Estos controles permiten crear con rapidez un formulario Web Forms a partir de componentes integrados reutilizables o personalizados, con un código de página simplificado. Este tipo de páginas acepta cualquier lenguaje compatible con CLR de .NET.

Una página Web ASP.NET se compone de dos partes:

* Elementos visuales, incluidos el formato, los controles de servidor y el texto estático.
* Lógica de programación para la página, que incluye controladores de eventos y otro tipo de código.

ASP.NET proporciona dos modelos para administrar el código y los elementos visuales: el modelo de página de un solo archivo y el modelo de página de código subyacente. Los dos modelos funcionan de la misma manera y se pueden utilizar los mismos controles y el mismo código para ambos modelos. Normalmente y en forma predeterminada se utiliza el modelo de código subyacente.



#### Modelo de página de código subyacente

Este modelo permite mantener el formato en un archivo (el archivo .**aspx**) y el código de programación en otro. El nombre del archivo de código varía según el lenguaje de programación que se esté utilizando. Por ejemplo, si está trabajando con una página llamada SamplePage, el formato se encuentra en el archivo SamplePage.aspx y el código se encuentra en un archivo denominado SamplePage.aspx.vb (en Visual Basic), SamplePage.aspx.cs (en C#), etc.

El archivo de código subyacente contiene las declaraciones de clase completas en el espacio de nombres predeterminado. Sin embargo, la clase se declara con la palabra clave **partial**, que indica que no está incluida totalmente en un archivo. En lugar de ello, cuando se ejecuta la página, el compilador lee la página .aspx y el archivo al que hace referencia en la directiva **@ Page**, los ensambla en una sola clase y, a continuación, los compila como una unidad en una sola clase.

Las páginas de código subyacente ofrecen una serie de ventajas que las hacen adecuadas para las aplicaciones Web con código importante o en las que varios desarrolladores están creando un sitio Web:

* Las páginas de código subyacente ofrecen una separación nítida entre el formato (la interfaz de usuario) y el código. Resulta práctico que el diseñador trabaje en el formato mientras el programador escribe el código.
* El código no se expone a los diseñadores de las páginas o a las demás personas que sólo estén trabajando con el formato de la página.
* El código se puede reutilizar para varias páginas.

#### Directiva @ Page

Define los atributos específicos de página (archivo .aspx) que utilizan el analizador y el compilador de páginas de ASP.NET.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" %>

Si desea definir varios atributos para la directiva @ Page, separe cada par de atributo y valor con un solo espacio. Para un atributo específico, no incluya un espacio a ambos lados del signo de igualdad (=) que conecta el atributo con su valor.

Permite configurar varios atributos, algunos de ellos son:

* **Language**: Especifica el lenguaje utilizado cuando se compilan todos los bloques de representación en línea (<% %> y <%= %>) y de declaraciones de código en la página. Los valores pueden representar cualquier lenguaje compatible con .NET Framework, incluidos Visual Basic, C# o JScript. Se puede utilizar y especificar sólo un lenguaje por página.
* **AutoEventWireup**: Indica si los eventos de la página están autoconectados. Su valor es true si la autoconexión de eventos está habilitada; en caso contrario, su valor es false. El valor predeterminado es true.
* **CodeFile**: Especifica una ruta de acceso al archivo de código subyacente al que se hace referencia para la página. Este atributo se utiliza junto con el atributo **Inherits** para asociar un archivo de código fuente subyacente a una página Web. Este atributo sólo es válido para las páginas compiladas.
* **MaintainScrollPositionOnPostback:** Obtiene o establece un valor que indica si el usuario regresará a la misma posición del explorador cliente después de la devolución de datos.

Se puede incluir sólo una directiva @ Page por cada archivo .aspx. Dado que se proporcionan los valores predeterminados más comunes para la mayoría de los atributos, ya sea en código fuente o en archivos de configuración, a menudo resulta necesario agregar un amplio conjunto de atributos a la directiva. Generalmente, se debe agregar el conjunto mínimo de atributos requeridos para especificar las características deseadas de una página. Si hay un atributo común que desee aplicar a todas las páginas, por ejemplo, si desea habilitar el seguimiento en todas las páginas, considere la posibilidad de habilitar la característica en el archivo **Web.config** en lugar de agregar el atributo a cada página.

#### Controles de Servidor

Los controles de servidor web ASP.NET son objetos de páginas web ASP.NET que se ejecutan cuando se solicita la página y que representan marcado en un explorador. Muchos controles de servidor web son similares a elementos HTML conocidos, como botones y cuadros de texto. Otros controles abarcan comportamiento complejo, por ejemplo controles de calendario y controles que administran las conexiones de datos.

Los controles de servidor Web ofrecen todas las funciones de los controles HTML y estas funciones adicionales:

* Un modelo de objetos enriquecido que proporciona capacidades de programación de tipo seguro.
* Detección automática del explorador. Los controles pueden detectar las funciones del explorador y representar el marcado adecuado.
* Para algunos controles, la capacidad para definir su propio diseño para el control utilizando Templates.
* Para algunos controles, la capacidad de especificar si un evento del control provoca un envío inmediato al servidor o, en su lugar, se almacena en caché y se activa cuando se envía la página.
* Compatibilidad para temas, lo que le permite definir un aspecto uniforme para los controles en todo el sitio.
* Capacidad para pasar eventos de un control anidado (como un botón en una tabla) al control contenedor.

Los controles utilizan una sintaxis como la que se muestra a continuación:

<asp:Button ID="Button1" runat="server" Text="Button" />

En este caso los atributos no son los de los elementos HTML. En lugar de ello, son propiedades del control Web.

Cuando se ejecuta la página Web ASP.NET, el control de servidor Web se representa en la página utilizando el marcado apropiado, que con frecuencia no sólo depende del tipo de explorador sino también de la configuración que haya realizado para el control. Por ejemplo, un control **TextBox** podría representarse como una etiqueta **input** o una etiqueta **textarea** dependiendo de sus propiedades.

La cláusula **runat=”server”** indica que el control es un control de servidor. Cualquier elemento HTML de una página se puede convertir en control de servidor HTML agregando el atributo runat="server". Durante el análisis, el marco de trabajo de la página ASP.NET creará instancias de todos los elementos que contienen el atributo runat="server". Si desea hacer referencia al control como un miembro dentro del código, también deberá asignarle un atributo **id** al control.

<input id="Text1" type="text" runat="server" />

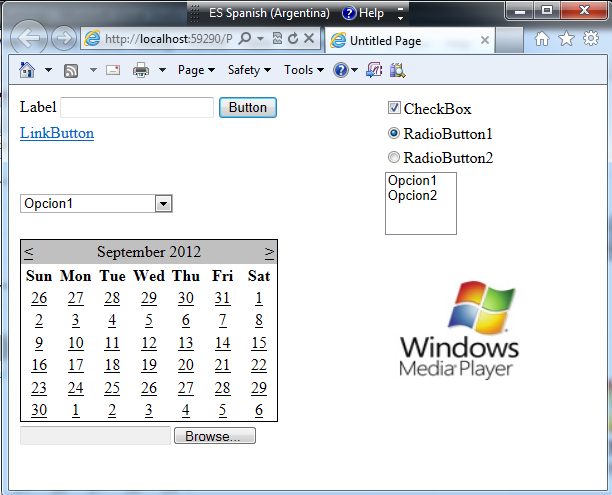
#### Tipos de Controles de Servidor

* Controles Intrínsecos: Se corresponden con elementos HTML simples
* Controles de Validación: Controles ocultos que permiten validar las entradas de los usuarios teniendo en cuenta patrones predeterminados
* Controles Ricos: Proveen una funcionalidad importante insertando funciones complejas
* Controles con Enlace a Datos: Permiten mostrar datos de fuentes de datos

#### Controles Estándar de Servidor Web

A continuación se detallan algunos controles de servidor:

* **Label**: Proporciona una forma de mostrar texto en una página Web ASP.NET.
* **Botón**: Los botones de una página Web ASP.NET permiten a los usuarios enviar comandos. Los botones envían la página al servidor y hacen que se procese junto con los eventos pendientes. Los controles de servidor Web incluyen tres tipos de botones: un botón de comando estándar (control **Button**), un botón de estilo hipervínculo (control **LinkButton**) y un botón gráfico (control **ImageButton**). Los tres realizan funciones parecidas, pero cada uno de ellos ofrece un aspecto diferente.
* **TextBox**: Proporciona a los usuarios un medio para escribir datos en una página de formulario Web Forms, incluidos texto, números y fechas.
* **DropDownList**: Permite a los usuarios realizar una selección en un cuadro de lista desplegable de selección simple. El control DropDownList es parecido al control de servidor Web ListBox. Difiere en que sólo muestra el elemento seleccionado en un cuadro, junto con un botón desplegable. Cuando los usuarios hacen clic en el botón, se muestra una lista de elementos.
* **ListBox**: Representa un control de cuadro de lista que permite la selección de uno o varios elementos.
* **RadioButton y RadioButtonList**: Proporcionan a los usuarios un medio para seleccionar entre opciones mutuamente excluyentes.
* **CheckBox y CheckBoxList**: Ofrecen a los usuarios la posibilidad de cambiar entre opciones excluyentes de tipo verdadero/falso (true/false), sí/no (yes/no) y activado/desactivado (on/off).
* **HyperLink**: Proporciona un medio para crear y manipular vínculos en una página Web desde el código del servidor.
* **Image**: Permite mostrar imágenes en una página Web ASP.NET y administrar las imágenes en su propio código.
* **ImageMap**: Este control permite crear una imagen con regiones individuales en las que los usuarios pueden hacer clic, denominadas zonas activas. Cada una de estas zonas activas puede ser un hipervínculo independiente o un evento de devolución de datos.
* **Calendar**: Muestra un calendario tradicional de un mes en una página ASP.NET. Los usuarios pueden ver y seleccionar fechas en el calendario.
* **FileUpload**: Proporciona a los usuarios una forma de enviar un archivo de su equipo al servidor.
* **PlaceHolder**: Permite colocar un control contenedor vacío en la página y, a continuación, agregarle dinámicamente elementos secundarios en tiempo de ejecución.
* **AdRotator**: Proporciona una manera práctica de mostrar los anuncios en las páginas Web ASP.NET. El control muestra la imagen gráfica que se ha proporcionado (un archivo .gif o una imagen similar). Cuando los usuarios hacen clic en el anuncio, se les redirige a la dirección URL de destino que se ha especificado.



#### ViewState o Estado de Vista

El estado de vista es el método que utiliza el marco de las páginas ASP.NET para conservar los valores de página y control entre viajes de ida y vuelta. Cuando se representa el marcado HTML de la página, el estado actual de la página y los valores que se deben conservar durante la devolución de datos se serializan en cadenas codificadas en base64. Utiliza **campos ocultos**.

Una aplicación Web **no tiene estado**. Cada vez que el servidor solicita una página se crea una nueva instancia de la clase de página web. Normalmente esto supondría que toda la información de la página y de sus controles se perdería en cada viaje de ida y vuelta. Por ejemplo, si un usuario inserta información en un cuadro de texto de una página web HTML, dicha información se envía al servidor de forma predeterminada. Sin embargo, no se devuelve al explorador en la respuesta.

Para superar esta limitación intrínseca de la programación web, se incluyen varias características de administración de estados para conservar los valores de página y control entre viajes de ida y vuelta al servidor web. Una de estas características es el estado de vista o ViewState.

<input type="hidden" name="\_\_VIEWSTATE" id="\_\_VIEWSTATE"

value="/wEPDwUKMTQ2OTkzNDMyMWRkqkemRVXuWimVdVDG7O8z3nQVOfE=" />

#### Consideraciones en el uso de ViewState

La información del estado de vista se serializa en XML y luego se codifican en base64, lo cual puede generar grandes cantidades de datos. Cuando se expone la página en el servidor, el contenido del estado de vista se envía como parte de la información de devolución de datos de la página. Si el estado de vista contiene una gran cantidad de información, puede afectar al rendimiento de la página. Debe probar el rendimiento de las páginas utilizando datos típicos para la aplicación y así determinar si el tamaño del estado de vista provoca problemas de rendimiento. Si no tiene que almacenar información de control de algún control individual, puede deshabilitar el estado de vista para un control.

Otra consideración que se debe tener en cuenta es que si la cantidad de datos de un campo oculto llega a ser muy grande, algunos servidores proxy y algunos firewalls impedirán el acceso a la página que los contiene. Debido a que la cantidad máxima permitida puede variar con distintas implementaciones de firewall y proxy, los campos ocultos de gran tamaño pueden causar problemas intermitentes (recuerde que el ViewState se implementa usando un campo oculto)

#### ControlState o Estado de Control

Además del estado de vista, ASP.NET es compatible con el estado de control. La página utiliza el estado de control para mantener información del control que debe conservarse entre devoluciones de datos, independientemente de que el estado de vista esté deshabilitado para la página o para un control. Al igual que el estado de vista, el estado de control se almacena en uno o más **campos ocultos**.

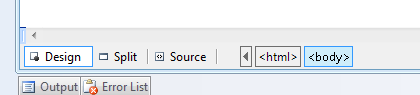
#### Creación de Formularios Web

Al crear una aplicación o sitio web se crea automáticamente un formulario llamado Default.aspx. Si quiere agregar nuevos formularios puede hacerlo presionando el botón derecho del mouse sobre el nombre del proyecto en el Explorador de Soluciones, **Add New Item**. Luego seleccione **Web Form**.

En la parte inferior del Diseñador de Web Forms existen tres fichas, **Design, Split, Source** que muestran diferentes vistas del archivo .aspx con el que está trabajando:

* La vista **Design** le proporciona una vista WYSIWYG donde puede arrastrar controles y utilizar la ventana Propiedades para configurarlos.
* La vista **Source** muestra el código XHTML o HTML.
* **Split** permite ver ambas al mismo tiempo

Puede trabajar en cualquiera de las dos vistas. Cuando cambia entre las dos vistas, cada una de ellas se actualiza con los cambios efectuados en la otra.



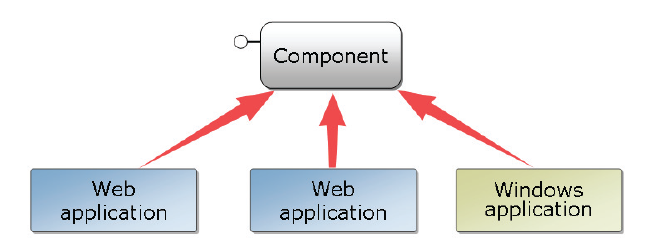
Puede agregar controles arrastrándolos desde la **Toolbox**.

Los controles de servidor de las páginas de formularios Web Forms pueden producir diversos eventos, muchos de los cuáles se inician por acciones que realiza el usuario en el explorador. Por ejemplo, un control Button de servidor Web puede provocar un evento Clic cuando un usuario haga clic en un botón de la página.

El código para gestionar el evento provocado se ejecuta en el servidor. Cuando el usuario hace clic en un botón, la página se envía al servidor. El marco de trabajo de páginas ASP.NET analiza la información del evento y si existe un controlador de eventos que se corresponda con el evento, se llama al código inmediatamente. Cuando el código finalice, la página se vuelve a enviar al explorador con cualquier cambio producido por el código del controlador de eventos.

#### Trabajar con Componentes

Cuando se trabaja con lenguajes orientados a objetos como c# o Vb.Net para desarrollar una aplicación se pueden usar clases y componentes existente. Los componentes generan sus propios archivos lo cual generan que puedan ser reusados por varias aplicaciones. Se pueden crear aplicaciones de tipo **Class Library** (Biblioteca de Clases) para crear este tipo de componente.



Las clases creadas en estos componentes están disponibles para todas las páginas. Primero se deberá crear una referencia al componente (dll). Luego puede acceder a las clases creando instancias de las mismas y trabajar con sus métodos, propiedades y eventos.

### 3- Master Pages

#### Master Pages o Páginas Principales

Las páginas principales de ASP.NET permiten crear un diseño de página (una página principal) que se puede utilizar en ciertas páginas o en todas (páginas de contenido) en un sitio Web. Las páginas principales pueden simplificar en gran medida la tarea de crear un aspecto coherente para el sitio.

Puede definir el aspecto, el diseño y el comportamiento estándar que desea que tengan todas las páginas (o un grupo de páginas) de la aplicación en una sola página principal. A continuación, puede crear páginas de contenido individuales que incluyan el contenido que desea mostrar. Cuando los usuarios solicitan las páginas de contenido, éstas se combinan con la página principal para dar como resultado una página con el diseño de la página principal y el contenido de la página de contenido.

Una página principal es un archivo de ASP.NET con la extensión **.master** (por ejemplo, MySite.master) que tiene un diseño predefinido que puede incluir texto estático, elementos HTML y controles de servidor. La página principal se identifica mediante una directiva **@ Master** especial que reemplaza la directiva @ Page utilizada en las páginas .aspx ordinarias. El aspecto de la directiva es el siguiente:

<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>

Además del texto estático y los controles que aparecerán en todas las páginas, la página principal también incluye uno o varios controles **ContentPlaceHolder**. Estos controles PlaceHolder definen las regiones que incluirán contenido reemplazable. A su vez, el contenido reemplazable se define en las páginas de contenido.

#### Páginas de contenido

Para definir el contenido de los controles PlaceHolder de la página principal, cree páginas de contenidos individuales, que son páginas ASP.NET (archivos .aspx y, opcionalmente, archivos de código subyacente) que están enlazadas a una página principal concreta. El enlace se establece en la directiva **@Page** de la página de contenido al incluir un atributo **MasterPageFile** que apunta a la página principal que se va a utilizar. Por ejemplo, una página de contenido podría tener la siguiente directiva @Page, que la enlaza con la página Master1.master.

<%@ Page Language="C#" MasterPageFile="~/MasterPages/Master1.master" Title="Content Page"%>

En la página de contenido, cree el contenido agregando los controles y asignándolos a los controles **ContentPlaceHolder** de la página principal. En una página de contenido, todo lo que no esté dentro de los controles ContentPlaceHolder (excepto los bloques de secuencias de comandos del código del servidor) producirá un error. En una página de contenido puede realizar las mismas tareas que en una página de ASP.NET.

#### Ventajas de las páginas principales

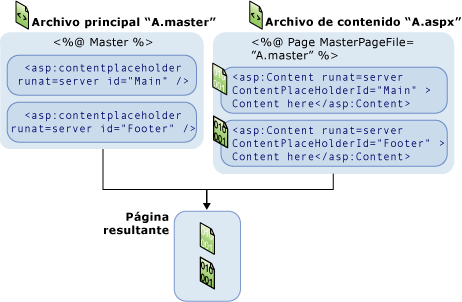
Las páginas principales proporcionan una funcionalidad que tradicionalmente los programadores creaban copiando el código, el texto y los elementos de control existentes repetidamente, mediante conjuntos de marcos (frames), archivos de inclusión de elementos comunes (#include), controles de usuario de ASP.NET, etc. Entre las ventajas de las páginas principales se incluyen las siguientes:

* Permiten centralizar las funciones comunes de las páginas para que las actualizaciones puedan llevarse a cabo en un solo lugar.
* Facilitan la creación de un conjunto de controles y código, y aplican los resultados en un conjunto de páginas. Por ejemplo, puede utilizar los controles en la página principal para crear un menú que se aplique a todas las páginas.
* Proporcionan un control más preciso sobre el diseño de la página final al permitir controlar el modo en que se representan los controles PlaceHolder.
* Proporcionan un modelo de objetos que permite personalizar la página principal a partir de páginas de contenido individuales.

#### Comportamiento en tiempo de ejecución de las páginas principales

En tiempo de ejecución, las páginas principales se controlan en la secuencia siguiente:

* Los usuarios solicitan una página escribiendo la dirección URL de la página de contenido.
* Cuando se obtiene la página, se lee la directiva @ Page. Si la directiva hace referencia a una página principal, también se lee la página principal. Si las páginas se solicitan por primera vez, se compilan las dos páginas.
* La página principal con el contenido actualizado se combina en el árbol de control de la página de contenido.
* El contenido de los controles individuales se combina en el control ContentPlaceHolder correspondiente de la página principal.
* La página combinada resultante se representa en el explorador.



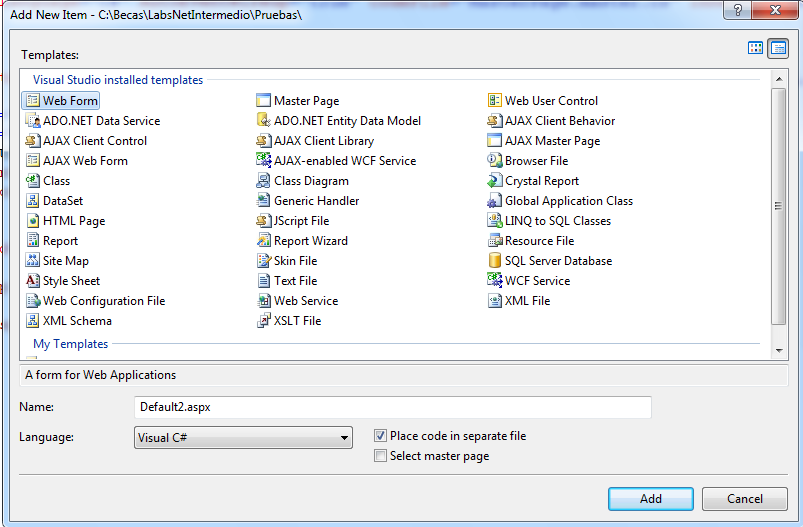
Desde la perspectiva del usuario, la combinación de las páginas principales y de contenido da como resultado una única página. La dirección URL de esta página es la de la página de contenido. Desde la perspectiva del programador, las dos páginas actúan como contenedores diferentes para sus respectivos controles. La página de contenido actúa como un contenedor de la página principal.

#### Creación de páginas principales

Presionando el botón derecho del mouse sobre el nombre del proyecto en el Explorador de Soluciones, **Add New Item**. Luego seleccione **Master Pages**.

Trabajar la página principal de la misma forma que una página web, por fuera de los controles ContentPlaceHolder.

Para agregar una página de contenido asociada a una página principal marque la opción **Select Master Pages.** Esta opción le permitirá seleccionar la página principal a la cual se la quiere asociar.



# Módulo II

# Trabajar con Formularios Web

### 1- Implementar Código Subyacente

#### Implementar Código

Código subyacente es un término empleado para describir el código unido con objetos definidos por marcado cuando una página HTML se compila por marcado. Contienen la lógica de la aplicación para esa página.

Se puede implementar de varias maneras:

* Código en Línea: Es el código está en una sección SCRIPT en el mismo archivo que el contenido HTML
* Páginas de Código Subyacente: El código esta en otro archivo. Es la opción predeterminada para ASP.NET

#### Código en Línea

<head runat="server">

<title>Untitled Page</title>

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

void Page\_Load(object sender, System.EventArgs e)

{

// Código

}

void btnLogon\_Click(object sender, System.EventArgs e)

{

// Código

}

</script>

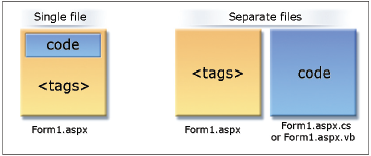
</head>

<body>

…………

</body>

#### Páginas de Código Subyacente



Cuando se solicita una página ASP.NET y se representa el marcado en un explorador, no sólo se ejecuta el código que se creó para la página, sino que ASP.NET genera y compila, en tiempo de ejecución, una o más clases que realizan efectivamente las tareas necesarias para ejecutar la página.

### 2- Trabajar con Eventos

#### Eventos

Se pueden usar procedimientos de evento para atajar la interacción del usuario en formularios Web, de manera de reaccionar a las acciones del usuario.

Existen 2 tipos de eventos:

* Del lado del **Cliente**: Se ataja en la máquina del navegador. No envían información al servidor. Son muy útiles para generar acciones que no requieren recursos del servidor, evitando así la ida y vuelta al servidor. Se usa el elemento **Script** para generar el código.
* Del lado del **Servidor**: Requieren del envío de información al servidor. Son muchos más poderoso que los del lado del cliente.

#### Eventos del lado de Servidor

Una característica importante de ASP.NET es que permite programar páginas Web utilizando un modelo basado en eventos similar al de las aplicaciones de cliente.

Los eventos producidos por los controles de servidor ASP.NET funcionan de manera diferente a los eventos de las páginas HTML tradicionales o de las aplicaciones Web basadas en el cliente. La diferencia se basa principalmente en la separación existente entre el propio evento y el lugar donde se controla el evento. En las aplicaciones basadas en cliente, los eventos se producen y controlan en el cliente. Sin embargo, en las páginas Web ASP.NET, los eventos asociados a los controles de servidor se originan en el cliente (navegador) pero los controla la página ASP.NET en el servidor Web.

ASP.NET controla la tarea de capturar, transmitir e interpretar el evento. Al crear controladores de eventos en una página Web ASP.NET, no es necesario saber capturar la información del evento y hacer que esté disponible para el código. En cambio, se pueden crear controladores de eventos casi de la misma forma que en un formulario de cliente tradicional.

Debido a que los eventos de controles de servidor ASP.NET requieren un viaje de ida y vuelta al servidor para procesarse, pueden afectar al rendimiento de una página. Por lo tanto, los controles de servidor ofrecen un conjunto limitado de eventos, normalmente sólo de tipo clic (**eventos de acción**). Algunos controles de servidor admiten **los eventos de cambio**. Por ejemplo, el control CheckBox de servidor Web produce un evento de cambio **CheckedChanged** cuando el usuario hace clic en el cuadro. Algunos controles de servidor admiten eventos más abstractos. Por ejemplo, el control de servidor Web Calendar provoca un evento **SelectionChanged** que es una versión más abstracta del evento de clic.

Los eventos que tienen lugar con frecuencia (y que pueden provocarse sin que el usuario lo sepa), como **onmouseover**, no se pueden usar en los controles de servidor. Los controles de servidor ASP.NET siguen pudiendo llamar a los controladores de cliente para esos eventos.

Los controles y la propia página también provocan eventos de ciclo de vida en cada paso del procesamiento, como **Init**, **Load** y **PreRender**. Puede aprovecharse de estos eventos de ciclo de vida en la aplicación. Por ejemplo, en el evento Load de una página, puede establecer valores predeterminados para los controles.

Ejemplo:

protected void Page\_Load(object sender, EventArgs e)

{

}

protected void Button1\_Click(object sender, EventArgs e)

{

}

#### Eventos de Cambios y Eventos de Acción

En los controles de servidor, algunos eventos, generalmente los de clic (**acción**), hacen que la página se envíe de vuelta inmediatamente al servidor. Los eventos de cambio en los controles de servidor HTML y de servidor Web, como el control TextBox, no ocasionan que se produzca una acción de envío inmediatamente. En su lugar, se ejecutan la próxima vez que tenga lugar una acción de envío.

Si resulta útil para una aplicación, se puede especificar que los eventos de cambio provoquen el envío de la página. Los controles de servidor Web que admiten un evento de cambio incluyen la propiedad **AutoPostBack**. Cuando esta propiedad está establecida como **true**, el evento de cambio del control provoca el envío inmediato de la página, sin esperar a que se produzca un evento de clic. Por ejemplo, de forma predeterminada, el evento CheckedChanged de un control CheckBox no provoca el envío de la página. Sin embargo, si se establece la propiedad AutoPostBack del control en true, en cuanto un usuario active la casilla de verificación, la página se envía al servidor para ser procesada.

<asp:CheckBox ID="CheckBox1" runat="server" Text="Activo" AutoPostBack="True" oncheckedchanged="CheckBox1\_CheckedChanged" />

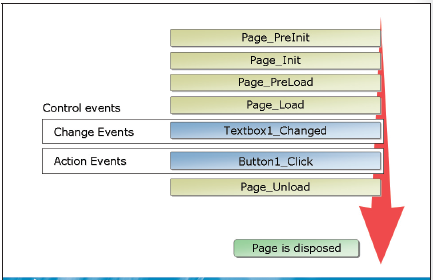
#### Ciclo de Vida de una Página Web

Cuando se ejecuta una página ASP.NET, ésta recorre un ciclo de vida en el que realiza una serie de pasos de procesamiento. Entre ellos se incluyen la inicialización, la creación de instancias de controles, la restauración y el mantenimiento del estado, la ejecución del código del controlador de eventos y la representación.

Dentro de cada fase del ciclo de vida de una página, ésta produce eventos que puede controlar para ejecutar su propio código.

En la tabla siguiente se muestran los eventos del ciclo de vida de la página que se utilizan con más frecuencia. Hay más eventos que los mostrados. Sin embargo, no se utilizan en la mayoría de los escenarios de procesamiento de páginas.

|  |  |
| --- | --- |
| **Evento** | **Uso Típico** |
| Page\_PreInit | Crear controles dinámicos, establecer temas o páginas principales, etc.  **Nota**: Si establece una propiedad de un control en esta fase puede que la misma sea sobre escrita en otra fase |
| **Page\_Init** | Leer o inicializar las propiedades de los controles |
| **Page\_Load** | Leer o inicializar las propiedades de los controles |
| Eventos de los Controles | Controla los eventos específicos de los controles, por ejemplo el Clic de un Button |
| **Page\_PreRender** | Realiza los cambios finales en el contenido de la página |
| **Page\_UnLoad** | Lleva a cabo el trabajo de limpieza final |



#### Page.IsPostBack

Algunos eventos, como por ejemplo Page\_Load, ejecutan en cada requerimiento de la página, tanto en el primer requerimiento como en la devolución de datos (postbacks).

Con lo cual es importante reconocer estas dos situaciones de manera que el código se ejecute cuando corresponde. Para esto se utiliza la propiedad Page.IsPostBack.

Esta propiedad obtiene un valor que indica si la página se está mostrando por primera vez o si se está cargando como respuesta a un postback. Es **true** si la página se carga como respuesta a un postback del cliente; en caso contrario, es **false**.

Ejemplo

protected void Page\_Load(object sender, EventArgs e)

{

if (Page.IsPostBack)

{

// es un postback

}

Else

{

// es la primera vez

}

}

# Módulo III

# Validación

### 1- Introducción a la Validación de Datos

#### ¿Por qué Validar?

Cuando se crean controles que permiten el ingreso de datos, como por ejemplo un TextBox, se debe controlar la información que el usuario ingresa. Si los datos ingresados son incorrectos no solo pueden demorar el ingreso de la información sino “quebrar” la aplicación.

Para controlar que los datos ingresados son correctos se deben verificar teniendo en cuenta valores, rangos o formatos esperados por la aplicación, etc.

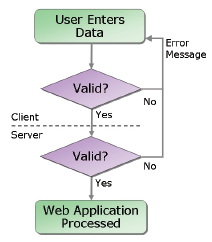
#### Validación del lado del Cliente o Servidor

La validación de entrada de datos se puede hacer tanto del lado del cliente como del lado servidor o ambas. La validación del lado del servidor es muy importante y debería programarse siempre. La validación del lado del cliente es optativa y solo puede hacerse en algunos navegadores.

La validación del lado del cliente verifica la entrada de datos en el cliente antes de que esta información sea enviada al servidor. Esto evita la ida y vuelta a servidor con datos incorrectos.

Si el usuario trabaja con un explorador que admite HTML dinámico (DHTML), los controles de validación ASP.NET podrán ejecutar la validación mediante una secuencia de comandos de cliente. Debido a que los controles pueden proporcionar una respuesta inmediata sin una acción de ida y vuelta al servidor, se mejora el trabajo del usuario con la página.

En la mayoría de los casos, no tiene que realizar cambios en la página ni en los controles de validación para utilizar la validación en el cliente. Los controles detectan automáticamente si el explorador admite DHTML y ejecutan su comprobación consecuentemente. La validación en el cliente utiliza el mismo mecanismo para mostrar el error que la validación en el servidor.



### 2- Controles de Validación

#### Controles de Validación

Mediante los controles de validación se puede agregar validación de entrada a las páginas Web ASP.NET. Los controles de validación proporcionan un mecanismo fácil de utilizar para todos los tipos comunes de validación estándar (por ejemplo, probar fechas válidas o valores comprendidos en un intervalo), además de otras formas para proporcionar validación escrita personalizada. Además, los controles de validación permiten personalizar completamente cómo se muestra la información de errores al usuario.

Los controles de validación se pueden utilizar con cualquier control que se coloque en una página Web ASP.NET, incluidos los controles HTML y de servidor Web.

Cada control de validación normalmente realiza una comprobación. No obstante, es posible que se deseen comprobar múltiples condiciones. Por ejemplo, supongamos que desea especificar que una entrada de usuario es necesaria y que sólo puede contener fechas dentro de un intervalo especificado.

En las páginas se puede asociar más de un control de validación a cada control de entrada. En ese caso, las comprobaciones realizadas por los controles se resuelven utilizando un operador lógico AND, lo que significa que los datos introducidos por el usuario deben pasar todas las comprobaciones para que se consideren válidos.

Normalmente, los controles de validación no son visibles en la página presentada. No obstante, si el control detecta un error, muestra el texto del mensaje de error que se especifique. El mensaje de error se puede mostrar de formas diferentes. Este tema se desarrolla más adelante, en este mismo módulo.

Los controles de validación de .Net son:

* RequiredFieldValidator
* CompareValidator
* RangeValidator
* RegularExpressionValidator
* CustomValidator

#### Propiedades Comunes de los Controles de Validación

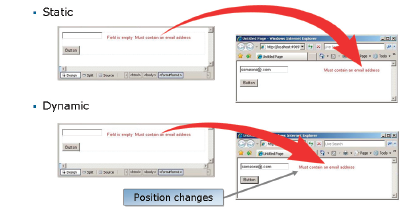
* **ControlToValidate**: Obtiene o establece el control de entrada de datos a validar.
* **ErrorMessage**: Obtiene o establece el texto del mensaje de error que aparece en un control de **ValidationSummary** cuando se produce un error en la validación.
* **Display**: Obtiene o establece el comportamiento de la presentación de mensajes de error en un control de validación.
* **EnableClientScript**: Obtiene o establece un valor que indica si la validación en el cliente está habilitada.
* **Text**: Obtiene o establece el texto mostrado en el control de validación cuando se produce un error en la validación.

#### Posicionamiento de los Mensajes de Error

Utilice la propiedad de **Display** para especificar el comportamiento de la presentación de mensajes de error en el control de validación. Es común combinar varios controles para validar una sola entrada, como los mismo podrían ser excluyentes, esta propiedad permite que los errores se muestren de forma prolija.

Tiene tres valores disponibles:

* **None**: El mensaje de validación nunca se realiza en línea.
* **Static**: El espacio para el mensaje de validación se asigna en el diseño de página. Predeterminado.
* **Dynamic**: El espacio para el mensaje de validación se agregan dinámicamente a la página si se produce un error en la validación.



#### Propiedad ValidationGroup

Estos controles tienen una propiedad llamada **ValidationGroup** que, cuando se establece, valida sólo los controles de validación dentro del grupo especificado cuando el control desencadena un envío al servidor. Utilice la propiedad de ValidationGroup para asignar un control de validación a un grupo de validación.

#### Control RequiredFieldValidator

Convierte el control de entrada asociado en un campo obligatorio. El control de entrada no supera la validación si su valor es igual a la propiedad **InitialValue** una vez que ha perdido el foco.

Ejemplo: Valida que el TextBox llamado TextBox1 tenga al menos algún carácter.

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<asp:RequiredFieldValidator ID="RequiredFieldValidator1"

runat="server" ControlToValidate="TextBox1"

ErrorMessage="El dato es obligatorio">\*

</asp:RequiredFieldValidator>

**Nota**: Si el control de entrada está vacío, no se llama a ninguna función de validación y la validación se realiza correctamente. Utilice un control **RequiredFieldValidator** para solicitar al usuario que especifique los datos en el control de entrada.

#### Control 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.

El control CompareValidator supera la validación si el valor del control de entrada coincide con los criterios especificados por las propiedades **ValueToCompare,** **ControlToCompare** u **Operator**.

También puede utilizar el control CompareValidator para indicar si el valor definido en un control de entrada se puede convertir al tipo de datos especificado por la propiedad **Type**.

Si desea comparar un control de entrada específico con otro control de entrada, establezca la propiedad **ControlToCompare** para especificar el control con el que desea comparar. Puede comparar también, el valor de un control de entrada con un valor constante. Especifique el valor constante con el que desee comparar estableciendo la propiedad **ValueToCompare**.

Utilice la propiedad **Operator** para especificar el tipo de comparación que se desea realizar, como, por ejemplo, mayor que, igual a, etc. Si establece la propiedad Operator en **DataTypeCheck**, el control CompareValidator omite las propiedades ControlToCompare y ValueToCompare y se limita a indicar si el valor definido en el control de entrada se puede convertir al tipo de datos especificado por la propiedad **Type**.

La propiedad **Type** se utiliza para especificar el tipo de datos de ambos valores de comparación. Ambos valores se convierten automáticamente a este tipo de datos antes de realizarse la comparación. Acepta los siguientes tipos:

* String: Tipo de datos de cadena.
* Integer: Tipo de datos entero de 32 bits con signo.
* Double: Un tipo de datos número de punto flotante de precisión doble.
* Date: Tipo de datos de fecha.
* Currency: Tipo de datos decimal que puede contener símbolos de moneda.

Ejemplos: En el primer ejemplo se utiliza un control CompareValidator para verificar el tipo de dato del dato ingresado, el cual debe ser entero. En el segundo ejemplo se utiliza otro control CompareValidator para verificar que el dato sea numérico y mayor a cero. El valor cero es tomado como una constante. En el tercer ejemplo se verifica que el valor de 2 TextBox sean iguales.

<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>

<asp:CompareValidator ID="CompareValidator1" runat="server"

ControlToValidate="TextBox2"

ErrorMessage="El dato debe ser numérico entero"

Operator="DataTypeCheck" Type="Integer">\*

</asp:CompareValidator>

<asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>

<asp:CompareValidator ID="CompareValidator2" runat="server"

ControlToValidate="TextBox3"

ErrorMessage="El dato debe ser numérico entero mayor a cero"

Operator="GreaterThan" ValueToCompare="0">\*</asp:CompareValidator>

<asp:TextBox ID="TextBox4" runat="server"></asp:TextBox> <asp:TextBox ID="TextBox5" runat="server"></asp:TextBox>

<asp:CompareValidator ID="CompareValidator3" runat="server"

ControlToCompare="TextBox5" ControlToValidate="TextBox4"

ErrorMessage="Los valores deben ser iguales">\*

</asp:CompareValidator>

#### Control RangeValidator

Comprueba si el valor de un control de entrada está comprendido en un intervalo especificado de valores.

El control RangeValidator utiliza cuatro propiedades claves para realizar la validación. La propiedad **ControlToValidate** contiene el control de entrada que se va a validar. Las propiedades **MinimumValue** y **MaximumValue** especifican los valores mínimo y máximo del intervalo válido (los valores ingresados en estas propiedades son considerados válidos).

La propiedad **Type** se utiliza para especificar el tipo de datos de los valores que se van a comparar. Los valores que se van a comparar se convierten a este tipo de datos antes de realizarse la validación.

Ejemplo:

<asp:TextBox ID="TextBox6" runat="server"></asp:TextBox>

<asp:RangeValidator ID="RangeValidator1" runat="server"

ControlToValidate="TextBox6"

ErrorMessage="El valor debe ser numerico entero y debe estar comprendido entre 0 y 100." MaximumValue="100" MinimumValue="0">\*

</asp:RangeValidator>

#### Control RegularExpressionValidator

Comprueba si el valor de un control de entrada asociado coincide con el modelo especificado por una expresión regular. Este tipo de validación permite comprobar secuencias de caracteres previsibles, como las de las direcciones de correo electrónico, los números de teléfono y los códigos postales.

La implementación de la validación mediante expresiones regulares presenta algunas diferencias en el cliente y en el servidor. En el cliente, se utiliza la sintaxis de expresiones regulares de **JScript**. En el servidor, se utiliza la sintaxis de la clase **System.Text. RegularExpressions.Regex**. Como la sintaxis de expresiones regulares de JScript es un subconjunto de la sintaxis de Regex, es aconsejable utilizar dicha sintaxis de expresiones regulares de JScript para obtener los mismos resultados en el cliente y en el servidor.

Visual Studio 2008 provee un set de expresiones regulares que incluyen direcciones de mail, números de teléfonos, etc. Se pueden construir sus propias expresiones usando los caracteres correspondientes.

La propiedad **ValidationExpression** se utiliza para ingresar la expresión regular para validar el dato.

Ejemplos: En el primer ejemplo se valida una dirección de mail utilizando una expresión regular predefinida por Visual Studio. En el segundo ejemplo se eligió una expresión personalizada y se usó el lenguaje de expresiones para crearla.

<asp:TextBox ID="TextBox7" runat="server"></asp:TextBox>

<asp:RegularExpressionValidator ID="RegularExpressionValidator1"

runat="server" ControlToValidate="TextBox7"

ErrorMessage="El dato no se corresponde con una dirección de mail"

ValidationExpression="\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*">\*

</asp:RegularExpressionValidator>

<asp:TextBox ID="TextBox8" runat="server"></asp:TextBox>

<asp:RegularExpressionValidator ID="RegularExpressionValidator2"

runat="server" ControlToValidate="TextBox8"

ErrorMessage="La sintaxis del CUIT es incorrecta"

ValidationExpression="\d{2}-\d{8}-\d{1}">\*

</asp:RegularExpressionValidator>

#### Control CustomValidator

Use el control CustomValidator para proporcionar una función de validación definida por el usuario para un control de entrada.

Los controles de validación siempre realizan la validación en el servidor. También tienen una implementación completa en el cliente que permite a los exploradores compatibles con secuencias de comandos (como Internet Explorer versión 4.0 o posterior) realizar la validación en el cliente. La validación en el cliente mejora el proceso de validación, ya que se comprueba la entrada del usuario antes de enviarla al servidor. De este modo, se puede detectar los errores en el cliente antes de enviar el formulario y se evita el recorrido de ida y vuelta de la información necesaria para la validación en el servidor.

Para crear una función de validación en el servidor, proporcione un controlador para el evento **ServerValidate** que realice la validación. El acceso a la cadena del control de entrada que se va a validar es posible mediante la propiedad **Value** del objeto **ServerValidateEventArgs** que se pasa al controlador de eventos como parámetro. Después, el resultado de la validación se almacena en la propiedad **IsValid** del objeto **ServerValidateEventArgs**.

Ejemplo

protected void CustomValidator1\_ServerValidate(object source, ServerValidateEventArgs args)

{

if (args.Value.Length <= 5)

args.IsValid = false;

else

args.IsValid = true;

}

Para crear una función de validación en el cliente, debe escribirla en un lenguaje que admita el explorador de destino, como VBScript o JScript. La función se asocia al control usando la propiedad **ClientValidationFunction**.

Ejemplo

<asp:TextBox ID="TextBox9" runat="server"></asp:TextBox>

<asp:CustomValidator ID="CustomValidator1" runat="server"

ClientValidationFunction="Validar" ControlToValidate="TextBox9"

ErrorMessage="El dato no puede contener menos de 5 caracteres"

onservervalidate="CustomValidator1\_ServerValidate">\*

</asp:CustomValidator>

<script type="text/javascript" >

function Validar(source, args) {

if (args.Value.length <= 5)

args.IsValid = false;

else

args.IsValid = true;

}

</script>

El parámetro **source** es una referencia al elemento <span> representado para el control CustomValidator. Esto le permite controlar mediante programación la etiqueta <span>, como modificar el atributo InnerHtml. El parámetro **args** es un objeto con dos propiedades: **Value** e **IsValid**. Este parámetro le permite obtener el valor del control para validar e indicar si el valor es válido basándose en la rutina de validación personalizada.

#### Page.IsValid

Cuando utilice controles de validación, debe comprobar siempre los resultados de la validación del lado del servidor antes de realizar cualquier procesamiento. Después de una devolución de datos pero antes de que se llame a los métodos del evento, la página llama a los controles de validación y agrega sus resultados en la propiedad **Page.IsValid**. (También puede llamar a los controles de validación utilizando explícitamente el método **Validate**.) En su propio código, debe comprobar que la propiedad **Page.IsValid** devuelve **true** antes de procesar la entrada. Aunque los exploradores compatibles con secuencias de comandos pueden impedir que se produzca una devolución de datos en el cliente si la comprobación de la validación produce un error, también debe comprobar siempre Page.IsValid en el código del servidor antes de procesar datos validados.

Ejemplo

protected void btnAceptar\_Click(object sender, EventArgs e)

{

if (Page.IsValid)

{

// Código

}

}

#### Control ValidationSummary

Se utiliza para resumir los mensajes de error de todos los controles de validación de una página Web, en una sola ubicación. Puede crear un resumen de los mensajes de error procedentes de un grupo de controles de validación de una página Web si asigna el control **ValidationSummary** a un grupo de validación estableciendo la propiedad **ValidationGroup**.

El resumen puede aparecer en forma de lista, de lista con viñetas o como un único párrafo, en función de la propiedad **DisplayMode**.

El resumen puede aparecer en una página Web y/o en un cuadro de mensaje estableciendo las propiedades **ShowSummary** y **ShowMessageBox**, respectivamente.

Ejemplo: En este ejemplo los mensajes de error se mostrarán en una MessageBox.

<asp:ValidationSummary ID="ValidationSummary1" runat="server"

ShowMessageBox="True" ShowSummary="False" />

