# *Desarrollo de Aplicaciones Web con Visual Studio 2005*

## ASP.net 2.0
## Controles de Validación

### Contenido




Moisés Grappín Palestina

Curso ASP.net 2.0


Comunidad de desarrolladores Puebla.NET
http://www.dotnetpuebla.com/

# Controles de Validación

## Introducción

Validar una aplicación Web puede llegar a ser absolutamente complejo, así que ASP.net nos provee de controles de validación, comúnmente llamados validators. Un control de validación (validator) es un control especial que asegura que el usuario introduzca el tipo de dato correcto y en el rango apropiado en un control dado. Cada tipo de control de validación realiza una revisión específica, así se pueden utilizar varios controles juntos para proveer una amplia cobertura en la introducción de datos en las aplicaciones.

Aunque los controles de validación varían en funcionalidad, todos estos controles requieren un mensaje de error colocado en la propiedad *Text*, el cual es el mensaje que el usuario ve y usa para corregir el error en la página Web. Por ejemplo cuando el usuario introduce un valor que esta fuera del rango correcto, el control de validación de rango *RangeValidator* despliega el mensaje de error que ha sido colocado en la propiedad *Text.* Además los controles de validación proveen una propiedad llamada *ControlToValidate* que se usa para asociar el control a validar con el control de validación. (Algunos controles de validación pueden requerir dos controles como entrada). Visual Web Developer soporta varios tipos de control de validación, a continuación se presentan los controles de validación más usados comúnmente en las aplicaciones Web.

## Tipos de Controles de Validación

ASP.net provee varios controles de validación los cuales los encontramos en el *ToolBox* en la opción de *Validation*, la figura 1 nos muestra los diferentes tipos de controles de validación que ASP.net provee:

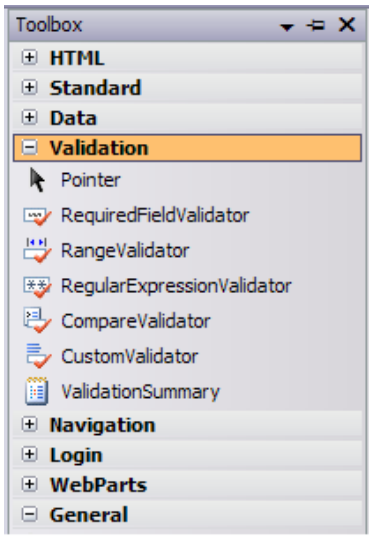
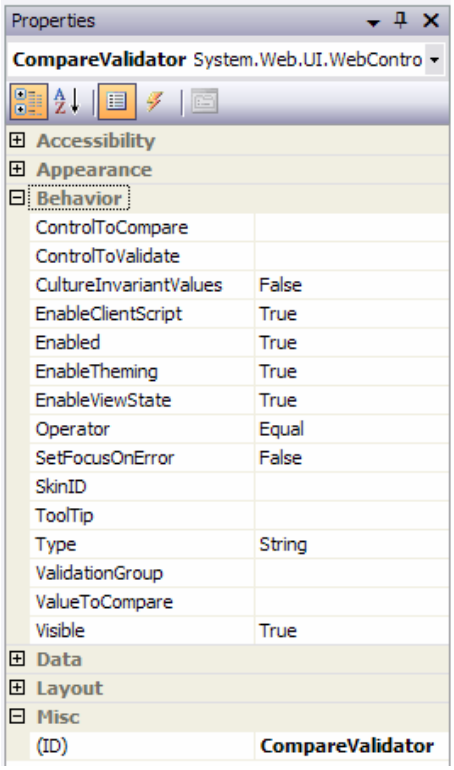



**Figura 1. Controles de Validación**

## CompareValidator

El control de validación de comparación acepta dos controles como entrada para comparar el valor de cada control. Si los dos controles no satisfacen la condición que se especificó, el control *CompareValidator* despliega un mensaje de error. La siguiente figura nos muestra las propiedades de comportamiento de este control y las cuales se describen a continuación:



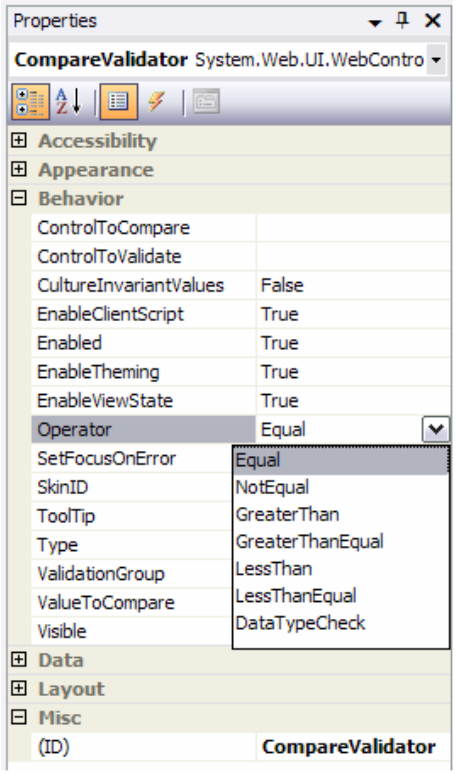

**Propiedades de comportamiento del control CompareValidator**

El nombre del segundo control o control a comparar debe aparecer en la propiedad *ControlToCompare*, (el control a comparar con).

La propiedad *ValueToCompare* se proporciona también para comparar el valor contra el del control al que se le ha asignado el control de validación en lugar de hacer la comparación con otro control.

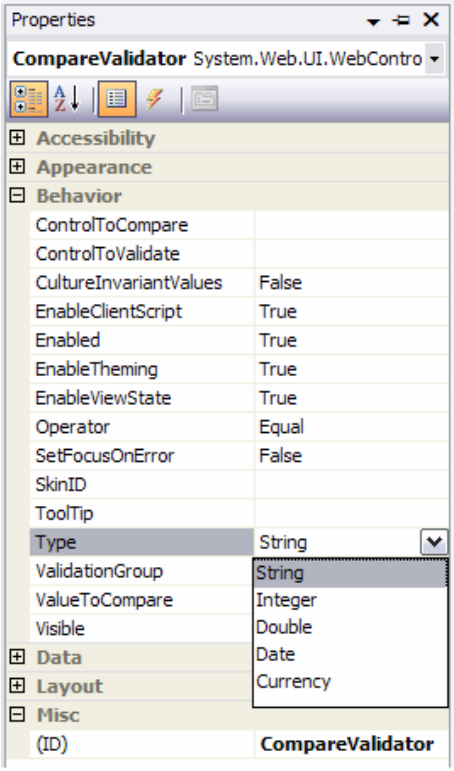
La propiedad *ControlToValidate* sirve para indicar a que control se le asignará en control de validación.

La propiedad *Operator* define la comparación entre los dos controles. Por ejemplo si se elije la opción *GreaterThan*, el valor del control listado en la propiedad *ControlToValidate* debe ser mayor que el valor del control listado en la propiedad *ControlToCompare,* otras opciones de la propiedad *Operator* se muestran en la siguiente figura.




**Opciones de la propiedad Operator**

La propiedad *Type* asegura que el segundo control contiene datos del tipo correcto que se desea obtener los cuales se muestran en la siguiente figura, los controles no compararán los valores si sus tipos no son iguales.

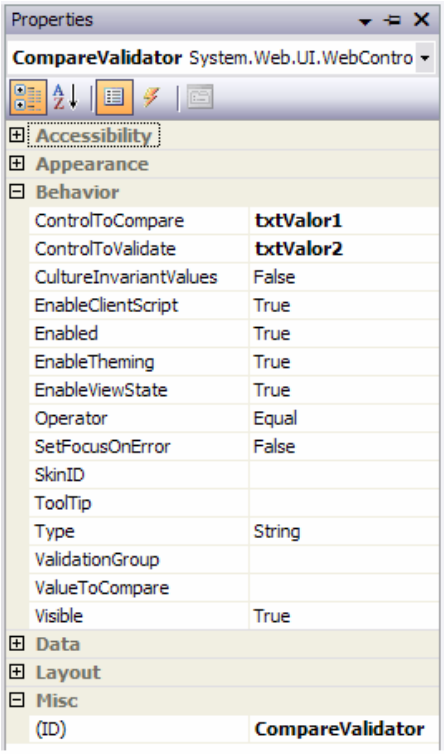



**Tipos de datos para validación**

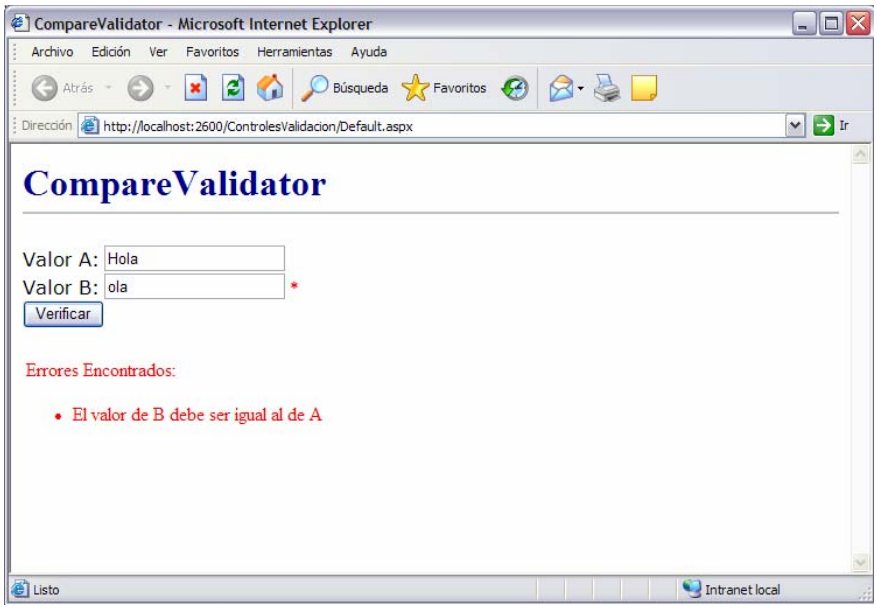
**Ejemplos:**

El siguiente ejemplo nos muestra la comparación del contenido de dos *TextBox* (txtValor1 y txtValor2) en los cuales se debe de poner el mismo dato y que sea del tipo string y un *Button* que verificará si el contenido de ambos *Textbox* es el mismo.

Creamos un control *CompareValidator* cuyas propiedades quedan de la siguiente manera:

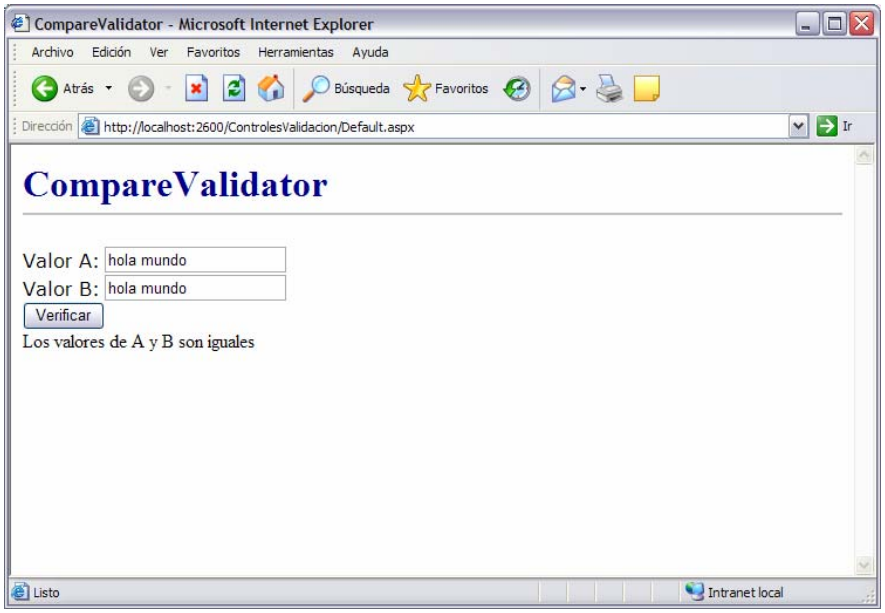



Probamos nuestra aplicación y verificamos si se realiza la comparación



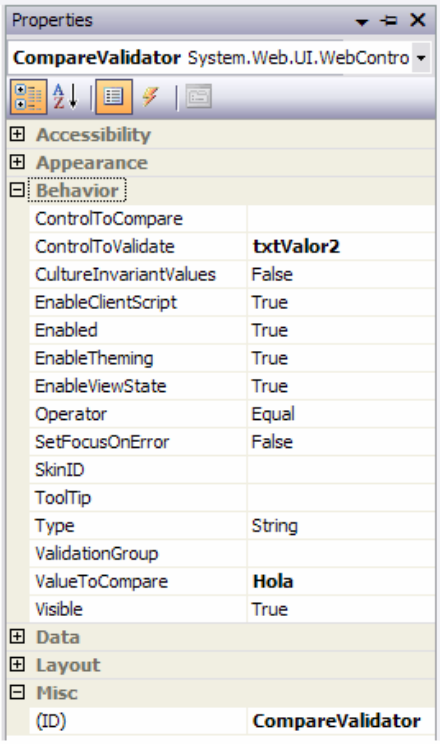
En este caso el valor de A no es el mismo que el de B y encontramos el error.

La siguiente figura nos muestra cuando la comparación se realizó correctamente.

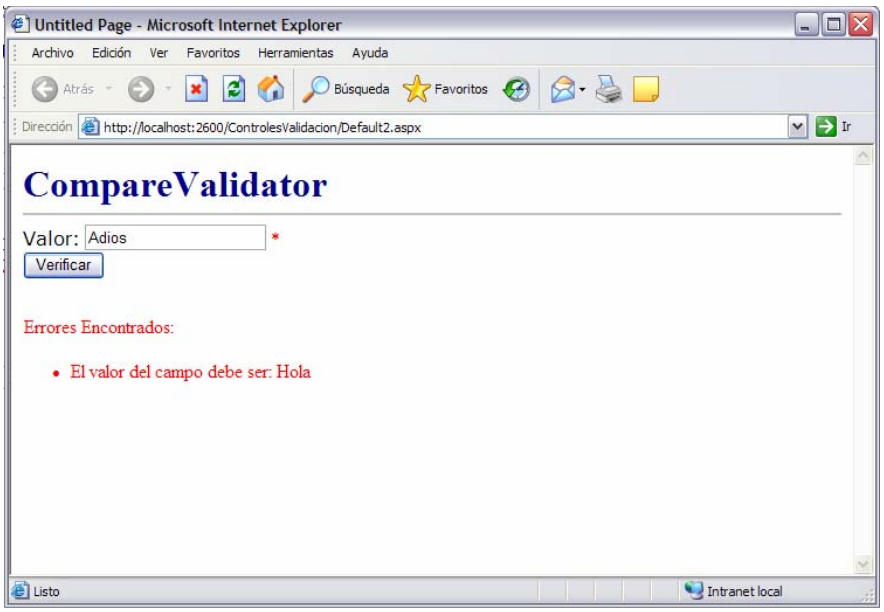



En el siguiente ejemplo realizamos lo mismo pero ahora quitamos el *TextBox* txtValor1 y le asignamos un valor a comparar a nuestro control de validación, en este caso le asignamos “Hola” a la propiedad *ValueToCompare*.

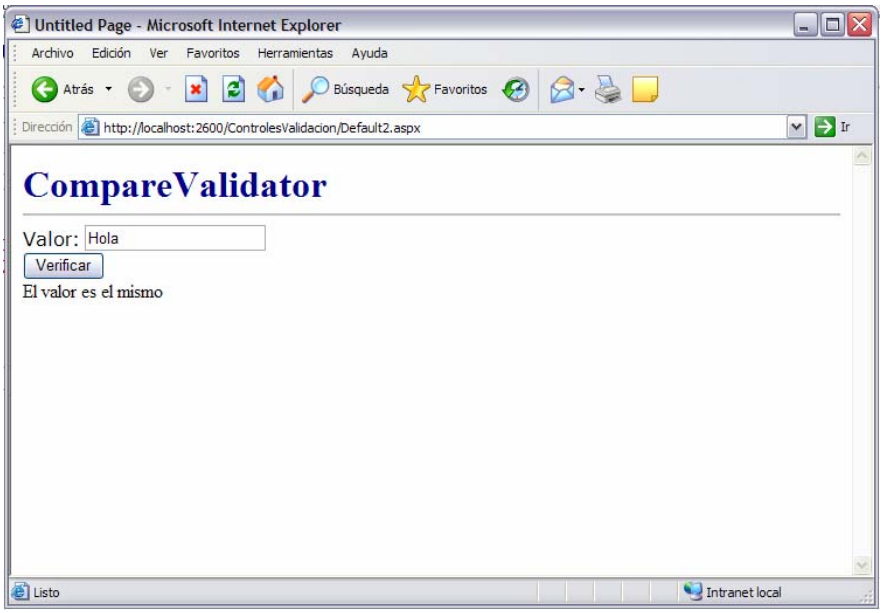
Las propiedades de nuestro control de validación queda de la siguiente manera:



En este caso la comparación se hace contra el valor establecido en la propiedad *ValueToCompare*, en nuestro ejemplo el valor es "*Hola*"

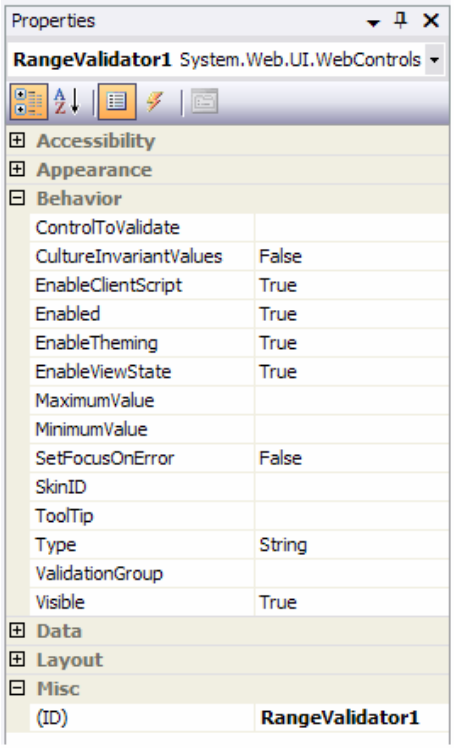



La comparación con el valor corregido es:



## RangeValidator

El control de validación de rango se asegura que la entrada en un control caiga dentro de un rango de valores. Las propiedades de comportamiento de este control se ilustran en la siguiente figura:




**Propiedades de Comportamiento de RangeValidator**

Las propiedades del control se describen de la siguiente manera:

Las propiedades *MinimumValue* y *MaximumValue* contienen el límite de valores que el usuario puede introducir.

La propiedad *Type* determina que tipo de dato aceptará el control, entre estos tipos están String, Integer, Double, Date y Currency.

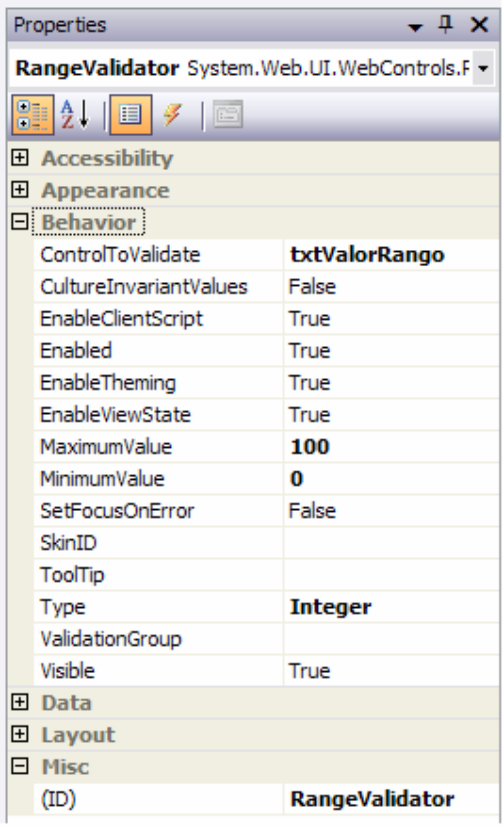
Si la entrada no cae dentro del rango seleccionado o es de tipo diferente al elegido el control desplegará un mensaje de error.

### Ejemplos:

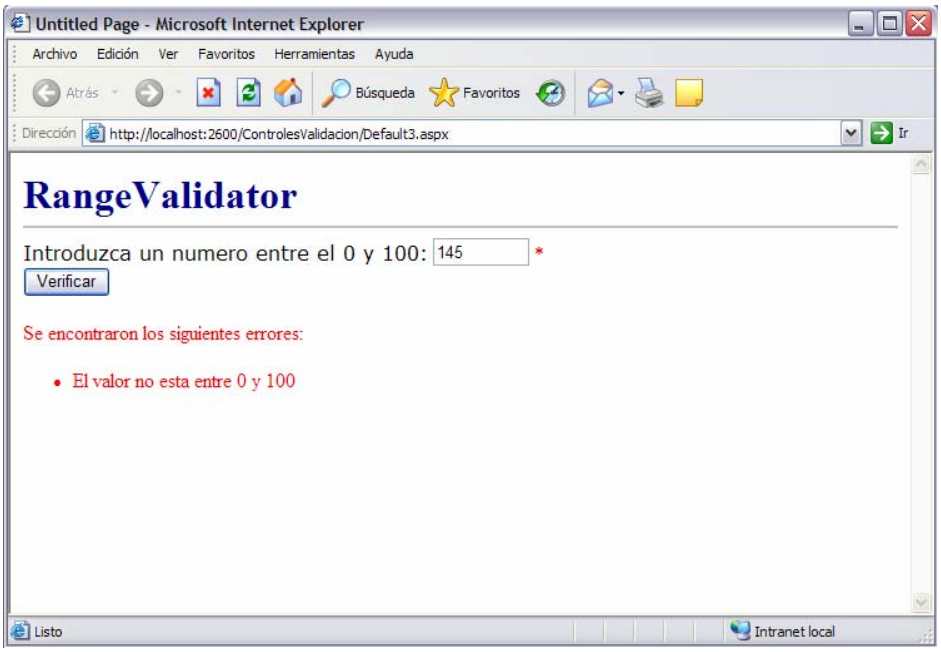
En el siguiente ejemplo se tendrá un *TextBox* llamado txtValorRango y un *Button* para verificar que está dentro del rango y un *Label* para mostrar el mensaje.

El rango en el que estará el valor deberá ser del tipo Integer y estar entre 0 y 100 para que el dato sea válido.

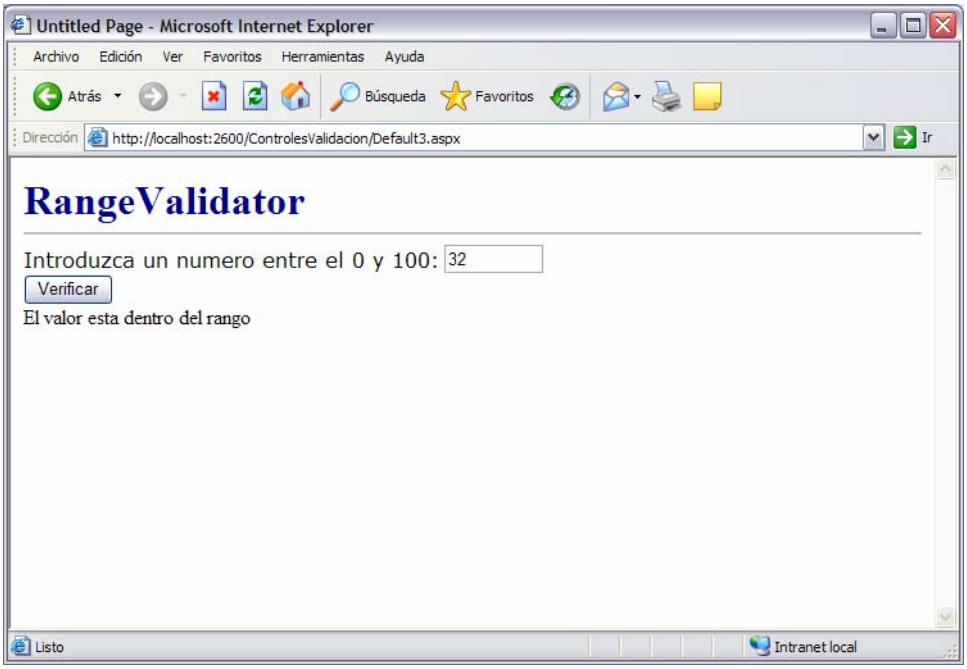
Nuestras propiedades de comportamiento deben estar de la siguiente manera:




Probamos nuestra aplicación e introducimos un valor fuera del rango de 0 y 100:



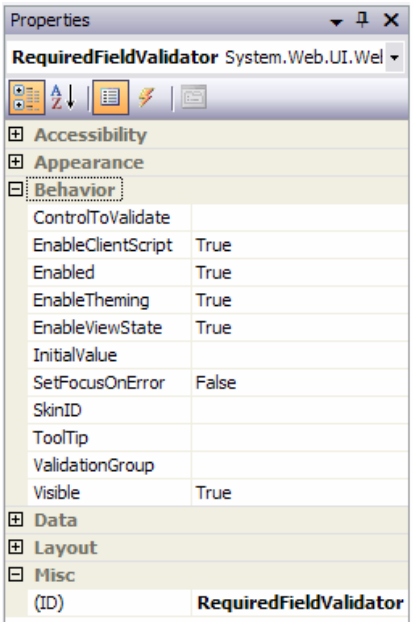
Ahora probamos la página con un valor que esté en el rango:




## RequiredFieldValidator

Este control de validación es el más fácil de entender. Asegura que el usuario no omita una entrada, es decir, que el usuario no deje en blanco un campo obligatorio, si es así el control despliega un mensaje de error.

Las propiedades de comportamiento se ilustran en la siguiente figura:




**Propiedades de comportamiento de RequiredFieldValidator**

Las propiedades del control de validación *RequiredFieldValidator* se describen como sigue:

La propiedad *ControlToValidate* indica a que control se le asignará el control de validación

La propiedad *InitialValue* se introduce para indicarle al usuario que valor no se requiere que el introduzca.

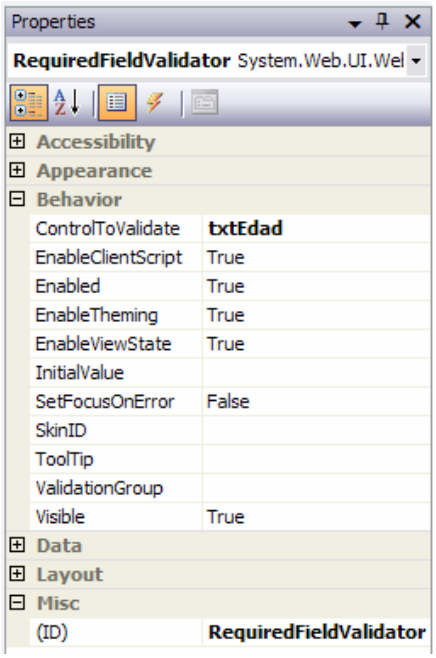
**Ejemplos:**

En el siguiente ejemplo se proporciona un *TextBox* llamado txtAño en donde se requiere que el usuario introduzca su año de nacimiento, si el usuario no introduce nada se le notificará que el campo no debe de estar vació.

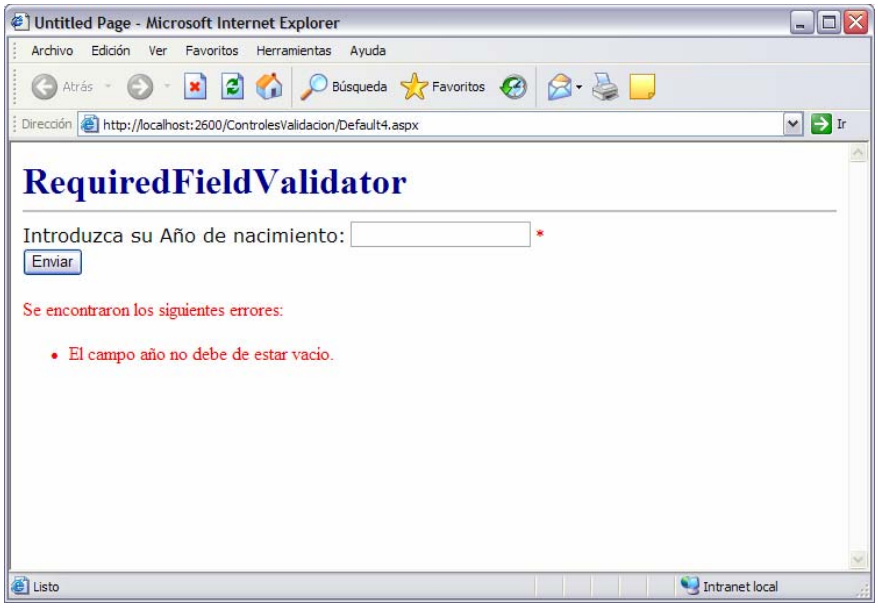
Posteriormente se establecerá la propiedad *InitialValue* a el año 1980 para indicarle al usuario que solo aceptaremos cualquier año de nacimiento excepto aquellos que han nacido en 1980.

Si el usuario introduce su año y este introduce 1980 la aplicación le lanzará el mensaje de error.

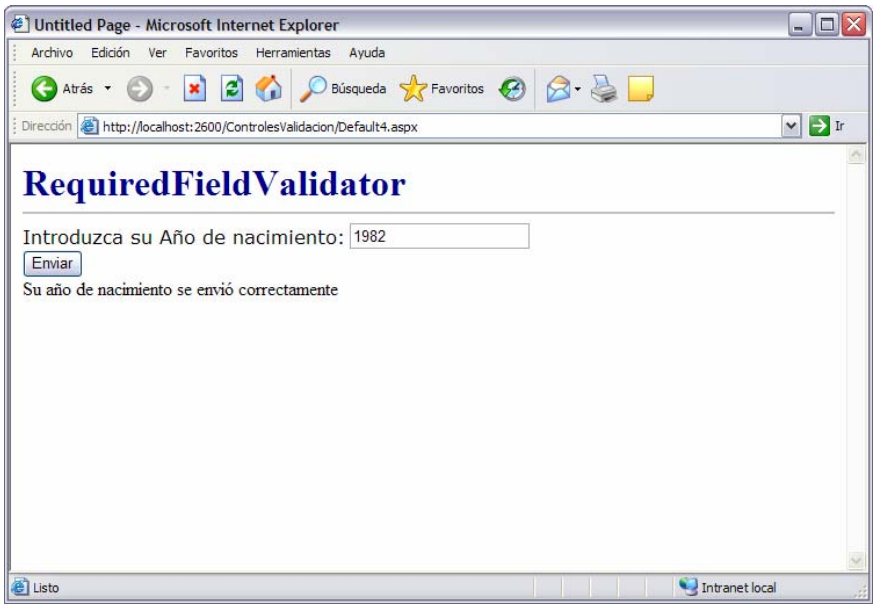
Las propiedades de comportamiento del control de validación *RequiredFieldValidator* deben estar de la siguiente manera:



Realizamos la comprobación del campo obligatorio dejando el *TextBox* txtAño vació.

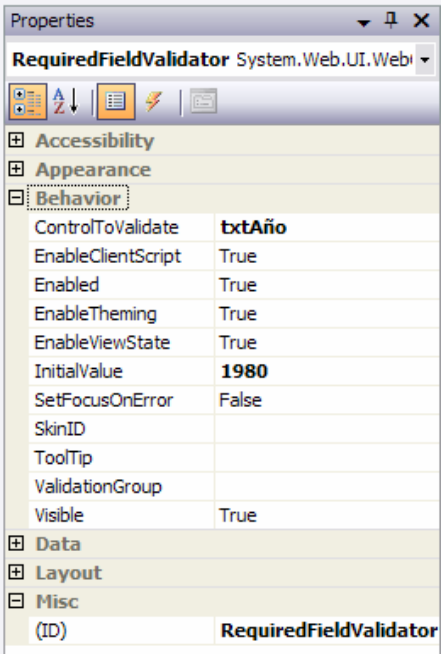



Ahora introducimos un año y veremos que el error ya no es mostrado y se ha ejecutado el envió del dato.

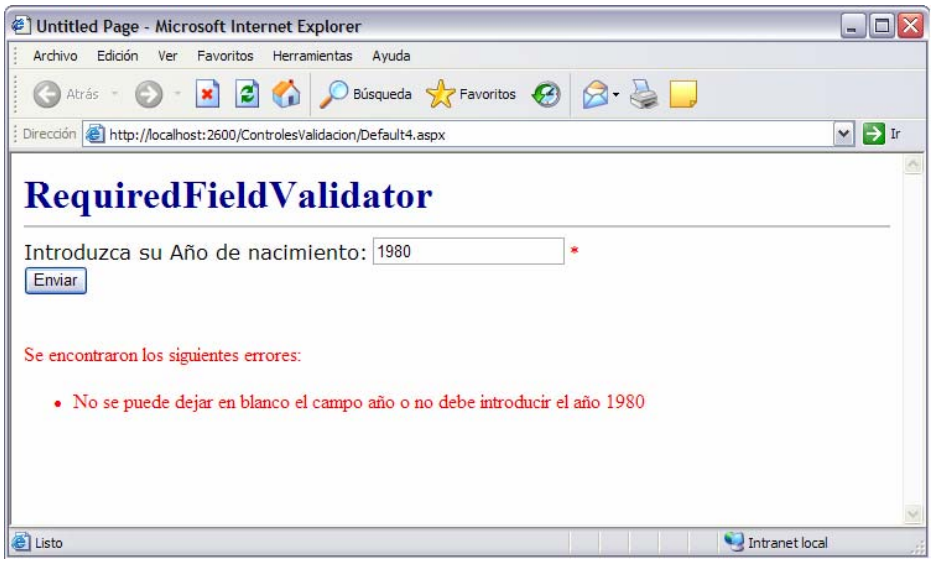



En el siguiente ejemplo introducimos el valor de 1980 en la propiedad *InitialValue* para no permitir que el usuario introduzca ese año

El siguiente diagrama nos muestra como debe de estar las propiedades de comportamiento del *RequiredFieldValidator*




En la siguiente figura se muestra el control de validación el cual acepta cualquier año excepto 1980, ya que este valor es el que se ha puesto en la propiedad *InitialValue*



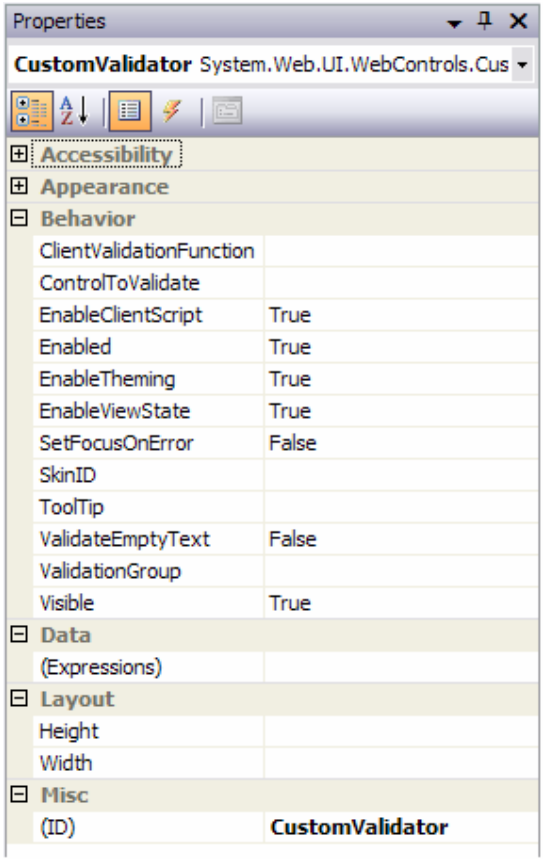

Así entonces se tiene otra opción de validación de datos con la propiedad *InitialValue* del control *RequiredFieldValidator*.

**CustomFieldValidator**

Este control permite escribir código para crear la expresión de validación, es decir, la lógica de validación puede ser personalizada para verificar la entrada del usuario contra una variable, formula o entrada de una segunda fuente, el control de validación *CustomValidator* es usado a menudo para situaciones, tal y como la verificación de un password, en el cual la entrada del usuario es comparada con un password que esta almacenado en la base de datos.

El control *CustomValidator* valida del lado del servidor y puede validar del lado del cliente si el navegador soporta validación, en los otros controles de validación ASP.net crea los scripts de validación del lado del cliente y del servidor, en este control el programador tiene que escribir ambos scripts.

Las propiedades del control de validación *CustomValidator* se muestran en la siguiente figura.




La propiedad *ClientValidationFunction* es el script que se requiere que se ejecute del lado del cliente.

La propiedad *ControlToValidate* sirve para indicar el control al que se le asignará el control de validación.

Para indicar la función que se ejecutará en el servidor tenemos que hacer en la parte del code-behind una función cuya firma sea que no regresa nada y los parámetros sean una variable del tipo object y otra del tipo ServerValidateEventArgs, esta función será indicada en el evento ServerValidate para su ejecución, este evento se encuentra en los eventos del objeto *CustomValidator*.

Ejemplo de script de lado del cliente:

```
<script language = "Jscript">

function MiFuncionCliente(s, args)
{

  alert("Estoy Corriendo del lado del Cliente! ");

  var intValue = args.Value;

   if (intValue % 2 == 0)
   {
       args.IsValid = true;

   }
   else
   {
       args.IsValid = false;
   }

}
</script>
```

Ejemplo de función del lado del servidor:

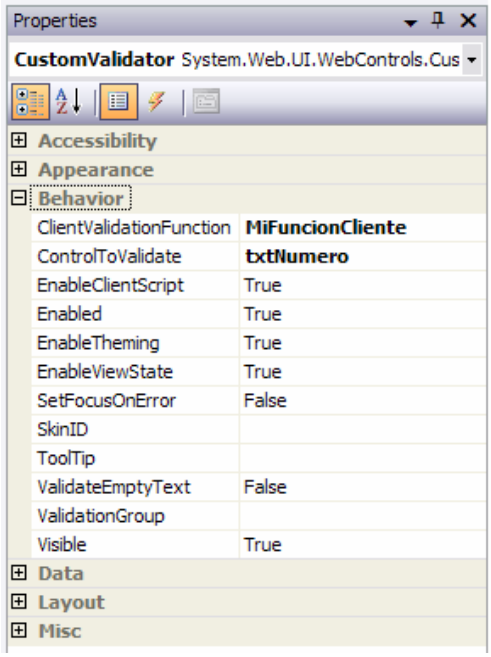
```
private void MiFuncionServidor(object objS, ServerValidateEventArgs args)
{
       int intValue = Convert.ToInt16(args.Value);

       if (intValue%2 == 0)
       {
              args.IsValid = true;
       }
       else
       {
              args.IsValid = false;
       }
}
```

La propiedad *EnabledClientScript* nos indica si se puede ejecutar el script del lado del cliente o solo se debe ejecutar el script del lado del servidor.

**Ejemplos:**

Esta vez se realizará un ejemplo donde el dato que proporcione el usuario debe ser un número divisible entre 2, así entonces, tendremos un *TextBox* llamado txtNumero y un *Button* para validar el número, se hará primero la validación del lado del cliente y luego del lado del servidor para comprobar ambas validaciones, los códigos de validación son los códigos proporcionados anteriormente.

Nuestras propiedades deberán de estar de la siguiente manera:

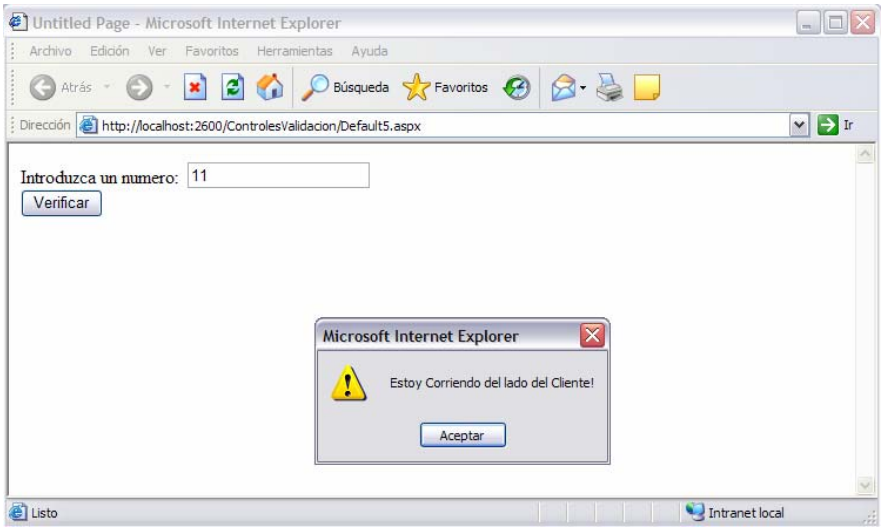



Donde la propiedad *ClientValidationFunction* contiene el nombre de la funcion que debe de ejecutarse, en este caso es “MiFuncionCliente”

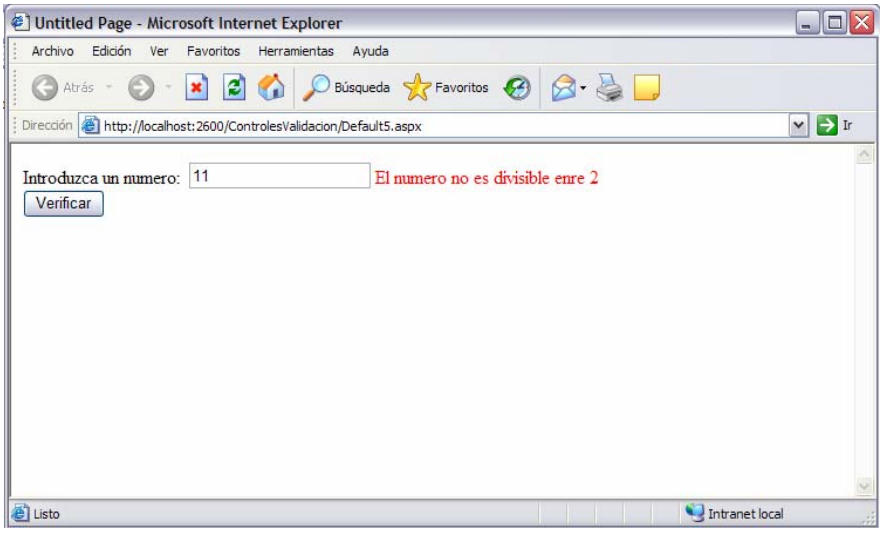
```
<script language = "Jscript">
function MiFuncionCliente(s, args)
{
  alert("Estoy Corriendo del lado del Cliente! ");
  var intValue = args.Value;

   if (intValue % 2 == 0)
   {
       args.IsValid = true;
   }
   else
   {
       args.IsValid = false;
   }
}
</script>
```

Este script debe de colocarse dentro del código HTML.

Cuando validamos del lado del cliente al presionar el boton debemos de ver primero el mensaje de *"Estoy Corriendo del lado del Cliente!"*, como se muestra en la siguiente figura:



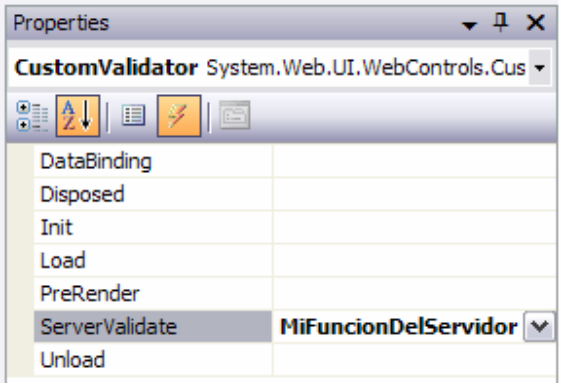

Luego se ejecutará el código de validación y marcará el error, esto siempre del lado del cliente.



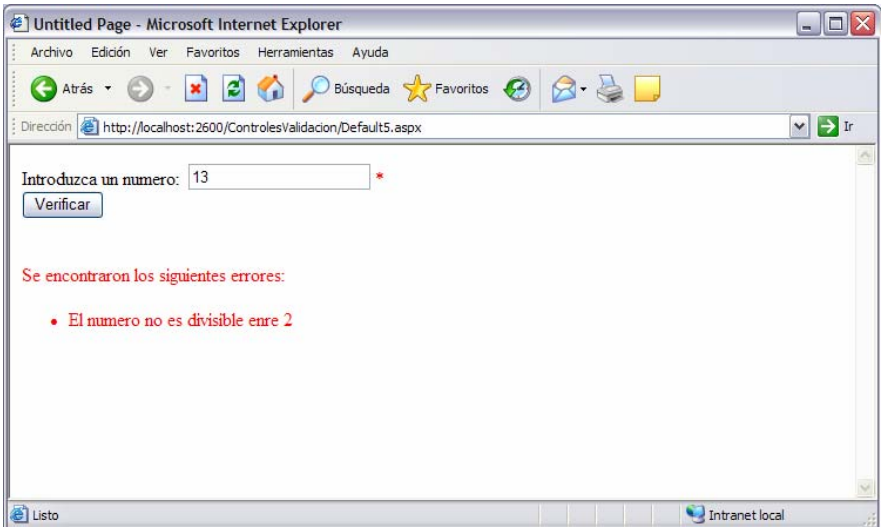

Ahora del lado del servidor tendremos nuestro código de la siguiente manera en nuestro archivo fuente, donde crearemos nuestra función con la siguiente firma:

```
private void MiFuncionServidor(object objS, ServerValidateEventArgs args)
{
        int intValue = Convert.ToInt16(args.Value);

        if (intValue%2 == 0)
                args.IsValid = true;
        else
                args.IsValid = false;
}
```

Y ahora agregamos esta función al evento *ServerValidate* del *CustomValidator*, como se muestra en la siguiente figura:



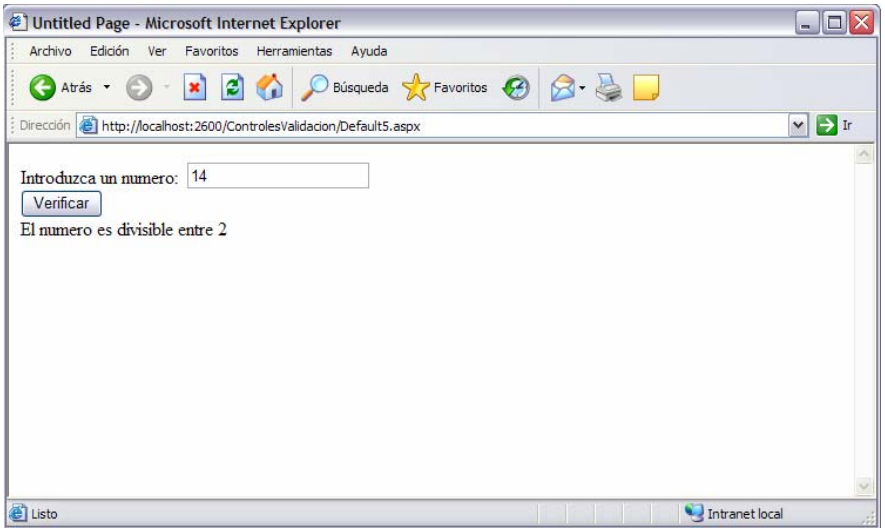

Las propiedades del *CustomValidator* serán las mismas como se mostró anteriormente solo que en este caso solo validaremos del lado del servidor, y en consecuencia la propiedad *EnableClientScript* tendrá que estar en *False*.




Ahora podremos observar que ya no se validó del lado del cliente, solo del servidor y el resultado es el mostrado en la figura anterior.

La validación también se puede hacer haciéndose de los dos lados al mismo tiempo, para esto la propiedad *EnableClientScript* debe de estar en *True* y se debe de proporcionar tanto la función a ejecutarse del lado del cliente como la que se tiene que ejecutar del lado del servidor.

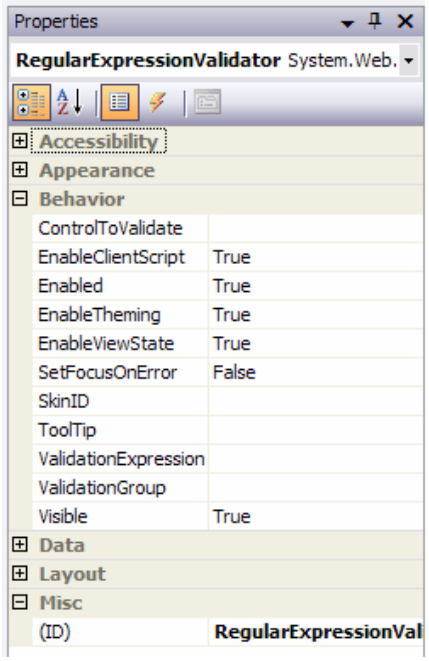
La siguiente figura muestra la validación correcta para ambos lados.



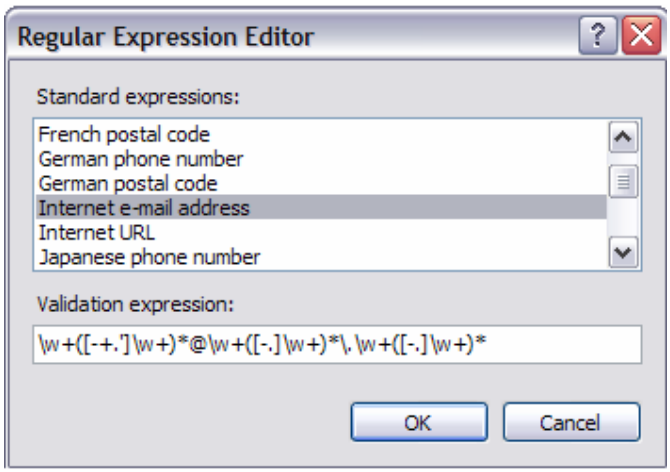

## RegularExpressionValidator

Este control de validación verifica que la entrada cumpla con un patrón que ha sido definido por una expresión regular, este control permite verificar secuencias de caracteres predecibles, tales como direcciones de correo electrónico, direcciones, números de teléfonos y códigos postales. Visual Studio.net proporciona patrones predefinidos para expresiones comunes tales como números telefónicos.

Este control de validación compara el patrón de caracteres, dígitos y símbolos, introducidos por el usuario con uno o más patrones en el control. Las propiedades de comportamiento del *RegularExpressionValidator* son las siguientes:



La propiedad *ValidationExpression* sirve para introducir la expresión regular, también existen patrones predefinidos, la siguiente figura nos muestra el editor de expresiones regulares con patrones predefinidos, tales como el correo electrónico:




**Construcción de una expresión regular.**

Para construir una expresión regular se debe de utilizar un conjunto de caracteres que son mostrados en la siguiente tabla:

| Carácter | Definición |
|---|---|
| a | Debe utilizar la letra **a** en minúscula, cualquier letra que no precedida por una diagonal invertida "\", o parte de un rango, es un requerimiento para ese valor literal |
| 1 | Debe utilizar el numero **1**, cualquier numero que no precedida por una diagonal invertida "\", o parte de un rango, es un requerimiento para ese valor literal |
| ? | 0 ó un elemento |
| * | 0 a muchos elementos |
| + | 1 ó N elementos (al menos 1) |
| [0-n] | Rango de valor entero de 0 a N |
| {n} | Longitud debe ser n caracteres |
| \| | Separación de múltiples patrones válidos |
| \ | El siguiente carácter debe ser un carácter comando |
| \w | Debe tener un carácter |
| \d | Debe tener un numero |
| \. | Debe tener un punto |

Con estos caracteres entonces podemos empezar a hacer nuestros patrones, como ejemplo sencillo podemos hacer la validación de un número de teléfono con clave lada que lleve entre paréntesis la lada de tres dígitos, puede usarse después un guión y después debe introducirse número de 7 dígitos.

Nuestra expresión regular quedaría de la siguiente manera:

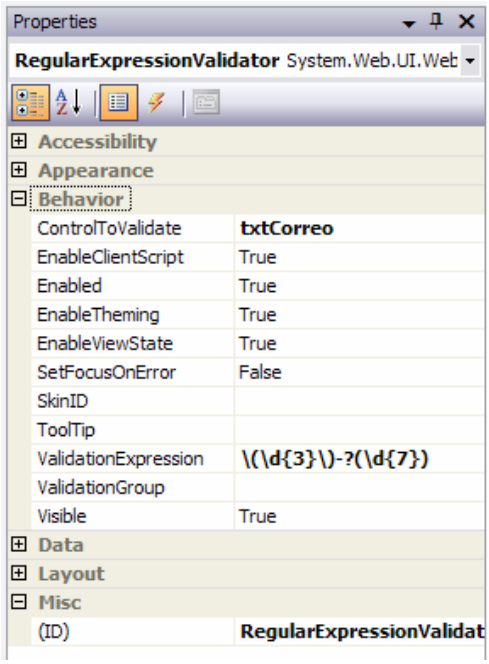
\(\d{3}\)-?(\d{7})

| Carácter | Definición |
|---|---|
| \( | Indica que se debe de usar un paréntesis que abre |
| \d{3} | Indica que debe de introducirse un numero entero de tres dígitos |
| -? | Indica que puede usarse un guión |
| \d{7} | Indica que debe introducirse un numero de 7dígitos |

Veamos los siguientes ejemplos de entrada:

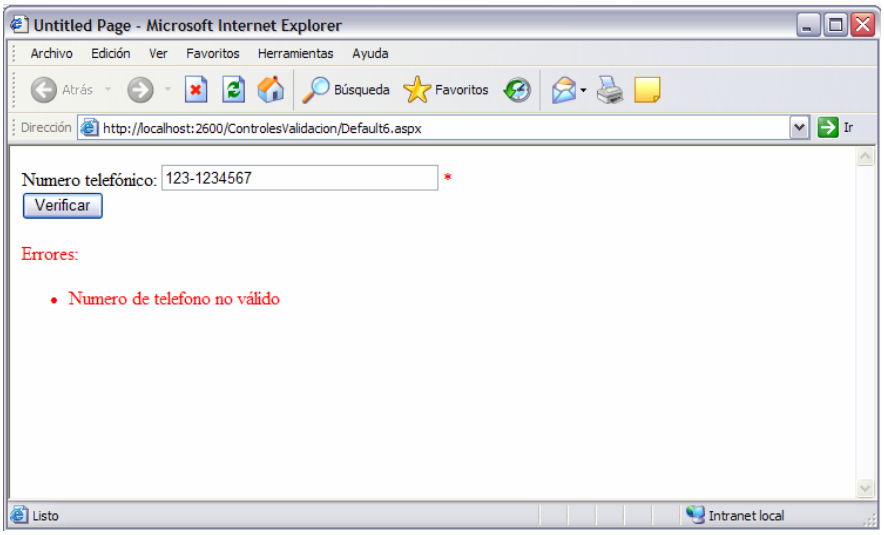
| Entrada | Descripción |
|---|---|
| (12)1234567 | Es incorrecto porque la lada solo tiene 2 elementos |
| (123)123456 | Es incorrecto porque el numero tiene 6 elementos |
| (123)-123456 | Es incorrecto porque el numero tiene 6 elementos |
| (123)1234567 | Es correcto porque no es necesario utilizar el guión |
| (123)-1234567 | Es correcto porque se puede utilizar el guión |

Así esta expresión se la asignamos a nuestro control de validación *RegularExpresionValidator,* nuestras propiedades de comportamiento se deben de ver de la siguiente manera:

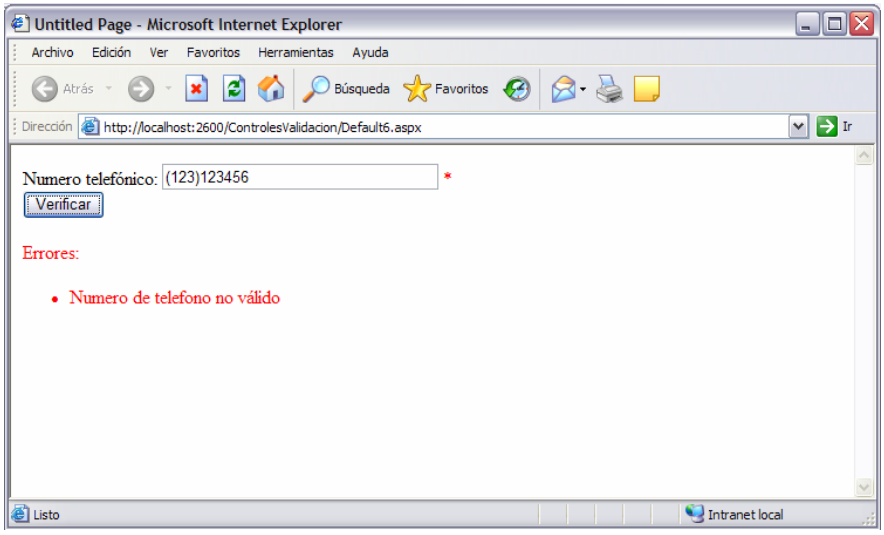



Se pueden utilizar paréntesis para agrupar, si se requiere los elementos de nuestra expresión regular, en nuestro caso lo utilizamos para (\d{7}), lo cual no afecta en nada a nuestra expresión, si quisiéramos los paréntesis utilizaríamos una diagonal invertida y nuestro paréntesis \(.

Probamos nuestra aplicación:

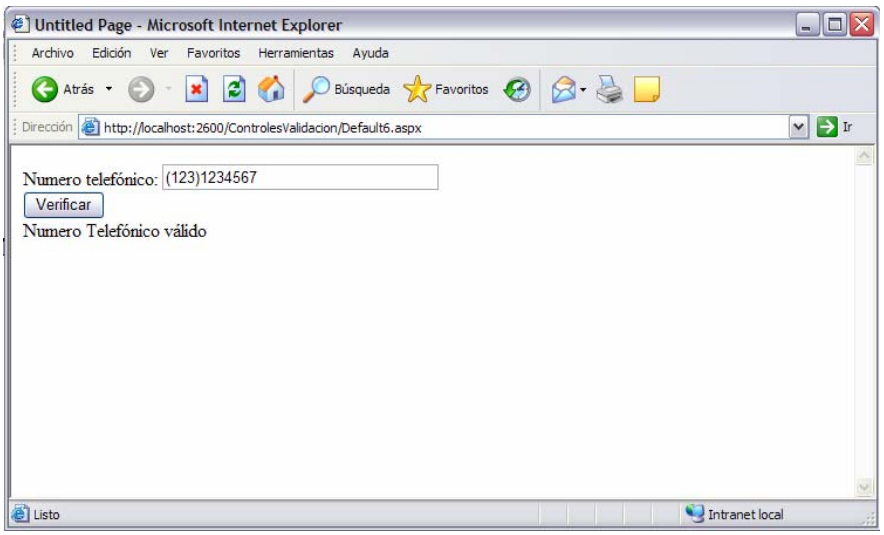


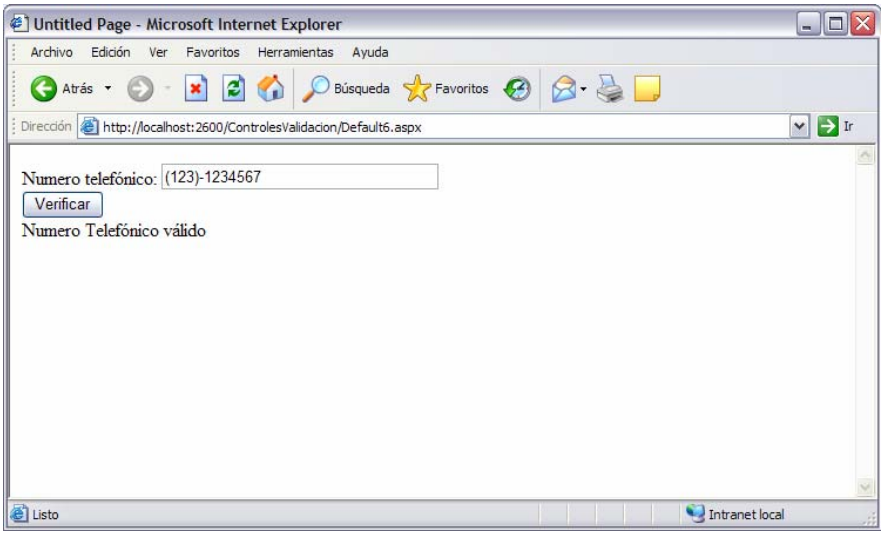

La figura anterior nos muestra un error, ya faltó poner la clave lada entre paréntesis




En este ejemplo tampoco es válido porque el numero telefónico es de seis números y son requeridos siete.

En las figuras siguientes se observa que las dos formas de introducir el número telefónico es valido, puede o no llevar el guión después del paréntesis que cierra en la clave lada.




**ValidationSummary**

Este control presenta un resumen de todos los errores de validación que han aparecido en un formulario, nos presenta el texto que ha sido introducido en la propiedad *ErrorMessage* de los controles de validación.

Estos mensajes los puede desplegar en forma de frase o en forma de lista con o sin viñetas, esto dependiendo del valor en el que se encuentre la propiedad *DisplayMode* del control *ValidationSummary*

## Bibliografía

“Web Development with Visual Studio 2005”
John Paul Mueller
Editorial Wiley, 2005