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

USERS

Microsoft

Curso teórico y práctico de programación

# Desarrollador .net 11

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

## Windows Forms

Conceptos fundamentales
Eventos y métodos - Ciclo de vida

## Interacción con el usuario

Tabulaciones - Cuadros de mensaje





ISBN 978-987-1347-43-8
00011
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

## El control ObjectDataSource

En las aplicaciones empresariales, de gran escala o con mucha lógica de negocios, es muy común trabajar con objetos de negocios, y no directamente con las tablas de la base de datos. En estos casos, el SqlDataSource no nos servirá, y tendremos que acudir a otro control de enlace de datos, ObjectDataSource. Como su nombre lo indica, funciona como origen de datos para enlazar controles Web con objetos de negocios de la aplicación.

A grandes rasgos, el funcionamiento de ObjectDataSource es muy similar al de SqlDataSource en cuanto a posibilidades de configuración, con la diferencia de que, en vez de especificar una consulta SQL para recuperar los datos, debemos indicar cuál es el método de la clase de negocios que se encarga de devolverlos.

Veamos un pequeño ejemplo que ilustre el funcionamiento de este interesante control. Supongamos que tenemos una clase llamada Provincia, que representa una provincia:

```
// C#
public class Provincia
{
 private string nombre;
 private int id;

 public Provincia(string nombre, int id)
 {
   Nombre = nombre;
   ID = id;
 }

 public string Nombre {
  get{ return nombre; }
  set{ nombre = value; }
 }
 public int ID {
  get{ return id; }
  set{ id = value; }
}
```

Esta clase, como vemos, tiene dos propiedades públicas y un constructor que recibe los valores de los atributos para hacer la inicialización. Es muy importante el uso de las propiedades, ya que DataBinding no funciona con atributos públicos, sino sólo con propiedades.

Ahora, lo que necesitamos es un método que devuelva una lista de provincias. Lo más correcto es crear uno estático dentro de la misma clase, aunque en la práctica, es posible emplear distintas técnicas, dependiendo de la arquitectura elegida para la aplicación. Para el ejemplo, vamos a "hardcodear" la lista que se devolverá:

```
//C#
public static List<Provincia>
ObtenerProvincias()
{
  List<Provincia> lista = new
  List<Provincia>();
  lista.Add( new Provincia("La Pampa", 1) );
  lista.Add( new Provincia("Buenos Aires", 2) );
  return lista;
}
```

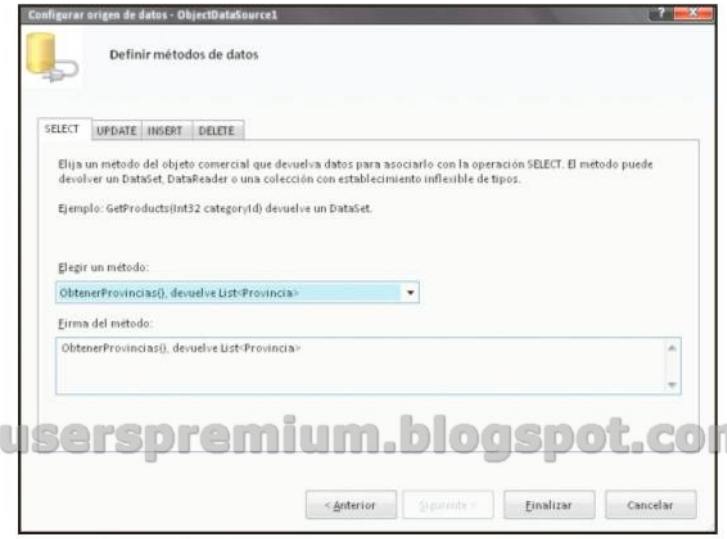


FIGURA 045 | El asistente de ObjectDataSource nos muestra los métodos de la clase, para que indiquemos cuál usar con el objetivo de recuperar los datos.

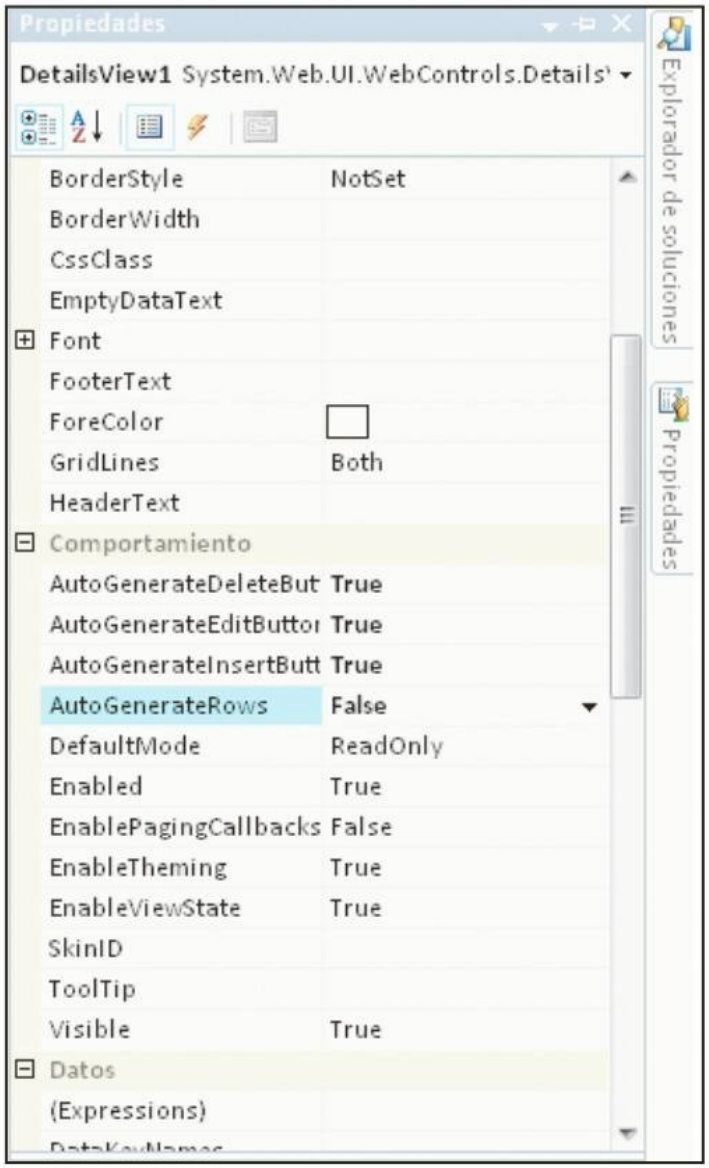


FIGURA 046 | Mediante las propiedades que aquí se presentan, logramos que el control muestre botones para editar, agregar y borrar registros, sin tener que escribir código.

Ya estamos listos para usar ObjectDataSource. Entonces, arrastramos uno de estos controles a una nueva página y abrimos el asistente, de la misma manera en que lo hicimos con el SqlDataSource.

El asistente nos mostrará primero una lista con las clases que detectó en la aplicación; seleccionamos Provincia. Al hacer clic en Siguiente, nos solicitará que indiquemos el método que vamos a usar para hacer el SELECT (es decir, para recuperar los datos). En la lista desplegable sólo estará ObtenerProvincias, de modo que lo elegimos.

Ya tenemos configurado nuestro control de origen de datos; sólo resta enlazarlo a un control Web para verlo en funcionamiento. Como hicimos anteriormente, arrastramos un control DropDownList a la página y lo asociamos con el ObjectDataSource creado. Veremos que nos pide las propiedades para mostrar y para usar como valor de la lista, pero no nos muestra nada, así que ingresamos manualmente Nombre como texto, e ID como valor. Listo, al ejecutar la aplicación, veremos que la lista se llena con las dos provincias que devolvemos en el método ObtenerProvincias de la clase Provincia.

## Edición de datos

Como adelantamos unas páginas antes, desde la versión 2.0 de ASP.NET es posible hacer enlace de datos de doble vía; es decir, para mostrar los datos en controles de la página, y para tomar los valores de esos controles y usarlos para actualizar los datos en la base. Además, con el uso de los controles de origen de datos, resulta realmente sencillo implementar funcionalidad de actualización, ya que los mismos controles hacen la mayoría del trabajo por nosotros.



### Controles para edición

Existe un par de controles que permiten mostrar de a un registro a la vez y que, al mismo tiempo, son muy útiles para modificar datos, porque ellos mismos se encargan de comunicarse con el control de origen para hacer todo

el trabajo. Se trata de DetailsView y FormView. Ambos son muy similares y, quizá, la diferencia más notable sea que FormView requiere que el usuario defina la forma en que se muestran los datos, tanto en modo lectura como en modo edición, mientras que DetailsView lo hace automáticamente mediante campos (incluso, podemos seleccionar cuáles de los campos de la tabla mostramos). Esta diferencia hace que DetailsView resulte más simple de usar pero sea menos flexible. La elección de uno u otro dependerá de cada caso particular.
Veamos un ejemplo sencillo. Siguiendo con la tabla de Provincias que ya usamos, en una nueva página vamos a configurar un SqlDataSource para esa tabla, haciendo que traiga todos los registros y ambos campos. Luego, arrastramos desde el cuadro de controles un objeto DetailsView y lo asociamos con el DataSource que creamos. Si ejecutamos la aplicación, veremos que el control mostrará el primer registro y nada más. Para navegar por todos ellos, basta con darle el valor True a la propiedad AlowPaging.

Hay tres propiedades que son muy útiles (Figura 046): AutoGenerateDeleteButton, AutoGenerateInsertButton y AutoGenerateEditButton. Éstas permiten que el control genere automáticamente botones de tipo Link para poner el control en modo de eliminación, inserción y edición, respectivamente. Para entender cómo funcionan, vamos a poner el valor True a las tres, y a ejecutar la aplicación. Veremos que al pie del control aparecen tres nuevos links: Editar, Eliminar y Nuevo. Si hacemos clic en el primero, la página se enviará al servidor y, al regresar, el control habrá cambiado: en vez de mostrar sólo el registro actual, habrá un control de tipo TextBox con el nombre de la provincia como contenido. Además, el pie del control también estará diferente, ya que tendrá dos nuevos botones: Actualizar y Cancelar. Esto significa que el control

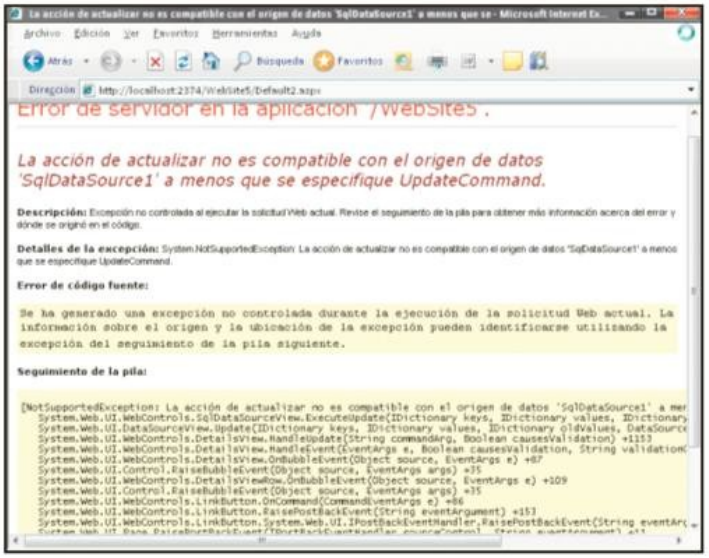


FIGURA 047 | Al querer actualizar los datos, recibimos un error indicando que falta configurar el UpdateCommand en SqlDataSource.

está en modo edición. Mediante esta funcionalidad, podemos darle al usuario la posibilidad de modificar el registro actual y, al hacer clic en Actualizar, de enviar esos cambios a la base. Bien, hagamos la prueba, cambiemos algo en el TextBox y presionemos Actualizar. ¿Qué sucedió? Nos encontramos con la clásica pantalla de error de ASP.NET, diciendo que el SqlDataSource que estamos usando no soporta actualización, porque no hemos configurado la propiedad UpdateCommand. ¿Qué significa esto? Bueno, DetailsView se comunica con SqlDataSource para llevar a

## Ayudar al asistente

Para que el asistente nos muestre la lista de propiedades para enlazar a un control, podemos decorarlas con el atributo System.ComponentModel.ObjectField. Así, éste las detectará y nos presentará una lista, y no tendremos que escribirlas nosotros mismos.

cabo las actualizaciones, pero el control SqlDataSource no es lo suficientemente inteligente para saber cómo debe actualizar los datos, así que debemos indicárselo nosotros.

Para hacer la actualización contra la base de datos, escribimos las consultas que SqlDataSource deber ejecutar.

## Editar con ObjectDataSource

Al igual que SqlDataSource, ObjectDataSource permite actualizar los datos. Para lograrlo, debemos especificar cuáles son los métodos de nuestra clase de negocios que se ocupan de realizar cada una de las operaciones de actualización, asegurándonos de que tienen los parámetros correctamente indicados.

Afortunadamente, Visual Studio también nos ayuda en ese aspecto. Abrimos el asistente de SqlDataSource y, en la sección en la que seleccionamos los campos por recuperar (el segundo paso), presionamos el botón Avanzado. Se abrirá una nueva ventana en la que podemos decidir, marcando un CheckBox, si queremos generar los comandos para Insertar, Actualizar y Eliminar. Lo hacemos y aceptamos. Al aceptar y cerrar el asistente, tendremos listo nuestro SqlDataSource para actualizar los datos a través de DetailsView. Ejecutamos la aplicación y ponemos el control en modo edición. Si cambiamos algo en el nombre de la provincia y hacemos clic en Actualizar, veremos que el control vuelve al modo de navegación (sólo lectura), y el cambio queda reflejado. Podemos probar a hacer esto mismo con la inserción y con la eliminación de registros.

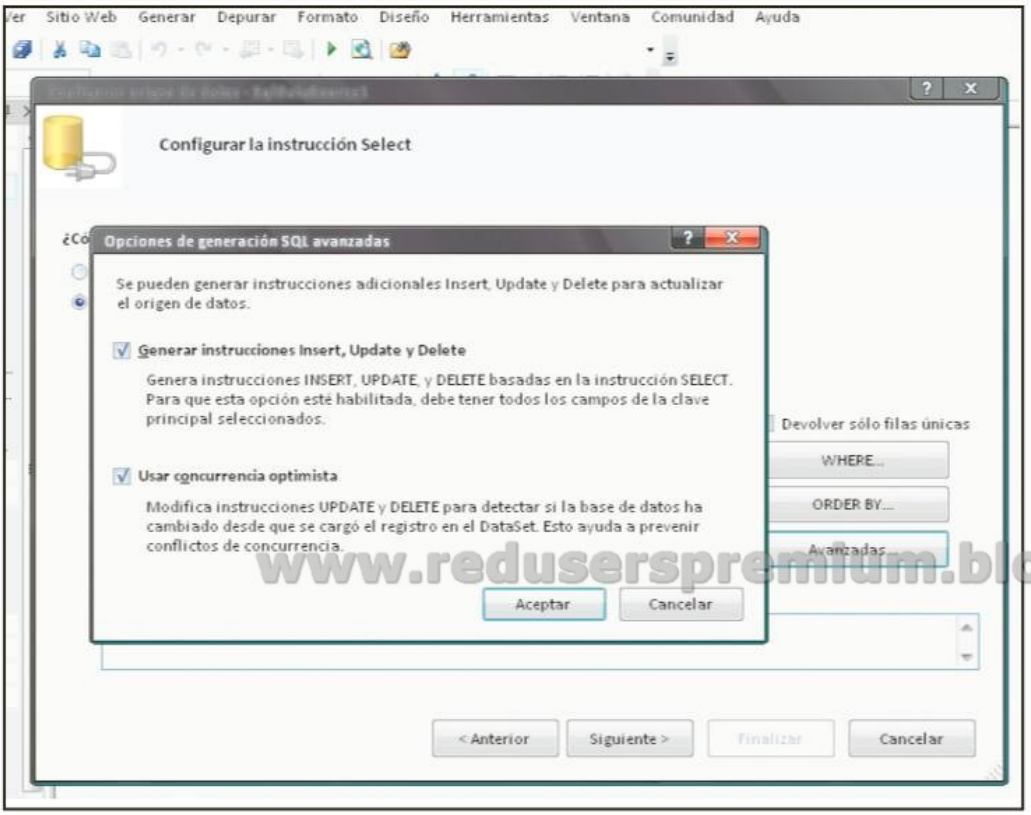


FIGURA 048 | El asistente genera por nosotros los comandos para actualizar la información en la base de datos.

# Compilación e instalación

Veremos los pasos requeridos para una correcta instalación del sitio Web, mediante su compilación y publicación.

El nuevo modelo de compilación instantánea de ASP.NET permite a los desarrolladores prescindir de una librería compilada con todas las clases del sitio por cada proyecto Web. Esto implica una serie de ventajas, como el famoso edit & continue: la capacidad de editar una página, guardarla y volver a ejecutarla en el depurador sin frenar la aplicación. Si bien la técnica de compilación instantánea es útil en un entorno de desarrollo, no es recomendada para un servidor en producción, ya que el rendimiento alcanzado no es el óptimo. Por este motivo, y cuando necesitemos "instalar" nuestro proyecto finalizado (o una versión de él) en el servidor de nuestro cliente, utilizaremos la precompilación ofrecida por la herramienta Aspnet_compiler.exe.

## Carpetas y archivos

Para realizar la compilación dinámica, ASP.NET permite organizar los archivos en una serie de carpetas, con nombres "reservados" para este fin. A continuación, veremos una breve lista de ellas.

### Bin

En esta carpeta se incluyen las librerías referenciadas (DLL) en nuestro proyecto. Es una carpeta obligatoria en el caso de la precompilación.

### App_Code

En este directorio guardaremos todos los archivos de código fuente (archivos .VB o .CS).

### App_WebReferences

Aquí se mantienen las referencias a servicios Web, y los correspondientes archivos .DISCO y .WSDL, en carpetas individuales para cada servicio referenciado.

### App_Data

Es el único directorio que se genera por defecto, y está destinado a guardar tanto bases de datos como cualquier archivo (XML por ejemplo) que utilicemos como origen de datos.

### App_LocalResources y App_GlobalResources

Están destinadas a almacenar archivos RESX, con recursos para páginas específicas o todo el sitio, respectivamente.

### App_Themes

Se utiliza para guardar los temas del sitio Web, como se vio en la sección de Temas y skins.

### Archivos .aspx en sitios precompilados

Cuando precompilamos un sitio, los archivos con extensión ASPX (los que tienen los controles y el código HTML de la página) también son compilados en una dll, y su contenido se reduce a un texto que indica que es un archivo generado por una herramienta de precompilación. Sin embargo, de todos modos deberemos subirlo al servidor de producción para que la aplicación funcione correctamente. La precompilación de archivos ASPX es uno de los cambios en el modelo de compilación que introdujo ASP.NET 2.0.


FIGURA 049 | La herramienta de compilación se encuentra en la carpeta correspondiente a la versión del Framework .NET.

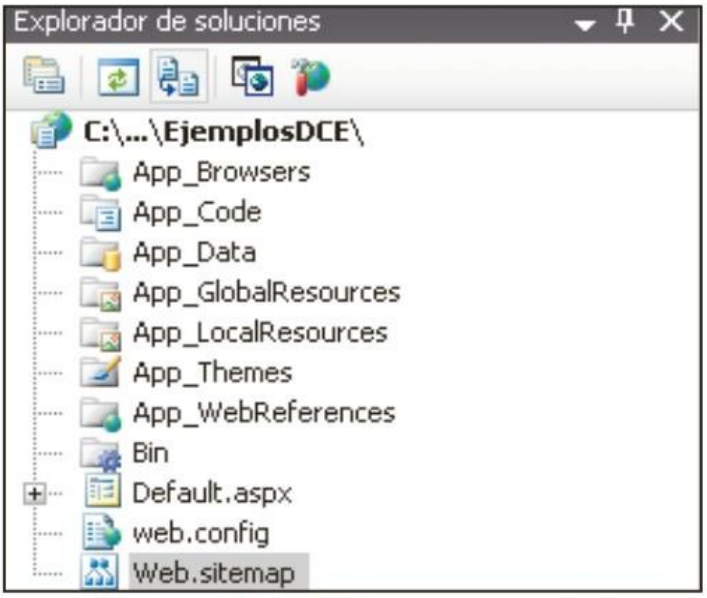


FIGURA 050 | Un sitio Web ASP.NET 2.0 con todas las "carpetas especiales" incluidas.

## Precompilacion en Visual Web Developer

Ésta es la única versión de Visual Studio que no posee la herramienta de publicación, pero toda la funcionalidad necesaria para precompilar un sitio está en la herramienta Aspnet_compile.exe. El resto de las versiones de Visual Studio 2005, a partir de la estándar, ya incluye esta característica.

# Utilizando el compilador

La herramienta Aspnet_compiler, de la línea de comandos del .NET Framework, nos permite realizar la precompilación de nuestro sitio. Esta herramienta está en C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727.

## Parámetros

Al ser una herramienta no visual, toda la configuración debe realizarse mediante parámetros opcionales que modificarán el resultado de la ejecución del EXE en cuestión. A continuación, veremos los principales parámetros para compilar aplicaciones con Aspnet_compiler.

### *v

Permite establecer la ruta virtual a nuestra aplicación. Basta con mencionar el nombre de la aplicación si ésta se encuentra instalada en Internet Information Server.

### *p

Ruta física de nuestra aplicación, donde están los archivos de código que deseamos compilar.

### *errorstack

Si lo incluimos, se agregará información sobre los errores generados durante la compilación.

### *targetDir

Especifica el directorio de salida, en donde se depositará el resultado de la compilación.

### *u

Esta opción fundamental indica al compilador si debe realizar un sitio actualizable o completamente compilado. En el primer caso, sólo se "ocultan" los archivos .VB, es decir que no podremos tocar el código fuente. En el segundo, si bien se precompila el sitio, los archivos ASPX permanecen intactos para poder modificarlos posteriormente.

# Consejos

Veremos una serie de trucos o tips avanzados para mejorar nuestra productividad durante el desarrollo de aplicaciones Web.

Existe una manera de tener "código mixto" en nuestra aplicación Web, y es a través de una directiva de compilación, ubicada en el archivo Web.Config, como veremos a continuación.
Para lograrlo, deberemos crear una carpeta dentro de la reservada App_Code, con un nombre particular, que luego referenciaremos al compilador para indicarle el lenguaje de los archivos que contiene. El siguiente código representa un Web.Config armado para compilar un subdirectorio con Visual Basic .NET y otro con C#:

```
<configuration>
  <system.web>
    <compilation>
      <codeSubDirectories>
        <add directoryName="CodigoVisual"/>
        <add directoryName="CodigoCSharp"/>
      </codeSubDirectories>
    </compilation>
  </system.web>
</configuration>
```

La estructura de nuestra aplicación Web quedará configurada de la siguiente manera:

```
+Raiz de la Aplicación
|-+App_Code
|-|—CodigoVisual
|-|—CodigoCSharp
```

## Plantillas

Si prestamos atención al asistente de Nuevo Sitio Web, veremos la opción Mis Plantillas. Este elemento, a veces sin consideración, permite crear sitios completos a partir de plantillas de código (también llamadas *templates*).
Por defecto, Visual Web Developer 2005 incluye la plantilla Atlas WebSite, que permite hacer un sitio con tecnología AJAX, prácticamente sin ningún esfuerzo adicional.

### Archivos y directorios

La plantilla Atlas WebSite genera todo lo necesario para ejecutar nuestra aplicación con AJAX, lo cual incluye: referencia a la librería de Atlas, carpeta contenedora de scripts (conocida como ScriptLibrary), página por defecto de Atlas, un archivo de ayuda (readme) y el contrato de usuario en formato RTF.

### Descargar plantillas

Junto con la opción Atlas WebSite del asistente Nuevo Sitio Web, se encuentra Buscar plantillas en línea. Si hacemos doble clic en este elemento, se abrirá la ayuda, configurada para la búsqueda de plantillas.

### AJAX y la experiencia del usuario

El concepto de AJAX no es exclusivo de ASP.NET, sino que es mucho más extenso y nació como una necesidad de ofrecer una mejor experiencia de usuario, haciendo menos viajes de ida y vuelta al servidor, y utilizando DHTML y JavaScript para actualizar sólo algunas porciones de la página. Con esto, las aplicaciones Web se comportan casi como aplicaciones de escritorio.

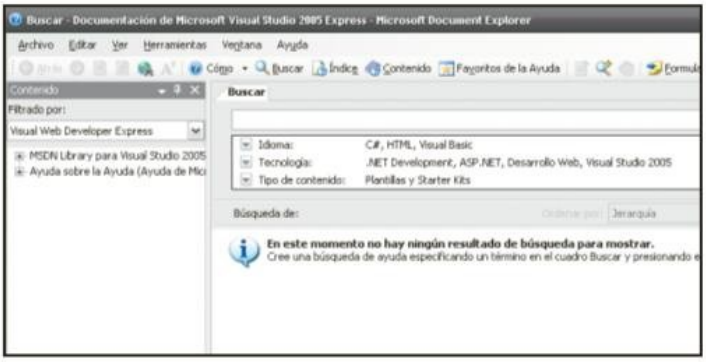


FIGURA 051 | Con la herramienta incluida dentro de la Documentación de Visual Studio Express, podemos buscar más plantillas para agregar a las ya presentes.

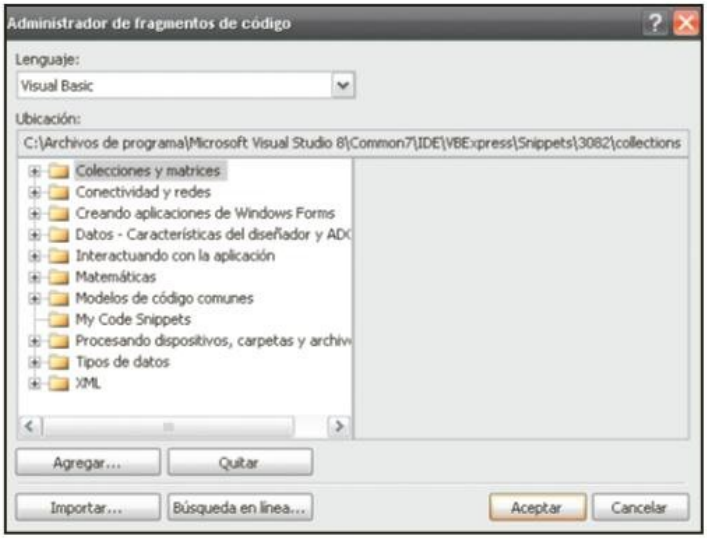


FIGURA 052 | El Administrador de fragmentos es práctico y muy fácil de utilizar.

## Consejo de productividad

Es muy aconsejable que, a medida que el programador adquiere cierta experiencia, vaya creando sus propios snippets o porciones de código personalizado. Teniendo una buena batería de snippets, logrará prácticamente automatizar la mayoría de las tareas repetitivas y monótonas con las que se encuentra durante la codificación de una aplicación.

# Snippets

Para una programación óptima y un alto nivel de productividad, no hay nada mejor que los llamados Code Snippets o, simplemente, fragmentos de código. Esta novedad incluida en Visual Studio 2005 y las versiones Express permite tener nuestra propia biblioteca de fuentes, con los fragmentos de código más utilizados, e insertar un fragmento usando Intellisense.

## Administrando código

Si accedemos al menú Herramientas/Administración de fragmentos de código, en Visual Web Developer 2005, se abrirá un cuadro de diálogo destinado a la organización, creación y modificación de snippets. Entre las categorías que incluye, podemos encontrar cientos de fragmentos de código, con su correspondiente descripción en español, y la forma de acceder directamente a él desde el código fuente.

## Lenguajes

La lista desplegable de Lenguaje funciona como filtro para la lista de snippets, y nos permite elegir entre Visual Basic .NET, C# e, incluso, XML. A su vez, en cada uno de ellos está la carpeta My xxxxx Snippets, en donde xxxxxx representa al lenguaje seleccionado.
Dicha carpeta se usa para almacenar los fragmentos de código que ingresemos por nuestra cuenta o descarguemos desde Internet.

## Importar, descargar

Desde el administrador es posible importar archivos con extensión .snippet. En el sitio http://msdn2.microsoft.com/en-us/vstudio podemos encontrar una lista de snippets listos para descargar. Para insertar un fragmento de código en nuestro archivo .VS o .CS basta con escribir el símbolo clave "?" y presionar <Tab>, o elegir la opción Insertar fragmento de código, del menú contextual del código fuente.

# Windows Forms

## Aplicaciones de escritorio

6

## Contenidos

En capítulos anteriores aprendimos a crear aplicaciones Web con ASP.NET. En este capítulo trataremos todos los temas relacionados con el desarrollo de aplicaciones de escritorio.

## Temas tratados

- Características de las aplicaciones de escritorio
- Qué es un formulario
- El diseñador de formularios de Visual Studio
- Eventos y métodos
- Ciclo de vida de un formulario
- Interacción con el usuario
- Controles

# Windows Forms

Veremos desde cero como desarrollar aplicaciones de escritorio para el sistema operativo Windows, utilizando los lenguajes C# y Visual .NET.

## Aplicaciones de escritorio

Primero conoceremos cuáles son los elementos y las características que forman parte de las aplicaciones de escritorio para el sistema operativo Windows.

- Creación de aplicaciones en Windows
- Qué es un Form
- Aplicaciones SDI y MDI

## Formularios

Veremos en detalle qué son los formularios y cómo crearlos con Visual Studio .NET.

- Creación de nuevos proyectos
- Diseño de formularios
- Principales propiedades de los formularios

## Eventos y métodos

Repasaremos los conceptos relacionados con los eventos y nos introduciremos en el mundo de los eventos generados por la interacción del usuario con la aplicación.

- Principales eventos
- Manejadores de eventos
- Uso de los argumentos del evento

## Ciclo de vida de un formulario

Para crear aplicaciones que logren una buena experiencia, es imprescindible conocer el ciclo de vida de los formularios con el fin de que la aplicación reaccione de manera apropiada.

- Ciclo de vida
- Principales eventos
- Ejemplos de uso de los eventos del ciclo de vida



## Interacción con el usuario

Aprenderemos a utilizar el mouse y el teclado, y veremos ciertas características que podemos aprovechar para que el usuario se sienta más cómodo con la aplicación.

- Eventos del mouse y del teclado
- Tabulaciones
- Cuadros de mensaje

# Formularios

En las siguientes páginas, nos introduciremos en el mundo de los formularios, para poder aprovechar todo su potencial.

El Framework .NET proporciona una gran cantidad de elementos para la creación y administración de aplicaciones Windows.
Todos estos elementos, clases, objetos y controles se encuentran bajo el namespace System.Windows.Forms.
Éste contiene todo lo necesario para la construcción de aplicaciones Windows, y si no hallamos lo adecuado, podemos crear nuestras propias clases que hereden de ellas, para así lograr la funcionalidad requerida.

## ¿Qué es un form?

Un form, formulario o Windows Form es un componente de diseño donde colocaremos los elementos gráficos que hacen a la interfaz con el usuario; son las "ventanas" de las aplicaciones. A diferencia de las aplicaciones Web, donde todo el contenido se coloca en páginas, en las de escritorio se utilizan los formularios para realizar cualquier tarea, tanto para que el usuario ingrese datos, como para mostrar información. Si prestamos atención, veremos que cada vez que usamos Windows, estamos ante formularios: una venta para enviar un mail, el cuadro de diálogo que presenta Word para abrir un archivo y hasta el juego de Solitario que viene con Windows son formularios.

## Diseñador de formularios

Como todo en .NET, los formularios también son clases, que en este caso heredan de System.Windows.Forms.Form, la cual contiene mucha funcionalidad que veremos en las próximas secciones.
Ahora que ya sabemos qué es un formulario, vayamos al teclado para crear nuestra primera aplicación Windows con este tipo de elemento. Para hacerlo, utilizamos la siguiente secuencia.

Seleccionamos el menú **File/New Project** y, en la ventana que se abre, elegimos **Windows Application**.
Al crear el proyecto, automáticamente se genera un formulario por defecto (Form1) y, dependiendo del lenguaje de programación, un conjunto de archivos asociados al proyecto (esto se debe a la manera en que cada lenguaje ejecuta las aplicaciones y genera los puntos de entrada a ellas), por ejemplo:

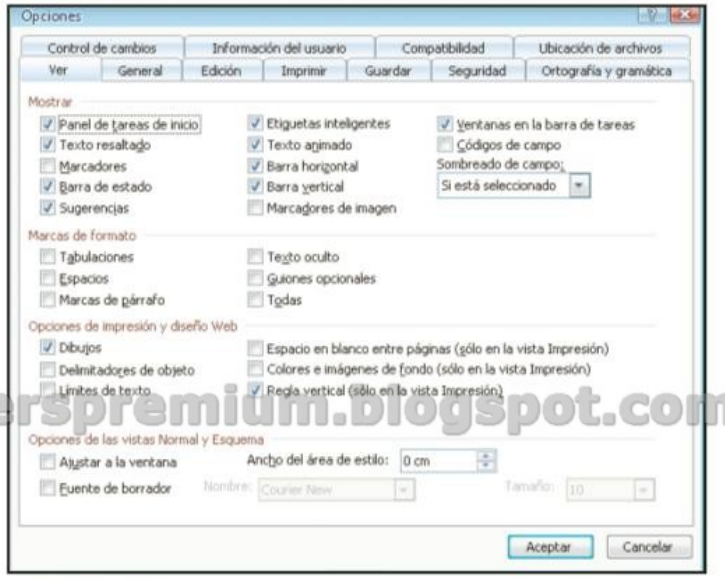


FIGURA 001 | Cualquier ventana de una aplicación para Windows es un formulario, como ésta de opciones de Word.

En las aplicaciones de escritorio, se utilizan los formularios para que el usuario ingrese datos, como también para mostrar información.

**C#:**

- Archivo Program.CS: Contiene una clase estática y un método Main(), que es el punto de entrada a la aplicación. Otros elementos referidos a la forma en que se inicia la aplicación se escriben en esta clase.
- Carpeta References: Contiene las referencias a los assemblies necesarios para que este tipo de aplicación funcione.

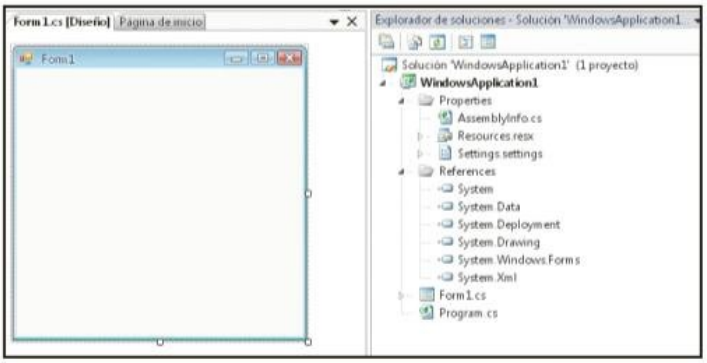


FIGURA 002 | Los diferentes elementos del proyecto Windows Application en el lenguaje C#.

## ⓘ Creación de aplicaciones Windows

Para los ejemplos utilizaremos las ediciones Express Edition de Visual C# o Visual Basic .NET, que proveen herramientas para diseñar los formularios de manera visual, arrastrando objetos (los controles), dando valor a ciertas propiedades y ayudando a generar parte del código necesario para manipular la interfaz con el usuario.

- Carpeta Properties: Contiene un conjunto de archivos que permiten administrar diferentes propiedades del proyecto
- Archivo AssemblyInfo.cs: Contiene la información del assembly al compilar, tal como versión, visibilidad COM, etc. Este archivo se edita manualmente en C#.
- Settings.settings: Contiene información referida a la configuración de la aplicación.

**VB.NET**

- Carpeta My Project: Contiene el acceso a las propiedades del proyecto; dentro de esa ventana, se puede establecer el punto de entrada a la aplicación. No se necesita una clase estática para tal fin en este lenguaje, aunque también es posible utilizar este método. Basta con indicar cuál es el formulario de inicio de la aplicación.
- Otras carpetas y archivos: Visual Basic .NET hace que la configuración de propiedades del proyecto y otros elementos se realice directamente desde los diseñadores de la carpeta My Project, en vez de hacerlo a mano (aunque esta posibilidad se deja para usuarios avanzados). Si deseamos visualizar los elementos de la misma manera que en C#, sólo tenemos que seleccionar Show All Files, en el menú del Solution Explorer, para ver los archivos ocultos del proyecto; automáticamente, se mostrarán los archivos AssemblyInfo.vb, la carpeta References, el archivo Settings.settings, etc.

El diseñador es el encargado de generar todo el código de las clases, y de asociar las propiedades de los formularios y los controles de manera automática. Recordemos que, básicamente, es un generador de código: todo lo realizado en forma visual, arrastrando y soltando, modificando sus propiedades a través de los diseñadores y ventana de propiedades, se genera en la clase del formulario.

El Form es el principal componente de una aplicación Windows. Todo su comportamiento se define modificando sus propiedades, y utilizando los métodos y eventos proporcionados por la clase Form.
Por ejemplo, podemos crear aplicaciones MultiDocumentos (MDI). Para hacerlo, basta con cambiar la propiedad IsMDIContainer al valor True. De esta manera, los formularios hijos (child) se cerrarán al cerrar el formulario principal. Para que un formulario sea hijo de un formulario MDI, hay que asignarle al hijo la referencia al padre, a través de la propiedad MdiParent; veamos algunos ejemplos:

VB.NET:

```
'Formulario Padre
Form1.IsMdiContainer=True

'Formulario Hijo
Form2.MdiParent=Form1

'ó si al crear la instancia está en el
mismo Form padre
Form2.MdiParent=Me
```

C#:

```
Form1.IsMdiContainer=true;

/*Formulario Hijo*/
Form2.MdiParent=Form1;

/*ó si al crear la instancia está en el
mismo Form padre*/
Form2.MdiParent=this;
```

Diferentes propiedades aceptan distintos tipos de datos. Algunos son tipos de datos simples de .NET:

VB.NET:

```
Form1.Text="Título del Form"
Form1.TopMost=True
```

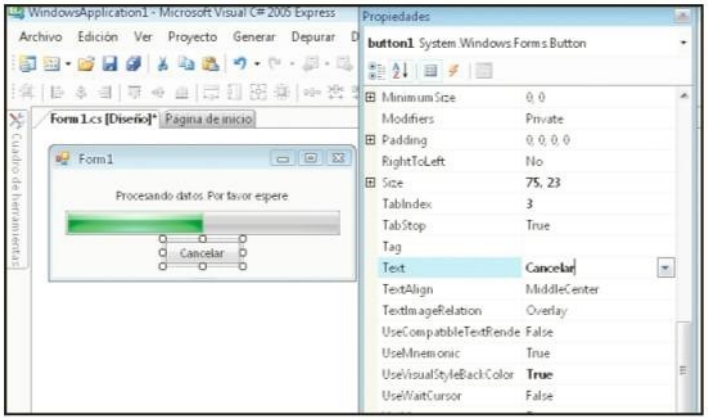


FIGURA 003 | El editor de formularios permite crear la interfaz de manera visual e interactiva.

C#:

```
Form1.Text="Título del Form";
Form1.TopMost=true;
```

## § Aplicaciones SDI y MDI

Existen dos tipos de aplicaciones de ventanas: simple documento (SDI, *Single Document Interface*) y múltiple documento (MDI, *Multiple Document Interface*). En las SDI hay una ventana principal y una cierta cantidad de ventanas secundarias independientes, lo que significa que, por ejemplo, podemos minimizar la principal y seguir trabajando en las otras. Las aplicaciones MDI constan de una ventana "madre" y de un conjunto de ventanas "hijas" que dependen de aquélla. Esto significa, por ejemplo, que al minimizar la ventana principal, se minimizan todas las otras. El programa Microsoft Word es un ejemplo de MDI, dado que, cuando se minimiza la ventana, también lo hacen todos los documentos que estén abiertos. Otra característica de las aplicaciones MDI es que el menú de cada ventana secundaria se fusiona con el de la ventana principal, y se provee un único punto de acceso a todas las opciones de la aplicación.

Otros son objetos:

VB.NET:

```
'Posición
Me.Location = New Point(0, 0)

'Tamaño del form
Me.Size = New Size(640, 480)

'Fuente por defecto para elementos del form
Dim DefaultFont As Font = New Font("Arial", 10)
Me.Font = DefaultFont
btnAceptar.Font = DefaultFont
lblTitulo.Font = DefaultFont
```

C#:

```
/*Posición*/
this.Location = New Point(0, 0);

/*Tamaño del form*/
this.Size = New Size(640, 480);

/*Fuente por defecto para elementos del form*/
Font DefaultFont = New Font("Arial", 10);
this.Font = DefaultFont;
btnAceptar.Font = DefaultFont;
lblTitulo.Font = DefaultFont;
```

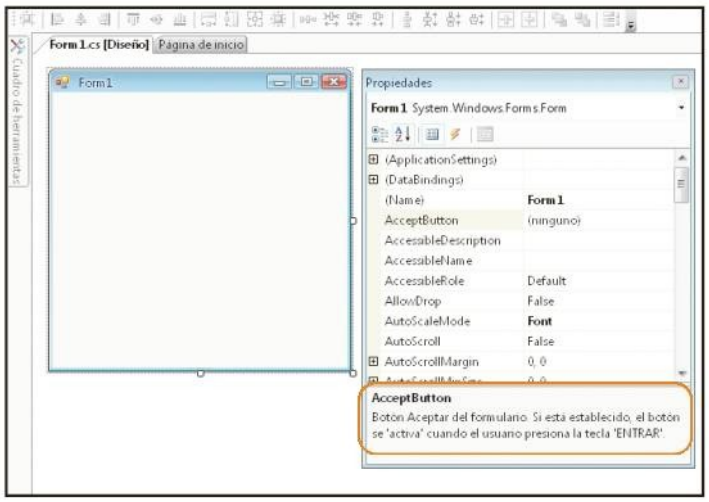


FIGURA 004 | Es aconsejable recorrer la ventana de propiedades, observando el texto de ayuda de cada una.

## Controles adicionales

Los controles que trae .NET no son los únicos que podemos usar. El modelo de desarrollo de .NET permite crear nuevos controles e incorporarlos a la barra de controles, con el fin de que estén listos para ser arrastrados al diseñador de formularios. También podemos comprar controles desarrollados por colegas o empresas que se dedican a comercializar estos elementos con funcionalidad específica.

Un formulario presenta diferentes propiedades que permiten mejorar su aspecto y proporcionar variadas funcionalidades. Dado que deriva de un control, muchas de las propiedades también son comunes a ellos y otras son específicas de él. En la Tabla 1 se muestran algunas.

## Eventos y métodos

Para trabajar con los formularios, es necesario conocer sus principales métodos y eventos. Como vimos en los primeros capítulos y en el libro que se adjunta a la obra, un método es un subprograma, una pequeña porción de código separada del resto y que puede ser invocado cuantas veces sean necesarias. Desde el punto de vista de los objetos, podemos ver un método como una acción que puede ejecutar el objeto. Los eventos, en cambio, son sucesos que ocurren por algún motivo, como puede ser una acción del usuario o cualquier suceso externo. Para que podamos reaccionar ante esos sucesos, el sistema operativo "dispara" un evento que puede ser atrapado por la aplicación. Al código que escribimos para actuar en

consecuencia al evento se lo denomina **manejador del evento**. Conocer ambas características en fundamental para el diseño de las aplicaciones Windows, ya que su funcionamiento se basa, principalmente, en estos dos elementos. Las aplicaciones Windows tienen un modelo de programación orientado a eventos. ¿Qué significa esto? Pues que la estructura de un programa debe basarse en las acciones que pueda realizar el usuario o el sistema. El formulario y los controles en él reaccionan ante diferentes acciones o circunstancias. Por ejemplo, cuando hacemos clic en un botón, se genera un evento asociado a esa acción (el evento Clic), y en su manejador ponemos el código para realizar alguna tarea y proveer de una respuesta a la acción del usuario (el clic que hizo en el botón). El formato de un manejador de eventos es el siguiente:

C#:

```
/*sólo admite la segunda opción mostrada
 en VB.NET*/
this.control.Click += new System.EventHandler
(this.NombreProcedimiento);
private void NombreProcedimiento(object
sender, EventArgs e)
{
   /*código*/
}
```

## Tabla 1 | Propiedades de la clase Form

| Propiedades | Descripción |
|---|---|
| BackColor, ForeColor, BackgroundImage, Font, Cursor | Definen la apariencia: color de fondo, color de texto, imagen de fondo, fuente y cursor, respectivamente. |
| 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. |
| Opacity | Permite modificar la opacidad. Varía de 0 a 1, aunque en la ventana de propiedades aparece como porcentaje (0 al 100). Es útil para hacer formularios traslúcidos o semitransparentes. |
| Text | Texto del control o formulario. |
| Controls | Es una colección que contiene todos los controles incorporados dentro del formulario o de un control contenedor. |
| AcceptButton | Indica cuál es el botón por defecto del formulario; es decir, cuál se ejecuta al presionar la tecla ENTER. |
| IsMDIContainer | Indica si el formulario va a presentar una interfaz MDI (puede contener otros formularios). |
| CancelButton | Indica cuál es el botón de cancelación del formulario; es decir, aquel que se ejecutará cuando el usuario presione la tecla ESC. |
| ControlBox, MinimizeBox, MaximizeBox | Indica si el formulario tiene botones de control, minimizado y maximizado, respectivamente. |
| FormBorderStyle | Permite cambiar el estilo de borde del formulario. |
| StartPosition | Indica dónde debe ser posicionado el formulario al visualizarse (centrado en la pantalla, en la posición por defecto, etc.). |
| WindowState | Indica cómo se muestra el formulario inicialmente (maximizado, minimizado, etc.). |

VB.NET:

```
'Manejador de evento generado automáticamente.
No puede ser eliminado en tiempo
'de ejecución. Es el que se crea por defecto.
Private Sub NombreProcedimiento (ByVal sender
As Object, ByVal e As System.EventArgs)
Handles Control.Evento
  'codigo
End Sub


'Ó también puede escribirse manualmente de
esta manera. De ésta forma podemos
'"atrapar" un evento en tiempo de ejecución
cuando lo necesitemos y controlarlo 'manera
dinámica.
AddHandler Me.Control.Click, AddressOf(Nombre
Procedimiento)
Private Sub NombreProcedimiento(ByVal sender
As Object, ByVal e As System.EventArgs)
'código
End Sub
```

Donde:

- NombreProcedimiento: Es el nombre del procedimiento que se utilizará para capturar el evento. Automáticamente, se genera un nombre de procedimiento asociando el control y el evento elegido, pero puede tener cualquier nombre.
- sender: Es el parámetro del procedimiento que indica quién genera el evento.
- e: Es el parámetro que contiene los argumentos del evento; es decir, toda aquella información extra asociada a lo que ha ocurrido y alguna más. Esta variable es de diferente tipo según el evento asociado. En el evento Clic, por ejemplo, su tipo es System.EventArgs; en un evento KeyPress, su tipo es System.KeyPressEventArgs.
- AddHandler (VB.NET) y += (C#): Realizan la asociación del evento del control con el manejador del evento. VB.NET tiene una posibilidad extra utilizando la palabra clave Handles al finalizar el procedimiento. Para quitar la asociación de los manejadores de evento en tiempo de ejecución, se emplean sus contrapartes RemoveHandler (VB.NET) y -=(C#).

Un formulario contiene una gran cantidad de eventos que podemos monitorizar o utilizar. Los más usados son:

- Load: Ocurre después de que el formulario se cargó en memoria, pero no es visible aún.
- Activated: Ocurre cuando el formulario es activado; es decir, pasa a ser el formulario con el que está interactuando el usuario.
- FormClosing, FormClosed: Se producen cuando el formulario está por cerrarse o ya se cerró, respectivamente. En estos eventos podemos determinar el origen del cierre, y cancelar.

## Práctica

Es muy importante "jugar" con el entorno. Es necesario acostumbrarse a él y probar los diferentes elementos que nos da el diseñador de formularios y, sobre todo, la ventana de propiedades. En última instancia, tenemos la opción de no grabar los cambios e, incluso, de borrar los directorios creados, sin problemas.



Como mencionamos unos párrafos atrás, además de los eventos, los formularios tienen métodos que nos permitirán darles "instrucciones", como mostrarlos, ocultarlos o maximizarlos. Veamos un ejemplo de uso de un método. Una aplicación normalmente está compuesta por muchos formularios. A excepción del principal, el resto debe crearse y utilizarse como cualquier otro objeto. Supongamos que en una

aplicación Windows tenemos dos formularios (Form1 y Form2), y deseamos visualizar el Form2 ante el evento Clic en el Form1.

El código por escribir es el siguiente:

VB.NET:

```
Private Sub Form1_Click(ByVal sender As
Object, ByVal e As System.EventArgs) Handles
Me.Click
    Dim oForm2 As Form2
    oForm2 = New Form2
    oForm2.Show()
 End Sub
```

C#:

```
private void Form1_Click(object sender,
EventArgs e)
{
    Form2 oForm2;
    oForm2=new Form2();
    oForm2.Show();
}
```

Como vemos, debemos declarar una variable del tipo adecuado, instanciarla como cualquier objeto y utilizar un método del objeto para visualizarlo (Show, en este caso).
El método Show permite visualizar un formulario y, a continuación, seguir con la ejecución del código desde donde fue llamado. A este método se le puede adicionar un parámetro extra para indicarle cuál es el "dueño" (owner) de este form. Esto hace que el formulario visualizado esté siempre por encima de su padre, y es útil para imitar el comportamiento de barras de herramientas flotantes de muchos programas. Si deseamos visualizar el formulario de manera modal –es decir, en forma de diálogo–, debemos utilizar otro método, en este caso, ShowDialog. En este modo, no podremos volver al formulario inicial hasta tanto no cerremos el nuevo creado.

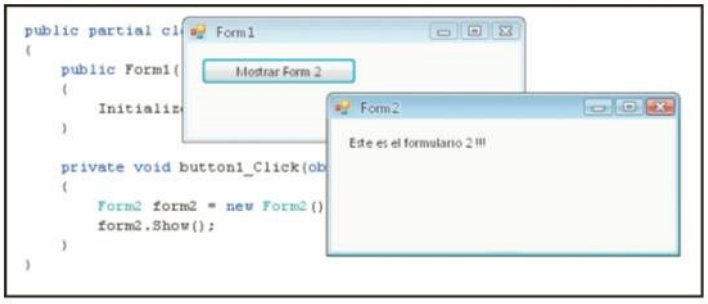


FIGURA 005 | Los métodos nos permiten darles instrucciones a los formularios, como mostrarse en pantalla.

Hay diferentes métodos que podemos utilizar con los formularios. Los principales son:

- Activate: Activa el formulario y le da el foco.
- Close: Cierra el formulario, y hace que se ejecuten los eventos FormClosing y FormClosed de él.
- Show: Muestra y activa el formulario. Si es la primera vez que se muestra, primero se carga en memoria, lo cual ocasiona el evento Load.
- ShowDialog: Se comporta de manera similar a Show, con la diferencia de que no devuelve el control al formulario padre hasta tanto no se cierre el nuevo.

## § Busy Waiting

Antes de que existiera el desarrollo orientado a eventos, las aplicaciones que necesitaban "enterarse" de algún suceso debían quedarse consultando continuamente algún valor que indicara si el suceso había ocurrido o no.
Por ejemplo, para detectar que se había presionado una tecla, era preciso escribir un ciclo while y leer el teclado en cada iteración. A esto se lo llamaba Busy Waiting (espera ocupada) y no era una buena práctica, ya que impide que la aplicación siga haciendo otra cosa, además de que consume procesador.

El método Show permite visualizar un formulario y seguir con la ejecución del código desde donde fue llamado.

- Hide: Oculta un formulario. Es el método opuesto a Show. Hide sólo lo oculta (cambia la propiedad Visible a False), no lo descarga de memoria, por lo que no se producen los eventos FormClosing y FormClosed al utilizar este método.

## § Signature

La estructura (cantidad y tipo) de parámetros de un procedimiento o función se denomina firma (signature) y queda fija en tiempo de compilación. Cuando escribimos un procedimiento para manejar un evento, su firma debe coincidir exactamente con la del tipo del manejador del evento que queremos atrapar.

## Ciclo de vida

Muchos de los eventos a los que responde el objeto Form pertenecen al ciclo de vida del formulario. Entre ellos están los siguientes, en orden de ocurrencia:

- Load: El formulario se carga en memoria pero no está visible. Se ejecuta una sola vez durante la vida del form.

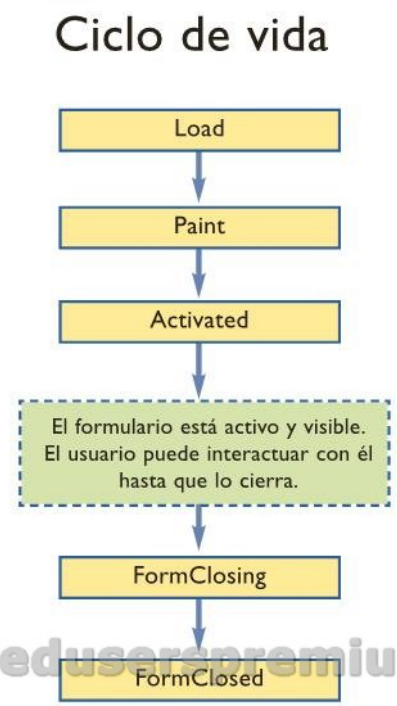




FIGURA 006 | El ciclo de vida de un formulario consta de una serie de eventos que se suceden en cadena, antes y después de que el usuario interactúa con él.

- Paint: Se "pintan" el formulario y también sus controles.
- Activated: El formulario recibe foco.
- FormClosing: Se producen cuando el formulario está por cerrarse. En este evento es posible cancelar la descarga.
- FormClosed: El formulario se ha cerrado. Es el lugar indicado para liberar los recursos utilizados por el form durante su vida.

Varios eventos ocurren varias veces durante la vida del Form, como Activated, Deactivated, VisibleChanged, etc. Conocer el ciclo de vida de un formulario nos permitirá controlar nuestra aplicación y determinar en qué momento escribir el código para que todo funcione como queremos. Por ejemplo, utilizando el evento FormClosing, podemos pedirle al usuario que confirme la descarga del form. Este evento nos da la posibilidad de detectar desde qué lugar se está cerrando y cancelar la descarga si así lo deseamos. Siguiendo con el ejemplo anterior, en el evento FormClosing del Form2 escribimos el siguiente código:

VB.NET:

```
Private Sub Form2_FormClosing(ByVal sender As
Object, _
        ByVal e As System.Windows.Forms.Form
        ClosingEventArgs) Handles Me.FormClosing

    'Verificamos el origen del cierre del form.
    'En este caso hemos elegido el cierre por
    una acción del usuario
    If e.CloseReason = CloseReason.UserClosing
    Then

        'Preguntamos si realmente desea cerrarlo
        If MessageBox.Show("Confirma el cierre
        del formulario?", _
                    Me.Text, MessageBox
                    Buttons.YesNo, _
                    MessageBoxIcon.Question, _
                    MessageBoxDefaultButton.
                    Button2) = Windows.Forms.
                    DialogResult.No Then

            'si la respuesta es No, cancelamos la
            descarga
            e.Cancel = True
        End If

    End If

End Sub
```

C#:

```
private void Form2_FormClosing(object sender,
FormClosingEventArgs e)
{
    /*Verificamos el origen del cierre del form.
    En este caso hemos elegido el cierre por una
    acción del usuario*/
    if (e.CloseReason == CloseReason.
    UserClosing)
     {
       //Preguntamos si realmente desea cerrarlo
       if (MessageBox.Show("Confirma el cierre
       del formulario?",
          this.Text, MessageBoxButtons.YesNo,
          MessageBoxIcon.Question, MessageBox
          DefaultButton.Button2)==DialogResult.No)
```

## Convenciones para los eventos

En .NET, todos los manejadores de eventos tienen dos parámetros: el primero es de tipo Object, y el segundo, generalmente, llamado "e", es de un tipo compatible con EventArgs (es decir, del mismo tipo o de un tipo descendiente). El primero siempre contiene una referencia al objeto que generó el evento, y el segundo, los argumentos del evento.

CloseReason nos permite saber el motivo por el cual se está cerrando un formulario.

```
        {

            //si la respuesta es No, cancelamos la
            descarga
            e.Cancel = true;
        }
    }
}
```

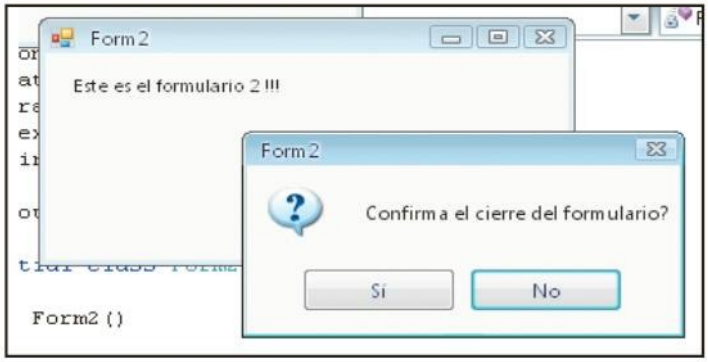


FIGURA 007 | Mediante el evento FormClosing, podemos detectar cuándo se está cerrando el formulario e, incluso, cancelar el evento, para evitar que la ventana se cierre.

## Anular el cierre de un formulario

Mediante la propiedad Cancel del parámetro e del evento FormClosing, podemos hacer que el formulario no se cierre, al darle un valor True. De este modo, como el formulario aún no se cerró, podemos hacer que se cancele el evento y no se lo cierre. Esto es útil, por ejemplo, si el usuario no grabó en su momento los cambios que hizo en la aplicación.

Lo importante de todos los eventos es inspeccionar el tipo de dato de *e* (el parámetro con los datos del evento). De ese objeto obtendremos toda la información necesaria para trabajar con ellos.

## Trabajando con el mouse y teclado

Como sabemos, las aplicaciones modernas hacen un uso intensivo del teclado y, sobre todo, del mouse. ¿Cómo hacemos para reaccionar ante acciones del mouse? La respuesta es simple: capturando eventos. Tanto el formulario como los controles tienen la posibilidad de capturar eventos de mouse y teclado. A continuación, se detallan los principales eventos, tanto del mouse como del teclado, que podemos capturar y cuándo se produce cada uno de ellos.

El parámetro *e* de los eventos del mouse es de tipo MouseEventArgs, y proporciona la información de los diferentes elementos:

- e.Button: Indica qué botón/es han sido presionados.
- e.Clicks: Cantidad de clics generados desde el último evento del mouse capturado
- e.Location: Proporciona las coordenadas del puntero del mouse durante el evento.
- e.X: Coordenada X de la posición del puntero del mouse.
- e.Y: Coordenada Y de la posición del puntero del mouse.

Veamos un ejemplo de cómo verificar si se presionó el botón derecho del mouse:



VB.NET:

```
Private Sub Form1_MouseDown(ByVal sender As
Object, ByVal e As MouseEventArgs) _
                        Handles Me.MouseDown
  If e.Button = MouseButtons.Right Then
```

```
    MessageBox.Show("Botón derecho del mouse
    presionado!")
  End If
End Sub
```

C#:

```
private void Form1_MouseDown(object sender,
MouseEventArgs e)
{
  if (e.Button == MouseButtons.Right)
  {
   MessageBox.Show("Botón derecho del mouse
   presionado!");
  }
}
```

¿Por qué tres eventos para cuando se presiona una tecla? Bueno, porque cada uno ocurre en un momento diferente y nos permite reaccionar cuando sea más apropiado. Por ejemplo, podemos necesitar hacer algo recién cuando el usuario suelte la tecla, o apenas la presione y aún no la haya soltado (como sucede con las teclas de control). El orden de ejecución de estos eventos es KeyDown, KeyPress y KeyUp. El evento KeyPress permite determinar qué tecla fue presionada.

Para esto, en su manejador de evento, el argumento *e* es del tipo KeyPressEventArgs y proporciona los siguientes elementos:

- e.KeyChar: Indica el carácter presionado y es del tipo Char.
- e.Handled: Permite indicar al framework que el evento ha sido procesado y "cancelar" su ejecución.

Por ejemplo, supongamos que debemos permitir al usuario sólo el ingreso de datos numéricos; es decir, de caracteres numéricos. Podríamos escribir un código como el siguiente para proporcionar dicha funcionalidad:

VB.NET:

```
Private Sub Form1_KeyPress(ByVal sender As
Object, ByVal e As _
                KeyPressEventArgs)Handles
                Me.KeyPress
    If Char.IsNumber(e.KeyChar) = False Then
      e.Handled = True
    End If
End Sub
```

C#:

```
private void Form1_KeyPress(object sender,
KeyPressEventArgs e)
{
  if (Char.IsNumber(e.KeyChar)==false)
  {
    e.Handled = true;
  }
}
```

En el ejemplo anterior, si el carácter ingresado no es un número, se cancela su ingreso utilizando la propiedad **handled** del argumento del evento.

## Tabla 2 | Principales eventos del mouse



| Evento | Descripción |
|---|---|
| MouseDown | Ocurre cuando presionamos un botón del mouse sin soltarlo. |
| MouseMove | Ocurre al mover el mouse sobre el control o formulario. |
| MouseUp | Ocurre al soltar un botón del mouse luego de haberlo presionado. |

MessageBox permite presentar un mensaje al usuario con diferentes opciones y tipo de información.

Si deseamos monitorear las teclas que presiona el usuario, como las de función F1, F2, CTRL, ALT, etc., el evento KeyPress no proporciona esa información, sino que debemos utilizar los eventos KeyDown y KeyUp. Éstos brindan más información acerca del teclado. La diferencia entre los dos eventos se da cuando se ejecutan KeyDown, al presionar una tecla; y KeyUp, al soltarla. Pero la información proporcionada por ambos es la misma. Como todo evento, su información viene en el parámetro *e* del manejador y es del tipo KeyEventArgs. Esta clase otorga mucha información, pero las propiedades más comúnmente utilizadas son las siguientes:

- e.Alt: Indica si está presionada al mismo tiempo la tecla ALT.
- e.Control: Indica si está presionada al mismo tiempo la tecla CTRL.
- e.Shift: Indica si está presionada al mismo tiempo la tecla SHIFT.
- e.Handled: Permite indicar al framework que el evento ha sido procesado y "cancelar" su ejecución.
- e.KeyCode: Proporciona el código de la tecla presionada.

Supongamos que queremos verificar cuándo el usuario presiona la combinación de teclas CTRL+F6, para realizar una determinada acción. El código que podríamos utilizar sería:

VB.NET:

```
Private Sub Form1_KeyUp(ByVal sender As
Object, ByVal e As KeyEventArgs) _
                                Handles Me.KeyUp
    If e.Control=True AndAlso e.KeyCode =
    Keys.F6 Then
      'realizar acción
    End If
End Sub
```

C#:

```
private void Form1_KeyUp(object sender,
KeyEventArgs e)
```

## Información de los eventos

Para sacar el máximo provecho de los eventos, es necesario conocer la información que proporciona el parámetro e en ellos y, sobre todo, los tipos de datos involucrados. Comprendiendo estos elementos, la validación de datos y la obtención de información se realizan mucho más fácilmente.

## Tabla 3 | Principales eventos del teclado



| Evento | Descripción |
|---|---|
| KeyDown | Ocurre cuando presionamos una tecla sin soltarla. |
| KeyPress | Ocurre cuando una tecla es presionada mientras el control tiene el foco. |
| KeyUp | Ocurre al soltar una tecla luego de haberla presionado. |

```
{
   if (e.Control==true && e.KeyCode == Keys.F6)
    {
     //realizar accion
    }
}
```

## Focos y tabulaciones

Cuando desarrollamos aplicaciones, siempre debemos tratar de lograr un buen diseño de **navegación** de los controles en el formulario; es decir, conseguir que la interacción con la aplicación sea intuitiva, y que el usuario se sienta cómodo y pueda familiarizarse rápidamente con la interfaz. Por navegación se entiende la forma en que el usuario se desplaza de un control a otro. Por ejemplo, si está cargando una factura, el orden de navegación será: nombre del cliente, primer artículo y su cantidad, segundo artículo y su cantidad, etc. En formularios diseñados para una carga intensiva de datos, es fundamental tener un orden adecuado de navegación de los controles, para que el usuario pueda desplazarse lo más cómodamente posible. La toma de foco de los controles debe ser coherente con el flujo de la carga de la información. Se dice que un control tiene **foco** cuando es el control que está activo y es el que recibe los mensajes que envía el sistema operativo al presionar una tecla. En el caso de un cuadro de texto (TextBox), el que tiene foco será el que tenga el cursor de escritura. El orden en que reciben foco los controles se denomina **orden de tabulación** (por el hecho de utilizar normalmente la tecla TAB para pasar de control en control) y está determinada por la propiedad **TabIndex** de cada uno.

Esta propiedad puede modificarse a mano utilizando la ventana de propiedades, pero en caso de tener una interfaz muy compleja, a veces es mejor utilizar una herramienta incorporada por Visual Studio para tal fin. Para habilitarla, debemos tener el formulario en vista de diseño y luego seleccionar la opción **Tab Order** del menú **View** del entorno. A continuación, cada control presentará un pequeño cuadro en la esquina su-

> Siempre debemos tratar de lograr un buen diseño de navegación de los controles.

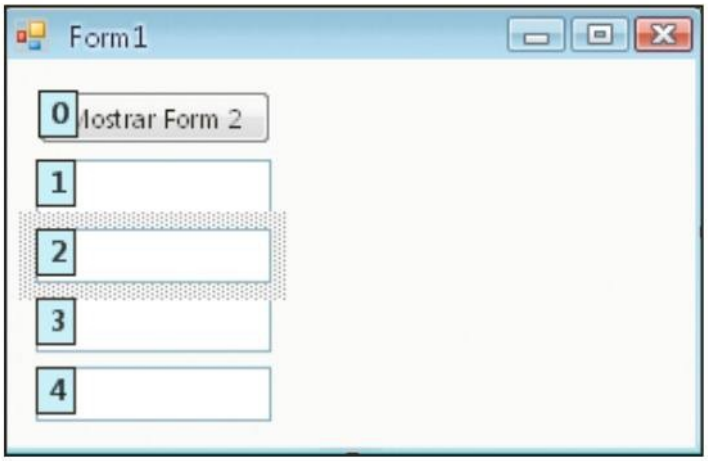


FIGURA 008 | Herramienta de TabOrder en funcionamiento: la figura muestra la información de TabIndex de cada control, para que pueda modificarse de manera visual.

### Aprender a diseñar formularios

Cuando tenemos poca experiencia, puede ser difícil lograr un buen diseño de formularios, sobre todo, en cuanto a la disposición y navegación de los controles. Es muy aconsejable tomar como referencia el software comercial que tengamos instalado, para ver cómo están organizadas las ventanas. Uno de los mejores ejemplos es el programa Microsoft Outlook.

perior izquierda con un número que indica su TabIndex actual. Para cambiarlo, basta con hacer clic en cada control en el orden deseado en ese pequeño cuadro y, automáticamente, se modificarán los TabIndex de cada control.

## MessageBox

Muchas veces es necesario solicitar información al usuario, pedir confirmación para determinadas tareas o sólo indicarle el resultado de una operación. Para estos casos, podemos hacer uso de la clase **MessageBox**, que permite presentar un mensaje al usuario con diferentes opciones y tipo de información. Se le puede indicar el título, el mensaje, las opciones que puede seleccionar (Sí-No-Cancelar, Aceptar-Cancelar, etc), el icono por visualizar, etc. La clase MessageBox no puede ser instanciada. Para mostrar el diálogo sólo debe llamarse al método estático Show correspondiente y seleccionar los parámetros adecuados. Este método tiene 21 combinaciones diferentes para elegir la que más nos convenga. A continuación, un ejemplo que solicita al usuario que confirme la salida de la aplicación:

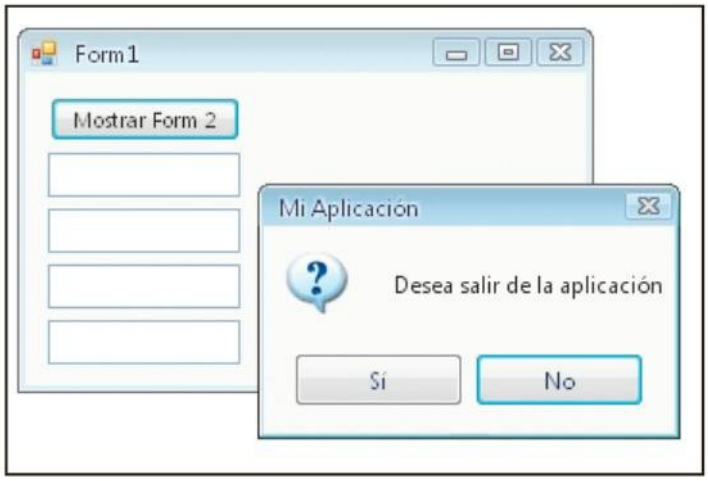


FIGURA 009 | El MessageBox en acción, solicitando una confirmación al usuario.

VB.NET:

```
Private Sub Button1_Click(ByVal sender As
System.Object, ByVal e As _
                        System.EventArgs)
                        Handles Button1.Click
 If MessageBox.Show("Desea salir de la
 aplicación", "Mi Aplicación", _
   MessageBoxButtons.YesNo, MessageBoxIcon.
   Question, _
   MessageBoxDefaultButton.Button2) =
   Windows.Forms.DialogResult.Yes Then
           Application.Exit()
 End If
End Sub
```

C#:

```
private void button1_Click(object sender,
EventArgs e)
{
    if (MessageBox.Show("Desea salir de la
    aplicación", "Mi Aplicación",
      MessageBoxButtons.YesNo,
      MessageBoxIcon.Question,
      MessageBoxDefaultButton.Button2) ==
      Windows.Forms.DialogResult.Yes)
         {
         Application.Exit();
         }
}
```

### Opciones de MessageBox

La clase MessageBox ofrece una gran cantidad de opciones. El lector deberá investigar los valores que puede tomar cada uno de los parámetros y experimentar con distintas combinaciones, para observar cómo varía el resultado final. Otro buen ejercicio es explorar la ventana de IntelliSense de Visual Studio y ver los valores del enumerado DialogResult.

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

USERS

Microsoft

Curso teórico y práctico de programación

# Desarrollador .net 11

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

## Windows Forms

Conceptos fundamentales
Eventos y métodos - Ciclo de vida

## Interacción con el usuario

Tabulaciones - Cuadros de mensaje





ISBN 978-987-1347-43-8
00011
9 789871 347438