Aplicación de ejemplo en VB.NET

Luís Rodríguez Baena
Universidad Pontifica de Salamanca en Madrid
Facultad de Informática
2010

Aplicación de ejemplo en VB.NET

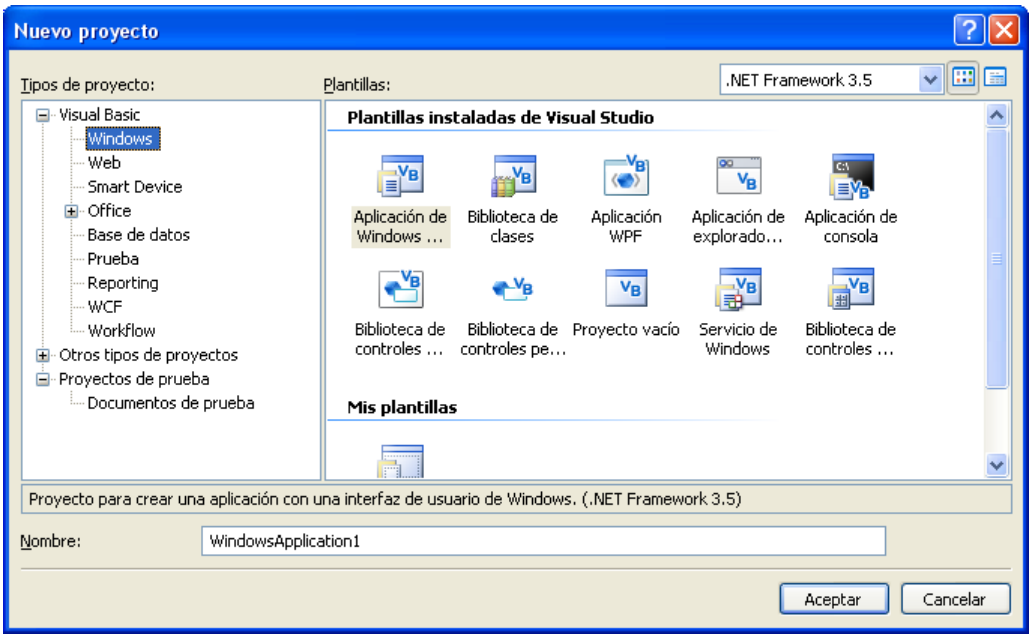
1. Propósito de la aplicación

Esta guía es un tutorial que explica el desarrollo completo de una aplicación en Visual Basic .NET. La aplicación formaría parte de otra mayor para la compra-venta de coches usados. En concreto permitiría la visualización de los vehículos a partir de su matrícula. La interfaz de la aplicación presentaría un aspecto similar al siguiente:

Al introducir la matrícula y pulsar sobre el botón ACEPTAR, se visualizará una foto con la imagen del vehículo, apareciendo su matrícula en la barra de título del formulario. El botón SALIR finalizaría la aplicación.

2. Crear un nuevo proyecto[1]

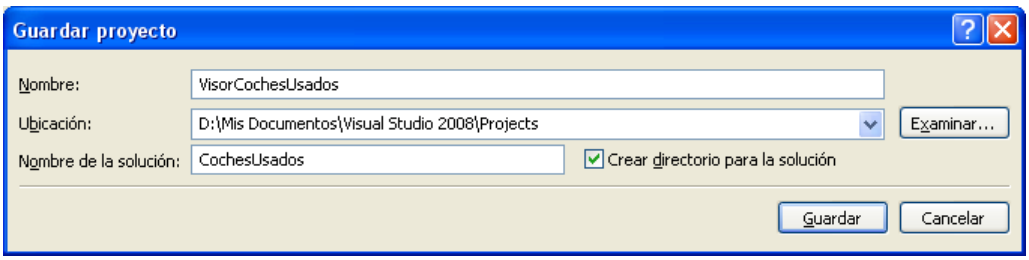
Una vez abierto el entorno de desarrollo, es necesario crear un nuevo proyecto, ya sea desde la página de inicio del entorno o mediante la opción PROYECTO del menú ARCHIVO/NUEVO (CTRL+MAYS+N).



En este cuadro de diálogo se debe elegir la plantilla APLICACIÓN PARA WINDOWS de la parte derecha. Además es conveniente dar un nombre a la aplicación para que la solución generada no tome el nombre por omisión (`WindowsApplicationXXX`).

[1] Para todas las pantallas y referencias al entorno de desarrollo, se supondrá que se está utilizando la "Configuración de desarrollo de Visual Basic". Dicha configuración se puede cambiar mediante la opción IMPORTAR Y EXPORTAR CONFIGURACIONES del menú HERRAMIENTAS.

También es posible dar un nombre a la solución y al proyecto, así como indicar la carpeta donde se guardará cuando por primera vez se da la opción GUARDAR TODO (CTRL+MAYÚS+S) del menú ARCHIVO.



Es recomendable dar al proyecto y a la solución un nombre distinto del nombre por omisión, para que, de esta forma, se pueda distinguir claramente la finalidad de cada proyecto.

3. Añadir componentes y establecer propiedades

Los componentes de la aplicación serán:

- Un contenedor heredado de `System.Windows.Forms.Form` para el formulario principal.
- Un campo de texto estático (control de la clase `Label`) para la etiqueta del cuadro de texto.
- Un campo de texto editable (control de la clase `TextBox`) para introducir la matrícula.
- Dos botones (controles de la clase `Button`) etiquetados como ACEPTAR y SALIR.
- Un control de la clase `PictureBox` que actuará como contenedor de archivo de imagen de los vehículos.

Además serán necesarias imágenes en formato JPG cuyo nombre será la matrícula del vehículo con extensión .jpg y con un tamaño de 140x100 píxel (algunas de estas imágenes están disponibles en el Campus Virtual o en la página www.colimbo.net).

3.1. El formulario principal

Al iniciar una aplicación para Windows, el entorno de desarrollo crea automáticamente un componente de la clase `Form` que actuará como formulario principal de la aplicación. El nombre por omisión del archivo que contiene el formulario es `Form1.vb` y el nombre del objeto `Form1`. Sin embargo, es interesante cambiar dichos nombres por otros más significativos. Para cambiar el nombre del archivo se puede recurrir a las propiedades del archivo, seleccionándolo desde el explorador de proyectos. La versión actual del entorno de desarrollo Visual Studio 2008, cambiará también automáticamente la propiedad `Name` del objeto de la clase `Form`.

En el formulario, también es importante cambiar el valor de la barra de títulos del formulario mediante la propiedad `Text`, y puede ser interesante cambiar el icono predeterminado de la aplicación mediante la propiedad `Icon`.

En el directorio de Visual Studio, en la subcarpeta `VS2008ImageLibrary` de la carpeta `Common7`, existen una serie de directorios con iconos y bitmaps que se pueden utilizar en las aplicaciones. Entre el material proporcionado en este tutorial, también aparece el archivo `car_icon.ico`, con el icono del formulario.

3.2. El control Label

Se utiliza para etiquetar otros campos, en este caso el cuadro de texto que contendrá la matrícula. Aparte de las propiedades que podemos usar para cambiar su aspecto (colores, tamaño, posición o tipo de letra), debemos cambiar la propiedad `Text` que contendrá el texto estático de la etiqueta. En estos controles (ocurrirá lo mismo en los botones de la aplicación), el carácter *ampersand* (`&`) antes de una letra definirá la tecla de acceso al control.

3.3. El control TextBox

Se utiliza para introducir texto editable por el usuario. En este control es muy conveniente cambiar la propiedad `Name` del control (por ejemplo, por el identificador `txtMatrícula`, utilizando el prefijo `txt` para indicar que se trata de un cuadro de texto.), ya que nos referiremos a él en el código del formulario. La propiedad `Text` permitirá referirnos al texto para obtener su valor o cambiarlo mediante la expresión `txtMatrícula.Text`.

3.4. Controles Button

En este caso, también será conveniente cambiar las propiedades `Name` y `Text` utilizando en esta última el carácter *ampersand* (`&`) para especificar la tecla de acceso. En el caso de la propiedad `Nam`e, para indicar que se trata de referencias a objetos de la clase `Button`, se puede utilizar el prefijo `btn` (los botones podrían tener los nombres `btnAceptar` y `btnSalir`).

3.5. Control PictureBox

La clase `PictureBox` sirve como contenedor de objetos gráficos. Además de la propiedad `Name` (se puede utilizar `pic` cómo prefijo de las referencias a esta clase, por lo que en nuestra aplicación el control se podría llamar `picFoto`) podemos modificar otras propiedades para cambiar su aspecto. El control, por omisión, aparece sin ningún borde que permita distinguirlo en el formulario cuando no contiene ninguna imagen. Es posible crear un borde modificando en tiempo de diseño la propiedad `BorderStyle`, a la que se asigna algún valor de la enumeración `BorderStyle`. La siguiente tabla muestra los miembros de esa enumeración.

Nombre de miembro	Descripción
Fixed3D	Borde tridimensional.
FixedSingle	Borde de una sola línea.
None	Sin borde.

De forma predeterminada, se muestra sólo la parte de la imagen que cabe en el control, pero mediante la propiedad `SizeMode`, podemos obligar a cambiar, bien el tamaño del control para que se adapta al tamaño de la imagen, o bien el tamaño de la imagen para que se adapte al tamaño del control. El valor de `SizeMode` es algún miembro de la enumeración `PictureBoxSizeMode`, cuyos miembros se puede ver en la tabla siguiente.

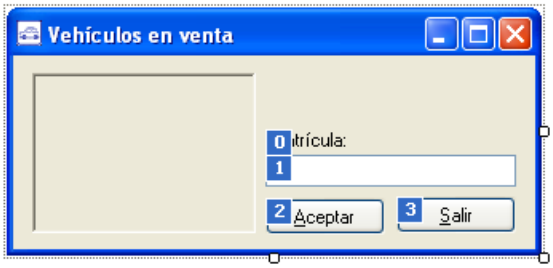
Nombre de miembro	Descripción
AutoSize	El tamaño de PictureBox debe ajustarse igual que el tamaño de la imagen que contiene.
CenterImage	La imagen se muestra en el centro si PictureBox es más grande que la imagen. Si la imagen es más grande que PictureBox, la imagen se coloca en el centro de PictureBox y se recortan los bordes exteriores.
Normal	La imagen se coloca en la esquina superior izquierda de PictureBox. La imagen se recorta si es más grande que el objeto PictureBox que la contiene.
StretchImage	La imagen situada dentro de PictureBox se estira o encoge para ajustarse al tamaño del mismo.
Zoom	Se aumenta el tamaño de la imagen o se disminuye, manteniendo la proporción de tamaño.

3.6. Control del orden de tabulación

Es importante verificar el orden de tabulación de los controles. En los controles que pueden tomar el foco, el orden de tabulación lo determina la propiedad `TabIndex` (también podemos evitar que esos controles entren en el orden de tabulación estableciendo la propiedad `TabStop` a `False`).

El control `Label` no puede tomar el foco, aunque si tiene propiedad `TabIndex`. En este caso, la propiedad se utiliza para establecer que control tomará el foco al utilizar su tecla de acceso, que será el siguiente en el orden de tabulación.

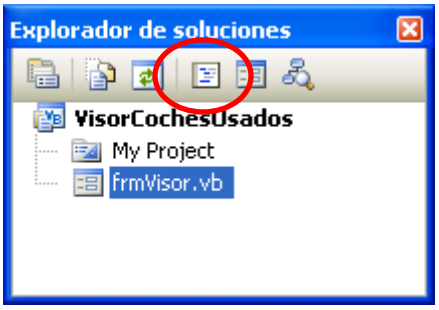
Podemos establecer manualmente la propiedad `TabIndex`, o bien utilizar la opción ORDEN DE TABULACIÓN del menú VER que permite establecerla de forma visual.



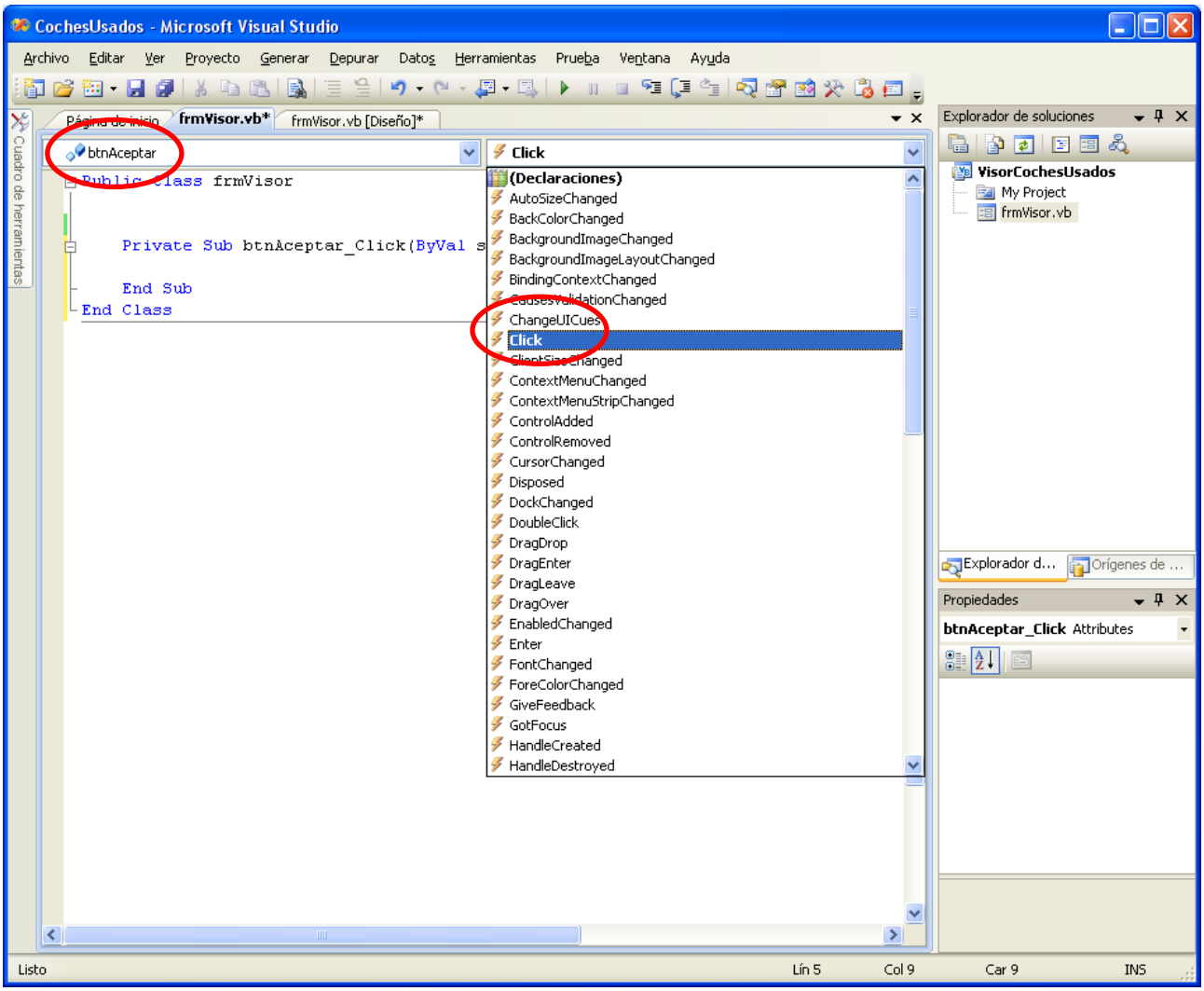
4. Codificación de los eventos

Para dotar de funcionalidad a la aplicación es necesario codificar los eventos seleccionados mediante el editor de código. Existen distintas formas de abrir el editor de código:

- Pulsando dos veces sobre un componente se abre el editor de código y aparece una plantilla con el evento predefinido para ese componente (por ejemplo, `Click` para los controles `Button`, `TextChanged` para los `TextBox`, etc.).
- Pulsando F7 o mediante la opción CÓDIGO del menú VER se abre el editor y el cursor se pone sobre la primera línea del código del módulo.
- Pulsando sobre el botón VER CÓDIGO de la parte superior del Explorador de soluciones.



En cualquiera de los casos aparecerá la ventana de código de la que podremos seleccionar un evento predefinido (seleccionando el componente en la lista de la izquierda) y el evento de la lista de la derecha o escribir nuestros propios procedimientos, declaraciones, etc.



Cada clase tiene una lista de eventos definidos para ella (que aparece en la lista de la derecha de la pantalla de código). Cada procedimiento de evento no es más que un procedimiento `Sub` con una cabecera ya definida. Por ejemplo, para el evento `Click` de un objeto de la clase `Button` sería:

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

End Sub
```

La cabecera del procedimiento de evento tiene distintas partes. Por una parte el nombre del procedimiento que, por omisión toma el nombre del control, el carácter de subrayado y el nombre del evento (`btnAceptar_Click`). Es posible cambiar el nombre del procedimiento, ya que quien controla el evento es la cláusula `Handles` del procedimiento.

Todos los eventos tienen dos argumentos. El argumento `sender` es una referencia al objeto que ha enviado al evento. A partir de ella podemos acceder a los distintos miembros del objeto. Sin embargo, su único miembro accesible es el método `GetType`, que devuelve una instancia del objeto `Type` que representa el tipo exacto de la clase actual. Por ejemplo, dentro del procedimiento de evento anterior `sender.GetType.Name`, devolvería el nombre de la clase del objeto que ha enviado el evento (en nuestro caso, `Button`). Si queremos acceder a los miembros del objeto que envía el evento, deberíamos convertirlo un objeto de una case concreta de forma explícita haciendo una conversión de tipos con los métodos `CType` o `DirectCast`.

```
Dim obj As Button = DirectCast(sender, Button)
```

El segundo argumento es una referencia a una instancia de la clase `EventArgs` o a alguna de sus derivadas y contiene información sobre el evento. Dependiendo del tipo de evento, este argumento será de una clase distinta, con distintos miembros para representar cada una de sus características.

Por último, la cláusula `Handles` indicará qué evento de qué control será el que lance el procedimiento. En el procedimiento anterior, la cláusula indica `Aceptar.Click`, por lo que se lanzará al hacer clic sobre el botón Aceptar. En la cláusula `Handles` pueden coexistir varios eventos de distintos controles separados por comas, con lo que un único procedimiento de evento podrá controlar varios eventos. Por ejemplo un procedimiento que incluyera la cláusula …`Handles Button1.Click, Button2.Click, Button1.DoubleClick` controlaría los enventos `Click` y `DoubleClick` de `Button1` y el evento `Click` de `Button2`.

La lista de eventos a codificar en la aplicación de ejemplo sería la siguiente:

- Al pulsar sobre el botón SALIR (evento `btnSalir.Click`), terminará la aplicación.
- Al pulsar sobre el botón ACEPTAR (evento `btnAceptar.Click`), se cargará en el control `PictureBox` la foto correspondiente a la matrícula introducida en el cuadro de texto. Al mismo tiempo aparecerá la matrícula en la barra de título del formulario.
- Esta misma operación se podrá realizar al pulsar la tecla ENTER en el cuadro de texto (evento `txtMatricula.KeyPress`).

4.1. Evento btnSalir.Click

Existen dos formas de terminar una aplicación Visual Basic. Una de ellas es utilizar la palabra reservada de Visual Basic `End`. `End` terminará la aplicación al tiempo que cerrará todos los archivos abiertos meditante la instrucción `Open` y eliminará las variables de memoria, pero sin ejecutar el método `Dispose()` del formulario. Este método permite incluir código de limpieza que permita cerrar *streams* abiertos, eliminar referencias de objetos, etc. Por ello, en muchas ocasiones es preferible utilizar una llamada directa al método `Close()` del formulario de inicio para terminar la aplicación. El método `Close()` cierra la ventana actual, al ser ésta la única que tiene abierta la aplicación, el programa finalizará.

```
Private Sub btnSalir_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles btnSalir.Click
        Me.Close()
End Sub
```

La palabra clave `Me` hace referencia a la clase actual, en nuestro caso a `frmVisor`.

4.2. Evento btnAceptar.Click

Este evento permitirá visualizar la imagen del vehículo en el `PictureBox`. La imagen de un `PictureBox` se almacena en un objeto de la clase `System.Drawing.Image` que posee un método estático `FromFile()` que sirve para cargar un bitmap almacenado en disco. El método debe recibir como argumento una cadena que contenga la especificación de archivo del mapa de bits. Estableciendo dicha propiedad al literal `Nothing`, desaparecerá la imagen del control.

El problema reside en averiguar cuál es la especificación del archivo que queremos cargar. Podemos utilizar una cadena con la dirección absoluta y concatenar el contenido del cuadro de texto y la extensión `.jpg` de la siguiente forma:

```
imagen = "C:\Carpeta de imágenes\" & txtMatricula.Text & ".jpg"
```

pero requeriría crear dicha carpeta en todas las máquinas en las que se ejecutara la aplicación. Otra solución sería utilizar el objeto `Application` que guarda información sobre la aplicación que se está ejecutando. Dicho objeto tiene como miembro la propiedad `StartUpPath` que devuelve una cadena con el directorio de inicio de la aplicación. Si se guardan las imágenes en una carpeta `Fotos` que cuelgue del directorio de la aplicación, la especificación de archivo podría ser:

```
imagen = Application.StartupPath & "\Fotos\" & txtMatricula.Text & ".jpg"
```

Obsérvese que la aplicación no es ni el archivo `.vb` ni la solución, sino el archivo ejecutable creado en la compilación. Dicho archivo se encuentra situado en la carpeta `bin\Debug` de la solución, por lo que en nuestro caso la propiedad `StartUpPath` devolvería *`carpetaDelProyecto`*`\bin\Debug`. En nuestra aplicación, las los archivos gráficos deberían estar en una carpeta `Fotos` que cuelga de la carpeta de la aplicación. Nótese también que la propiedad no añade la barra invertida final y (`\`) y la carpeta `Fotos`, por lo que es necesario añadirla en la expresión

Suponiendo que la carpeta de la solución sea `CochesUsados`, que la carpeta del proyecto sea `VisorCochesUsados` y que estén almacenados en la carpeta que utiliza Visual Studio por omisión para guardar los proyectos, `Application.StartUp` devolvería,

```
D:\Mis Documentos\Visual Studio
2008\Projects\CochesUsados\VisorCochesUsados\bin\Debug
```

por lo que la expresión `Application.StartupPath & "\Fotos\" & txtMatricula.Text & ".jpg"` devolvería:

```
D:\Mis Documentos\Visual Studio
2008\Projects\CochesUsados\VisorCochesUsados\bin\Debug\Fotos\contenidoDelTextBox.jpg
```

Si en el cuadro de texto se hubiera introducido el texto “BI-1111-A” la expresión anterior haría referencia al archivo `BI-1111-A.jpg` que se encontraría en la carpeta `Fotos` del directorio de la aplicación.

```
Private Sub btnAceptar_Click(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles btnAceptar.Click
    Const titulo = "Vehículos en venta" 'Título base del formulario
    Dim imagen As String 'Especificación del archivo que contiene la imagen
    imagen = Application.StartupPath & "\Fotos\" & txtMatrícula.Text & ".jpg"
    picFoto.Image = Image.FromFile(imagen)
    Me.Text = titulo & " - " & txtMatrícula.Text
End Sub
```

Este procedimiento de evento utiliza una variable de tipo cadena `imagen` para guardar la ubicación y el nombre del archivo gráfico. Además cambia el título del formulario (propiedad `Me.Text`) añadiendo al título original (“Vehículos en venta”, almacenado en la constante `titulo`) la matrícula del coche que se está visualizando (recordad el heurístico de visibilidad y orientación inmediata).

4.3. Evento txtMatricula.KeyPress

La detección de la tecla ENTER en un control se puede controlar con los eventos `KeyPress`, `KeyDown` o `KeyUp`, aunque como los dos últimos se utilizan sobre todo para detectar la pulsación de teclas especiales (Alt, Ctrl, teclas de función, teclas de cursor, etc.) se utilizará `KeyPress`. El segundo argumento de `KeyPress` pertenece a la clase `KeyPressEventArgs`, uno de cuyos miembros es la propiedad `KeyChar`, que representa el carácter de la tecla pulsada; para detectar

si se ha pulsado la tecla ENTER habrá que comprobar si `KeyChar` coincide con el código de la tecla (`Keys.Enter`).

Si esto ocurre, la acción a realizar será la misma que la del evento `Click` del botón aceptar. Esto se puede conseguir, además de copiando el código, llamando al evento `Click` de forma directa (mediante una llamada a `btnAceptar_Click(sender,e)`), creando un nuevo procedimiento que cargue la imagen (por ejemplo un procedimiento `Sub MostrarVehiculo()`), o utilizando el método `PerformClick` de la clase `Button`. (`btnAceptar.PerformClick`)

```
Private Sub txtMatrícula_KeyPress(ByVal sender As Object, _
            ByVal e As System.Windows.Forms.KeyPressEventArgs) _
            Handles txtMatrícula.KeyPress
    If AscW(e.KeyChar) = Keys.Enter Then
        MostrarVehículo()
        'También se podrían haber utilizado las siguientes llamadas:
        '
        'Aceptar.PerformClick()
        '
        'Aceptar_Click(sender, e)
    End If
End Sub

Private Sub MostrarVehículo()
    Const titulo = "Vehículos en venta" 'Título base del formulario
    Dim imagen As String 'Especificación del archivo que contiene la imagen
    imagen = Application.StartupPath & "\Fotos\" & txtMatrícula.Text & ".jpg"
    picCoche.Image = Image.FromFile(imagen)
    Me.Text = titulo & " - " & txtMatrícula.Text
End Sub

Private Sub btnAceptar_Click(ByVal sender As System.Object, _
          ByVal e As System.EventArgs) Handles btnAceptar.Click
    MostrarVehículo()
End Sub
```

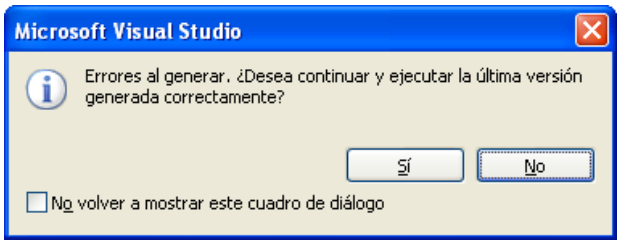
Otra forma de ejecutar las mismas acciones cuando se pulsa la tecla ENTER es convertir el botón “Aceptar” en el botón predeterminado del formulario. Para ello habría que modificar la propiedad `AcceptButton` del formulario y asignarla al botón Aceptar. El botón quedaría rodeado de un recuadro negro y se activaría al pulsar la tecla ENTER desde cualquier parte del formulario.

5. Ejecución y depuración

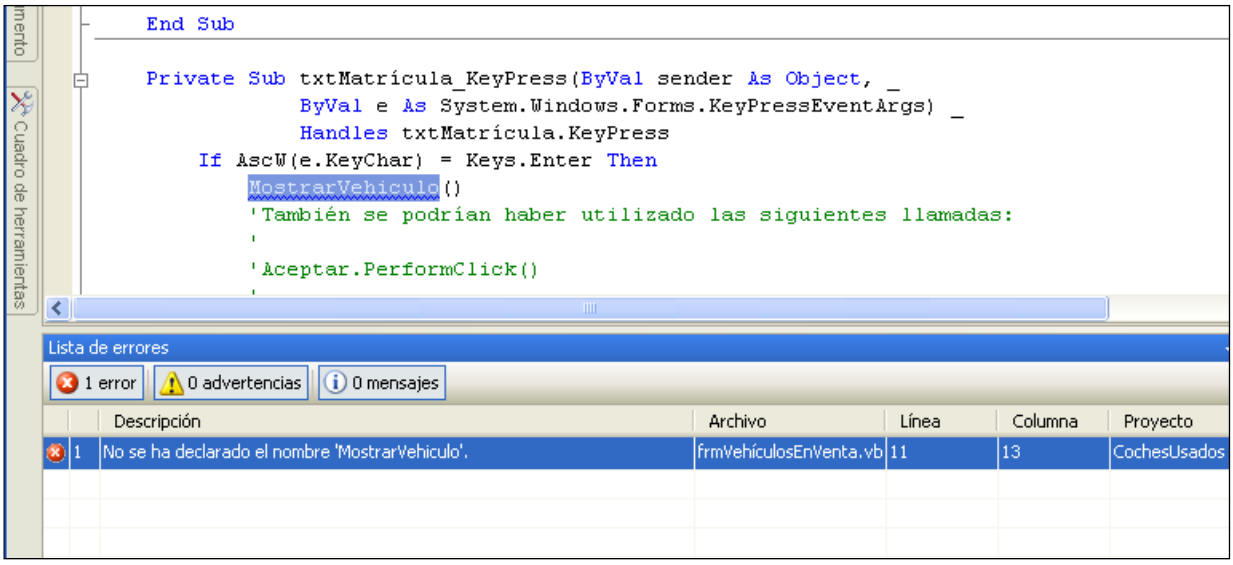
5.1. Generación de código y control de errores sintácticos

La ejecución del proyecto se realizará mediante la opción INICIAR del menú DEPURAR (o pulsando la tecla F5). Se generará entonces el ejecutable de la solución y comenzará la ejecución en modo de depuración hasta encontrar un punto de interrupción, un error o hasta que se detenga la ejecución de forma manual. También es posible ejecutar la solución paso a paso (teclas F10 o F11).

Si se detecta un error sintáctico, no se podrá generar la solución y aparecerá una ventana indicándolo.



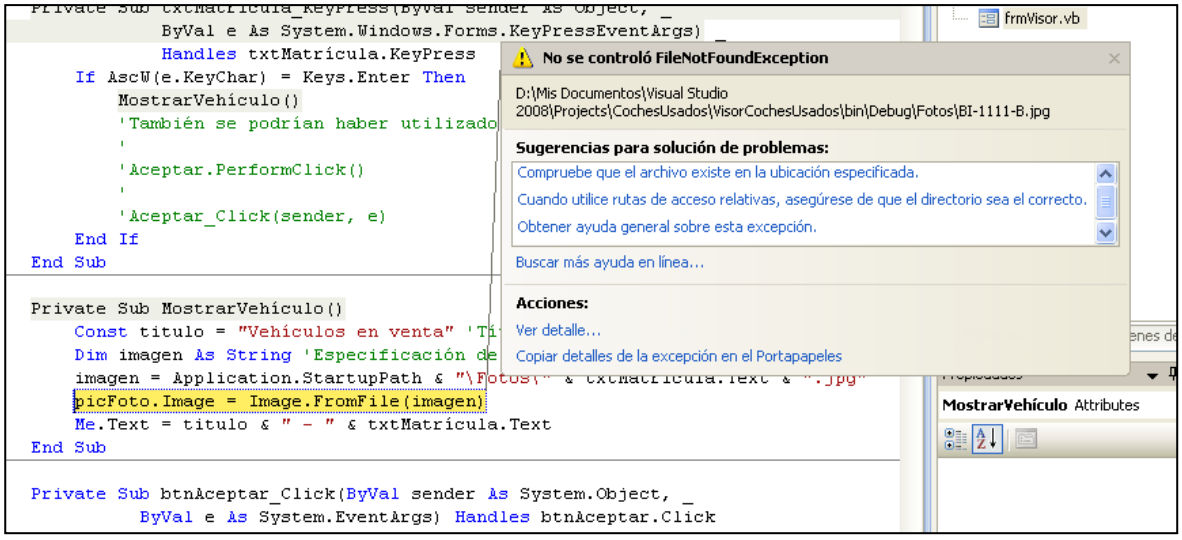
Si se continúa la ejecución se ejecutará la última solución generada. Si se interrumpe, aparecerá en la ventana LISTA DE DE ERRORES información del error y se podrá ir a la línea que ha producido el error.

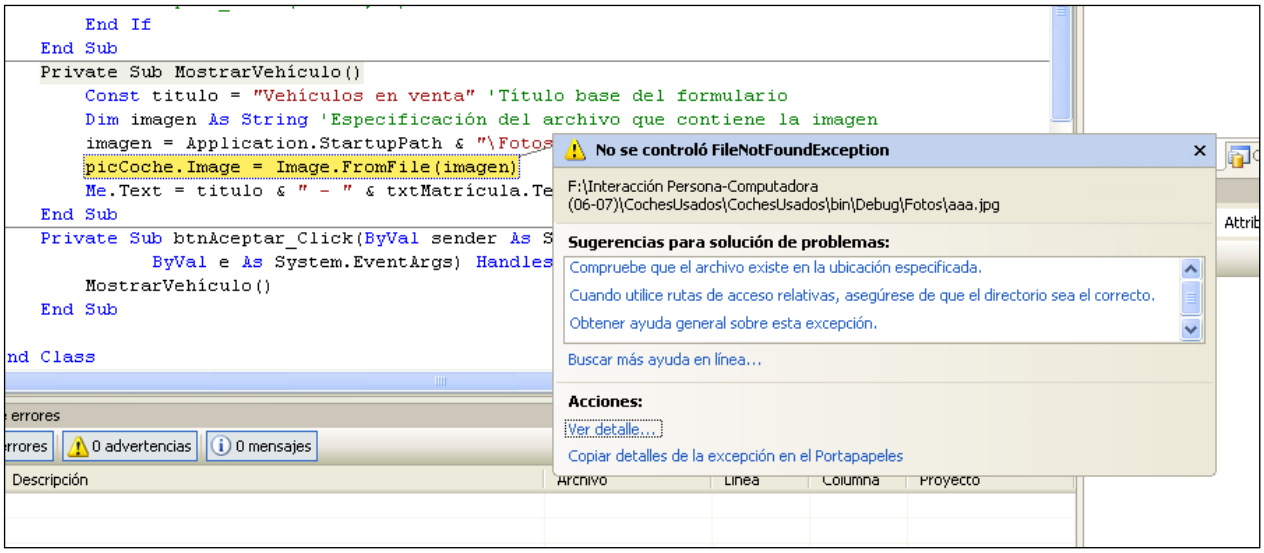


Obsérvese que la llamada dentro del procedimiento es a `MostrarVehiculo()` (sin acento), mientras que el procedimiento declarado se llama `MostrarVehículo()` (con acento).

5.2. Errores en tiempo de ejecución

Cuando el entorno detecta un error en tiempo de ejecución, aparecerá una ventana indicando el error.

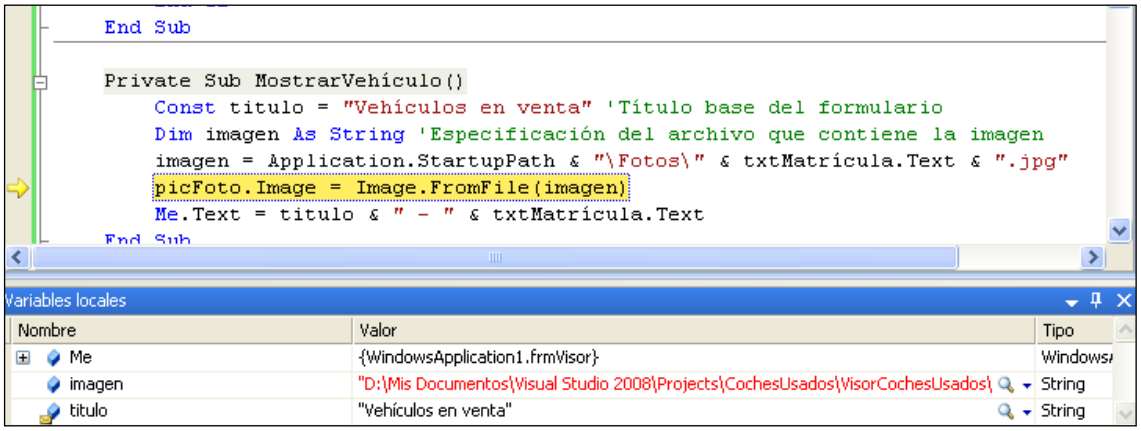




En la ventana de código aparecerá señalada la línea que ha producido el error. En la ventana del error aparecerá el tipo de error que se ha producido (en este caso una excepción del tipo `FileNotFoundException`, es decir, archivo no encontrado), una explicación más detallada de la instrucción que ha producido el error (en este caso el nombre del archivo que no se ha encontrado), algunas sugerencias para controlar el problema, y la posibilidad de ver el detalle de la excepción.

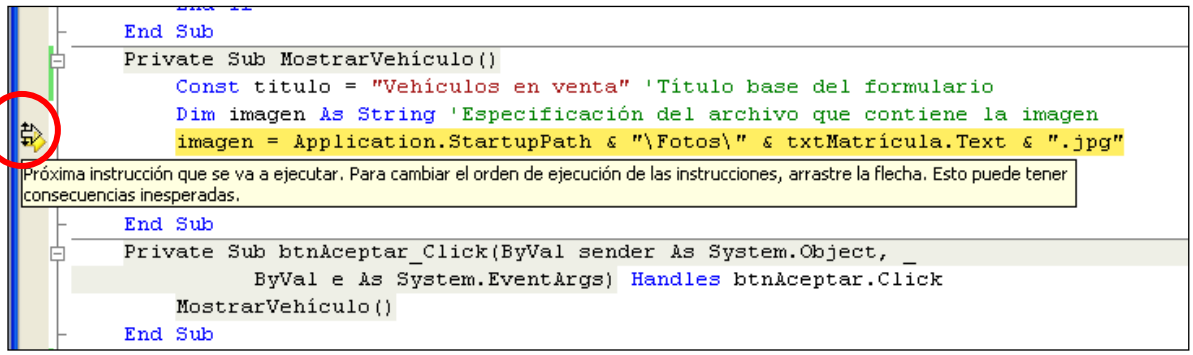
5.3. Depuración del código en tiempo de ejecución

Una vez cerrada la ventana de error, será posible corregir la línea o líneas de código erróneas y continuar la ejecución seleccionando la opción CONTINUAR del menú DEPURAR o pulsando la tecla F5. También es posible ver el valor de las variables sacando la ventana AUTOMÁTICO que permite visualizar las variables utilizadas en esa línea y en la línea de código anterior, o la ventana de VARIABLES LOCALES, que permite visualizar las variables locales del procedimiento actual. Para seleccionar esas ventanas hay que acceder a la opción VENTANAS del menú DEPURAR.



Se puede modificar el valor de las variables pulsando dos veces sobre la variable, modificando su valor y pulsando la tecla ENTER.

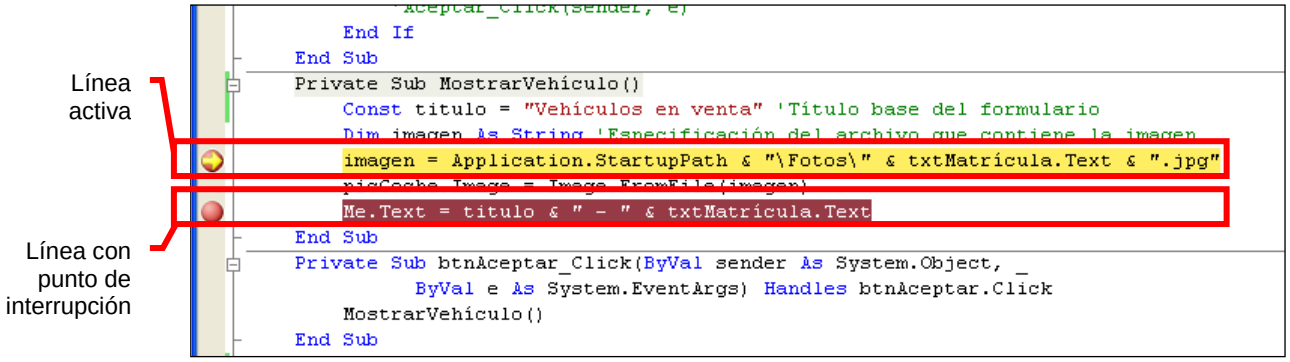
También se puede modificar la ejecución normal del programa y desplazar la línea actual a otra posición. Para ello hay que pulsar sobre la flecha que indica la línea actual y desplazar la flecha a otra nueva línea.



Mientras se está en el modo de depuración interrumpido (y siempre antes de arrancar la depuración), es posible ejecutar las instrucciones una a una, haciendo un seguimiento preciso del flujo del programa. Podemos avanzar paso a paso de las siguientes formas:

- Paso a paso por instrucciones (F11). Ejecuta la siguiente instrucción.
- Paso a paso por procedimientos (F10). Ejecuta la siguiente instrucción, pero si se trata de una línea de código con una llamada a una función, la ejecuta totalmente y salta a la siguiente instrucción dentro del ámbito actual.
- Paso a paso para salir (SHIFT+F11). Termina la función actual y lleva el cursor a la última llamada.

Otra posibilidad de depuración es forzar la detención de la ejecución colocando puntos de interrupción en las instrucciones ejecutables. Para establecer un punto de interrupción, simplemente hay que pulsar en el margen de la ventana de edición de código en cualquier línea que contenga una instrucción ejecutable. Aparecerá entonces un punto rojo indicando el *breakpoint* y la ejecución se detendrá en ese punto. Al llegar la ejecución del programa a ese punto, el programa se detendrá, aparecerá en el título de la ventana de Visual Studio la aclaración "(Depurando)" y se marcará como línea activa la línea donde esté colocado el punto de interrupción.



Con el programa detenido, podremos analizar las variables, modificar el código, cambiar la línea activa o ejecutar paso a paso el programa tal y como se ha indicado anteriormente.

6. Control estructurado de excepciones

Cuando un método detecta un problema en tiempo de ejecución, se genera una excepción (un error). Si el método no está preparado para controlarla, se manda la excepción al siguiente método en la pila de llamadas y así sucesivamente hasta que se encuentra un controlador de excepciones que la capture o se llega al final de la pila de llamadas con lo que aparecerá un mensaje indicando la excepción que se ha producido y finalizando la aplicación.

Mediante el control estructurado de excepciones el código que potencialmente pude producir un error se encapsula y captura las excepciones para las que ha sido programado mediante las instrucciones `Try..Catch..Finally`.

```
Try
   [ instrucciones protegidas ]
[ Catch [ excepción [ As tipoExcepción ] ]
   [ instrucciones Catch ] ]
[ Exit Try ]
...
[ Finally
   [ instrucciones Finally ] ]
End Try
```

El bloque `Try..End Try` se encarga de encerrar el código que puede producir una excepción. Si se produce algún error dentro de ese bloque, Visual Basic revisará todas las instrucciones `Catch` del bloque para comprobar si alguna de ellas hace referencia a la excepción que se ha producido. De ser así, ejecutará las instrucciones de la cláusula `Catch` correspondiente. Por último, la cláusula `Finally` se ejecutará tanto si se produce como si no se produce una excepción.

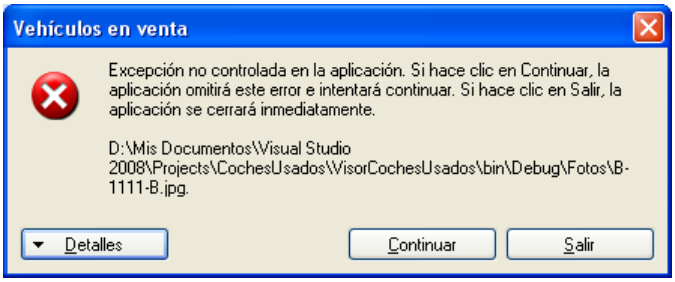
Un bloque `Try` puede tener tantas cláusulas `Catch` como excepciones se deseen controlar. Cuando se produce una excepción se establece el objeto global `Err` y se genera una instancia de la clase `Exception` o de alguna de sus derivadas. Visual Basic comprobará todas las cláusulas `Catch` hasta encontrar una que coincida con el tipo de excepción generada, ejecutándose el código asociado. De no encontrar ninguna, se pasará esa excepción al siguiente método en la pila de llamadas.

El objeto `Err` contiene información sobre el error producido, como las propiedades `Err.Number`, con el código del error que se ha producido, `Err.Source` con el nombre del objeto o la aplicación donde se ha producido, o `Err.Description` con la descripción del error.

La clase `Exception` tiene, entre otras, las propiedades `Source`, con información de la aplicación o el objeto que la ha producido, `Message`, con información sobre la excepción o `StackTrace` con una lista de la pila de llamadas cuando se produjo la excepción actual. Además cada una de las clases derivadas de `Exception` tiene su propiedades características (en la ayuda de Visual Studio, aparece la jerarquía de clases de `Exception` con sus clases derivadas, propiedades, métodos, etc.)

6.1. Controlar excepciones en la aplicación de ejemplo

La aplicación que se está desarrollando puede generar una excepción cuando no existe ninguna imagen del vehículo cuya matrícula se ha introducido al ejecutar la instrucción `FromFile()`. La excepción producida pertenecería a la clase `System.IO.FileNotFoundException`, y, si no se controla, en la aplicación definitiva aparecería el siguiente mensaje por pantalla y terminaría la aplicación.



Un nuevo procedimiento `MostrarVehículo()` con el controlador de excepciones podría quedar de la siguiente forma.

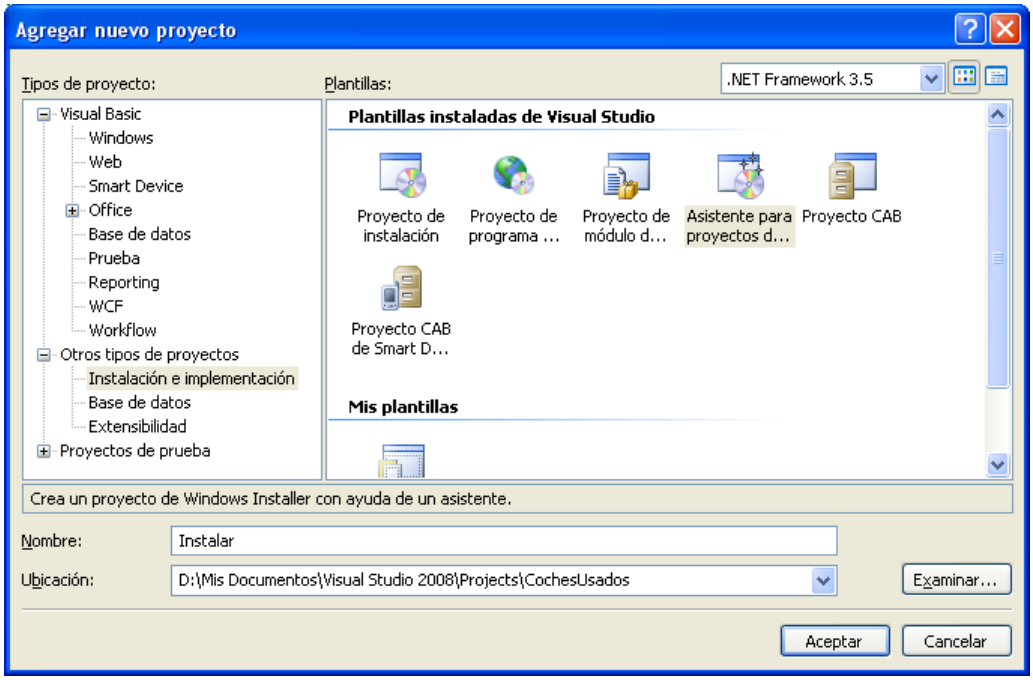
```
Private Sub MostrarVehículo()
    Const titulo = "Vehículos en venta" 'Título base del formulario
    Dim imagen As String 'Especificación del archivo que contiene la imagen
    imagen = Application.StartupPath & "\Fotos\" & txtMatrícula.Text & ".jpg"
    Try
        'Si no hay errores, se carga la imagen
        'y se establece el título del formulario
        picCoche.Image = Image.FromFile(imagen)
        Me.Text = titulo & " - " & txtMatrícula.Text
    Catch e As System.IO.FileNotFoundException
        'Si no se encuentra el archivo aparece un cuadro de diálogo
        'indicándolo, se vacía el cuadro de texto y
        'se restablece el título de la ventana
        MessageBox.Show("El vehículo con matrícula " & txtMatrícula.Text & _
                        " no tiene imagen asociada.", _
                        "Vehículos en venta", _
                        MessageBoxButtons.OK, _
                        MessageBoxIcon.Exclamation)
        'Se vacía el texto del cuadro de texto
        txtMatrícula.Text = String.Empty
        'Se vacía el PictureBox
        picCoche.Image = Nothing
        'Se restablece el título de la ventana
         Me.Text = titulo
    Catch e As Exception
        'Si se produce cualquier otro error aparece un cuadro de diálogo
        'indicando la circunstancia, el código de error,
        'la descripción del error y la pila de llamadas.
        'Además se vacía el cuadro de texto y
        'se restablece el título de la ventana
        MessageBox.Show("No se ha podido cargar la imagen." & _
                        ControlChars.CrLf & _
                        "Error: " & Err.Number & " - " & e.Message & _
                        " " & e.StackTrace & ".", _
                        "Vehículos en venta", MessageBoxButtons.OK, _
                        MessageBoxIcon.Exclamation)
        'Se vacía el PictureBox
        picCoche.Image = Nothing
        'Se vacía el texto del cuadro de texto
        txtMatrícula.Text = String.Empty
        'Se restablece el título de la ventana
        Me.Text = titulo
    Finally
        'De cualquier forma el foco vuelve al cuadro de texto
         txtMatrícula.Select()
    End Try
End Sub
```

7. Distribución de la aplicación

A diferencia de las aplicaciones Win32, una aplicación .NET no precisa instalar y registrar los componentes y referencias de la solución para distribuir la aplicación en otros equipos: un simple `XCopy` o un FTP del directorio de la aplicación permitirá ejecutarla sin problemas, siempre y cuando se encuentre instalado .NET Framework en los ordenadores donde se debe ejecutar la aplicación.

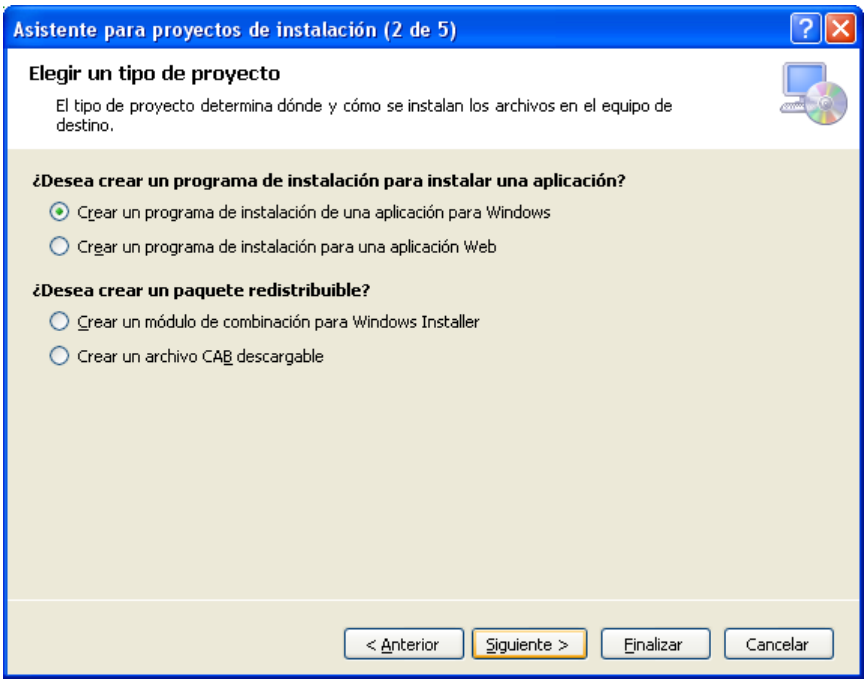
A pesar de esto, Visual Studio tiene entre sus plantillas de proyectos aplicaciones para instalar y distribuir aplicaciones. Este tipo de aplicaciones generarán un archivo `Setup.exe` que permitirá instalar la aplicación en otros equipos[2].

Para añadir un proyecto de instalación a nuestra aplicación, hay que seleccionar la opción NUEVO PROYECTO del menú ARCHIVO/AGREGAR. En la lista de plantillas habría que elegir entre los tipos de proyecto OTROS TIPOS DE PROYECTOS y PROYECTOS DE INSTALACIÓN E IMPLEMENTACIÓN y elegir la plantilla ASISTENTE PARA PROYECTOS DE INSTALACIÓN.

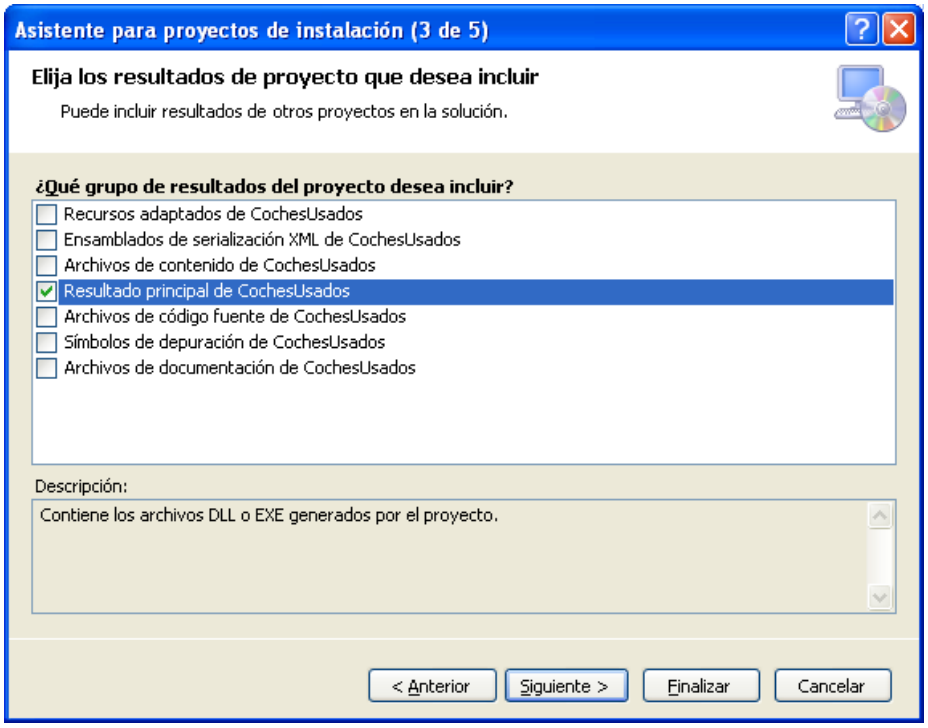


Se abrirá entonces el asistente y, después de la pantalla de bienvenida, habrá que seleccionar el tipo de instalación que deseamos hacer, que en nuestro caso será una instalación de una aplicación para Windows.

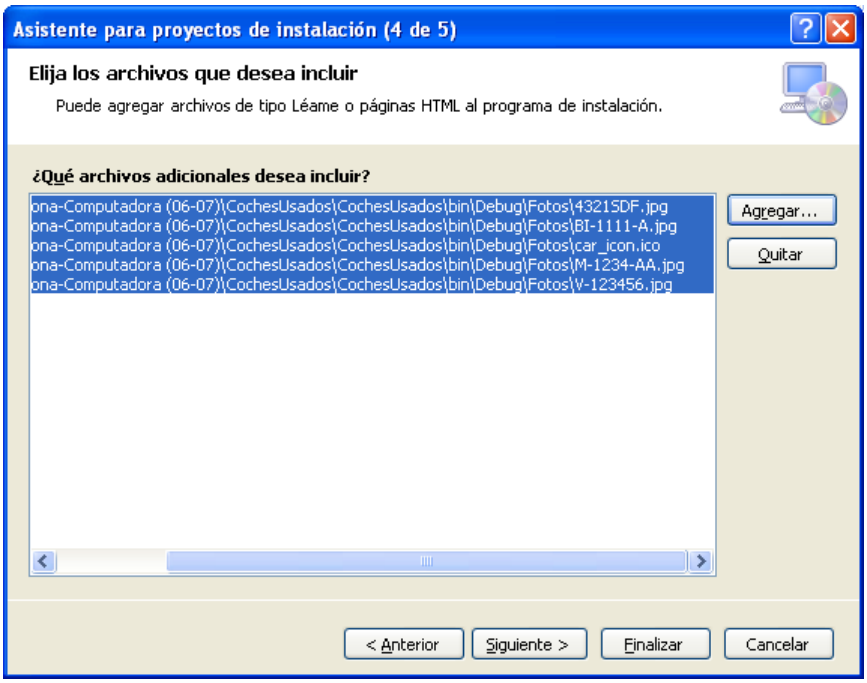
[2] Esa aplicación también es una aplicación .NET por lo que todavía será necesario tener instalado .NET Framework. Para más información sobre cómo instalar también .NET Framework, se puede acceder a las páginas `http://support.microsoft.com/?scid=kb%3Bes%3B324733&x=12&y=11` o `http://msdn.microsoft.com/es-es/library/ms994427(en-us).aspx`. También es posible obtener más información sobre los proyectos de instalación en `http://www.elguille.info/colabora/puntoNET/jmbeas_instaladores/jmbeas_InstaladoresNET.htm`



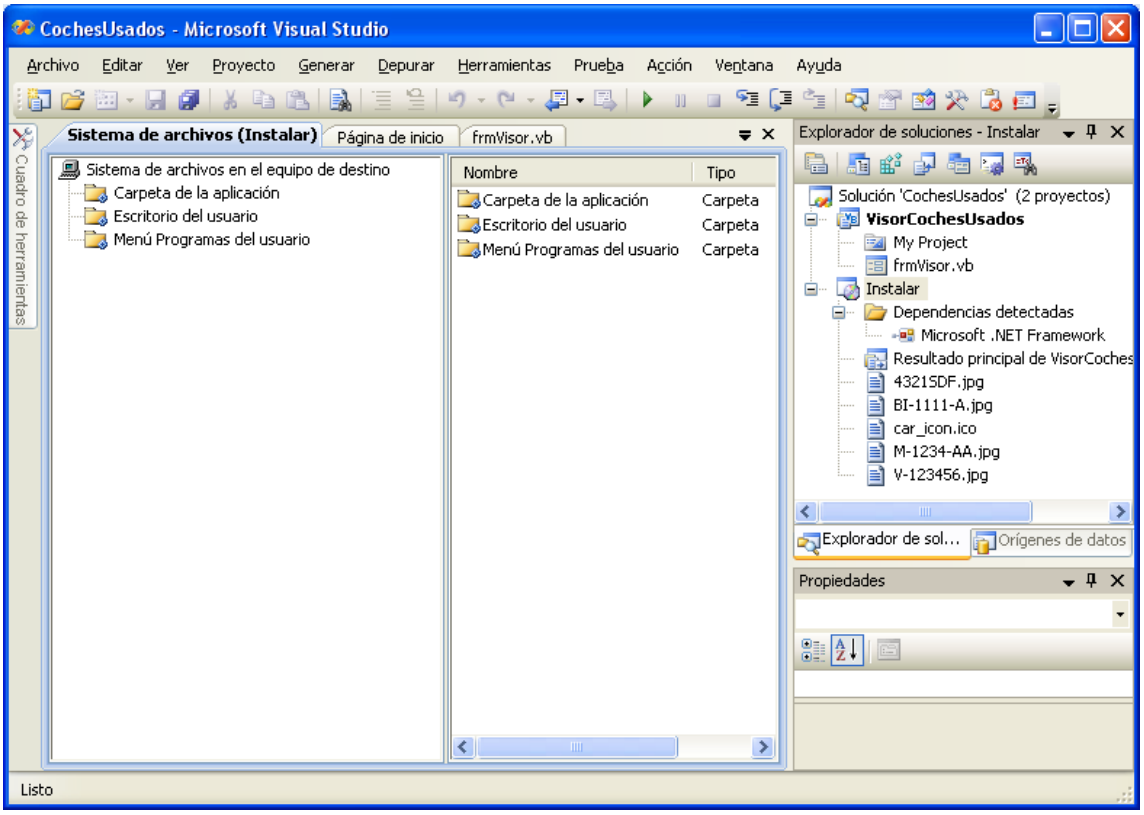
La siguiente pantalla permite indicar que elementos de la aplicación se incluirán en la instalación. El RESULTADO PRINCIPAL DE COCHESUSADOS es el que incluye el archivo ejecutable de la instalación y las DLL necesarias.



El paso siguiente será indicar si se desea incluir algún archivo extra en la instalación. Aquí se deberían incluir, si son necesarios, archivos gráficos, archivos de tipo `readme.txt`, las bases de datos, etc. En nuestro caso puede ser interesante incluir las imágenes de los vehículos y el icono del programa.



Después de la pantalla de despedida y pulsar el botón FINALIZAR se creará un nuevo proyecto. En la ventana del Explorador de soluciones aparecerá una nueva carpeta en la solución con el nombre de `Instalar` (este es el nombre que hemos dado al proyecto en la ventana Agregar nuevo proyecto) que será el nuevo proyecto. La ventana de los diseñadores de los proyectos de instalación es distinta de las del diseñador de proyectos. Lo que aparecerá es el sistema de archivos que creará la instalación, tanto en la carpeta de la aplicación, como en el escritorio del usuario como en el menú programas del usuario. En la ventana de la izquierda aparecen las carpetas y en el de la derecha su contenido. Es posible crear nuevas carpetas y añadir nuevos contenidos a los creados inicialmente mediante los menús contextuales.



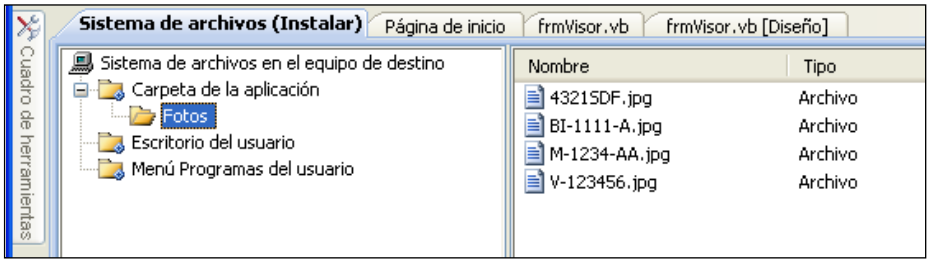
El proyecto de instalación, por omisión, creará una proyecto llamado `Instalar`, que se instalará en la carpeta de ARCHIVOS DE PROGRAMA con ese nombre y no creará ninguna entrada en el menú Programas. Para cambiar esto es necesario modificar las propiedades del proyecto.

Para cambiar las propiedades del proyecto es necesario marcar el proyecto Instalar del Explorador de proyecto. Algunas de las propiedades son:

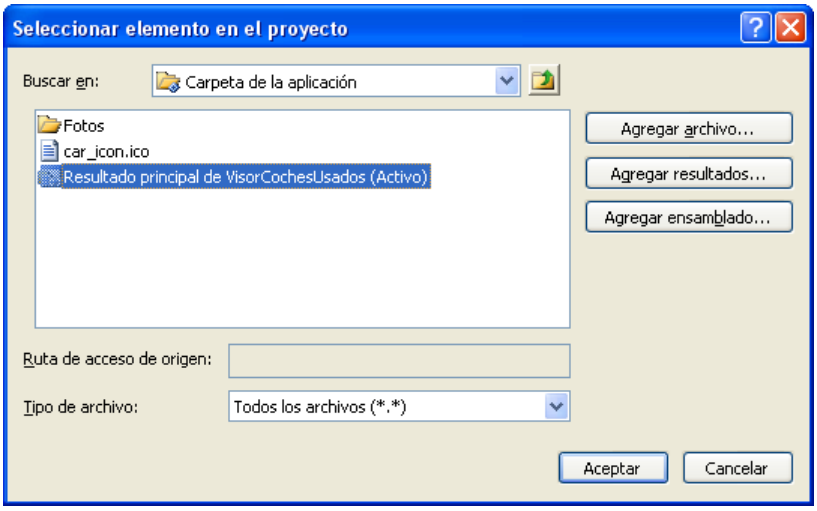
- Autor, con el nombre del creador del programa. Aparece al pasar el cursor por el archivo .msi de la aplicación.
- Description, es la descripción del programa que aparecerá en el archivo .msi
- Manufacturer, nombre de la compañía que será de forma predeterminada el nombre de la carpeta donde se instalará la carpeta de la aplicación.
- ProductName, nombre del producto que aparecerá en el título de la ventana del programa de instalación.
- Title, título del programa que aparecerá en el archivo .msi.

La carpeta de la aplicación representa la ubicación de la carpeta donde se ubicará el programa. Por omisión se instalará en `[ProgramFilesFolder][Manufacturer]\[ProductName]`, pero podemos cambiar el destino mediante la propiedad `DefaultLocation`.

En esta carpeta también se podrían crear subcarpetas para determinadas funciones. Por ejemplo, podríamos crear una subcarpeta `Fotos` y guardar allí los archivos gráficos con las fotos, tal y como aparece en la figura anterior.



En MENÚ PROGRAMAS DEL USUARIO podemos crear la entrada que aparecerá en el menú de Inicio de Windows. Inicialmente esa carpeta permanece vacía, por lo que si deseamos crear un nuevo elemento, deberemos crear un nuevo acceso directo al resultado principal de la aplicación. Para ello, se deberá pulsar con el botón derecho en la ventana de la derecha y selecciona la opción CREAR NUEVO ACCESO DIRECTO del menú contextual.



En la ventana resultante deberemos seleccionar Resultado principal de CochesUsados de la Carpeta de la aplicación. Pulsando sobre el nuevo acceso directo accedemos a sus propiedades:

- Name, con el nombre que aparecerá en el menú Programas.
- Description, con el texto que aparecerá al pasar el cursor por la entrada del menú.
- Icon, con el icono asociado a la aplicación. El icono debe estar incluido en la carpeta de la aplicación, y podemos añadirlo seleccionando la carpeta de la aplicación y añadiendo el archivo .ico.

Una vez modificadas las propiedades, seleccionando GENERAR del menú secundario del proyecto `Instalar`, se generará el programa de instalación. En la carpeta indicada al crear el proyecto, en la subcarpeta Debug, aparecerán dos archivos: `Instalar.msi`, el programa de instalación que se puede utilizar si la máquina cliente no tiene instalado Windows Installer, y `setup.exe`, el programa que es necesario utilizar si se tiene instalado Windows Installer. En condiciones normales, cualquiera de los dos últimos permitirá instalar la aplicación.