Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

12

Controles

Controles de Windows
Menús - Controles contenedores

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Diseño de aplicaciones

Snaplines - Anchor y Docking - Paneles

ISBN 978-987-1347-43-8
00012
9 789871 347438

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Controles

Desarrollaremos a fondo uno de los temas más importantes vinculados a la programación de aplicaciones de escritorio.

Los controles son los elementos visuales fundamentales para la creación de las interfaces de usuario. Permiten el ingreso de información por parte del usuario y dan la posibilidad de organizar visualmente aquella que la aplicación proporciona. Cada elemento que vemos en una ventana es un control: botones, listas, textos; todos son controles, con sus propiedades, eventos y métodos.

Pero, entonces, en lo que respecta a .NET, ¿qué es un control? Bien, podemos decir que un control es el elemento básico para crear la interfaz gráfica de la aplicación. Desde el punto de vista sintáctico, es una clase que hereda de **System.Windows.Forms.Control**. La clase Control contiene la definición y la funcionalidad básica de todos los controles; proporciona eventos y propiedades comunes, por lo que su comportamiento es idéntico para todos. Cada control, al heredar de esta clase, agrega y/o modifica propiedades, eventos y métodos, para proporcionar una funcionalidad única.

> Los controles permiten organizar visualmente la información que la aplicación brinda.

La Tabla 4 muestra las propiedades más comunes de la clase Control, y en la Tabla 5 podemos ver algunos de sus eventos.

Controles básicos

El éxito de una aplicación Windows consiste, básicamente, en la correcta elección de los controles proporcionados por .NET, para presentarle la información al usuario y que éste pueda interactuar con ella.

Dada la gran cantidad de controles de la que disponemos, y su amplia variedad de propiedades y eventos particulares, a continuación sólo

Tabla 4 | Principales propiedades de la clase Control

Propiedades	Descripción
BackColor, ForeColor, BackgroundImage, Font, Cursor	Definen la apariencia: color de fondo, color de texto, imagen de fondo, fuente y cursor, respectivamente.
Anchor, Dock, AutoSize	Indican cómo deben reposicionarse los controles al cambiar de tamaño el contenedor.
Location, Size	Definen la ubicación y el tamaño.
Enabled, Visible	Permiten cambiar el estado, habilitando o no, o visualizando o no el control o el formulario.
TabIndex, TabStop	Estas propiedades configuran el orden de tabulación de los controles.
Text	Texto del control. Reemplaza a la propiedad caption en los controles de las versiones anteriores de Visual Basic.

nombraremos los más utilizados, y sus propiedades y eventos más útiles, con una breve reseña sobre su funcionalidad.

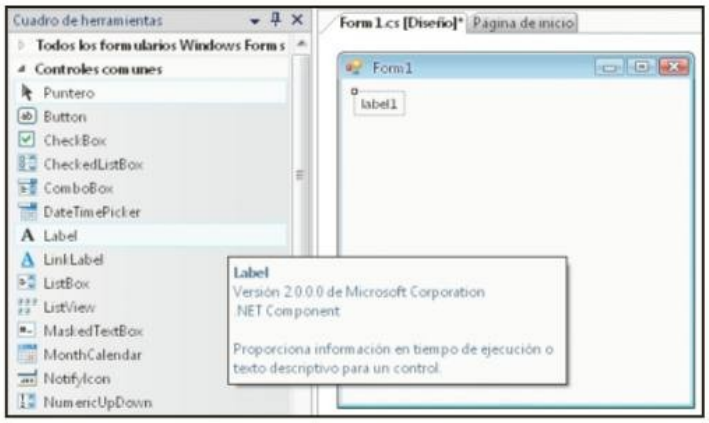

FIGURA 010 | Label. Control Label en la Toolbox y representación visual en el formulario.

Los controles y la herencia

Como se explica en el libro que acompaña a la obra, la herencia permite reutilizar código. El modelo arquitectónico de los controles en .NET es un excelente ejemplo de uso y aprovechamiento de la herencia: podemos crear nuestros propios controles heredando de Control, y ya tendremos resueltas (gracias a la herencia) muchas cuestiones complejas, como el manejo de eventos e, incluso, el soporte para tiempo de diseño.

Label

El Label es un control capaz de mostrar un texto o una imagen en forma de sólo lectura. Es meramente informativo; por ejemplo, podemos colocar un Label con el texto "Número de Cliente", para indicarle al usuario que en el TextBox que está junto a él debe ingresar esa información. Las propiedades más relevantes del control son:

- Text: Texto que se mostrará en el control. Sin dudas, la propiedad que más utilizaremos.
- TextAlign: Determina cómo se posiciona el texto dentro del control.
- AutoSize: Permite que el control cambie su tamaño de manera automática en función del texto que contiene.
- Image: Imagen para mostrar en el control.

En cuanto a los eventos, el Label no tiene ninguno especial. En algunos casos, se utilizan los eventos del mouse sobre el control para informar algo al usuario o realizar alguna acción particular.

Button

Como vimos en los primeros capítulos, el Button es el control de acción por excelencia. El usuario indica, confirma o cancela cualquier acción por realizar en la aplicación a través de un clic del mouse o presionando una tecla. Entre las principales propiedades de Button podemos mencionar:

Tabla 5 | Principales eventos de la clase Control

Eventos	Descripción
Click, DoubleClick, MouseEnter, MouseLeave, MouseDown, MouseUp, MouseMove	Permiten interactuar con el mouse.
KeyPress, KeyUp, KeyDown	Procesan información del teclado.
Paint	Se dispara cuando Windows necesita redibujar el control (por ejemplo, luego de que una ventana pasó por encima).

- Text: Texto que se mostrará en el control.
- TextAlign: Determina cómo se posiciona el texto dentro del control.
- Image: Imagen por mostrar en el control.
- ImageAlign: Determina cómo se posiciona la imagen dentro del control.

El evento por defecto y el más utilizado del botón es Click, que se ejecuta al presionar con el mouse o el teclado. En este evento es donde pondremos el código para realizar la acción solicitada por el usuario.

TextBox

El TextBox es el control más básico y más usado para el ingreso de datos por parte del usuario. Por ejemplo, si necesitamos solicitarle al usuario su nombre y contraseña para validarlo, podremos utilizar dos controles TextBox, uno para cada dato. Debido a que es utilizado para el ingreso de diferentes tipos de información, posee propiedades y eventos adecuados para su validación.
La Tabla 6 muestra algunas de las propiedades del control TextBox.

En cuanto a los eventos, el TextBox presenta muchos, pero sin dudas el más útil es **TextChanged**, que ocurre cuando el texto del control cambia. Este evento nos permitirá detectar interacción del usuario y actuar en consecuencia. Por ejemplo, podemos ejecutar una función de validación cuando el usuario modifica el texto de un TextBox.

MaskedTextbox

El control MaskedTextbox es similar a un TextBox. La diferencia radica en que nos permite especificar la secuencia de caracteres que son aceptados por el control. ¿Para qué sirve? Para controlar mejor lo que hace el usuario; por ejemplo, si sólo aceptamos dígitos numéricos, no necesitaremos validar que lo ingresado

> El label se utiliza tanto para mostrar información como para identificar otros elementos.

es un número y no contiene caracteres extraños. Esta secuencia y formato están especificados en la propiedad Mask (la máscara), donde se indica un conjunto de caracteres utilizados para enmascarar y validar los datos ingresados.

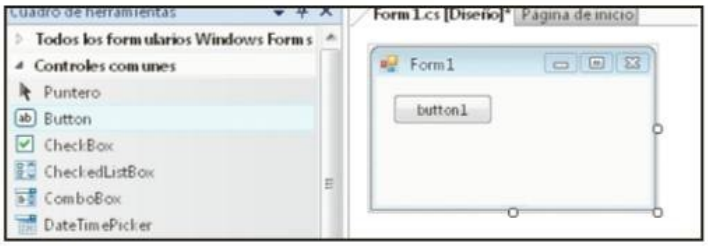

FIGURA 011 | Control Button en la Toolbox y representación visual en el formulario.

FIGURA 012 | Con los controles Label, TextBox y Button, ya podemos crear ventanas para ingresar datos.

TextBox y PassworChar

La propiedad PasswordChar del TextBox no encripta el contenido del texto, sino que sólo impide que los datos sean visibles al ingresarlos. Si deseamos realizar algún tipo de encriptación, podemos recurrir a algunas de las clases que contiene el namespace System.Security.Cryptography.

Es útil para dar una idea concreta de lo que se espera que el usuario ponga; por ejemplo, números de teléfono, códigos postales, datos numéricos, sólo letras, no permitir espacios, etc. Si no se especifica una máscara, funciona como un TextBox normal. Algunos de los caracteres que se utilizan para armar una máscara son los que se muestran en la Tabla 7.

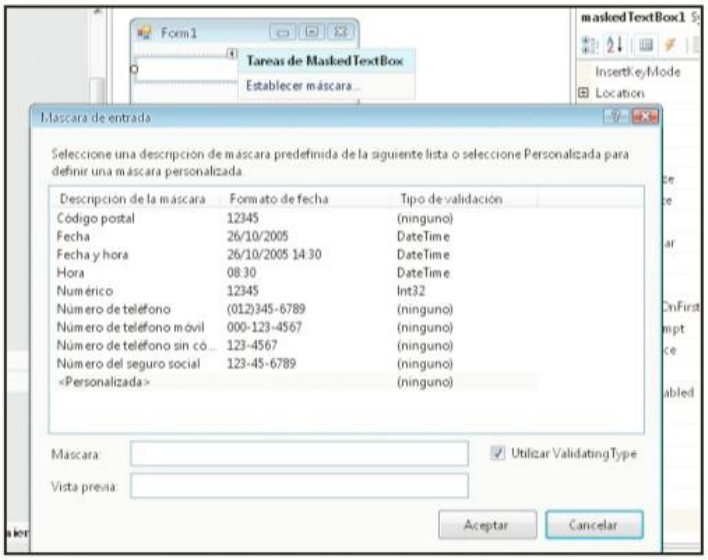

FIGURA 013 | Selección de máscaras. El control MaskedTextbox tiene máscaras por defecto que pueden utilizarse seleccionándolas desde el SmartTag o eligiendo la propiedad Mask en la ventana de Propiedades.

Además de las propiedades Mask y Text, otras muy útiles son:

- TextMaskFormat: Indica cómo deben obtenerse los valores de la propiedad Text del control (si incluye los literales, sólo el texto, etc.).
- BeepOnError: Indica si el control emitirá un sonido al ingresar un carácter que no sea válido.

El control MaskedTextBox se comporta de manera similar al TextBox, por lo que sus eventos son parecidos. Un evento particular es **MaskInputRejected**, que se produce cuando el usuario ingresa un carácter no válido. En este evento el parámetro es del tipo **MaskInputRejectedEventArgs**, y contiene la posición del texto donde ocurrió el error y su tipo (las propiedades **Position** y **RejectionHint** del parámetro, respectivamente). Con esta información podemos indicar al usuario lo que se espera que ingrese y cuál fue el error, si es necesario. En el siguiente ejemplo, capturamos el evento e informamos al usuario en un Label cuál fue el error:

Tabla 6 | Principales propiedades del control TextBox

Propiedad	Descripción
Text	Texto por mostrar en el control. Esta propiedad es la más utilizada. De aquí se obtiene lo que el usuario ha ingresado.
TextAlign	Determina cómo se posiciona el texto dentro del control.
MaxLength	Determina cuántos caracteres se permiten ingresar en el control. Útil para validar la longitud de los datos que se pondrán.
CharacterCasing	Indica cómo debe ser procesado el texto que se ingresa. Puede ser convertido en mayúsculas, minúsculas o dejarlo sin cambios, modificando esta propiedad.
PasswordChar	Indica qué caácter se utilizará para mostrar cuando se lo utilice para el ingreso de contraseñas.
MultiLine	Indica si el control puede aceptar varias líneas de texto.
ScrollBars	Indica qué barras de scroll son visibles en el modo MultiLine del control.
ReadOnly	Indica si el usuario puede modificar el texto o no.

VB.NET:

```
Private Sub txtMaskedTextBox_MaskInput
Rejected(ByVal sender As System.Object, ByVal
e As
System.Windows.Forms.MaskInputRejectedEventArgs)
Handles txtMaskedTextBox.MaskInputRejected

        lblError.Text = String.Format
        ("Error: {0}", e.RejectionHint.
        ToString)

End Sub
```

C#:

```
private void txtMaskedTextBox_MaskInput
Rejected(object sender, MaskInputRejected
EventArgs e)
    {
      lblError.Text = String.Format("Error:
      {0}", e.RejectionHint.ToString());
    }
```

CheckBox y RadioButton

Además de ingresar datos, es muy común pedirle al usuario que seleccione una opción entre varias. Por ejemplo, podemos precisar que escoja entre "Hombre" y "Mujer" para almacenar el sexo con los datos de una persona. Los CheckBox y RadioButton son los controles más básicos para realizar esta selección. El primero permite elegir opciones independientes, mientras que segundo admite opciones mutuamente excluyentes. Para que las opciones de los RadioButton sean excluyentes entre sí deben estar agrupadas en un control contenedor (el formulario, un Panel, un GroupBox, por ejemplo, controles que veremos más adelante en este capítulo) para que funcionen en conjunto.

> El TextBox es el control más básico y más usado para el ingreso de datos por parte del usuario.

ⓘ Controles nuevos

La versión 2.0 del framework .NET incorporó una cantidad interesante de nuevos controles que nos facilitan la tarea de crear formularios atractivos y con un buen grado de "usabilidad", sin tener que escribir mucha lógica ni recurrir a controles de terceros. El control MaskedTextBox es uno de los que no existían en las versiones anteriores de .NET.

Tabla 7 | Caracteres para la máscara del MaskedTextBox

Caracter	Significado
0	Indica un dígito numérico obligatorio del 0 al 9.
9	Indica un dígito numérico o un espacio. Es opcional.
L	Indica una letra obligatoria de la A a la Z.
?	Indica una letra opcional de la A a la Z.
,	Indica un separador de miles.
:	Indica un separador de hora.
/	Indica un separador de fecha.
$	Indica un símbolo de moneda.

MaskedTextBox es útil para evitar que el usuario ingrese caracteres que no corresponden.

Como estos controles permiten que el usuario marque cuál de las opciones escoge, la propiedad más utilizada es **Checked,** que es de tipo Boolean y permite saber si la opción está seleccionada o no. Otra propiedad necesaria es **Text**, que nos permitirá configurar el texto de la opción.
Entre los eventos, el más utilizado en ambos controles es **CheckedChanged**. Al marcar o desmarcar el CheckBox o el RadioButton, se produce este evento para indicar el cambio de estado del control.
Suponiendo que tenemos un CheckBox que indica si deseamos incluir datos de domicilio, por ejemplo, podemos utilizarlo para habilitar o deshabilitar esa sección del formulario:

VB.NET:

```
Private Sub chkDomicilio_CheckedChanged(ByVal
sender As System.Object, ByVal e As
System.EventArgs) Handles
chkDomicilio.CheckedChanged

        'Si el checkbox esta marcado,
        habilita la seccion de domicilios
        'sino la deshabilita

        grpDatosDomicilio.Enabled =
        chkDomicilio.Checked

End Sub
```

C#:

```
private void chkDomicilio_CheckedChanged
(object sender, EventArgs e)
{
    /*Si el checkbox esta marcado, habilita
    la seccion de domicilios
     sino la deshabilita*/

     grpDatosDomicilio.Enabled =
     chkDomicilio.Checked;

}
```

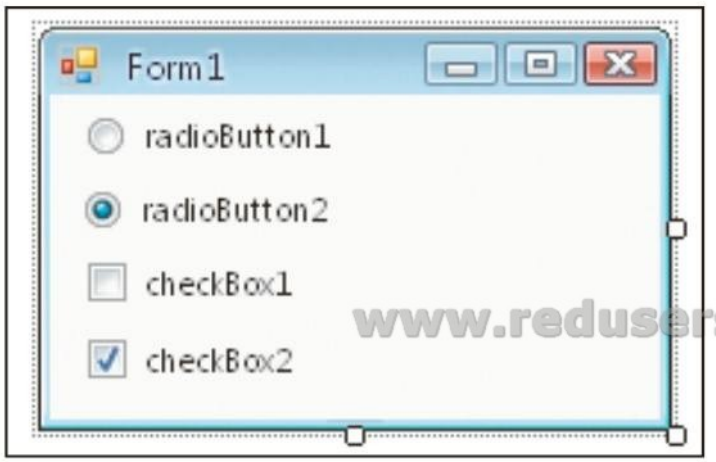

FIGURA 014 | Estos controles permiten seleccionar una opción entre varias posibles dentro del formulario.

DateTimePicker y MonthCalendar

Estos dos controles son muy útiles para el manejo de fechas y horas, soportan múltiples formatos de fecha y realizan automáticamente la validación del dato ingresado (por ejemplo, validan que el usuario no escriba "30/02/2008"). **DateTimePicker** presenta una lista desplegable para permitir la selección de las fechas o escribirlas a la manera de un TextBox. **MonthCalendar** permite la visualización de meses completos y la selección de rangos de fechas como principal característica. DateTimePicker es útil para la selección de una fecha u hora. En cambio, el MonthCalendar está pensado para realizar aplicaciones tipo agenda en las que se requiere una visualización y selección de varias fechas.

Entre las propiedades más importantes de DateTimePicker podemos citar:

- Value: Permite asignar u obtener el valor seleccionado del calendario.
- MinDate: Fecha mínima que puede elegir el usuario.
- MaxDate: Fecha máxima que puede elegir el usuario.
- Format: Permite especificar cuál es el formato de visualización de la fecha en el cuadro de texto del control.

Por el lado de MonthCalendar, además de las fechas mínimas y máximas, una propiedad muy interesante es CalendarDimensions, que indica cuántos meses visualiza el control, tanto horizontal como verticalmente. Así, podemos hacer un gran formulario para todo un año, con un control MonthCalendar de cuatro meses de ancho por tres de alto.
En cuanto a los eventos, ambos controles tienen eventos que indican cuándo el valor seleccionado ha cambiado. El evento en el DateTime Picker se denomina **ValueChanged**, mientras que en el MonthCalendar es **DateChanged**.

PictureBox

El control PictureBox permite visualizar una imagen en el formulario. Es bastante sencillo de utilizar, y sirve para mejorar la interfaz del usuario o mostrar datos relacionados con imágenes (fotografías, mapas, etc.).
Sus dos propiedades más importantes son:

- Image: Permite asignar u obtener la imagen que muestra el control.
- SizeMode: Indica cómo se comportan el control y la imagen en cuanto al tamaño; es decir, si el tamaño del control se adapta al de la imagen, o si el de la imagen se adapta al del control.

ListBox y ComboBox

Como vimos unos párrafos atrás, los controles CheckBox y RadioButton permiten elegir entre varias opciones; sin embargo, cuando la cantidad es grande (por ejemplo, la lista de países del mundo), estos controles ocuparían mucho espacio en el formulario. Los controles ListBox y ComboBox permiten hacer selecciones, pero ocupan menos espacio en la ventana, por lo cual resultan excelentes si tenemos muchas opciones.

> Los CheckBox y RadioButton son los controles más básicos para permitir la selección de opciones a los usuarios.

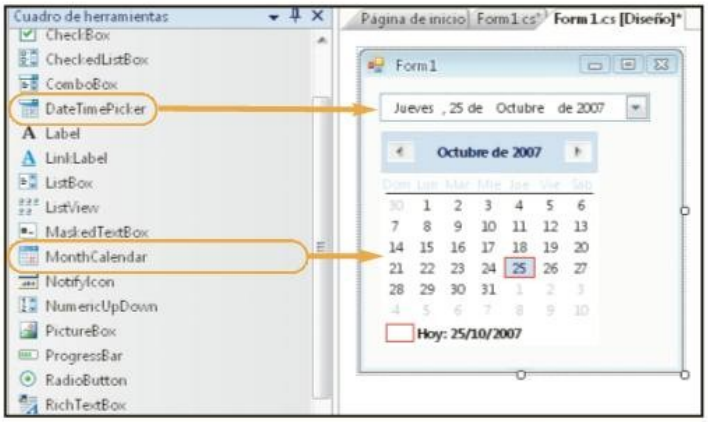

FIGURA 015 | Controles DateTimePicker y MonthCalendar en la Toolbox y representación visual en el formulario.

§ La propiedad Enabled

Una propiedad que usaremos muchas veces en nuestros desarrollos, y que pertenece a la mayoría de los controles, es Enabled. Su función es indicar si el control está habilitado o no. Al estar deshabilitado, el usuario no puede interactuar con él. Por ejemplo, un botón deshabilitado no responde al clic del mouse, y un TextBox deshabilitado no permite modificar su texto.

MonthCalendar se utiliza para aplicaciones en la que se necesita una visualización de fechas.

Los controles ListBox y ComboBox son los más simples de una categoría en la que el acceso a la selección del valor y las opciones que se le dan al usuario están basados en colecciones. Los controles que hemos nombrado anteriormente obtienen su valor a través de una única propiedad (Text, Checked, Value, etc.). Estos nuevos controles tienen como propiedad fundamental una o varias colecciones que administran los datos por presentar al usuario y las que ha seleccionado. El ComboBox presenta una lista desplegable para la selección de los datos, y es más compacto en la interfaz; el ListBox presenta una lista con todos los ítem y barras de desplazamiento para cuando la cantidad de elementos excede a los que se pueden mostrar. Además, el ComboBox puede funcionar como un TextBox, al permitir que el usuario ingrese un texto libre, cuando ninguno de los elementos de la lista es el deseado.
La Tabla 8 muestra las principales propiedades del ComboBox, y en la Tabla 9 vemos algunas de las propiedades del ListBox.

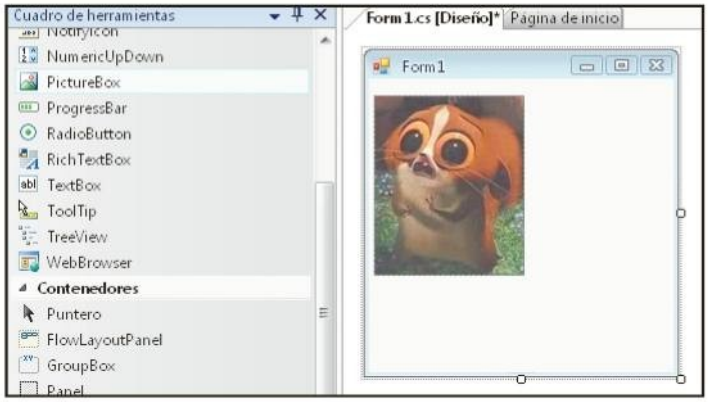

FIGURA 016 | Control PictureBox en la Toolbox y representación visual en el formulario.

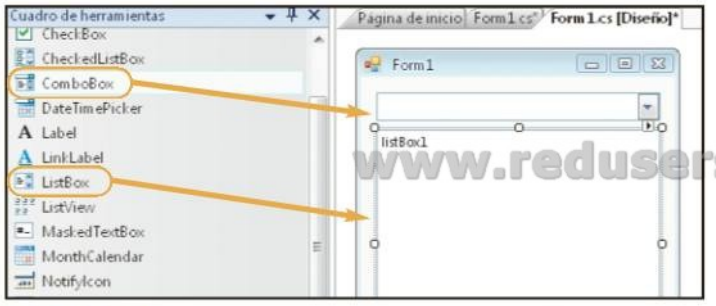

FIGURA 017 | Controles ComboBox y ListBox en la Toolbox y representación visual en el formulario.

Eventos: Ambos controles tienen como evento más utilizado el **SelectedIndex Changed**. Este evento se produce cuando el usuario está "eligiendo" alguna de las opciones mostradas por el control y va seleccionando alternativamente una u otra (cambia la selección). El ComboBox tiene, además, otro evento que se dispara luego de que el usuario eligió una determinada opción, para evitar así el constante cambio de índice sin selección definitiva. Este evento es **SelectionChange Commited** y es el que deberíamos utilizar en vez de **SeletedIndexChanged**. En el siguiente ejemplo se carga un combo con los valores de categorías de alimentos, y un segundo con la lista de éstos al elegir alguna categoría indicada por el anterior:

VB.NET:

```
Private Sub Form1_Load(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
Me.Load
    'Cargamos items en la coleccion
    cmbCategorias.Items.AddRange(New String()
    {"Carnes", "Vegetales", "Frutas"})
End Sub

Private Sub cmbCategorias_SelectionChange
Committed(ByVal sender As Object, ByVal e As
System.EventArgs) Handles cmbCategorias.
SelectionChangeCommitted
```

```
    'Eliminamos datos previos

    If cmbAlimentos.Items.Count > 0 Then
    cmbAlimentos.Items.Clear()
    Select Case cmbCategorias.SelectedItem

      Case "Carnes"
        cmbAlimentos.Items.AddRange(New
        String() {"Vaca", "Cerdo", "Pollo"})

      Case "Vegetales"
        cmbAlimentos.Items.AddRange(New
        String() {"Lechuga", "Tomate"})

      Case "Frutas"
        cmbAlimentos.Items.AddRange(New
        String() {"Manzana", "Naranja",
        "Banana"})
    End Select
    cmbAlimentos.SelectedIndex = 0
End Sub
```

C#:

```
private void Form1_Load(object sender,
EventArgs e)
{
 //Cargamos items en la coleccion
 cmbCategorias.Items.AddRange(new string[] {
 "Carnes", "Vegetales", "Frutas" });
}
```

ListBox y ComboBox permiten hacer selecciones, pero ocupan menos espacio en el formulario.

¿ComboBox o ListBox?

Muchas veces nos encontraremos ante la necesidad de decidir si utilizar un ComboBox o un ListBox. Para ayudarnos a elegir, podemos formularnos estas preguntas: ¿Cuánto espacio tengo en el formulario? ¿Voy a permitir escribir cuando el valor deseado no está en la lista? ¿Voy a permitir seleccionar más de un elemento? Con las respuestas a estas preguntas y lo aprendido en este capítulo, podremos decidir qué control utilizar.

Tabla 8 | Principales propiedades del control ComboBox

Propiedad	Descripción
DropDownStyle	Indica el estilo de selección del usuario. La opción más utilizada es DropDownList. De esta manera, el usuario puede seleccionar un elemento de la lista únicamente.
MaxDropDownItems	Indica cuántos ítem muestra la lista desplegable a la vez.
Sorted	Determina si los ítem serán ordenados por el control para su visualización.
Items	Colección que almacena los elementos que el usuario puede seleccionar. Puede contener cualquier elemento, ya que el tipo de dato que acepta es un Object.
SelectedIndex	Permite obtener o indicar cuál es el elemento de la colección de ítem que está seleccionado a través de su índice.
SelectedItem	Permite obtener o indicar el elemento seleccionado. El tipo de dato de esta propiedad es Object, y en ejecución corresponde al tipo de dato de los elementos que contiene la propiedad Ítems.

El evento más utilizado, tanto en el ListBox como en el ComboBox, es SelectedIndexChanged.

⊛ ComboBox, ListBox y colecciones

Los controles que basan su selección de valores en una colección de ítem son bastante particulares. De hecho, dicha colección, en su mayoría, acepta un Object como ítem para visualizar, con lo que podemos ingresar cualquier cosa que queramos. Esta característica es de suma utilidad, dado que los controles tratan de visualizar cualquier objeto a la manera de un String, llamando el método ToString de cada uno ellos para verlo.

```
private void cmbCategorias_Selection
ChangeCommitted(object sender, EventArgs e)
{
    //Eliminamos datos previos
    if (cmbAlimentos.Items.Count > 0)
    cmbAlimentos.Items.Clear();
    switch (cmbCategorias.SelectedItem)
    {

     case "Carnes":
        cmbAlimentos.Items.AddRange(new
        string[] { "Vaca", "Cerdo", "Pollo" });
        break;

     case "Vegetales":
        cmbAlimentos.Items.AddRange(new
        string[] { "Lechuga", "Tomate" });
        break;

     case "Frutas":
        cmbAlimentos.Items.AddRange(new
        string[] { "Manzana", "Naranja",
```

Tabla 9 | Principales propiedades del ListBox

Propiedad	Descripción
Items	Colección que almacena los elementos que el usuario puede seleccionar. Puede contener cualquier elemento, ya que el tipo de dato que acepta es un Object.
Sorted	Determina si los ítem serán ordenados por el control para su visualización.
SelectionMode	Indica la manera en que el usuario puede seleccionar los elementos. A diferencia del ComboBox, permite selección múltiple (el usuario puede seleccionar más de un elemento, manteniendo presionada la tecla <Ctrl>).
MultiColumn	Permite visualizar los ítem en columnas.
HorizontalScrollBar	Visualiza una ScrollBar cuando los ítem en columnas no son completamente visibles.
SelectedIndex	Permite obtener o indicar cuál es el elemento de la colección de ítem que está seleccionado a través de su índice. En caso de que no haya ningún elemento seleccionado, el valor de esta propiedad es -1.
SelectedItem	Permite obtener o indicar el elemento seleccionado.
SelectedIndices	Contiene los índices de los elementos seleccionados en el ListBox. Debe utilizarse cuando está permitida la selección múltiple.
SelectedItems	Contiene los elementos seleccionados por el usuario. Debe utilizarse cuando está permitida la selección múltiple.

```
            "Banana" });
            break;
        }
        cmbAlimentos.SelectedIndex = 0;

    }
```

ListView

El control ListView, en conjunto con TreeView, es uno de los más utilizados para crear interfaces similares a un explorador de Windows. Es un control muy versátil, ya que puede utilizarse para mostrar variada información, tanto simulando una grilla, mostrando texto e iconos agrupados por categorías, permitiendo ordenamiento, etc. Los detalles de uso de este control exceden un poco el alcance de este curso, por lo que veremos algunos ejemplos y, si el lector quiere profundizar aún más, puede consultar http://msdn2.microsoft.com/es-es/library/system.windows.forms.listview(VS.80).aspx.
La Tabla 10 muestra algunas de las propiedades más importantes de ListView.
En lo que respecta a los eventos, los dos más utilizados son **ItemSelectionChanged** e **ItemCheck**. El primero ocurre al cambiar la selección de un ítem. Se produce tanto cuando se deselecciona como cuando se selecciona un elemento de la lista. Para conocer qué ítem cambia y a qué estado, se deben consultar las propiedades del parámetro *e* del evento, que es de tipo **ListViewItemSelectionChangedEventArgs**. En él las propiedades **IsSelected**, **Item** e **ItemIndex** nos indicarán sobre quién se produjo el evento y en qué estado quedó finalmente. El segundo evento, en cambio, sólo se produce cuando la propiedad CheckBoxes está en True, y se marca o se desmarca el checkbox del ítem. Al igual que en el otro evento, el parámetro es del tipo **ItemCheckEventArgs**, y contiene la información para determinar el estado del checkbox y en qué ítem sucedió el evento.

Dado que un ítem marcado (checked) no necesariamente puede estar seleccionado, existen dos colecciones que mantienen por separado los ítem que están seleccionados y los que están marcados (checked): **SelectedItems** y **CheckedItems**, respectivamente. Con estas colecciones nos evitamos recorrer todo el ListView para obtener dichos elementos.
En la página siguiente, veremos un ejemplo de carga de elementos de un ListView. Se asume que su nombre es ListViewCtl, está asignada la propiedad SmallImageList con los iconos por visualizar, la vista de Details y GridLines en True.

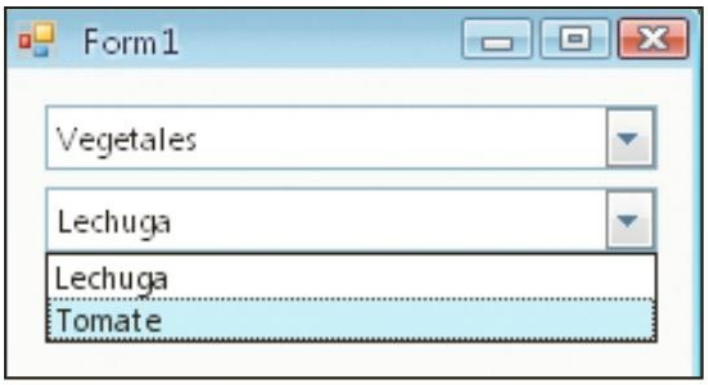

FIGURA 018 | Resultado de la ejecución del código anterior. Al seleccionar una categoría, se llena el combo con los alimentos asociados en el segundo.

⊛ Columnas, ítem y subítem

Al principio, puede resultar difícil entender cómo se relacionan las columnas, los ítem y los subítem. El tema es así: las columnas definen los encabezados de cada una, los ítem corresponden a cada elemento que se muestra (cada fila en modo detalle) y su texto se asocia con la primera columna; finalmente, los subítem corresponden a "subelementos" del ítem, y se corresponden con la segunda columna en adelante.

El control ListView es muy versátil, ya que puede utilizarse para mostrar información de todo tipo.

VB.NET:

```
Private Sub Form1_Load(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
Me.Load
        Dim oItem As ListViewItem

        'carga un pais y asigna el icono a
        visualizar
        oItem = ListViewCtl.Items.Add
        ("Argentina", 0)
        'agrega los otros items de la fila
        oItem.SubItems.Add("Buenos Aires")
        oItem.SubItems.Add("América")

        'repetimos la operación por cada
        fila
        oItem = ListViewCtl.Items.
        Add("Brasil", 1)
        oItem.SubItems.Add("Brasilia")
        oItem.SubItems.Add("América")

        oItem = ListViewCtl.Items.
        Add("Canadá", 2)
        oItem.SubItems.Add("Ottawa")
        oItem.SubItems.Add("América")

End Sub
```

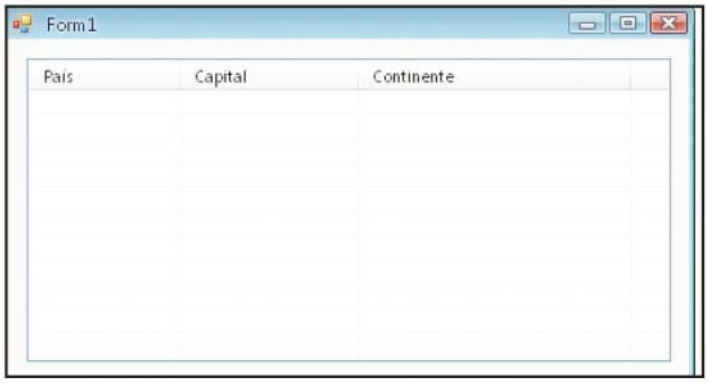

FIGURA 019 | Control ListView con algunas columnas agregadas y visualización en vista de detalle.

FIGURA 020 | Resultado de la ejecución del código de ejemplo. Al cargarse el formulario, se llena el ListView con algunas filas y se asocian imágenes a él.

C#:

```
private void Form1_Load(object sender,
EventArgs e)
{
      ListViewItem oItem;

      //carga un pais y asigna el icono a
      visualizar
      oItem = ListViewCtl.Items.Add
      ("Argentina", 0);
      //agrega los otros items de la fila
      oItem.SubItems.Add("Buenos Aires");
      oItem.SubItems.Add("América");

      //repetimos la operación por cada fila
      oItem = ListViewCtl.Items.Add("Brasil",
      1);
      oItem.SubItems.Add("Brasilia");
      oItem.SubItems.Add("América");

      oItem = ListViewCtl.Items.Add("Canadá",
      2);
      oItem.SubItems.Add("Ottawa");
      oItem.SubItems.Add("América");
}
```

TreeView

El control TreeView permite visualizar jerarquías de datos, como estructuras de directorios, organigramas, maestros-detalle, etc. Se comporta como un árbol en el que cada elemento se denomina Nodo y es del tipo TreeNode. El nodo tiene una propiedad llamada Text, que se muestra en el árbol y puede tener asociada una imagen.

Al permitir mostrar datos jerárquicos, el control provee funcionalidad para expandir las ramas del árbol, ya sea de manera interactiva (por acción del usuario) o mediante código.

El TreeView se utiliza, en general, acompañado de un ListView, para crear aplicaciones con interfaces similares a un Explorador de Windows.

Además, permite asociar casillas tipo CheckBox para seleccionar elementos individuales o ramas enteras del árbol.
Entre las principales propiedades del control TreeView, podemos mencionar:

Tabla 10 | Principales propiedades del control ListView

Propiedad	Descripción
Columns	Colección de objetos ColumnHeader que definen cuántos elementos contendrán las filas.
Items	Colección de objetos ListViewItem que definen las filas de datos del ListView.
AllowColumnReorder	Indica si el usuario puede mover las columnas para reordenarlas en la vista de Details.
FullRowSelect	Indica si la selección del ítem debe marcar toda la fila o sólo la primera celda.
CheckBoxes	Permite visualizar CheckBoxes en cada fila del ListView.
GridLines	Permite visualizar o no las líneas de división de filas y columnas simulando una grilla.
Groups	Colección de grupos en los que pueden agruparse los ítem.
LabelEdit	Permite modificar el texto de un ListViewItem directamente en el control.
MultiSelect	Permite o no la selección múltiple de ítem.
SmallImageList	Permite asociar un control ImageList que contiene las imágenes por utilizar en el ListView en una vista con iconos pequeños (16x16).
LargeImageList	Permite asociar un control ImageList que contiene las imágenes por utilizar en el ListView en una vista con iconos grandes (32x32).
StateImageList	Permite asociar un control ImageList que contiene las imágenes por utilizar en el ListView en cada ítem para mostrar un estado personalizado.
SelectedIndices	Colección que mantiene los índices de los ítem seleccionados.
SelectedItems	Colección que mantiene los ítem seleccionados.
CheckedIndices	Colección que mantiene los índices de los ítem marcados (checked) cuando la propiedad CheckBoxes es igual a True.
CheckedItems	Colección que mantiene los ítem marcados (checked) cuando la propiedad CheckBoxes es igual a True.
View	Permite cambiar la vista del ListView. Con esta opción tenemos la posibilidad de visualizar los elementos como una grilla con texto e iconos, como una lista de iconos (grandes y pequeños), etc. (tal como sucede con el menú Vistas del explorador de carpetas de Windows XP).

- SelectedNode: Contiene el nodo seleccionado.
- PathSeparator: Carácter utilizado como separador en la propiedad FullPath del nodo inspeccionado para poder identificarlo.
- FullRowSelect: Indica si selecciona sólo el nodo o toda la línea del ítem.
- ShowRootLines: Obtiene o establece un valor que indica si se dibujan líneas entre los nodos de árbol situados en la raíz de la vista de éste.
- ShowPlusMinus: Obtiene o establece un valor que indica si se dibujan líneas entre los nodos de árbol situados en la raíz de la vista de éste.
- ImageList: Permite asociar un control ImageList que contiene las imágenes por utilizar en los nodos del TreeView.
- StateImageList: Permite asociar un control ImageList que contiene las imágenes por utilizar en el Treeview, para mostrar un estado personalizado.

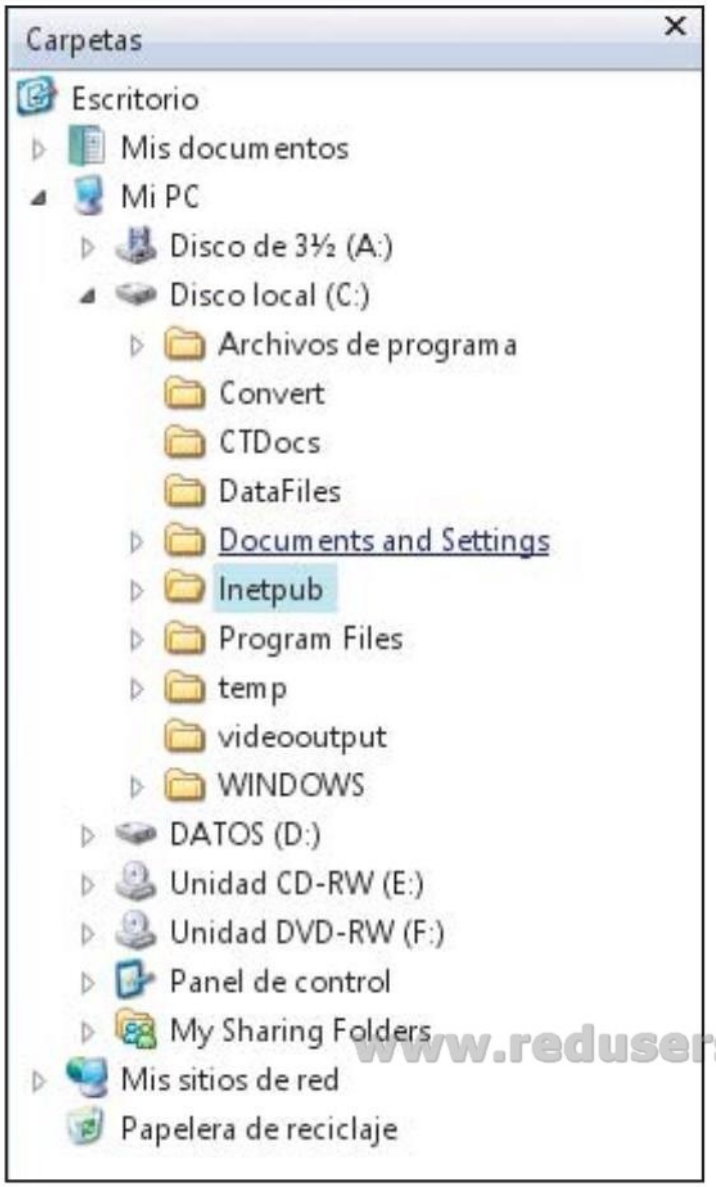

FIGURA 021 | El control TreeView es ideal para mostrar datos jerárquicos, como las estructuras de directorios de un disco.

En cuando a los eventos, el más importante es **AfterSelect**, que se produce cuando un nodo es seleccionado. El parámetro es del tipo **TreeViewEventArgs**, y contiene en sus propiedades la acción que originó el evento y qué nodo fue seleccionado (Action y Node, respectivamente). Otros eventos útiles son **AfterExpand** y **AfterCollapse**, que ocurren al expandir o colapsar los nodos del árbol; y **NodeMouseClick**, que se produce al seleccionar un nodo con el mouse.

Para entender el funcionamiento y la utilidad del control TreeView, veremos un pequeño ejemplo de carga de elementos, en el que los nodos raíz son continentes, y los nodos hijos de primer nivel son países (podríamos tener un segundo nivel con provincias o estados, y otro nivel más con ciudades). Asume que su nombre es TreeViewCtl y está asignada la propiedad ImageList con los iconos por visualizar:

VB.NET:

```
Private Sub Form1_Load(ByVal sender As
Object, ByVal e As System.EventArgs)
Handles Me.Load
    Dim oRaiz As TreeNode
    Dim oPais As TreeNode

    'cargo continente: America
    oRaiz = TreeViewCtl.Nodes.
    Add("América")
```

```
        'pongo el mismo icono para imagen y
        seleccionado
        oRaiz.ImageKey = "folder"
        oRaiz.SelectedImageKey = "folder"

        'cargo paises del continente
        Americano
        oPais = oRaiz.Nodes.Add("Argentina")
        oPais.ImageKey = "argentina"
        oPais.SelectedImageKey = "argentina"

        oPais = oRaiz.Nodes.Add("Brasil")
        oPais.ImageKey = "brasil"
        oPais.SelectedImageKey = "brasil"

        oPais = oRaiz.Nodes.Add("Canadá")
        oPais.ImageKey = "canada"
        oPais.SelectedImageKey = "canada"

        'cargo continente: Europa
        oRaiz = TreeViewCtl.Nodes.
        Add("Europa")

        'pongo el mismo icono para imagen y
        seleccionado
        oRaiz.ImageKey = "folder"
        oRaiz.SelectedImageKey = "folder"

        'cargo paises del continente Europeo
        oPais = oRaiz.Nodes.Add("Francia")
        oPais.ImageKey = "francia"
        oPais.SelectedImageKey = "francia"

        oPais = oRaiz.Nodes.Add("Grecia")
        oPais.ImageKey = "grecia"
        oPais.SelectedImageKey = "grecia"

        'cargo continente: Asia
        oRaiz = TreeViewCtl.Nodes.
        Add("Asia")
        'pongo el mismo icono para imagen y
        seleccionado
        oRaiz.ImageKey = "folder"
        oRaiz.SelectedImageKey = "folder"

        'cargo paises del continente
        Asiatico

        oPais = oRaiz.Nodes.Add("Japón")
        oPais.ImageKey = "japon"
        oPais.SelectedImageKey = "japon"

        oPais = oRaiz.Nodes.Add("Corea")
        oPais.ImageKey = "corea"
        oPais.SelectedImageKey = "corea"

        'Expande todos los items y los
        visualiza
        TreeViewCtl.ExpandAll()

    End Sub
```

TreeView permite asociar casillas tipo CheckBox para seleccionar elementos.

! El TreeView como estructura jerárquica

El TreeView tiene una colección de nodos (llamada Nodes) que funcionan como nodos raíz, y cada uno tiene, a su vez, su propia colección Nodes (que podemos llamar nodos hijos). De esa manera, permite armar la estructura jerárquica y recursiva. Por ejemplo, podemos asociar los discos de la PC con los nodos raíz del árbol; luego, las carpetas de la raíz de cada disco con cada uno de los nodos correspondientes; las subcarpetas de cada carpeta como nodos hijos de cada nodo, y así sucesivamente.

C#:

```
private void Form1_Load(object
sender, EventArgs e)
{
    TreeNode oRaiz;
    TreeNode oPais;

    //cargo continente: America
    oRaiz = TreeViewCtl.Nodes.
    Add("América");

    //pongo el mismo icono para
    imagen y seleccionado
    oRaiz.ImageKey = "folder";
    oRaiz.SelectedImageKey =
    "folder";

    //cargo paises del continente
    Americano
    oPais = oRaiz.Nodes.Add
    ("Argentina");
    oPais.ImageKey = "argentina";
    oPais.SelectedImageKey =
    "argentina";

    oPais = oRaiz.Nodes.Add
    ("Brasil");
    oPais.ImageKey = "brasil";
    oPais.SelectedImageKey =
    "brasil";

    oPais = oRaiz.Nodes.Add
    ("Canadá");
    oPais.ImageKey = "canada";
    oPais.SelectedImageKey = "canada";

    //cargo continente: Europa
    oRaiz = TreeViewCtl.Nodes.Add
    ("Europa");

    //pongo el mismo icono para
    imagen y seleccionado
    oRaiz.ImageKey = "folder";
    oRaiz.SelectedImageKey =
    "folder";

    //cargo paises del continente
    Europeo
    oPais = oRaiz.Nodes.Add
    ("Francia");
    oPais.ImageKey = "francia";
    oPais.SelectedImageKey =
```

FIGURA 022 | Nuestro ejemplo en ejecución. Al cargarse el formulario, se llena el TreeView con algunos nodos e imágenes asociadas a los textos.

ⓘ Averiguando qué nodo está bajo el puntero

Si bien en la mayoría de los casos con los eventos mencionados anteriormente será suficiente para trabajar con el control TreeView, pueden darse situaciones en las que necesitemos capturar el evento MouseMove y determinar cuál de los nodos está en la posición del puntero, si es que hay alguno. Para hacerlo, podemos utilizar el método HitTest, que recibe una coordenada y nos devuelve un objeto de tipo TreeViewHitTestInfo, que entre otras cosas posee una propiedad con el TreeNode que está en esa coordenada.

```
            "francia";

            oPais = oRaiz.Nodes.Add
            ("Grecia");
            oPais.ImageKey = "grecia";
            oPais.SelectedImageKey = "grecia";

            //cargo continente: Asia
            oRaiz = TreeViewCtl.Nodes.Add
            ("Asia");

            //pongo el mismo icono para
            imagen y seleccionado
            oRaiz.ImageKey = "folder";
            oRaiz.SelectedImageKey = "folder";

            //cargo paises del continente
            Asiatico
            oPais = oRaiz.Nodes.Add
            ("Japón");
            oPais.ImageKey = "japon";
            oPais.SelectedImageKey = "japon";

            oPais = oRaiz.Nodes.Add
            ("Corea");
            oPais.ImageKey = "corea";
            oPais.SelectedImageKey = "corea";

            //Expande todos los items y los
            visualiza
            TreeViewCtl.ExpandAll();
        }
```

Personalizar el dibujado

Al igual que otros controles, TreeView permite personalizar la forma en que se dibujan tanto los elementos en particular como el control en general. Utilizando la propiedad DrawMode y el evento DrawNode, podemos codificar nosotros el método para dibujar los nodos, con lo cual será posible variar la fuente, los colores o lo que precisemos para cada nodo del árbol.

Controles contenedores

Hasta ahora hemos visto controles simples; es decir, cada control que colocamos en el formulario actúa de manera independiente, y debemos programarlo y manipularlo como tal. Sin embargo, existen controles que permiten la inclusión de otros controles dentro de ellos, o sea que actúan como contenedores. Estos controles heredan de la clase ContainerControl, que proporciona una colección mediante la cual podemos tener acceso a los controles alojados en él. Los controles contenedores son útiles para agrupar la información que se muestra al usuario y proporcionar una estética más profesional a nuestros desarrollos.
A continuación, veremos los principales controles contenedores de que disponemos.

GroupBox

Éste es un control contenedor útil para sectorizar la información en un formulario. También se utiliza con los RadioButton para permitir que sean mutuamente excluyentes. Se caracteriza por tener un borde y un texto en la parte superior (determinado por la propiedad Text), que podemos usar para lograr una interfaz más clara e intuitiva.

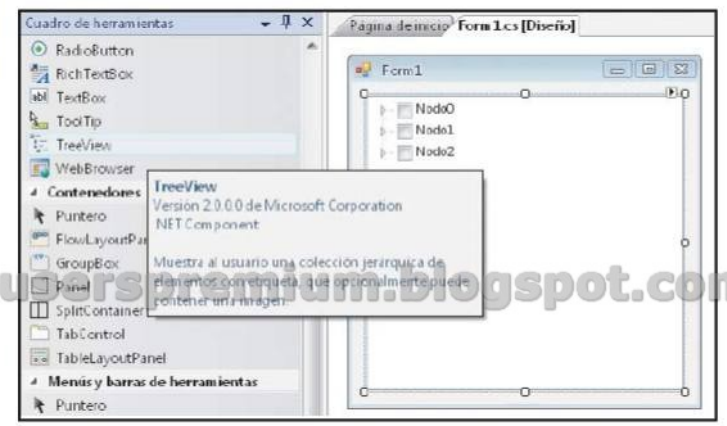

FIGURA 023 | Control TreeView en la Toolbox y representación visual en el formulario con ítem de prueba.

Mediante un menú contextual, Visual Studio nos permite agregar solapas al TabControl en tiempo de diseño.

Panel

El Panel es un control contenedor que tiene la particularidad de poseer bordes transparentes, por lo que permite sectorizar el formulario en regiones sin mostrar bordes de unión. Se utiliza para organizar los controles dentro del formulario y presenta mucha más flexibilidad que un GroupBox. Permite hacer un uso extensivo de las propiedades Dock y Anchor para su posicionamiento y el de los controles contenidos en él. A diferencia de GroupBox, no posee una propiedad Text para indicar el contenido del control, y debe realizarse con un Label o alguno similar. Al ser un control contenedor, puede tener ScrollBars para mostrar su contenido si la parte visible del control no alcanza.

FIGURA 024 | En la sección Contenedores del Cuadro de Herramientas tenemos todos los que vienen con .NET.

Entre las propiedades más destacadas del control Panel, podemos mencionar:

- BorderStyle: Permite visualizar o no el borde del control para mejorar su aspecto. Por defecto, no lo muestra.
- AutoScroll: Habilita o no las barras de desplazamiento en el control para mostrar el contenido si el espacio visual no alcanza.

TabControl

El control TabControl también es contenedor, pero los controles pueden organizarse en diferentes solapas o TabPages. Cada una de ellas es configurable, y puede tener controles y aspectos diferentes. Es muy útil cuando disponemos de poco espacio en un formulario y debemos mostrar muchos datos. Cada solapa puede representar una actividad o conjunto de datos particulares. Por ejemplo, si estamos desarrollando un formulario para cargar datos de clientes, podemos colocar los controles para

Tabla 11 | Principales propiedades del TabControl

Propiedad	Descripción
TabPages	Es una colección de solapas (TabPages), cada una de las cuales contendrá diferentes controles. Pueden modificarse el texto y la imagen que se mostrará, entre otras opciones.
HotTrack	Indica si el Tab cambia visualmente al pasar el mouse por encima.
Alignment	Indica en qué posición aparecerán las solapas en el control.

los datos generales en una solapa, los datos de facturación en otra y los de contactos en otra. De esta manera, el usuario tendrá una pantalla sencilla, pero con muchas opciones a la vez, y podrá seleccionar la solapa que desea ver o completar, concentrándose sólo en la parte del formulario que le interesa.
En la Tabla 11 vemos las principales propiedades del control TabControl.

Entre los eventos del TabControl, el más importante y útil es **Selected**, cuyo parámetro es de tipo **TabControlEventArgs** y nos informa la acción realizada (selección o deselección), qué solapa la llevó a cabo (**TabPage**) y el índice de la solapa dentro de la colección de TabPages (**TabPageIndex**).

Menús y toolbars

En la mayoría de las aplicaciones para Windows nos veremos ante la necesidad de crear menús y barras de herramientas para colocar las opciones de la aplicación y desarrollar interfaces más completas e intuitivas. Para lograrlo, .NET 2.0 incluye un conjunto de controles muy interesantes. Éstos soportan los estilos visuales de Windows XP y de Office 2003, permiten su movimiento y reorganización en

> El Panel es un control que se utiliza para organizar los controles dentro del formulario.

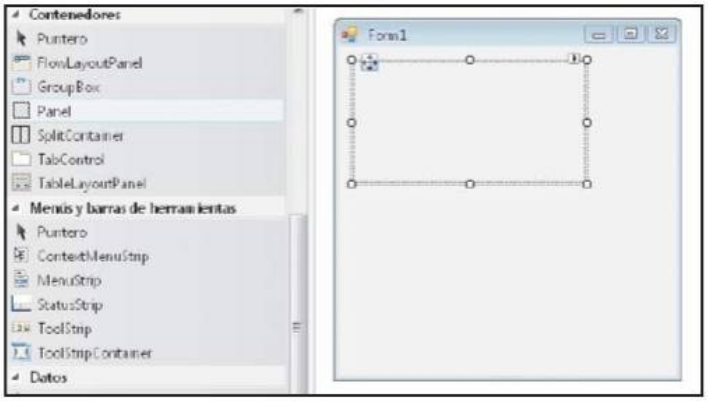

FIGURA 025 | Control Panel en la Toolbox y representación visual en el formulario.

Tabla 12 | Lista de controles Strip disponibles en .NET 2.0

Control	Descripción
ToolStripContainer	Este contenedor es necesario si deseamos darle al usuario la posibilidad de que reacomode las toolbars en todos los márgenes del formulario.
ToolStrip	Es una Toolbar propiamente dicha. Puede albergar botones, labels, texboxes, combos y cualquier otro control. Es la barra de herramienta típica.
MenuStrip	Permite crear los clásicos menús de las aplicaciones. Esta nueva versión permite el redibujado completo de cada ítem si es necesario e incorpora la posibilidad de asignarle imágenes a cada uno en particular (lo que no se podía hacer en .NET 1.1).
ContextMenuStrip	Es similar al MenuStrip, pero está diseñado para crear menús contextuales. No hace falta monitorizar los eventos del mouse para mostrar un menú, ya que cada control puede incorporar un menú contextual con sólo asignarle el ContextMenuStrip apropiado.
StatusStrip	Es la base de estado o StatusBar modernizada. Se comporta como una toolbar, al permitir adicionar controles personalizados si es necesario. Podemos ubicar, entre otros elementos, ProgressBar, labels, etc.

En la mayoría de las aplicaciones, nos veremos ante la necesidad de crear menús y herramientas.

tiempo de ejecución (el usuario puede cambiar las barras y los menús de posición, colocándolos, por ejemplo, contra el borde izquierdo o el inferior) y, en caso de que necesitemos algo especial, podemos incorporar controles personalizados e, incluso, tener el control del dibujado de cada ítem en particular.

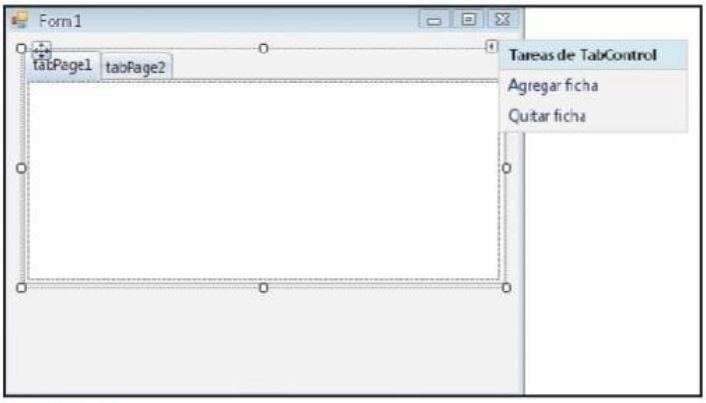

FIGURA 026 | El control TabControl permite organizar los controles en solapas o fichas.

Imágenes como recursos

Cuando asignamos una imagen a un control, el asistente de Visual Studio nos da la opción de importar una como recurso local o de utilizar un archivo de recursos. Siempre es preferible crear un archivo de recursos, por varios motivos: principalmente, por el tamaño del archivo ejecutable (que se verá reducido por no repetir los recursos en cada formulario) y también por la comodidad de tener todas las imágenes y los recursos en un mismo lugar.

A estos menús y toolbars se los conoce como controles "strip", ya que están basados en una clase en común llamada ToolStrip. Los controles strip con los cuales contamos son los que se mencionan en la Tabla 12.

Estos controles tienen muchas funcionalidades y son muy fáciles de utilizar. Debido a que tienen un modelo común, entender cómo funcionan es bastante sencillo una vez que aprendemos a usar uno. Los asistentes para la creación de menús y toolbars son muy simples e intuitivos de manejar, con lo cual el tiempo de aprendizaje se reduce al mínimo. Básicamente, todos los controles Strip están compuestos por una colección de ítem del tipo **ToolStripItem**. La mejor manera de entenderlos es "jugar" con cada uno de ellos y ver los resultados. Estos controles pueden encontrase en la categoría Menús y barras de herramientas, del Cuadro de Herramientas de Visual Studio. Cuando empecemos a "jugar" con esta familia de controles, deberemos tener presentes algunas propiedades de los objetos ToolStripItem, por ejemplo:

- Text: Contiene el texto que se visualizará en el ítem.
- TextAlign: Indica cómo debe alinearse el texto en el ítem.
- DisplayStyle: Indica cómo debe visualizarse el ítem (sólo texto, sólo imagen, imagen y texto, etc.).
- Image: Imagen por visualizar en el ítem.
- ImageAlign: Cómo se alineará la imagen en el ítem.
- TextImageRelation: Indica cómo debe ubicarse la imagen en relación con el texto del ítem (arriba, abajo, etc.).
- Alignment: Indica cómo se alinea el ítem (derecho o izquierda).

Además de estas propiedades –que usaremos en general en tiempo de diseño–, hay otras

relacionadas con el comportamiento visual de los controles Strip en tiempo de ejecución, para reflejar ciertos cambios en respuesta a las acciones del usuario. Por ejemplo, para mostrar una marca (check) en un comando de menú, debemos asignar **true** a la propiedad **Checked**, y si queremos que se haga automáticamente cuando el usuario hace Clic en el control, podemos darle el valor **true** a la propiedad **CheckOnClick.** También, como sucede con la mayoría de los controles, podemos utilizar la propiedad Enabled para deshabilitar el control y, así, evitar que el usuario haga clic en él y ejecute acciones no convenientes en el contexto actual de la aplicación.

Hasta aquí todo muy bien, pero ¿qué pasa cuando el usuario selecciona una opción del menú o un botón de la barra de herramientas? Bien, es entonces cuando tenemos que comenzar a programar, atrapando los eventos necesarios y escribiendo el código para ejecutar la acción seleccionada. El evento por monitorizar en la mayoría de los ítem es **Click**. Cada ítem se comporta como un **Button** y responde a los mismos eventos que él. En caso de que ubiquemos controles personalizados, deberemos monitorizar los eventos particulares de esa clase de control. ¿Qué sucede con las aplicaciones MDI? En aplicaciones de este tipo debemos tener algunas consideraciones particulares. Recordemos que las aplicaciones MDI son aquellas en las que hay una ventana principal y un conjunto de ventanas hijas, dependientes de la primera. En estas aplicaciones, el tipo de ventanas hijas podría ser diferente de la ventana MDI padre. Por ejemplo, la ventana MDI podría contener una planilla de cálculo, y la ventana hija, un gráfico estadístico. En tal caso, si queremos actualizar las opciones del menú principal (el de la ventana MDI) con las que contiene el menú de la ventana hija cuando ésta esté activa, debemos seguir estos pasos:

> Los controles de tipo strip tienen muchas funcionalidades y son muy fáciles de utilizar.

- Establecer la propiedad **AllowMerge** del menú principal en **true**.
- Establecer la propiedad **AllowMerge** del menú de la ventana hija en **true**.
- Establecer la propiedad **MergeAction** de cada uno de los elementos del menú de la ventana hija en **Append**.

Así, cuando la ventana hija sea maximizada, su menú se unirá al de la ventana madre, y se obtendrá una única barra de menú. Como podemos tener varias ventanas hijas, una acción del menú seleccionada por el usuario cuando ambos menús están mezclados afectará a la ventana hija que esté activa. Al principio, el funcionamiento de los menús y las barras de herramientas en aplicaciones MDI puede resultar un poco complejo, pero con un poco de práctica, y aprendiendo a hacer un buen uso de las tres propiedades que acabamos de aprender, rápidamente podremos desarrollar aplicaciones muy profesionales.

! ToolStrip e ítem estándar

El ToolStrip tiene la posibilidad de generar un conjunto de tools comunes a la mayoría de las aplicaciones de manera automática. Para utilizar esta característica debemos utilizar el comando Insert Standard Ítems, ubicado en el SmartTag del control.

El evento por monitorizar en la mayoría de los ítem es Click.

Otros controles

Existen muchos controles más en .NET, pero se nos hace imposible enumerarlos todos y explicar por completo su funcionalidad, a la cual puede accederse a través de la ayuda del entorno o desde el sitio oficial de Microsoft con la ayuda en línea en http://msdn2.microsoft.com/es-ar/library/default.aspx. A continuación, nombramos algunos de ellos y su funcionalidad básica, que puede profundizarse leyendo la documentación en línea de MSDN, muy completa y muy bien organizada.

Controles de datos

Tal como vimos en capítulos anteriores cuando estudiamos ASP.NET, habitualmente tendremos que recuperar datos de una base y mostrarlos en un formulario. Cuando el volumen de datos es grande, no podemos recorrer todos los registros y actualizar un control manualmente, sino que deberemos recurrir a técnicas de DataBinding y a controles que lo soporten. En futuros capítulos aprenderemos en detalle cómo enlazar controles como el TextBox a campos de la base de datos, pero mientras tanto, vamos a conocer algunos controles que permiten mostrar datos en pantalla:

- GridView (Grilla): Permite mostrar datos tabulares, es decir, en forma de filas y columnas. Es el control por defecto para mostrar tablas enteras o una gran cantidad de datos. Permite configurar las columnas que queremos mostrar, soporta ordenamiento en tiempo de ejecución e, incluso, podemos hacer que actualice los datos en la base luego de modificarlos en la grilla.
- BindingSource: Permite realizar el enlace de datos (binding) entre diferentes orígenes de datos y los controles. Funciona como un intermediario entre los controles del formulario enlazados a datos y el origen real de éstos.
- BindingNavigator: Permite realizar la navegación de los datos enlazados a los controles de manera visual. Actúa en conjunto con BindingSource. Además de la navegación, brinda facilidades para la manipulación de los datos, como altas, bajas y modificaciones de registros, todo de manera visual, como una barra de herramientas.

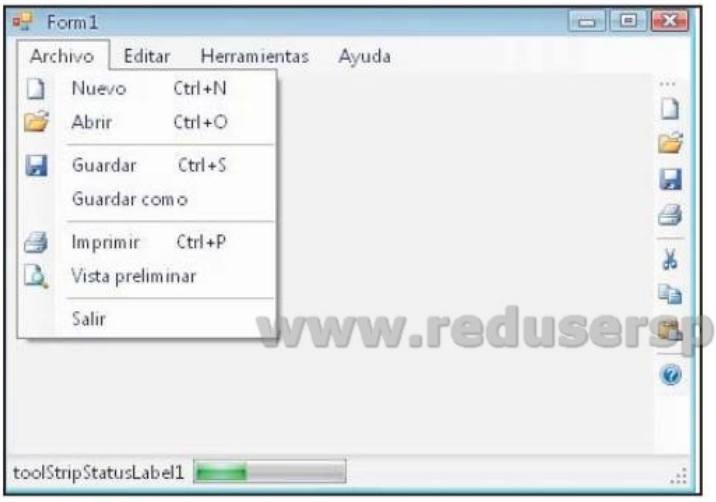

FIGURA 027 | Los controles strip permiten crear interfaces muy completas, con menús, barras de herramientas y de estado.

Controles no visuales

Todos los controles que hemos visto a lo largo de este capítulo son controles que, en tiempo de diseño, se ven igual a como se verán en ejecución; son controles "visuales". Pero existen otros que podemos colocar en el formulario y que, luego, en ejecución, no se verán o se verán de una manera diferente. Estos controles no visuales se utilizan para completar la interfaz con características adicionales que le dan un toque de distinción. Los controles no

visuales se destacan porque en tiempo de diseño no aparecen dentro del formulario, sino que se ubican en la parte inferior de la ventana de Visual Studio. Veamos algunos de ellos:

- ToopTip: Adiciona la propiedad Tooltip a los controles existentes, para visualizar la ayuda al pasar el mouse sobre ellos. Su uso es muy sencillo: basta con colocarlo en el formulario e inmediatamente a todos los controles les aparecerá una propiedad llamada ToolTip, que nos permitirá escribir el texto que queremos mostrar cuando se apoye el mouse sobre el control.
- ErrorProvider: Este interesante control permite mostrar una pequeña imagen parpadeante junto a un control, para indicar que hay un error en el ingreso de los datos. Tiene asociado un texto que se muestra cuando el usuario coloca el mouse sobre la imagen. La utilidad de ErrorProvider, a diferencia de MessageBox, radica en que podemos mostrar varios mensajes al mismo tiempo y, además, indicar visualmente dónde está el error.
- Timer: En ocasiones, debemos realizar una tarea de manera repetida, cada cierto intervalo de tiempo. Naturalmente, no podemos escribir un ciclo for o while para realizarla, porque la aplicación no podrá continuar con otras acciones. En estas situaciones, se recurre al control Timer, que permite indicar un intervalo de tiempo en milisegundos, tras el cual dispara un evento en el que podemos escribir el código de la tarea por realizar.
- ImageList: Este control permite crear una colección de imágenes a las que se les asocia un índice. Muchos de los controles que vimos anteriormente, como ListView, TreeView y TabControl, poseen una propiedad para asociarlos con un ImageList, y cada elemento tiene una propiedad para indicar el índice de la imagen por mostrar.

> La utilidad de ErrorProvider radica en que podemos mostrar varios mensajes al mismo tiempo.

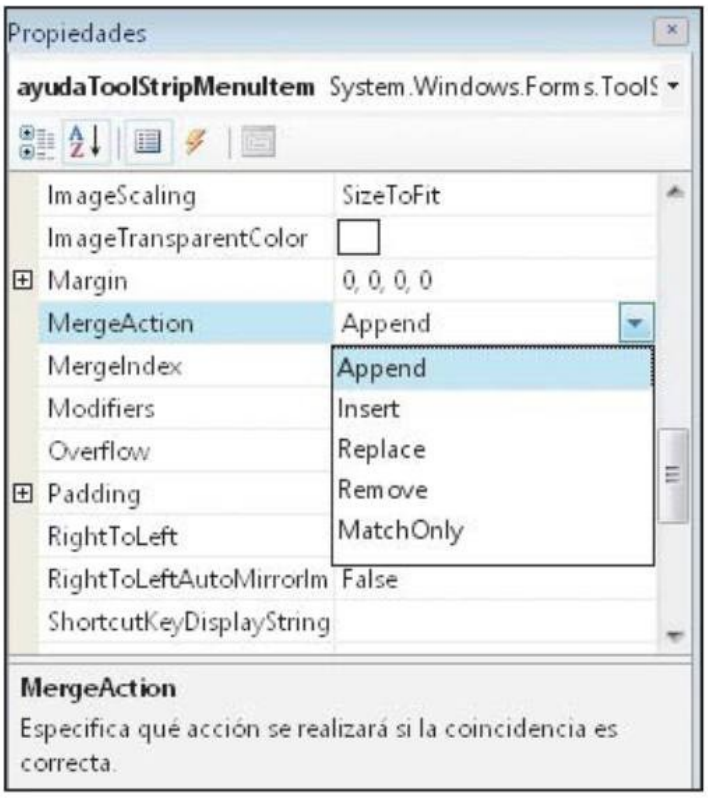

FIGURA 028 | Cuando colocamos menús en ventanas de tipo MDI, debemos configurar correctamente las propiedades MergeAction y AllowMerge.

FIGURA 029 | El control GridView permite mostrar datos en forma de tabla.

El framework provee muchos controles que nos permiten realizar tareas bastante complejas.

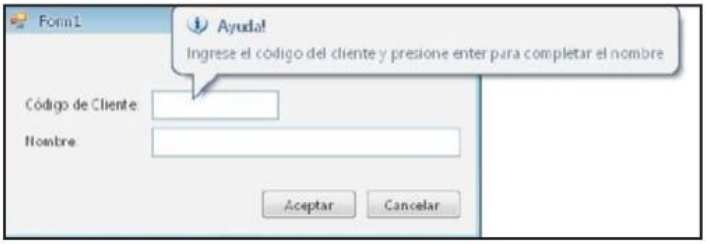

FIGURA 030 | ToolTip permite mostrar una ayuda al usuario cuando pasa el mouse sobre los controles.

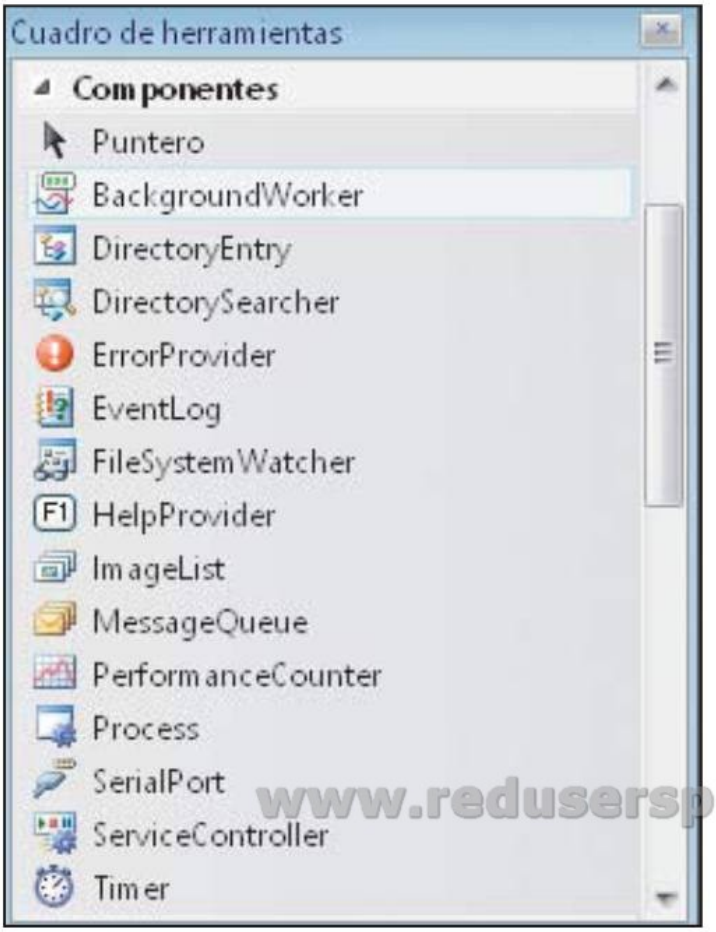

FIGURA 031 | Los controles no visuales se encuentran en la sección Componentes, del Cuadro de Herramientas de las versiones Express de Visual Studio.

Controles especiales

El framework provee muchos controles que nos permiten realizar tareas bastante complejas, como leer datos del puerto serie de la computadora. Estos controles también son del tipo "no visual":

- EventLog: Proporciona acceso al Registro de Eventos de Windows, para ingresar mensajes.
- FileSystemWatcher: Monitoriza el sistema de archivos y genera eventos al cambiar un directorio o archivo. Es muy útil para hacer que la aplicación reaccione cuando un archivo es modificado o creado en una carpeta del disco.
- SerialPort: Proporciona acceso a los puertos serie de la PC. Un detalle interesante es que funciona por eventos; es decir, cuando llegan datos al puerto, dispara un evento para que podamos leerlo.
- MessageQueue: Proporciona acceso a una cola en un servidor Message Queue de Microsoft. Esta tecnología es muy utilizada para comunicar aplicaciones de manera desatendida, donde una aplicación coloca un mensaje en la cola y sigue con sus tareas. La otra aplicación "desencola" el mensaje y hace con él lo que necesite.
- BackgroundWorker: Permite realizar una operación en un thread (hilo) separado de nuestra aplicación. Una de las ventajas de usar este control sobre la programación manual del hilo independiente es la sencillez de uso y la seguridad en cuanto al uso de los recursos.
- ServiceController: Proporciona el acceso y el control a los servicios del sistema operativo. Mediante este control, podemos manipular los servicios de manera muy sencilla, invocando métodos del control para pausar, detener e iniciar cualquier servicio.

Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

12

Controles

Controles de Windows
Menús - Controles contenedores

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Diseño de aplicaciones

Snaplines - Anchor y Docking - Paneles

ISBN 978-987-1347-43-8
00012
9 789871 347438