

Módulo 3: Desarrollo de aplicaciones Windows

- Uso del diseñador de Visual Studio 2008
- Controles de Windows Forms
- Trabajo con controles
- Trabajo con imágenes y gráficos
- Despliegue de aplicaciones

Módulo 3 - Desarrollo de aplicaciones Windows

En este módulo, conoceremos las partes generales y más importantes del entorno de desarrollo rápido Visual Studio 2008 para la programación de aplicaciones con este lenguaje de la familia .NET.

Veremos las partes principales del entorno, también veremos como desarrollar nuestros propios controles Windows, aprenderemos a trabajar con imágenes y gráficos con Visual Basic y finalmente, conoceremos como desplegar nuestras aplicaciones desarrolladas en Visual Basic 2008.

Las partes que forman parte de este módulo son las siguientes:

Capítulo 1

- Uso del diseñador de Visual Studio 2008

Capítulo 2

- Controles de Windows Forms

Capítulo 3

- Desarrollo de controles

Capítulo 4

- Trabajo con imágenes y gráficos

Capítulo 5

- Despliegue de aplicaciones

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 1: Diseñador de Visual Studio 2008

- Cuadro de herramientas
- Explorador de base de datos
- Explorador de soluciones
- Propiedades
- Menús y barras de botones
- Otras consideraciones

Introducción

Cuando nos encontramos con Visual Studio 2008 por primera vez, saltan a la vista, algunos de los cambios más importantes de este novedoso entorno de desarrollo de aplicaciones Windows.

Para un desarrollador, familiarizarse con el entorno de Visual Studio 2008 es una tarea que no debe entrañar una complejidad excesivamente grande. Como nos ocurre a todos los que nos encontramos delante de un nuevo entorno de trabajo, lo único que se requiere es constancia y práctica, mucha práctica. Sin embargo, si usted es ya un desarrollador habitual de otros entornos de desarrollo, notará que sus avances van a ser significativos en muy poco tiempo.

Nota:

Si está utilizando Visual Basic 2008 Express para seguir este curso debe saber que este entorno está especializado en desarrollar aplicaciones Windows con Visual Basic 2008, aunque podrá usar controles y librerías escritas en otros lenguajes de la plataforma .NET .

Módulo 3 - Capítulo 1

- 1. Cuadro de herramientas
- 2. Explorador de base de datos
- 3. Explorador de soluciones
- 4. Propiedades
- 5. Menus y barra de botones
- 6. Otras consideraciones

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 1: Diseñador de Visual Studio 2008

Cuadro de herramientas

- Explorador de base de datos
- Explorador de soluciones
- Propiedades
- Menús y barras de botones
- Otras consideraciones

Módulo 3 - Capítulo 1

1. Cuadro de herramientas

El cuadro o barra de herramientas de Visual Studio 2008, nos permite utilizar los distintos componentes que .NET Framework pone a nuestra disposición. Dentro de Visual Studio 2008 tenemos una gran cantidad de controles dispuestos en diferentes categorías.

En la figura 1 podemos ver la barra de herramientas de Visual Studio 2008.

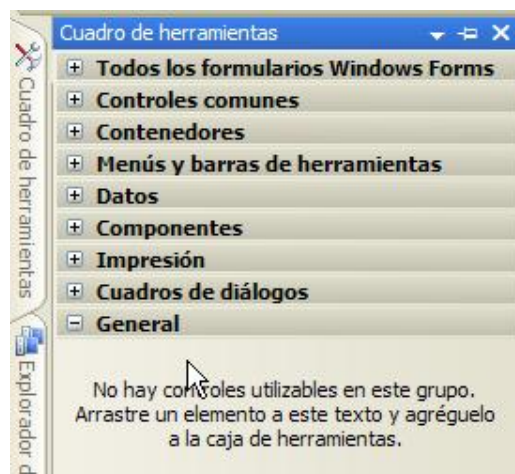


Figura 1

El *Cuadro de herramientas*, lo localizará en la parte izquierda del entorno *Visual Studio 2008*.

Cuando iniciamos un nuevo proyecto con Visual Studio 2008, el cuadro de herramientas queda relleno con los controles que podemos utilizar en el proyecto.

Si abrimos un formulario Windows, los controles quedan habilitados para que los podamos insertar en el formulario Windows. En la figura 2 se muestra la barra de herramientas con los controles preparados para ser insertados en el formulario Windows.



Toolbox de Visual Studio 2008 con controles Windows preparados para ser insertados en el formulario Windows

Figura 2

Nota:

Para insertar un control en un formulario Windows, se requiere que el formulario Windows sobre el que deseamos insertar un control, esté abierto. Una vez que está abierto, bastará con realizar una de las tres siguientes acciones para insertar un control al formulario:

- *Hacer doble clic sobre un control del cuadro de herramientas*
- *Hacer clic sobre un control del cuadro de herramientas, y sin soltar el botón del mouse, arrastrarlo sobre el formulario*
- *Hacer clic sobre un control del cuadro de herramientas, y luego hacer clic sobre el formulario y arrastrar para marcar una zona que cubrirá nuestro control y soltar el ratón*

El control quedará entonces insertado dentro del formulario.

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 1: Diseñador de Visual Studio 2008

- Cuadro de herramientas
- Explorador de base de datos
- Explorador de soluciones
- Propiedades
- Menús y barras de botones
- Otras consideraciones

Módulo 3 - Capítulo 1

2. Explorador de base de datos

Si ha sido lo suficientemente observador cuando se explicaban los detalles del cuadro o barra de herramientas, y ha prestado especial atención a las figuras o a las ventanas del entorno de desarrollo de Visual Studio 2008, quizás haya notado que en la parte izquierda además de la solapa *cuadro de herramientas*, aparece otra solapa de nombre *explorador de base de datos*.

Desde esta solapa, un programador puede acceder a diferentes recursos del sistema. El principal y más importante recurso, es el que tiene que ver con las conexiones con bases de datos, ya sean Microsoft Access, Microsoft SQL Server o cualquier otra fuente de datos.

En la figura 1 puede observar la solapa *Explorador de base de datos* extendida con parte de sus opciones.

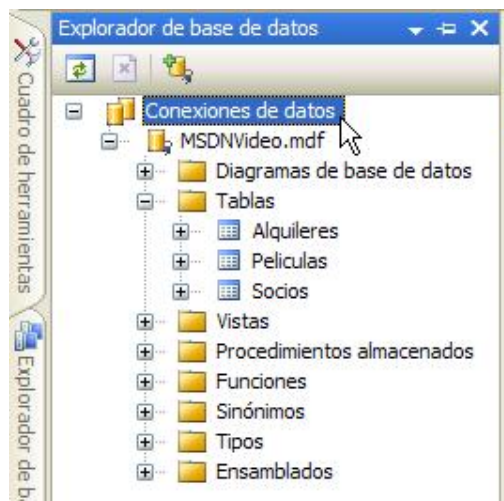


Figura 1

Conectando con una base de datos Microsoft Access a través de OLE DB

Para muestra un botón, y dado el carácter práctico de este tutorial, aprenderá a crear una conexión con cualquier base de datos, en nuestro caso de ejemplo una base de datos Microsoft Access, para poder utilizarla fácilmente en nuestra aplicación Windows.


Haga clic sobre el botón representado por la siguiente imagen . En este instante, se abrirá una nueva ventana como la que se muestra en la figura 2.



Figura 2

Por defecto, la ventana **Agregar conexión** queda preparada para establecer una conexión con una fuente de datos de origen de datos *OLE DB*, por lo que si nuestra intención es establecer una conexión con otra fuente de datos, entonces deberemos hacer clic sobre el botón **Cambiar...** que se indica en la figura 3.

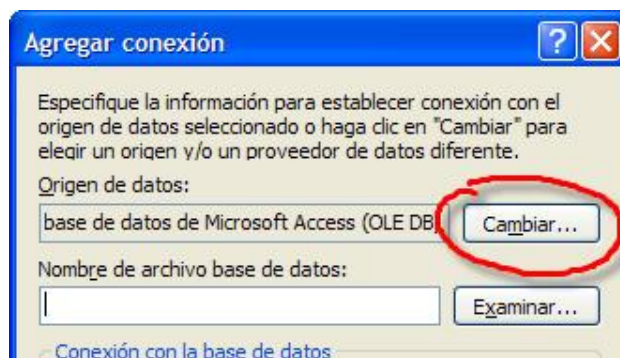


Figura 3

De esta manera, podemos indicar el origen de acceso a datos que necesitamos para establecer la conexión con nuestra fuente de datos, y que en nuestro ejemplo, no será un proveedor de SQL Server, por lo que el origen de datos OLE DB es válido para nosotros.

Una vez que hemos hecho clic sobre el botón **Cambiar...**, nos aseguramos por lo tanto, que nuestro origen de datos es *base de datos de Microsoft Access (OLE DB)*, como se indica en la figura 4.

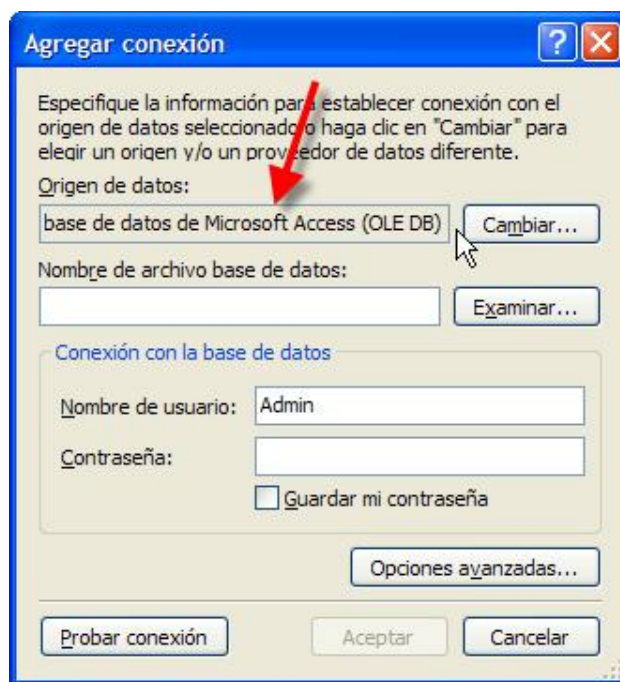


Figura 4

FAQ:

¿Puedo utilizar el proveedor OLE DB en lugar del proveedor de SQL Server para conectar con una base de datos SQL Server?

Con OLE DB, puede acceder a fuentes de datos SQL Server u otras fuentes de datos como Microsoft Access, sin embargo, si utiliza SQL Server 7.0, SQL Server 2000, SQL Server 2005 ó SQL Server 2008, se recomienda el uso del proveedor de SQL Server, que es un proveedor de acceso a datos nativo que aumenta el rendimiento de nuestras aplicaciones con SQL Server. Sólo si utiliza una versión de SQL Server anterior a SQL Server 7.0, deberá utilizar necesariamente el proveedor de acceso a datos OLE DB.

Una vez que hemos seleccionado el proveedor de acceso a datos, nos centraremos en la opción **Nombre de archivo base de datos** como se muestra en la figura 5.



Figura 5

Para agregar el fichero de base de datos a la conexión, presionaremos el botón **Examinar...** y seleccionaremos el fichero de base de datos de nuestro disco duro. De esta manera, la base de datos quedará indicada en la conexión y tan sólo deberemos probar nuestra conexión pulsando el botón **Probar conexión** como se indica en la figura 6.

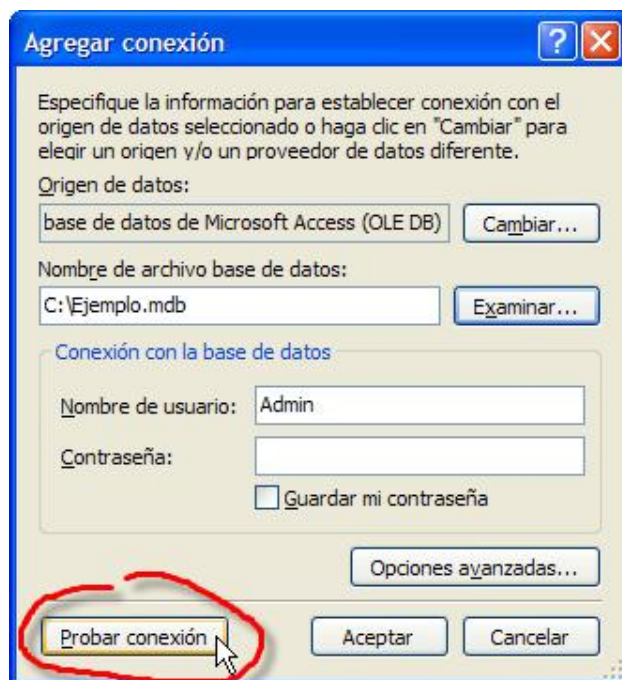


Figura 6

Si la prueba de conexión se ha realizado satisfactoriamente, recibiremos un mensaje en pantalla afirmativo como el que se indica en la figura 7.



Figura 7

A tener en cuenta:

En este ejemplo, la conexión con la base de datos Microsoft Access, no tiene ningún tipo de usuario y contraseña. Tenga en cuenta que en la parte identificada como **Conexión con la base de datos**, podríamos indicar el usuario y contraseña si fuera necesario.

En este punto, tan sólo deberemos presionar sobre el botón **Aceptar** para que la base de datos con la que hemos establecido la conexión, quede ahora insertada en la ventana del **Explorador de base de datos** como se muestra en la figura 8.

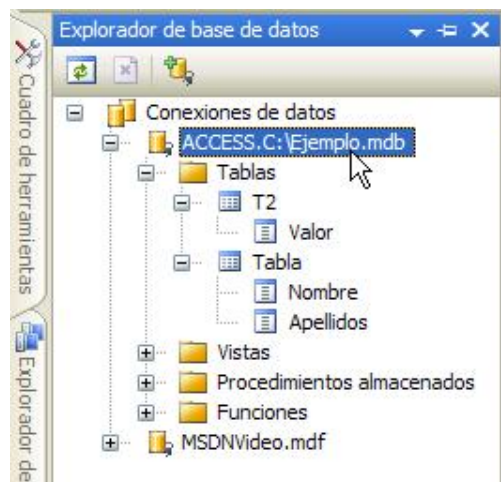


Figura 8

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 1: Diseñador de Visual Studio 2008

- Cuadro de herramientas
- Explorador de base de datos
- Explorador de soluciones**
- Propiedades
- Menús y barras de botones
- Otras consideraciones

Módulo 3 - Capítulo 1

3. Explorador de soluciones

El *Explorador de soluciones* lo podemos encontrar en la parte derecha de nuestro entorno de desarrollo.

Una solución se compone de proyectos y éstos, de recursos y objetos. Por lo general, una solución contendrá un proyecto, pero podemos encontrarnos con más de un proyecto dentro de una misma solución.

Sin embargo, estos conceptos son muy sencillos de comprender y controlar, y para nada debe hacernos pensar que esto es algo complejo que nos costará mucho tiempo dominar.

En la figura 1, podemos observar el explorador de soluciones de *Visual Studio 2008*.

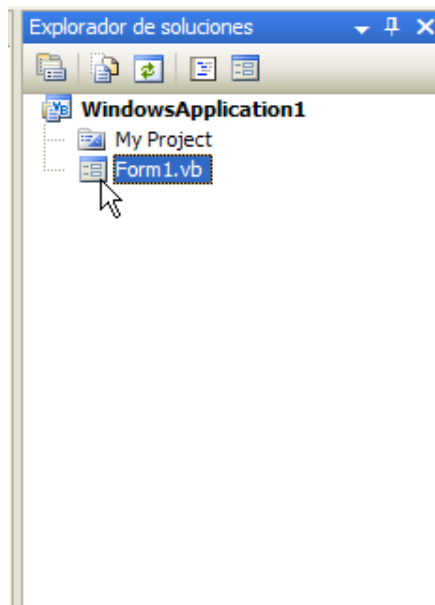


Figura 1

Si queremos añadir un nuevo formulario al proyecto, lo haremos presionando con el botón secundario en cualquier parte de la ventana del explorador de soluciones, pero si esa pulsación la hacemos en alguno de los objetos que contiene el proyecto, no podremos hacerlo, ya que el IDE de Visual Studio 2008 muestra un menú diferente según el objeto presionado, por ejemplo si queremos añadir un nuevo proyecto, podemos hacerlo presionando con el botón secundario del mouse sobre la "solución".

Nota:

Para abrir un recurso de la solución, basta con situarnos en el recurso determinado, por ejemplo un formulario Windows de nombre Form1.vb y hacer doble clic sobre él. El recurso se abrirá automáticamente en Visual Studio 2008. Además, en Visual Studio 2008 sabremos en todo momento sobre qué recurso estamos trabajando en un momento dado.

» [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 1: Diseñador de Visual Studio 2008

- Cuadro de herramientas
- Explorador de base de datos
- Explorador de soluciones
- Propiedades**
- Menús y barras de botones
- Otras consideraciones

Módulo 3 - Capítulo 1

4. Propiedades

La ventana de propiedades la encontraremos en la parte derecha y más abajo de la ventana **Explorador de soluciones** en nuestro entorno de desarrollo.

Esta ventana nos permitirá acceder a las propiedades de los objetos insertados en nuestros formularios Windows, como se muestra en la figura 1.

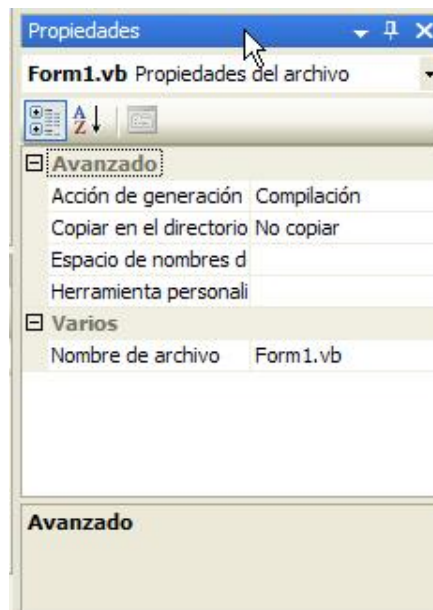


Figura 1

Para acceder a las propiedades de un determinado control, deberemos seleccionar el control en el formulario Windows y acudir a la ventana **Propiedades**, o bien, seleccionar el control en el formulario Windows y presionar la tecla **F4**.

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 1: Diseñador de Visual Studio 2008

- Cuadro de herramientas
- Explorador de base de datos
- Explorador de soluciones
- Propiedades
- Menús y barras de botones
- Otras consideraciones

Módulo 3 - Capítulo 1

5. Menús y barra de botones

Respecto a los menús y barra de botones, son muchas las opciones que tenemos disponibles, tal como podemos comprobar en la figura 1. Las barras de botones son configurables, además de que podemos elegir las que queremos que se muestren de forma permanente en el entorno de desarrollo de Visual Studio 2008.

Algunas de las barras de botones se mostrarán automáticamente según las tareas que estemos realizando, por ejemplo, cuando estamos en modo depuración o diseñando las tablas de una base de datos.

Con el contenido de los menús ocurre lo mismo, según el elemento que tengamos seleccionado se mostrarán ciertas opciones que sea relevantes para ese elemento del IDE de Visual Studio 2008.



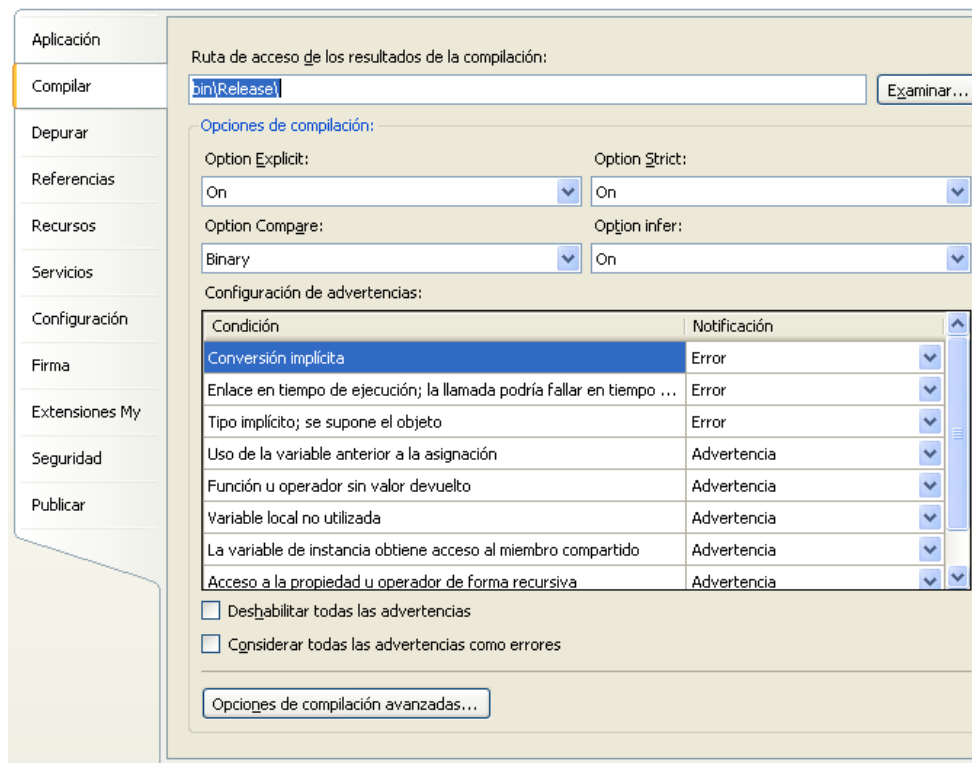
Los menús y barras de botones de Visual Studio 2008

Figura 1

Algunas de las opciones que tenemos en los menús también las podemos conseguir usando los menús contextuales (el mostrado al presionar con el botón secundario del mouse), y como es de esperar, también serán diferentes según el elemento sobre el que hemos presionado.

Por ejemplo, para configurar el proyecto actual, podemos elegir la opción **Propiedades** del menú **Proyecto** o bien presionar con el botón secundario del mouse sobre el proyecto mostrado en el Explorador de soluciones.

Al seleccionar las propiedades del proyecto, tendremos una nueva ventana desde la que podemos configurar algunas de las características de nuestro proyecto. En la figura 2, tenemos esa ventana de propiedades del proyecto, en la que podemos apreciar que está dividida según el tipo de configuración que queremos realizar, en este caso concreto las opciones de generación o compilación del proyecto.



Propiedades del proyecto sobre la que se trabaja en Visual Basic 2008

Figura 2

Como vemos en la figura 2, existen sin embargo multitud de opciones y apartados diferentes relacionados todos ellos con nuestra solución. Otro de los apartados destacables, es el apartado denominado **Publicar**.

Aún así, éste es el corazón o parte fundamental que debemos controlar a la hora de desarrollar una aplicación o a la hora de gestionar una solución, porque dentro de esta ventana, se resume buena parte de los menús y barra de botones del entorno de *Visual Studio 2008*.

De todos los modos, tendremos la oportunidad de ver más adelante, algunos usos de algunas de las opciones de la barra de botones del entorno.

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 1: Diseñador de Visual Studio 2008

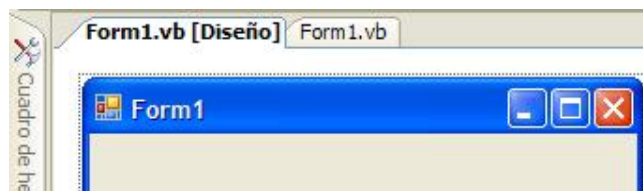
- Cuadro de herramientas
- Explorador de base de datos
- Explorador de soluciones
- Propiedades
- Menús y barras de botones
- Otras consideraciones

Módulo 3 - Capítulo 1

6. Otras consideraciones

El desarrollador que haya utilizado previamente otros entornos de desarrollo distinto a los de la familia de Visual Studio .NET, encontrará muy interesantes algunos de los cambios incorporados en *Visual Studio 2008*. Al principio, quizás se encuentre un poco desorientado, pero rápidamente y gracias a su experiencia en otros entornos de desarrollo, se acostumbrará al cambio. Entre algunos de estos cambios, destacaría los siguientes:

- En *Visual Studio 2008*, acceder a los objetos de nuestra aplicación es mucho más fácil. Dentro del entorno, observaremos que se van creando diferentes solapas que nos permite acceder y localizar los recursos con los que estamos trabajando de forma rápida. En la figura 1 podemos observar justamente esto que comento.

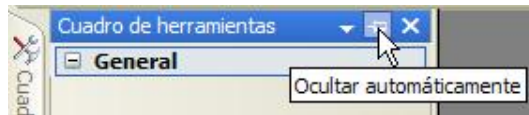


Solapas de los objetos abiertos en Visual Studio 2008

Figura 1

- *Visual Basic 2008* permite, hacer un **Stop & Go** (editar y continuar), de nuestras aplicaciones, es decir, pausar la ejecución de una aplicación en modo depuración y modificar los valores o propiedades que deseemos y continuar ejecutándola. Esta opción que los programadores de Visual Basic 6 utilizan con mucha frecuencia en el desarrollo de sus aplicaciones, se ha mantenido en *Visual Basic 2008*, pero no en Visual Studio .NET 2002 y Visual Studio .NET 2003. Si por alguna razón, debe trabajar con alguno de estos entornos, debe saber que esta opción no está disponible para las versiones comentadas.

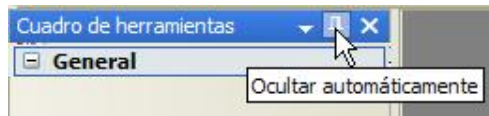
- Otra característica que debemos conocer de nuestro entorno de desarrollo, es la capacidad de anclar o fijar una ventana de las comentadas anteriormente o de permitir que se haga visible cuando acercamos el puntero del mouse sobre ella. Esta opción es la que puede verse en la figura 2.



Opción de ocultar o mostrar la ventana seleccionada en Visual Studio 2008

Figura 2

Nótese que al presionar el icono indicado en la figura 2, haremos que esta ventana quede fija en el entorno de desarrollo. Cuando pulsamos este icono, la ventana queda fija y queda representado por un icono como el que se muestra en la figura 3.



Icono para ocultar o mostrar la ventana seleccionada cuando se encuentra en modo anclado

Figura 3

- Algo que *oculta* el entorno de *Visual Studio 2008* por defecto, son las denominadas *clases parciales*. Se trata de una nueva característica añadida a .NET 2.0 y por lo tanto a Visual Basic 2008, que permite *separar* o *partir* una clase en varias porciones de código.

La explicación ruda de esto, es que el programador puede tener dos ficheros de código fuente independientes, que posean el mismo nombre de clase. Para indicar que pertenece a la misma clase, ésta debe tener la palabra clave ***Partial*** como parte de su definición para indicar que es una clase parcial. Un ejemplo que aclare esto es el siguiente:

```

Código

Public Class Class1

    Public Function Accion1() As Integer

        Return 1

    End Function

End Class

Código

Partial Public Class Class1

```

```

Public Function Accion2() As Integer

    Return 2

End Function

End Class

```

El comportamiento de la clase es el de una única clase, por lo que su declaración y uso es como el de cualquier clase normal, tal y como se indica en el siguiente código:

```

Código

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button1.Click

        Dim MiClase As New Class1

        MessageBox.Show(MiClase.Accion2.ToString() & vbCrLf &
MiClase.Accion1.ToString())

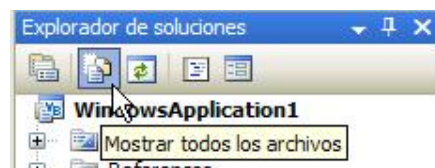
    End Sub

End Class

```

De todas las maneras, el entorno nos oculta muchas veces las clases parciales de un aplicación.

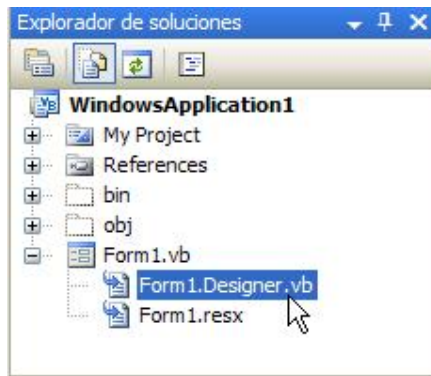
Para ello, presionaremos la opción **Mostrar todos los archivos** de la ventana *Explorador de soluciones* como se indica en la figura 4.



Icono u opción para mostrar todos los archivos del proyecto

Figura 4

De esta manera, podremos acceder a los archivos y recursos del proyecto, incluidas las clases parciales como se indica en la figura 5. En el archivo Form1.Designer.vb estará el código utilizado por el diseñador de formularios de Windows Forms, en el que se incluye la declaración de todos los controles y controladores de eventos que hemos definido en nuestro proyecto.



Clase parcial mostrada en los archivos del proyecto

Figura 5

A tener en cuenta:

Cuando se genera un proyecto con Visual Studio 2008, el entorno genera diferentes clases parciales, como por ejemplo la que se genera para un formulario.

▶ [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

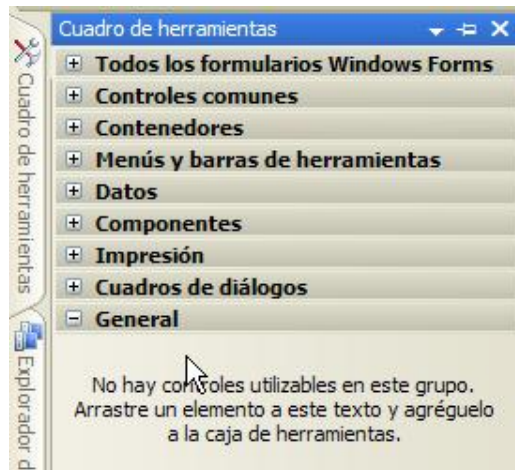
Lección 2: Controles de Windows Forms

- Datos
- Componentes
- Controles Comunes
- General
- Otras consideraciones

Introducción

Dentro del entorno de desarrollo de *Visual Studio 2008*, nos encontramos un enorme conjunto de librerías y controles que podemos utilizar en nuestras aplicaciones Windows. Dependiendo del tipo de aplicación que llevemos a cabo, el entorno habilitará los controles correspondientes para cada tipo de aplicación. En nuestro caso, nos centraremos en los controles más habituales de Windows, e indicaremos como utilizarlos en nuestros desarrollos.

En nuestro entorno de desarrollo, encontraremos diferentes grupos de controles o componentes dispuestos de ser utilizados. En la figura 1 encontraremos los grupos de controles y componentes más habituales.



Grupos de controles en Visual Studio 2008

Figura 1

Estos controles se dividen en los grupos representados en la figura anterior. A continuación veremos los más representativos.

Módulo 3 - Capítulo 2

- 1. Datos
- 2. Componentes
- 3. Controles comunes
- 4. General
- 5. Otras consideraciones

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 2: Controles de Windows Forms

Datos

- Componentes
- Controles Comunes
- General
- Otras consideraciones

Módulo 3 - Capítulo 2

1. Datos

El grupo *Datos* corresponde con el grupo que tiene relación directa con los componentes de acceso a datos, como se muestra en la figura 1.



Controles de Datos en Visual Studio 2008

Figura 1

Para muchos desarrolladores, los controles, componentes y métodos de acceso a datos, contiene dentro de sí un especial misterio, es como el *Santo Grial* de la programación. Casi siempre nos atascamos ahí, siempre en el mismo sitio. Pero no se preocupe ni lo más mínimo por ello, aprenderemos a utilizarlos a base de práctica, y lo que es más importante, los dominaremos rápidamente. Solo como curiosidad y por ahora, le presentaré uno de los componentes más destacables en *Visual Studio 2008*, por su semejanza con otro muy utilizado en "otros" entornos de desarrollo, estoy hablando del control y componente ***BindingNavigator*** que usaremos frecuentemente en nuestras aplicaciones con acceso a fuentes de datos.

Este control insertado en un formulario Windows, es el que se puede ver en la figura 2.



Control BindingNavigator insertado en un formulario Windows en Visual Studio 2008

Figura 2

Como puede observar, este control, tiene un aspecto muy similar al del famoso *Recordset* de Visual Basic 6 o al *DataNavigator* de Borland. Lógicamente, este control tiene un aspecto mucho más vistoso y moderno, pero es uno de los controles estrella de *Visual Basic 2005* y que también están incluidos en *Visual Studio 2008*, ya que en Visual Studio .NET 2002 y Visual Studio .NET 2003 no existía este control en el entorno.

Desde Visual Studio 2005 y por lo tanto, desde Visual Basic 2005, tenemos la posibilidad de trabajar con el control **BindingNavigator**.

Comunidad dotNet:

Visual Studio 2008 le proporciona un amplio conjunto de controles y componentes así como un no menos completo conjunto de clases que le facilita al desarrollador las tareas de programación requeridas. Sin embargo, existen contribuciones gratuitas y otras de pago, que el programador puede utilizar según lo requiera. A continuación le indico el que a mi modo de ver es el lugar más representativo de este tipo de contribuciones a la Comunidad de desarrolladores .NET.

» [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 2: Controles de Windows Forms

- Datos
- Componentes**
- Controles Comunes
- General
- Otras consideraciones

Módulo 3 - Capítulo 2

2. Componentes

Windows Forms incluye un conjunto de componentes muy nutrido y variado. Algunos de estos componentes, han sido mejorados y otros ampliados. En la figura 1 podemos observar estos componentes.

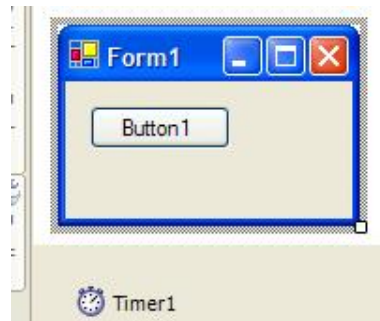


Componentes de Windows Forms

Figura 1

Los componentes son *como controles no visibles*, o dicho de otra forma, son controles que realizan ciertas tareas, pero no tienen un interfaz que mostrar, como puede ser el caso de un botón o una caja de textos.

Por ejemplo, el componente *Timer* nos permite recibir una notificación cada x tiempo, pero no muestra nada al usuario de nuestra aplicación. Si hacemos doble clic sobre el componente *Timer* para insertarlo en el formulario, éste quedará dispuesto en la parte inferior del formulario como se indica en la figura 2.



Control Timer insertado en un formulario de Windows Forms

Figura 2

Este tipo de componentes no son visibles en tiempo de ejecución.

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 2: Controles de Windows Forms

- Datos
- Componentes
- Controles Comunes**
- General
- Otras consideraciones

Módulo 3 - Capítulo 2

3. Controles comunes

Con este nombre, se aglutinan los controles más generales y variados que podemos utilizar en nuestras aplicaciones Windows. Sería algo así, como el resto de controles y componentes no contenidos en ninguna de las secciones que hemos visto anteriormente, aunque esto no es siempre así. Por esa razón, si encuentra dos controles o componentes iguales en dos o más secciones, no lo tenga en consideración.

Digamos que en esta solapa se aglutinan por lo tanto, los controles que utilizaremos con más frecuencia.

En la figura 1 podemos observar los controles y componentes citados.



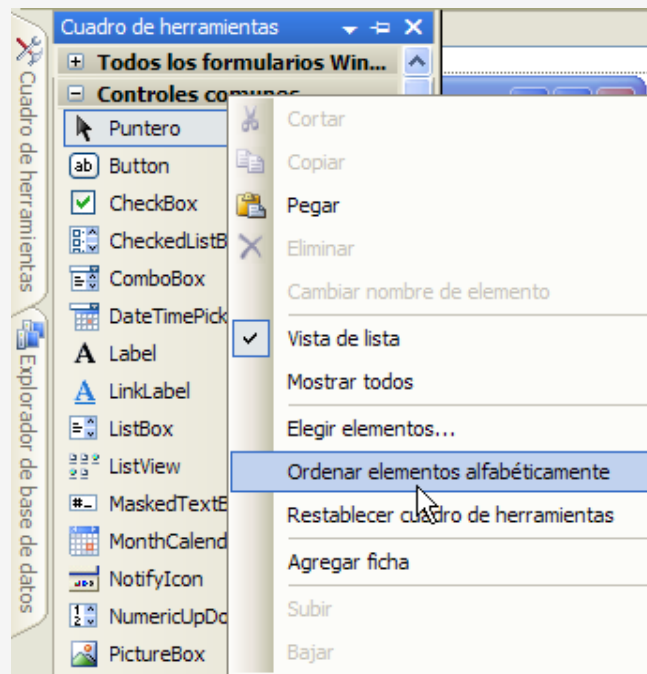
Controles Windows Forms en Visual Studio 2008

Figura 1

Debido a la cantidad de controles y componentes de los distintos grupos del Cuadro de herramientas, podemos usar el siguiente truco para que nos resulte más fácil su localización.

Truco:

*Como puede observar, a veces cuesta localizar un control debido a la enorme cantidad de controles que hay. Para ordenarlos, puede arrastrar y soltar los controles y componentes en la barra de herramientas o bien, si quiere hacer una ordenación por orden alfabético, puede hacer clic con el botón secundario del mouse sobre una determinada sección de controles y seleccionar la opción **Ordenar elementos alfabéticamente** como se indica en la siguiente figura*



Los controles y componentes de esa sección quedarán ordenados alfabéticamente.

Lo más destacable para el desarrollador habituado a otros entornos, es que aquí veremos una gran cantidad de controles que nos resultarán muy familiares. Controles como: **Label**, **PictureBox**, **TextBox**, **Frame** que ahora pasa a llamarse **GroupBox**, **CommandButton** que ahora pasa a llamarse **Button**, **CheckBox**, **OptionButton** que ahora pasa a llamarse **RadioButton**, **ComboBox**, **Listbox**, **HScrollbar**, **VScrollbar**, **Timer**, etc.

Pero además tenemos muchos otros que no son tan habituales en todos los entornos de desarrollo diferentes de Visual Studio .NET. Controles que proporcionan nuevas y ventajosas características a la hora de desarrollar aplicaciones con Visual Basic 2008.

Entre estos controles, podemos encontrar el control **PrintDocument** y **PrintPreviewControl**, para imprimir y realizar vistas preliminares, **ErrorProvider**, **WebBrowser**, **FolderBrowserDialog**, **ToolTip** para aportar *tooltips* a nuestros controles, **TrackBar**, **NumericUpDown**, **SplitContainer**, **MonthCalendar**, **DateTimePicker**, etc.

Cada uno de los controles, tiene unas características y cualidades determinadas. Sólo a base de práctica, aprenderemos a utilizarlos y lo único que debemos saber, es cuál de ellos utilizar en un momento dado.

El abanico de controles y componentes es lo suficientemente amplio como para poder abordar con ellos, cualquier tipo de proyecto y aplicación Windows que nos sea demandada.

» [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

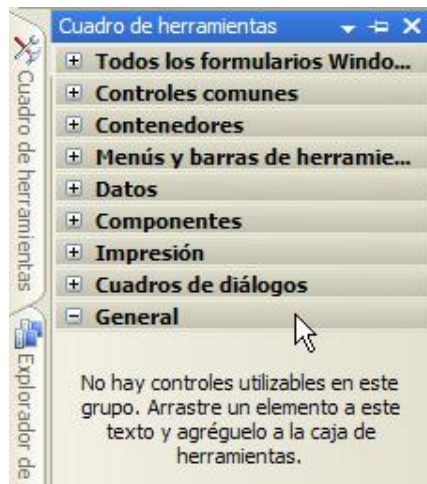
Lección 2: Controles de Windows Forms

- Datos
- Componentes
- Controles Comunes
- General**
- Otras consideraciones

Módulo 3 - Capítulo 2

4. General

Esta sección es como el cajón desastre, un lugar dónde podemos insertar otros controles o componentes desarrollados por terceros, por ejemplo.



Sección General en Visual Studio 2008

Figura 1

Esta sección de todos los modos, la puede utilizar un desarrollador en muchos casos. Por ejemplo, los desarrolladores que desean arrastrar y soltar aquí los controles y componentes que más utiliza o los que utiliza en un determinado proyecto.

Otro caso de ejemplo es cuando se trabaja con controles o componentes similares desarrollados por dos empresas diferentes que queremos tener localizados o separados para no mezclarlos. En otras circunstancias, tampoco es raro encontrarse con controles o componentes con iconos similares, por lo que aclararse cuál es el que nos interesa puede ser una tarea obligada.

Aún así, otra de las posibilidades con la que nos podemos encontrar para utilizar esta sección es la de tener que utilizar un control o componente circunstancialmente en un momento dado, y por eso, que no deseemos añadir este control o componente a otra sección como la de *Controles comunes* por ejemplo.

Utilice por lo tanto esta sección como lo considere oportuno.

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 2: Controles de Windows Forms

- Datos
 - Componentes
 - Controles Comunes
 - General
- Otras consideraciones

Módulo 3 - Capítulo 2

5. Otras consideraciones

La sección *General* nos indica un repositorio de ámbito y carácter general, sin embargo, el desarrollador puede querer ordenar su propio repositorio o sección de controles y componentes.

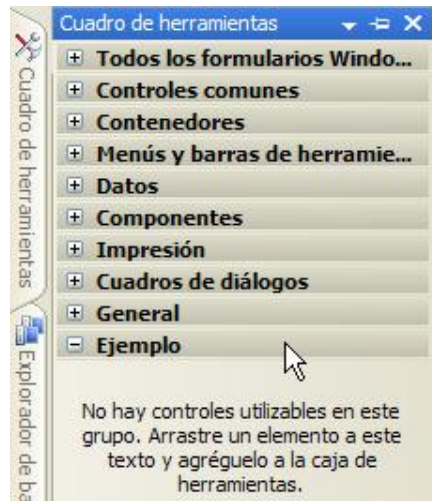
Manipulando el Cuadro de herramientas

Para ello, nos posicionaremos en la barra de herramientas y presionaremos el botón secundario del mouse sobre la parte gris de la barra de herramientas desplegada y seleccionaremos la opción *Agregar ficha* del menú emergente, como se muestra en la figura 1.



Figura 1

Cuando seleccionamos esta opción, aparecerá una caja de texto en la barra de herramientas dónde podremos escribir el nombre que consideremos oportuno, como se muestra en la figura 2.



Personalización de un grupo de controles y componentes en Visual Studio 2008

Figura 2

Si se hace la siguiente pregunta, ¿cómo cambiar el nombre de una sección ya creada o una existente?, sepa que deberá realizar los siguiente pasos.

Haga clic con el botón secundario del mouse sobre la sección sobre la que desea cambiar el nombre y seleccione la opción **Cambiar nombre de ficha** como se muestra en la figura 3.

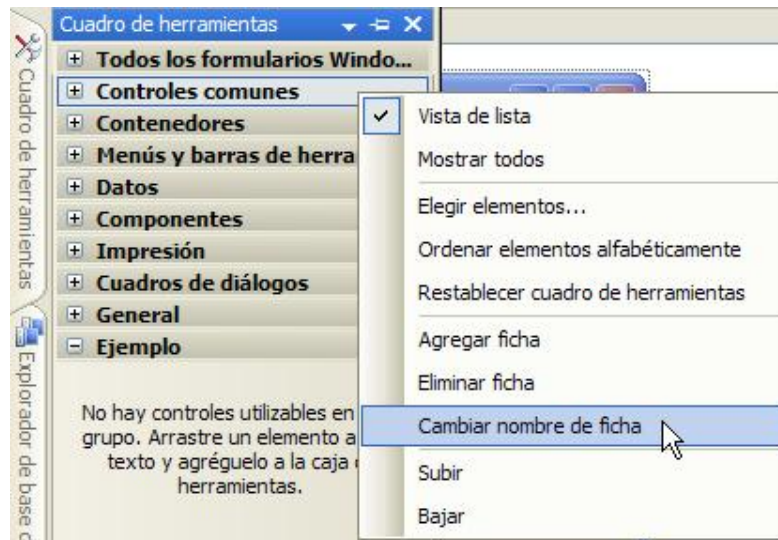


Figura 3

De igual forma, puede cambiar también el nombre de los controles o componentes insertados.

Para hacer eso, haga clic con el botón secundario del mouse sobre un control o componente y seleccione la opción **Cambiar nombre de elemento** como se muestra en la figura 4.

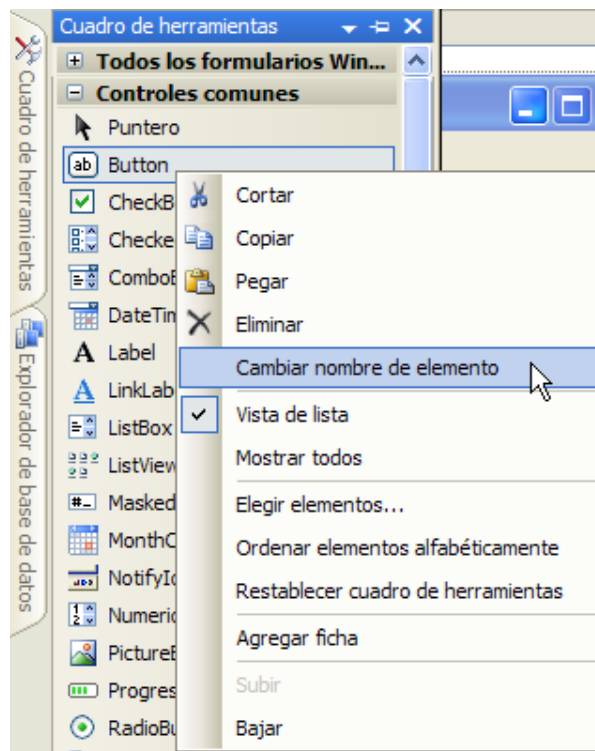


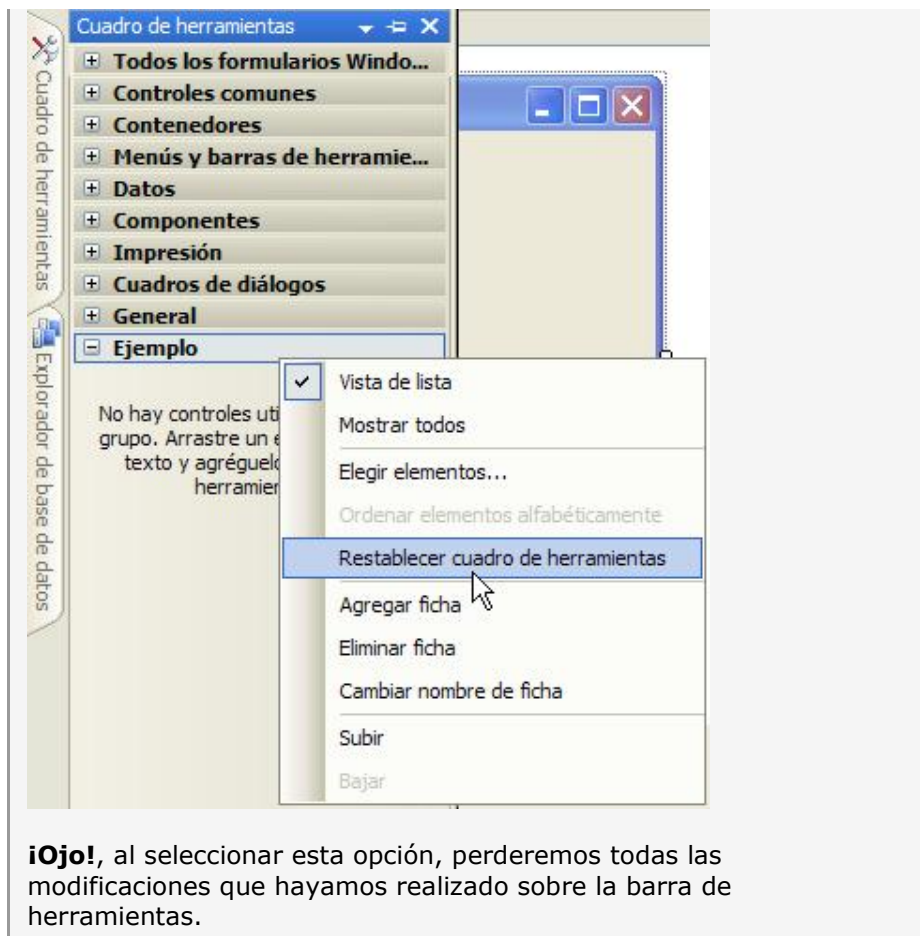
Figura 4

Visual Basic 2008, nos proporciona un amplio conjunto de opciones de personalización del entorno de trabajo, para que se ajuste a las exigencias de los desarrolladores.

FAQ:

¿Qué ocurre si me equivoco personalizando mi barra de herramientas?

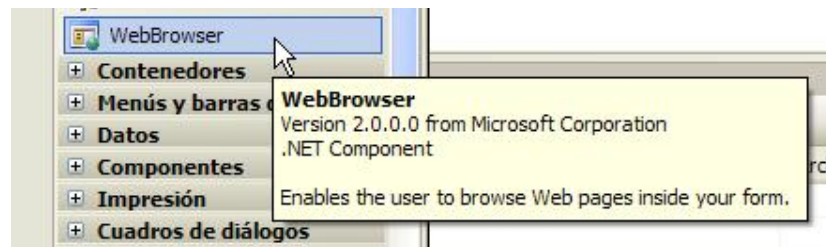
*Visual Studio 2008 nos proporciona la posibilidad de resetear o restaurar el estado inicial de la barra de herramientas en el entorno de desarrollo. Para hacer esto, haremos clic con el botón secundario del mouse la barra de herramientas y seleccionaremos la opción **Restablecer cuadro de herramientas** del menú emergente, como se muestra en la siguiente figura.*



Otros controles a tener en cuenta

Dentro del entorno de *Visual Studio 2008* y en .NET en general, tenemos la posibilidad de utilizar diferentes controles muy interesantes que conviene comentar.

Uno de estos controles, se llama *WebBrowser*, tal y como se indica en la figura 5.



Control WebBrowser en el Cuadro de herramientas

Figura 5

Este control es la representación de un control específico para mostrar contenido XML o contenido HTML, como si de una página Web se tratara.

Sirva el siguiente ejemplo de código fuente para demostrar como usar el control y como se muestra dicho control en una aplicación Windows.

El código de la aplicación quedaría como se detalla a continuación:

```
Código

Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load

        Me.webBrowser1.Navigate( "http://localhost/Bienvenido.aspx" )

    End Sub

End Class Form1
```

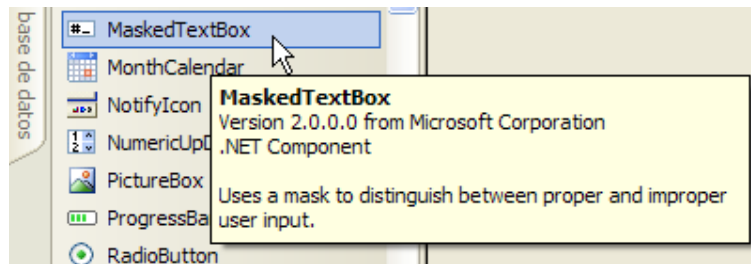
Nuestro ejemplo en ejecución es el que se muestra en la figura 6.



Control WebBrowser en ejecución

Figura 6

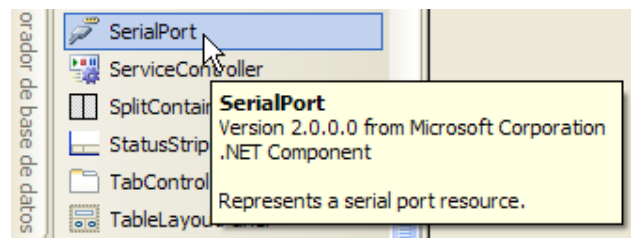
Hay más controles que representan una novedad para el desarrollador de .NET, como puede ser por ejemplo, el control **MaskedTextBox**, como se muestra en la figura 7.



Control MaskedTextBox en Visual Basic 2008

Figura 7

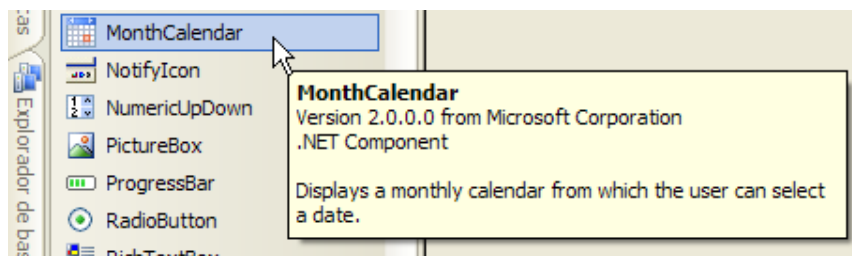
Sin embargo, hay otros controles clásicamente demandados por los desarrolladores, como los controles de accesos a puertos COM y puertos serie, como es el caso del control **SerialPort** que se muestra en la figura 8.



Control SerialPort en Visual Basic 2008

Figura 8

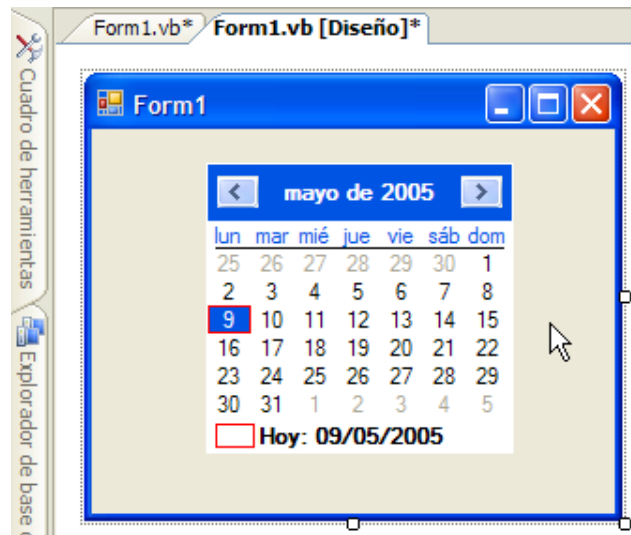
No es cuestión de repasar cada uno de los controles que el programador puede encontrar en *Visual Studio 2008*, sin embargo, no me gustaría dejar de comentar, uno de los controles más usados y útiles para las aplicaciones Windows, que tiene a su vez su equivalente para el desarrollo de aplicaciones Web en ASP.NET. Me refiero al control **MonthCalendar** que se muestra en la figura 9.



Control MonthCalendar en Visual Basic 2008

Figura 9

Este control, que se muestra en la figura 10 cuando lo insertamos en un formulario, es un control que nos facilita la entrada de fechas en el sistema y permite asegurarnos, que la fecha seleccionada es una fecha válida.



Control MonthCalendar insertado en un formulario Windows

Figura 10

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 3: Trabajo con controles

- Dominando los controles en el entorno de trabajo
- Creación de controles en tiempo de ejecución
- Creación de una matriz de controles
- Creación de controles nuevos
- Otras consideraciones

Introducción

Hasta ahora, hemos aprendido a identificar las partes más importantes del entorno de desarrollo de *Visual Studio 2008*, hemos visto igualmente como se separan los controles y componentes, y hemos visto también que existe un grupo de solapas donde podemos añadir nuestros propios controles y componentes, incluso hemos aprendido a crear nuestro propio grupo o sección de controles y componentes, en el que podemos también incluir los controles y componentes que por ejemplo hayamos desarrollado nosotros mismos, sin embargo, para poder insertar ahí un control o componente desarrollado por nosotros, deberíamos aprender a crearlos.

Eso es justamente lo que veremos a continuación además de ver otras técnicas que conviene repasar antes de aprender a desarrollar nuestros propios controles y componentes.

Por eso, lo primero que haremos será aprender a desenvolvernos adecuadamente en el entorno de desarrollo con los controles que ya conocemos.

Módulo 3 - Capítulo 3

- 1. Dominando los controles en el entorno de trabajo
- 2. Creación de controles en tiempo de ejecución
- 3. Creación de una matriz de controles
- 4. Creación de controles nuevos
- 5. Otras consideraciones

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 3: Trabajo con controles

Dominando los controles en el entorno de trabajo

- Creación de controles en tiempo de ejecución
- Creación de una matriz de controles
- Creación de controles nuevos
- Otras consideraciones

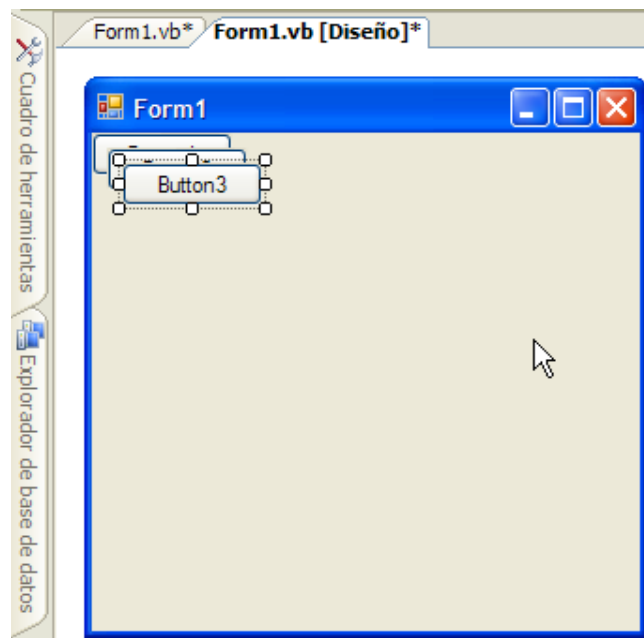
Módulo 3 - Capítulo 3

1. Dominando los controles en el entorno de trabajo

Si queremos aprender a crear nuestros propios controles para poder distribuirlos y utilizarlos en nuestro entorno, lo mejor es dominar el uso de los controles en nuestro entorno de desarrollo.

Ya hemos visto anteriormente como insertar un control a nuestro formulario, pero quizás no sepamos como manejarlos de forma eficiente en ese formulario.

Inserte en un formulario Windows por ejemplo, tres controles **Button** como se muestra en la figura 1.

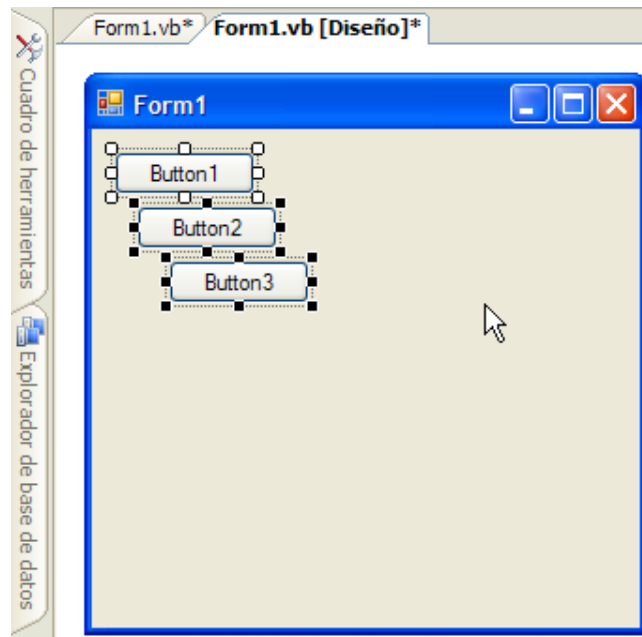


Controles Button insertados en un formulario Windows

Figura 1

Como podemos observar, los controles están dispuestos de una forma desordenada, ya que al insertarlos por ejemplo haciendo doble clic tres veces sobre un control **Button**, éstos quedan dispuestos anárquicamente en el formulario.

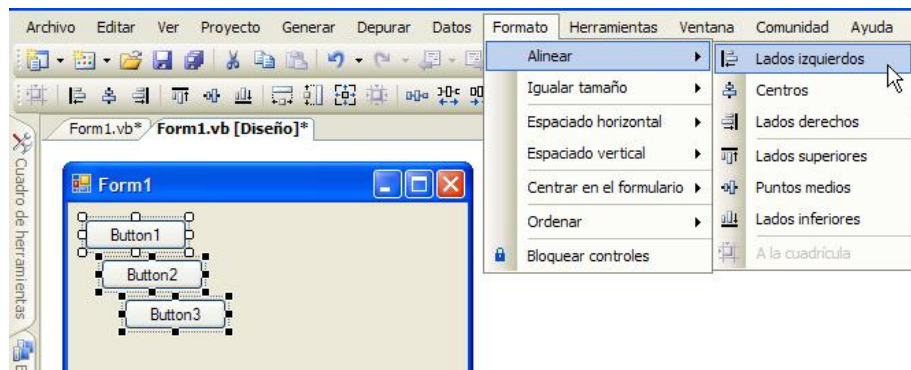
Separe los controles **Button** de forma que queden ahora esparcidos en el formulario de alguna manera tal y como se muestra en la figura 2.



Controles Button separados en el formulario

Figura 2

Seleccione todos los controles **Button** como se mostraba en la figura 2 y seleccione del menú las opciones **Formato > Alinear > Lados izquierdos** como se indica en la figura 3.

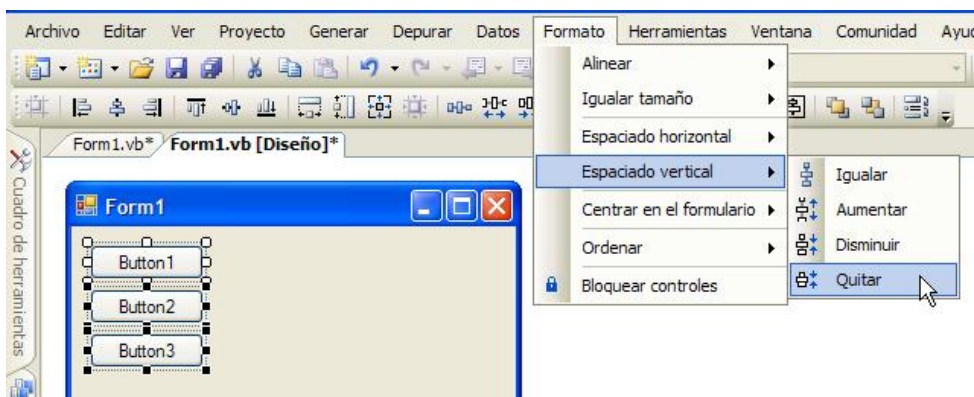


Los controles Button quedarán ordenados correctamente dentro del formulario

Figura 3

Sin embargo, podemos extender el uso de las propiedades especiales para alinear los controles en un formulario.

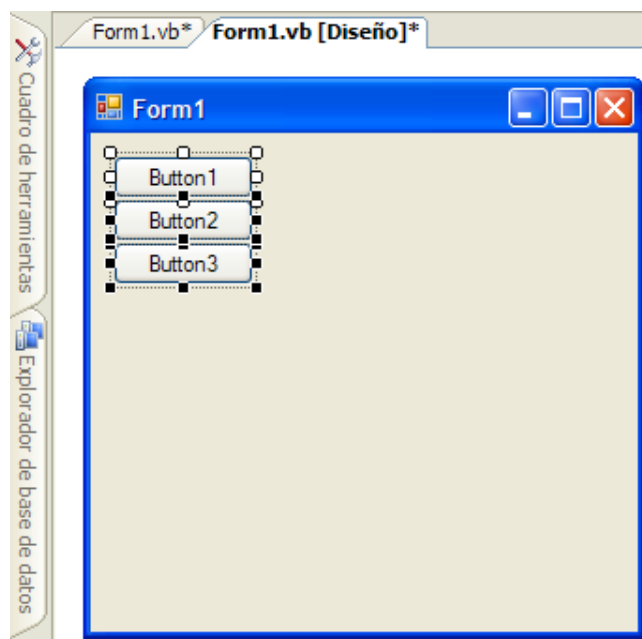
Por ejemplo, ahora que tenemos los controles **Button** alineados correctamente, podemos hacer uso de la opción de menú **Formato > Espaciado vertical > Quitar** como se indica en la figura 4.



Otras opciones especiales, nos permiten alinear o trabajar con los controles de forma rápida y segura

Figura 4

En este caso, los controles quedarán dispuestos en el formulario como se indica en la figura 5.

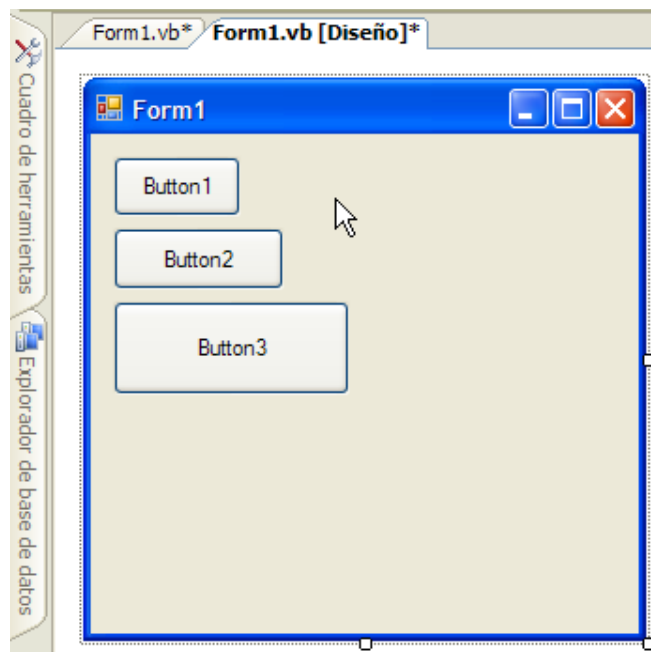


Controles alineados y espaciados según la elección de opciones del entorno

Figura 5

Como podemos apreciar, alinear los controles en el entorno es realmente sencillo. Visual Studio 2008 nos proporciona una gran cantidad de opciones para llevar a cabo este tipo de tareas.

Incluso si nos encontramos con un controles de diferente tamaño entre sí como se muestra en la figura 6, podemos hacer uso de la opción del menú **Formato > Igualar tamaño** permitiéndonos cambiar el tamaño de los controles seleccionados dentro del formulario Windows.

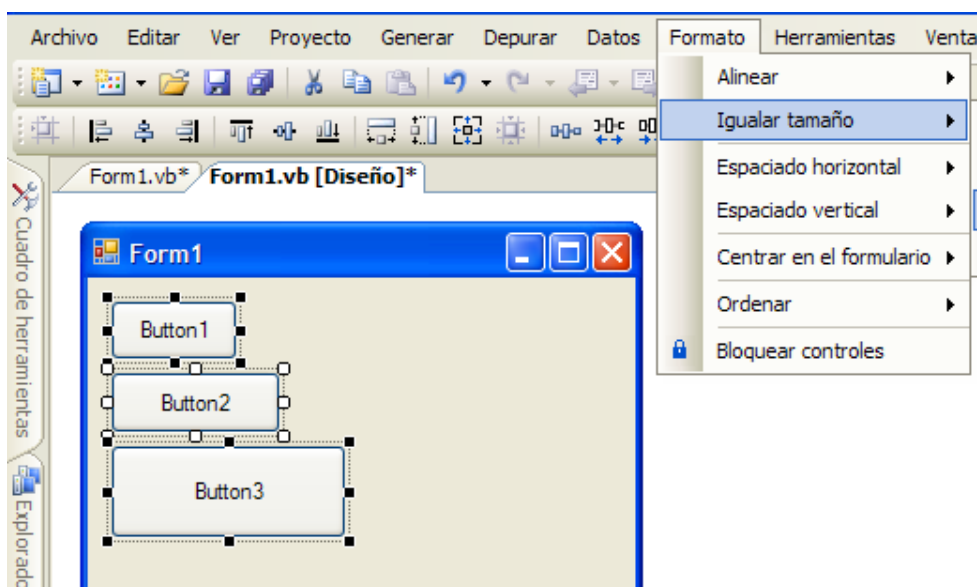


Controles de diferentes tamaños dispuestos en el formulario Windows

Figura 6

El menú que nos permite cambiar los tamaños de los controles insertados en un formulario posee diferentes posibilidades.

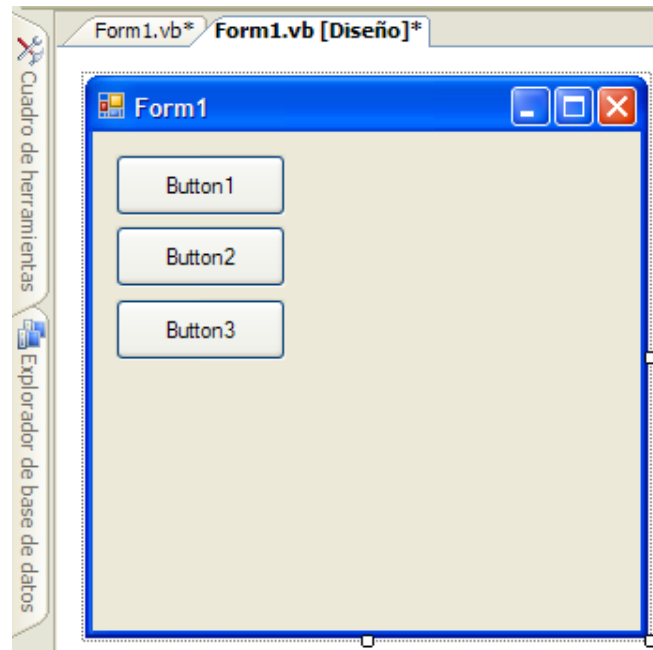
En nuestro caso, seleccionaremos del menú, la opción **Formato > Igualar tamaño > Ambos** tal y como se muestra en la figura 7.



Cambiando los tamaños ancho y alto de los controles seleccionados de un formulario

Figura 7

Una vez seleccionada esta opción, los controles se modificarán con el mismo tamaño tal y como se muestra en la figura 8.



Controles del formulario con su tamaño modificado

Figura 8

Una vez seleccionada esta opción, los controles se modificarán con el mismo tamaño tal y como se muestra en la figura 8.

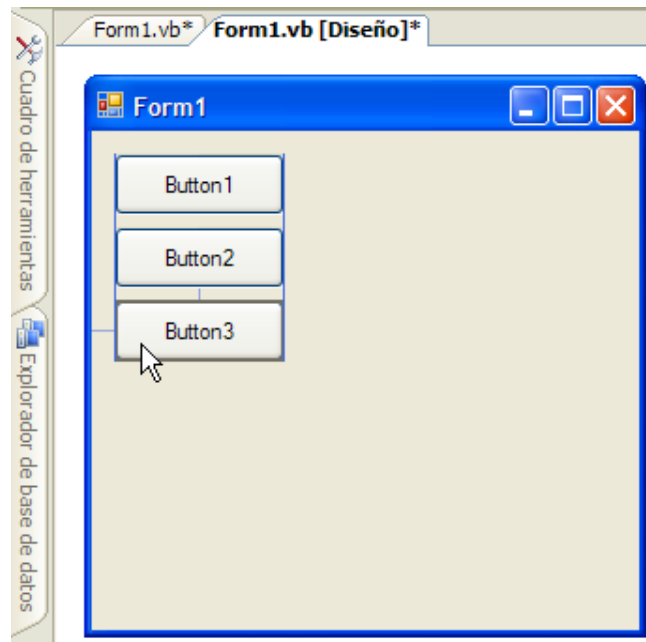
Truco:

*Suponiendo que tengamos tres controles **Button** de diferentes tamaños y que queramos que todos tengan el mismo tamaño que el segundo de sus controles, seleccionaremos **siempre** como primer control, el control que queremos como base de tamaño para el resto de controles, y posteriormente con la tecla **Ctrl** seleccionaremos uno a uno el resto de controles.*

Sin embargo, para alinear los controles en un formulario tenemos más opciones. Hemos visto algunas de ellas, quizás las más habituales, pero como podemos deducir del menú **Formato**, podremos alinear los controles, espaciarlos entre sí horizontal o verticalmente, modificar su tamaño, centrarlos en el formulario horizontal o verticalmente, etc.

Por otro lado, Visual Studio 2008, nos proporciona una utilidad en tiempo de diseño muy útil. Se trata de las guías de representación que veremos en tono azul claro, y que aparecen cuando movemos un control en el formulario.

Estas guías indican la situación y posición de un control respecto a otro próximo. La representación de estas guías que se muestran en la figura 9, nos facilita enormemente la disposición de los controles en un formulario.



Guías o reglas de dirección o separación entre controles

Figura 9

» [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 3: Trabajo con controles

- Dominando los controles en el entorno de trabajo
- Creación de controles en tiempo de ejecución
- Creación de una matriz de controles
- Creación de controles nuevos
- Otras consideraciones

Módulo 3 - Capítulo 3

2. Creación de controles en tiempo de ejecución

Prácticamente todo en .NET son objetos. Los controles también, así que para añadir un control a un formulario en tiempo de ejecución, deberemos hacerlo tratándolo como un objeto.

La declaración principal de un objeto, se realiza con la instrucción **New**.

El siguiente ejemplo de código, crea un control **Button** en tiempo de ejecución.

Código

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Declaramos el objeto Button

    Dim MiControl As New Button

    ' Declaramos un nombre al control (si queremos)

    MiControl.Name = "btn1"

    ' Cambiamos algunas de sus propiedades

    MiControl.Text = "Ejemplo de Botón"

    MiControl.Top = 50

    MiControl.Left = 50
```

```

MiControl.Width = 200

MiControl.Height = 50

' Añadimos el control al Formulario

Me.Controls.Add(MiControl)

End Sub

```

En la figura 1 podemos ver el resultado en ejecución del código escrito anteriormente.

Para ejecutar nuestra aplicación, presionaremos el botón **F5**, que es la forma habitual de ejecutar una aplicación.



Creación de un control en tiempo de ejecución en Visual Basic 2008

Figura 1

Otra de las características de los controles, es la posibilidad de manipular los controles en tiempo de ejecución. Sin embargo, en nuestro caso, vamos a modificar la propiedad **Text** del control **Button** que hemos insertado en tiempo de ejecución.

Para hacer esto, lo más habitual es poner el nombre del control, su propiedad y el valor correspondiente. En nuestro caso, el código quedaría como el que se indica a continuación:

```

Código

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Declaramos el objeto Button

    Dim MiControl As New Button

    ' Declaramos un nombre al control (si queremos)

```



```

MiControl.Name = "btn1"

' Cambiamos algunas de sus propiedades

MiControl.Text = "Ejemplo de Botón"

MiControl.Top = 50

MiControl.Left = 50

MiControl.Width = 200

MiControl.Height = 50

' Añadimos el control al Formulario

Me.Controls.Add(MiControl)

' Modificamos la propiedad Text del control insertado

btn1.Text = "Otro texto"

End Sub

```

Analizando este código, parece estar bien escrito, pero al presionar **F5** para ejecutar nuestro proyecto, nos encontramos con que Visual Basic 2008 nos muestra un error. Nos indica que **btn1** no está declarado. ¿Qué ocurre?.

Al buscar la clase de ensamblados de la aplicación, el control **Button** de nombre **btn1** no existe, por lo que Visual Basic 2008 detecta que debemos declarar el control, sin embargo y en nuestro caso, esta declaración la hacemos en tiempo de ejecución. ¿Cómo acceder a la propiedad **Text** del control **Button** creado en tiempo de ejecución?.

El siguiente código resuelve esta duda:

```

Código

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Declaramos el objeto Button

    Dim MiControl As New Button

    ' Declaramos un nombre al control (si queremos)

    MiControl.Name = "btn1"

    ' Cambiamos algunas de sus propiedades

    MiControl.Text = "Ejemplo de Botón"

    MiControl.Top = 50

    MiControl.Left = 50

    MiControl.Width = 200

    MiControl.Height = 50

```

```

' Añadimos el control al Formulario

Me.Controls.Add(MiControl)

' Modificamos la propiedad Text del control insertado

CType(Me.FindForm.Controls("btn1"), Button).Text = "Otro
texto"

End Sub

```

Básicamente, utilizamos una conversión explícita del objeto devuelto por la búsqueda realizada en los controles del formulario principal, que será un control **Button** de nombre **btn1**, para poder así, cambiar la propiedad **Text** de este control.

Aún así, también podríamos haber accedido a la propiedad **Text** del control mediante otra acción complementaria, como se muestra en el siguiente código:

Código

```

' Declaramos el objeto Button

Dim MiControl As New Button

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Declaramos un nombre al control

    MiControl.Name = "btn1"

    ' Cambiamos algunas de sus propiedades

    MiControl.Text = "Ejemplo de Botón"

    MiControl.Location = New Point(50, 50)

    MiControl.Size = New Size(200, 50)

    ' Añadimos el control al Formulario

    Me.Controls.Add(MiControl)

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click

    MiControl.Text = "Otro texto"

End Sub

```

¡Ojo!

Tenga en cuenta que esta acción se puede realizar si la declaración del objeto **Button** está dentro del mismo ámbito de llamada de la

propiedad **Text**. Por esa razón, hemos sacado la declaración **MiControl** del objeto **Button** fuera del procedimiento de creación dinámica del control, pero tenga en cuenta también, que en este caso, tendremos declarada siempre en memoria la variable **MiControl**. El uso de la conversión (**Button**) es siempre más seguro.

El caso anterior utilizando únicamente la conversión explícita, (**Button**), quedaría como se detalla a continuación (para estos dos ejemplos, he añadido además un control **Button** al formulario *Windows*, desde el cuál cambiaremos la propiedad **Text** del control **Button** creado en tiempo de ejecución):

Código

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Declaramos el objeto Button

    Dim MiControl As New Button

    ' Declaramos un nombre al control (si queremos)

    MiControl.Name = "btn1"

    ' Cambiamos algunas de sus propiedades

    MiControl.Text = "Ejemplo de Botón"

    MiControl.Location = New Point(50, 50)

    MiControl.Size = New Size(200, 50)

    ' Añadimos el control al Formulario

    Me.Controls.Add(MiControl)

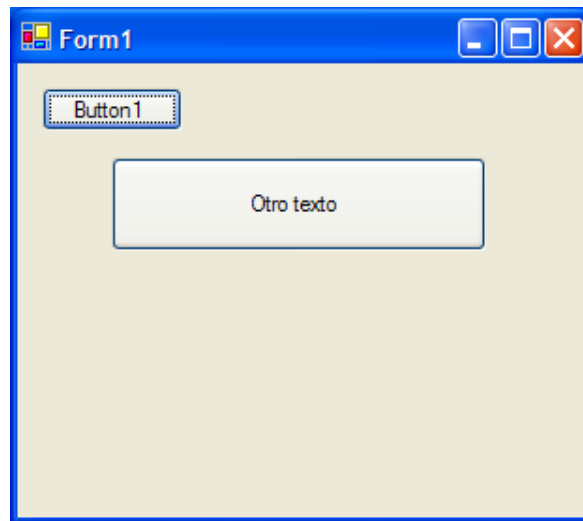
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click

    CType(Me.FindForm.Controls("btn1"), Button).Text = "Otro
texto"

End Sub
```

Este ejemplo en ejecución es el que se muestra en la figura 2.



Ejecución del ejemplo anterior en Visual Basic 2008

Figura 2

Antes de continuar con el siguiente apartado en el que aprenderemos a crear un array de controles, comentaré que a la hora de posicionar un determinado control en un formulario *Windows*, lo podemos hacer con las propiedades **Top** y **Left**, o bien, utilizando la propiedad **Location**. Lo mismo ocurre con las propiedades **Width** y **Height** que pueden ser sustituidas por la propiedad **Size**.

El mismo ejemplo de creación de controles en tiempo de ejecución utilizando el método **Location** y **Size**, quedaría como se detalla a continuación:

Código

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Declaramos el objeto Button

    Dim MiControl As New Button

    ' Declaramos un nombre al control (si queremos)

    MiControl.Name = "btn1"

    ' Cambiamos algunas de sus propiedades

    MiControl.Text = "Ejemplo de Botón"

    ' Propiedad Location

    MiControl.Location = New Point(50, 50)

    ' Propiedad Size

    MiControl.Size = New Size(200, 50)

    ' Añadimos el control al Formulario

    Me.Controls.Add(MiControl)
```

End Sub

Hasta ahora hemos visto como crear controles en tiempo de ejecución, pero en muchas ocasiones, nos es útil no sólo crear un control en tiempo de ejecución, sino relacionarlo con un evento. En nuestro ejemplo del control **Button**, cuando hacemos clic sobre el control, no ocurre nada, y sería interesante mostrar un mensaje o realizar una acción determinada.

Para hacer esto, deberemos utilizar el método **AddHandler** para asignar un evento al control creado dinámicamente. El código de nuestro ejemplo *mejorado* es el que se detalla a continuación:

Código

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Declaramos el objeto Button

    Dim MiControl As New Button

    ' Declaramos un nombre al control (si queremos)

    MiControl.Name = "btn1"

    ' Cambiamos algunas de sus propiedades

    MiControl.Text = "Ejemplo de Botón"

    MiControl.Location = New Point(50, 50)

    MiControl.Size = New Size(200, 50)

    ' Añadimos el control al Formulario

    Me.Controls.Add(MiControl)

    ' Añadimos el evento Click al control creado dinámicamente

    AddHandler MiControl.Click, AddressOf btn1Click

End Sub


Public Sub btn1Click(ByVal Sender As Object, ByVal e As
System.EventArgs)

    ' Mostramos un Mensaje

    MessageBox.Show("Soy el Control Button con texto: '" +
CType(CType(Sender, Button).Text, String) + "'")

End Sub
```

Nuestro ejemplo en ejecución es el que se muestra en la figura 3.



Ejecución del ejemplo con asociación de evento desarrollado en Visual Basic 2008

Figura 3

» [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 3: Trabajo con controles

- Dominando los controles en el entorno de trabajo
- Creación de controles en tiempo de ejecución

Creación de una matriz de controles

- Creación de controles nuevos
- Otras consideraciones

Módulo 3 - Capítulo 3

3. Creación de una matriz de controles

En el capítulo anterior, hemos visto como crear controles en tiempo de ejecución e incluso como asociar un evento a un control creado dinámicamente en Visual Basic 2008, pero, ¿qué ocurre si queremos crear un array o matriz de controles?.

Podemos simular una matriz de controles de muchas formas. Yo en este ejemplo, aplicaré la siguiente forma de llevar a cabo esta tarea:

Código

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    ' Declaramos la variable contador del bucle Para

    Dim I As Byte

    ' Declaramos la variable contador del número de controles a
    crear

    Dim intNumControles As Byte = 5

    ' Iniciamos el bucle Para

    For I = 0 To intNumControles - 1

        ' Declaramos el objeto TextBox
```

```

        Dim MiControl As New TextBox

        ' Le asignamos un nombre al control

        MiControl.Name = "txt1"

        ' Utilizamos la propiedad Tag para almacenar ahí el valor
        del control de la matriz virtual

        MiControl.Tag = I

        ' Le asignamos un tamaño en el Formulario Windows

        MiControl.Size = New Size(100, 20)

        ' Le asignamos una posición en el formulario Windows

        MiControl.Location = New Point(50, 22 * (I + 1))

        ' Le cambiamos la propiedad Text

        MiControl.Text = MiControl.Name + "(" + I.ToString() + ")"

        ' Añadimos el control al Formulario

        Me.Controls.Add(MiControl)

        ' Añadimos el evento Click al control creado dinámicamente

        AddHandler MiControl.Click, AddressOf txt1Click

    Next

End Sub

Public Sub txt1Click(ByVal Sender As Object, ByVal e As
System.EventArgs)

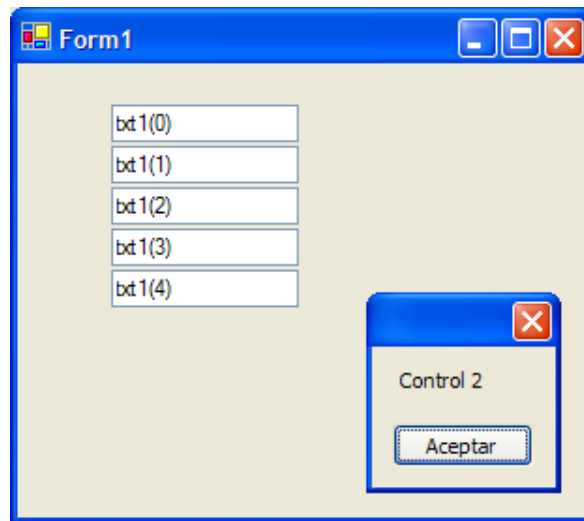
    ' Mostramos un Mensaje

    MessageBox.Show("Control " + CType(Sender,
TextBox).Tag.ToString())

End Sub

```


Nuestro ejemplo de demostración en ejecución es el que se puede ver en la figura 1.



Ejecución de la simulación de una matriz de controles

Figura 1

Obviamente, existen diferentes formas y técnicas de simular un array o matriz de controles. Sirva esta que hemos visto, como ejemplo de lo que se puede hacer, pero para nada se trata de una norma o regla fija.

Debemos destacar que en .NET no existe el concepto de array o matriz de controles ya que ésta, es una característica no propia del lenguaje, pero que para ciertas ocasiones conviene conocer.

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 3: Trabajo con controles

- Dominando los controles en el entorno de trabajo
- Creación de controles en tiempo de ejecución
- Creación de una matriz de controles

Creación de controles nuevos

- Otras consideraciones

Módulo 3 - Capítulo 3

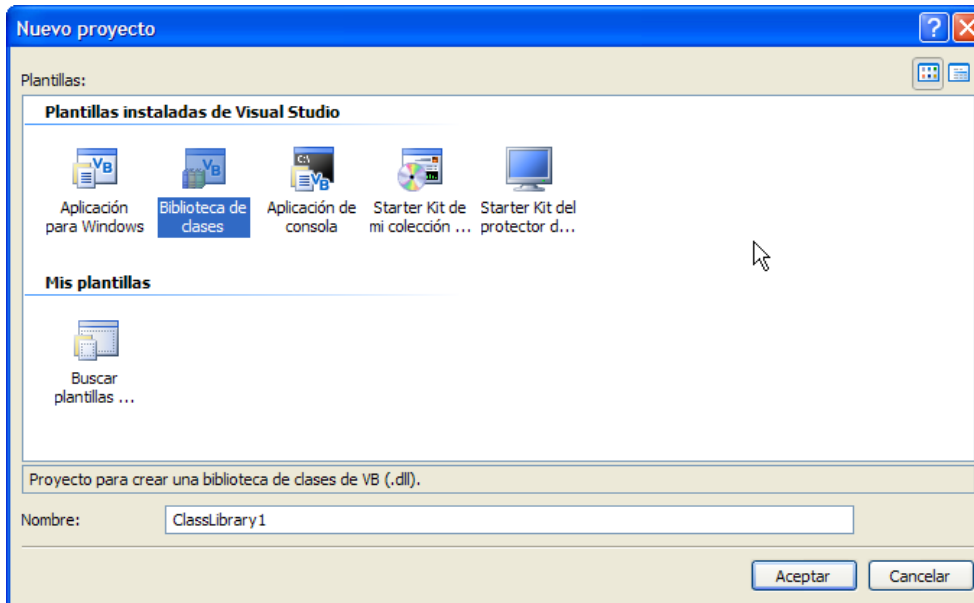
4. Creación de controles nuevos

Ya hemos visto la diferencia más genérica entre un componente y un control, pero aún no sabemos como desarrollar nuestros propios controles en Visual Basic 2008.

En primer lugar y antes de adentrarnos en la creación de nuestros propios controles con Visual Basic 2008, debo indicarle que debemos obviar todo lo relacionado con los ActiveX OCX y ActiveX en general.

En Visual Basic 2008, la palabra ActiveX ya no existe. El modelo de programación ha cambiado y por eso, los componentes y controles se generan ahora siguiendo otras normas que aprenderemos a utilizar de forma inmediata.

Iniciaremos *Visual Studio 2008* y seleccionaremos un proyecto de tipo **Biblioteca de clases**. En el nombre de proyecto, podemos indicarle el nombre que deseemos tal y como se muestra en la figura 1, y a continuación presionaremos el botón **OK**.



Selección de nuevo proyecto Biblioteca de clases en Visual Basic 2008

Figura 1

La diferencia mayor que reside entre el desarrollo de componentes y controles en .NET, es que en lugar de heredar de la clase **Component** como en el caso de la creación de los componentes, se ha de heredar de la clase **Control** o **System.Windows.Forms.UserControl**.

El tipo de proyecto seleccionado no posee por defecto como ocurre con los controles ActiveX, de la superficie contenedora sobre la cuál podremos insertar otros controles o realizar las acciones que consideremos pertinentes para crear así nuestro control personalizado.

En este caso, la superficie contenedora la deberemos crear añadiendo las referencias necesarias a nuestro programa.

Haga clic con el botón secundario del mouse sobre el proyecto o solución de la ventana **Explorador de soluciones** y a continuación, seleccione la opción **Propiedades** del menú emergente. A continuación, agregue las referencias a las librerías de clases **System.Drawing** y **System.Windows.Forms**.

Por último, escriba las siguientes instrucciones básicas.

```

Código

Imports System.ComponentModel

Imports System.Windows.Forms

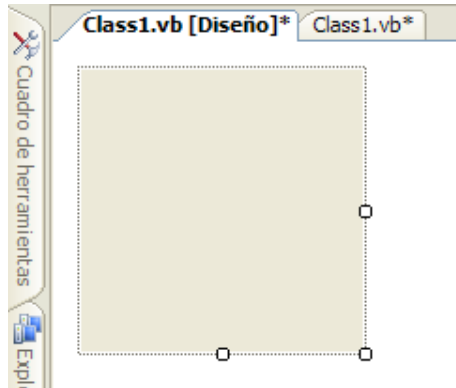
Imports System.Drawing

Public Class Class1

    Inherits UserControl
  
```

```
End Class
```

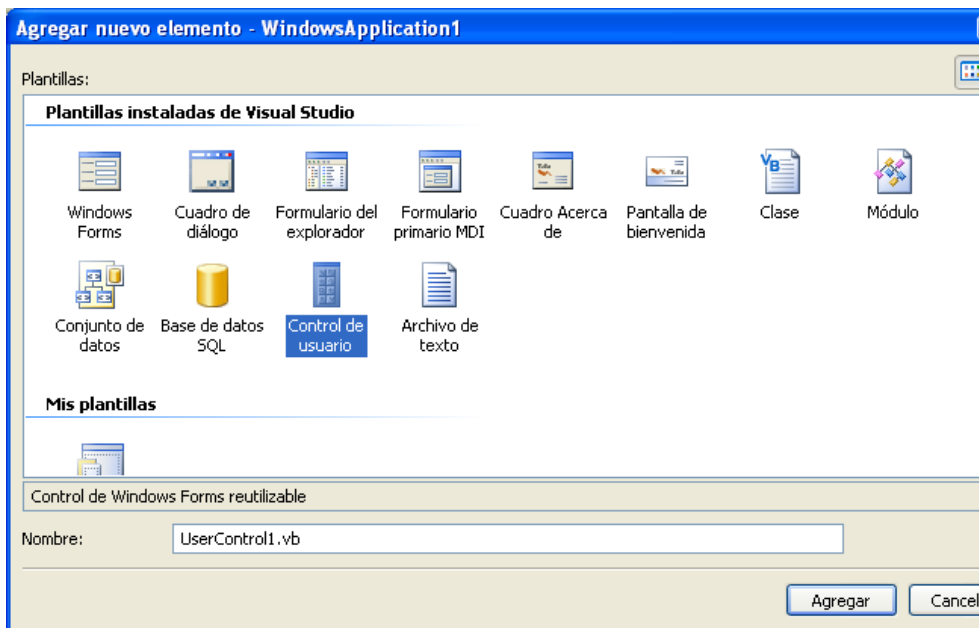
En este punto, nuestra clase habrá sido transformada en la clase contenedora de un control que es la que puede verse en la figura 2.



Superficie contenedora por defecto de un control

Figura 2

Aún así, sino quiere realizar esta acción, otra forma de tener lista la superficie contenedora del control, es la de eliminar la clase del proyecto y presionar el botón secundario del mouse sobre la ventana del *Explorador de soluciones* y seleccionar la opción de **Agregar > Nuevo elemento...** del menú emergente. De las opciones que salen, (ver figura 2B), deberíamos seleccionar entonces la plantilla **Control de usuario**.



Seleccionar un control de usuario

Figura 2B

Sin más dilación, añadiremos sobre la superficie del control contenedor, (al que habremos cambiado el nombre a **MiControl**), dos controles *Label* y dos controles *TextBox*.

Debido a que al añadirle los controles tendremos mucho código que simplemente sirve para el diseño del interfaz de usuario, podemos hacer lo que el propio diseñador hace: dividir la clase en dos partes o clases parciales, de esta forma conseguiremos el propósito de las clases parciales, (al menos en cuanto al diseño de formularios y controles de usuario se refiere), que es, como ya comentamos anteriormente, separar el código de diseño del código que nosotros vamos a escribir. Para ello, añadiremos un nuevo elemento del tipo Class, al que le daremos el nombre **MiControl.Designer.vb**, y en esa clase pegaremos el código que actualmente hay en la clase, pero añadiendo la partícula *Partial* antes de *Class*, con idea de que el compilador de Visual Basic 2008 sepa que nuestra intención es crear una clase parcial.

Realmente es casi lo mismo que tenemos actualmente, tan solo tendremos que añadir la instrucción *partial*, (no hace falta que esté declarada como *public*, ya que en el otro "trozo" tendremos declarada esta clase como pública), además del constructor de la clase, desde el que llamaremos al método que se encarga de toda la inicialización: **InitializeComponent()**.

A continuación, escribiremos el siguiente código, en el archivo que teníamos originalmente (MiControl.vb):

```
Código

Imports System.ComponentModel

Imports System.Windows.Forms

Public Class MiControl

    Private _Acceso As Boolean

    <Category("Acceso"), Description("Indica si se permite o no el
    acceso"), DefaultValue("False"), [ReadOnly](True)> _

    Public Property Acceso() As Boolean

        Get

            Return _Acceso

        End Get

        Set(ByVal Val As Boolean)

            _Acceso = Val

        End Set

    End Property
```

```

Private Sub UserControl1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load

    _Acceso = False

End Sub

Public Sub Validar()

    If TextBox1.Text = "ejemplo" And TextBox2.Text = "ejemplo"
Then

        _Acceso = True

    Else

        _Acceso = False

    End If

End Sub

End Class

```

Observando el código, vemos que hemos creado una propiedad **Acceso** que nos permitirá saber si un usuario y contraseña han sido validados o no.

Ahora nuestro control, está listo ya para ser compilado y probado en una aplicación Windows.

Para compilar nuestro control, haga clic en el menú **Generar > Generar Solución** como se muestra en la figura 3.



Opción para compilar nuestro control

Figura 3

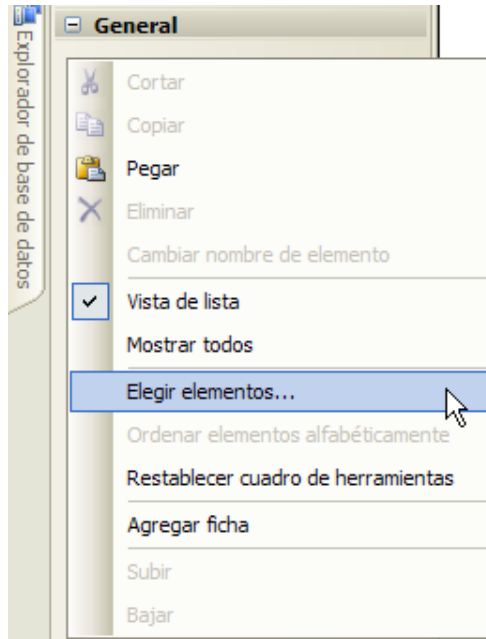
A continuación añadida un nuevo proyecto **Aplicación para Windows** a la solución. Para añadir un proyecto a la solución actual, tendremos que ir al menú **Archivo** y seleccionar **Nuevo proyecto**, cuando nos muestre el cuadro de diálogo, indicaremos que lo añada a la misma solución.

Establézcalo como proyecto inicial de la solución.

Acuda a la barra de herramientas y busque el control que hemos creado y compilado para insertarlo en el formulario Windows.

Sino aparece en la barra de herramientas, (que debería aparecer en el grupo **ClassLibrary1**), deberá añadirlo de la siguiente manera.

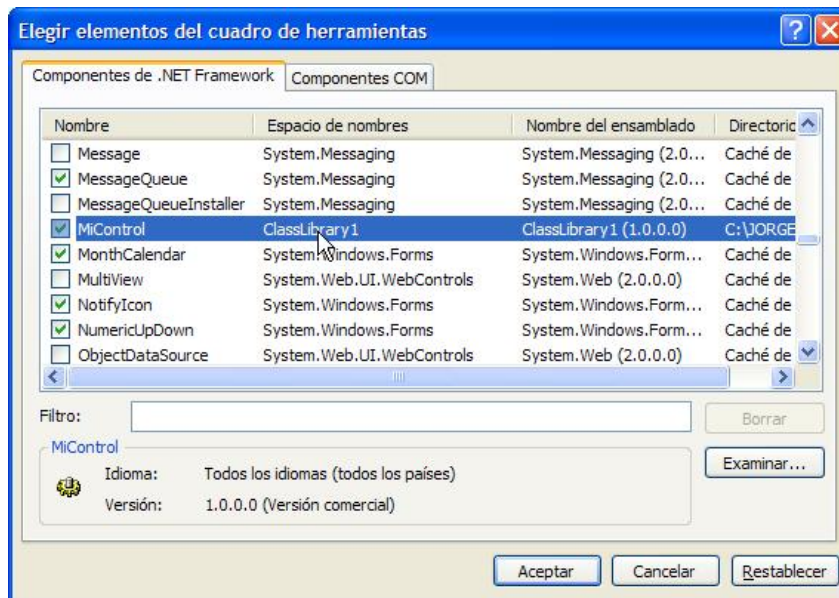
Haga clic sobre la barra de herramientas y seleccione la opción **Elegir Elementos...** del menú emergente que aparece en la figura 4.



Opción para añadir un control o componente a la barra de herramientas

Figura 4

Aparecerá una ventana para buscar el control ya compilado en el disco duro. Presionaremos el botón **Examinar...** y buscaremos nuestro control para seleccionarlo. Una vez hecho esto, tal y como se indica en la figura 5, haremos clic sobre el botón **OK**.



Selección del control compilado anteriormente

Figura 5

Nuestro control quedará insertado en en la barra de herramientas como se muestra en la figura 6.



Control insertado en la barra de herramientas

Figura 6

Para insertar el control en el formulario, haremos doble clic sobre él. Este quedará dispuesto en el formulario Windows como se indica en la figura 7.



Control insertado en el formulario Windows de prueba

Figura 7

Nuestro formulario Windows de prueba en ejecución con el control insertado en él, es el que puede verse en la figura 8.



Formulario Windows de prueba en ejecución con el control insertado

Figura 8

Como ha podido comprobar, la creación de controles en Visual Basic 2008, tampoco requiere de una gran habilidad o destreza, y su similitud con la creación de componentes es enorme. De hecho, todo se reduce en el uso y programación de una clase con la salvedad de que dentro de esa clase, indicamos si se trata de una clase como tal, la clase de un componente o la clase de un control.

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Módulo 3 - Capítulo 3

5. Otras consideraciones

Controles contenedores

Cuando trabajamos con controles, surgen muchas veces muchas dudas de carácter habitual. Definamos que hay dos tipos de controles, los controles contenedores y los controles no contenedores.

La diferencia entre ambos es que los controles contenedores, pueden como su propia palabra dice, contener otros controles dentro de éste.

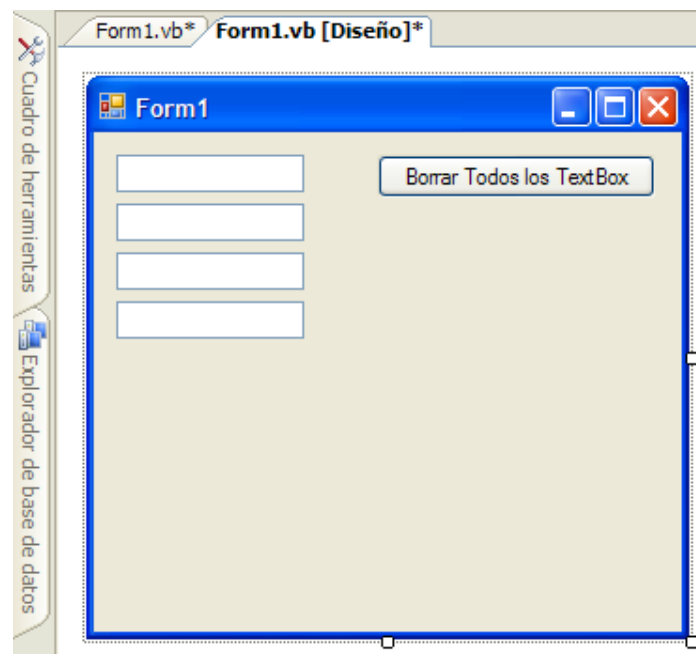
Los controles no contenedores no pueden. Un claro ejemplo de control contenedor, es el control **TabControl**, el cuál permite incluir dentro de él otros controles, incluso controles contenedores a su vez.

Un ejemplo de control no contenedor es el control **Button** por ejemplo. Si intenta poner un control **TextBox** encima de un control **Button**, observará que no puede.

Cuando un control se inserta dentro de un contenedor (no olvidemos que el formulario Windows es otro contenedor), éste control queda alojado dentro de su contenedor, de tal manera que si movemos el contenedor, estaremos moviendo también el *contenido* de éste.

El problema sin embargo cuando trabajamos con controles, viene cuando deseamos recorrer los controles de un formulario. Esto es muy habitual cuando deseamos borrar el contenido de todos los controles **TextBox** de un formulario Windows por ejemplo.

Dentro de un formulario y supuesto que tengamos 4 controles **TextBox** insertados dentro de él, y luego un botón que nos permita eliminar el contenido de cada caja de texto, tal y como se indica en la figura 1, deberíamos escribir un código similar al que veremos a continuación:



Formulario Windows de ejemplo con sus controles insertados

Figura 1

```
Código

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button1.Click

        Dim obj As Object

    End Sub

End Class
```

```

For Each obj In Controls

    If TypeOf obj Is TextBox Then

        CType(obj, TextBox).Text = ""

    End If

Next

End Sub

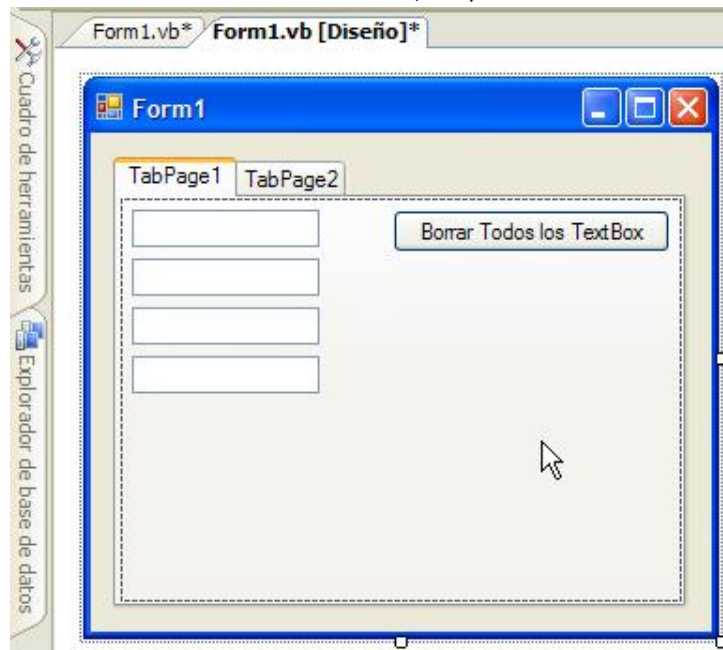
End Class

```

Observando este código, vemos que lo que hacemos no es otra cosa que recorrer los objetos como controles dentro del formulario y miramos si se trata de un control de tipo **TextBox**.

En este caso, modificamos la propiedad **Text** para dejarla en blanco. Ejecute este ejemplo con la tecla de función **F5**, escriba algo en las cajas de texto y pulse el botón. Observará que logramos conseguir nuestro objetivo de borrar el contenido de todos los controles **TextBox**.

A continuación, lo que haremos será añadir un control **TabControl** a nuestro formulario Windows, y dentro de él, añadiremos los cuatro controles **TextBox** y el control **Button** anteriormente comentados, tal y como se muestra en la figura 2.



Controles insertados dentro de un control contenedor como el control TabControl

Figura 2

El código anterior no hará falta modificarlo, por lo que ejecutaremos la aplicación nuevamente presionando el botón **F5**, escribiremos algo de texto en las cajas de texto y pulsaremos el botón como hicimos antes. Observaremos que en este caso, los controles **TextBox** no se han quedado en blanco como antes.

El contenedor no es el formulario Windows, sino el control **TabControl**. Dicho control es tratado en el bucle anterior del control como control dependiente del formulario Windows, pero los controles **TextBox** y el propio control **Button**, quedan encerrados dentro de un ámbito contenedor diferente.

Para solventar este problema, deberemos recorrer los controles del contenedor. Una forma de solventar esto es de la siguiente manera:

```
Código

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

        For I As Byte = 0 To TabControl1.TabPages.Count - 1

            For Each obj As Object In TabControl1.TabPages(I).Controls

                If TypeOf obj Is TextBox Then

                    CType(obj, TextBox).Text = " "

                End If

            Next

        Next

    End Sub

End Class
```

De esta manera, recorreremos todos los controles que residen en las páginas de un control **TabControl** y si se trata de un control **TextBox**, modificaremos la propiedad **Text** correspondiente.

Por esta razón, cuando trabajamos con controles dentro de un formulario y queremos actuar sobre un conjunto de controles determinado, debemos tener en cuenta entre otras cosas, si se trata de un conjunto de controles o un control simplemente, se encuentra dentro de un control contenedor o fuera de él.

Sin embargo, para resolver el problema de recorrer todos los controles de un formulario, estén o no dentro de un control contenedor, lo mejor es ampliar la función anterior y hacerla recursiva, de modo que permita recorrer todos los controles del formulario, estén o no dentro de un contenedor.

El siguiente código, refleja un ejemplo de como hacer esto posible. En el mismo proyecto, podemos añadir nuevos **TextBox** a la otra ficha del **TabControl**, además uno en el propio formulario, (para que no esté dentro del **TabControl**).

Rellene algo de contenido en los controles **TextBox** y ejecute el código:

```
Código

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
```

```

VaciartextBox(Me)

End Sub

Private Sub VaciartextBox(ByVal Parent As Control)

    For Each obj As Control In Parent.Controls

        If obj.Controls.Count > 0 Then

            VaciartextBox(obj)

        End If

        If TypeOf obj Is TextBox Then

            CType(obj, TextBox).Text = ""

        End If

    Next

End Sub

End Class


```

Smart Tags

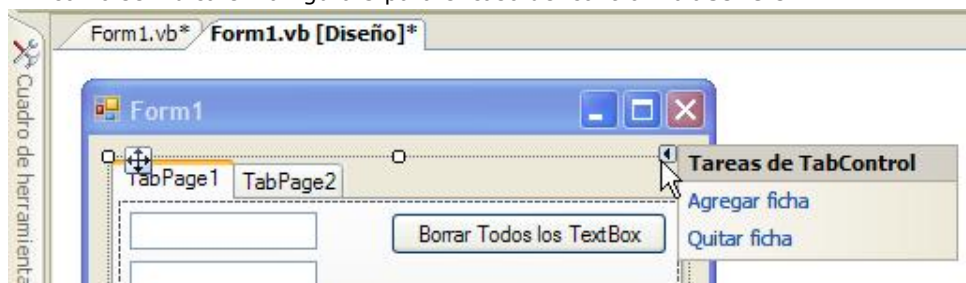
Otras de las novedades del diseñador de formularios de Visual Studio 2008, es el hecho de poder utilizar las denominadas *Smart Tags*.

Si usted es usuario de *Office* en sus últimas versiones, sabrá a que me refiero. Las **Smart Tags** nos indican trucos y consejos directa a la cuál se puede acceder desde Visual Basic 2008 de forma rápida, ahorrándonos tiempo y aportándonos productividad en nuestros desarrollos.

Las **Smart Tags** aparecen en Visual Studio 2008 tanto en el lado del diseño del formulario Windows como en la parte correspondiente del código fuente escrito.

En el caso de los controles insertados en un formulario Windows, estas **Smart Tags** aparecerán por lo general en el borde superior derecho del control y se representará con un icono parecido a este .

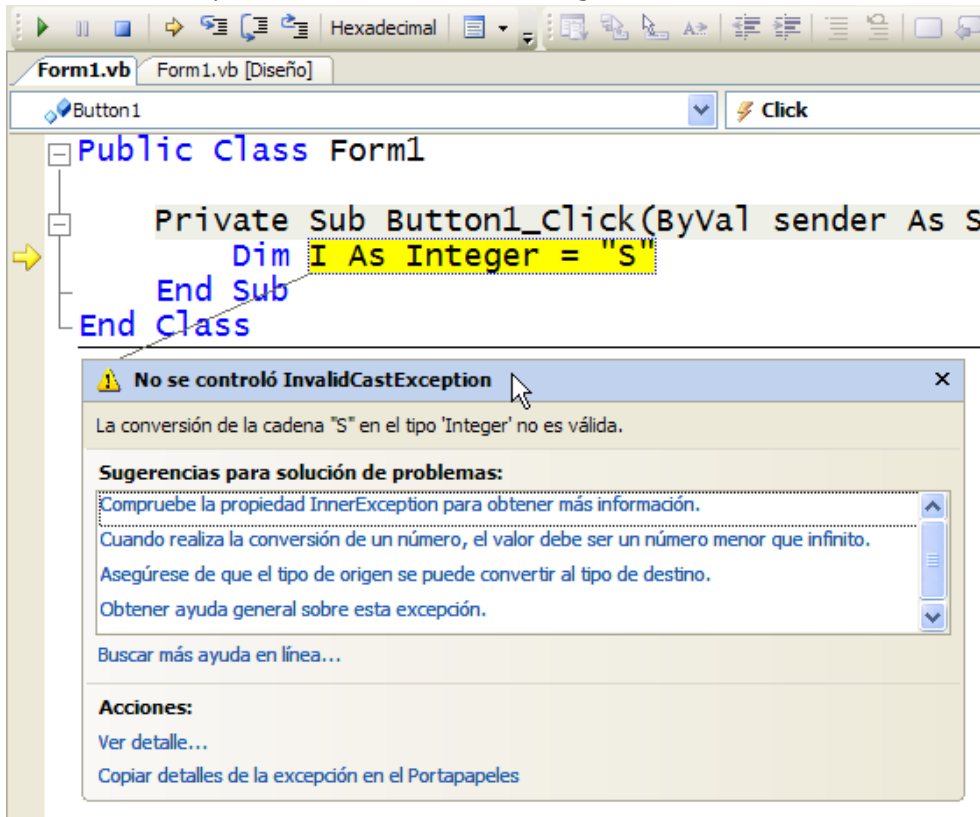
Si hacemos clic sobre ese icono, aparecerá una ayuda gráfica con varias opciones que permite realizar las acciones más comunes sobre el control seleccionado, tal y como se indica en la figura 3 para el caso del control **TabControl**.



Controles insertados dentro de un control contenedor como el control TabControl

Figura 3

En el caso del código, las **Smart Tags** poseen un comportamiento ligeramente diferente dependiendo del caso. Un ejemplo habitual es el hecho de declarar una variable incorrecta tratando de trabajar con ella. Cuando ejecutamos nuestra aplicación, el depurador se detiene en la declaración incorrecta y nos muestra una ventana de depuración, como se indica en la figura 4.

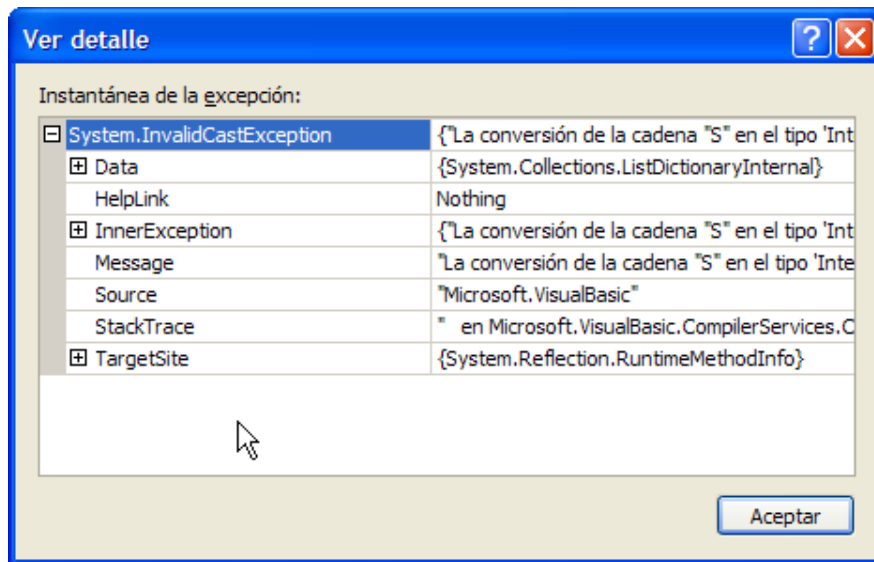


Ventana de depuración detallada

Figura 4

La ventana emergente de depuración nos indica no sólo el tipo de error encontrado, sino que además, nos indica las posibles soluciones para solventar el problema. Estos consejos, sugerencias o *Tips*, nos ofrecen una ventaja sobre nuestros desarrollos, proporcionándonos una rápida solución a un problema y permitiéndonos ahorrar mucho tiempo resolviendo el porqué del problema.

Si en la ventana de depuración emergente presionamos sobre la opción **Ver detalle...**, obtendremos una nueva ventana como la que se muestra en la figura 5.



Ventana de detalles del error

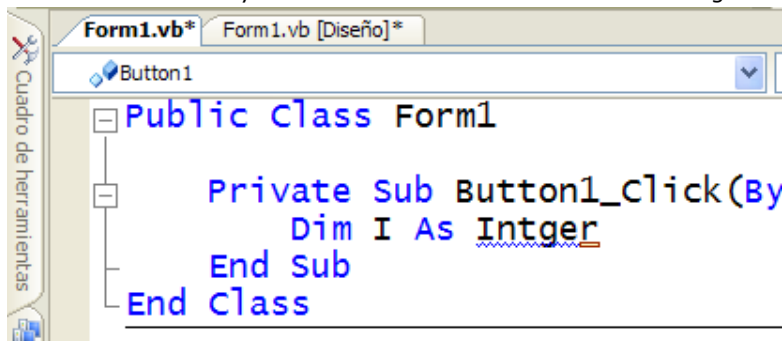
Figura 5

De esta manera, conoceremos los detalles del problema detectado por Studio 2008 permitiéndonos corregirlo cómodamente.

Otra posibilidad de las **Smart Tags** es la detección de una declaración de variable o cualquier parte de código, no esperada. Como ejemplo, declararemos una variable de la forma:




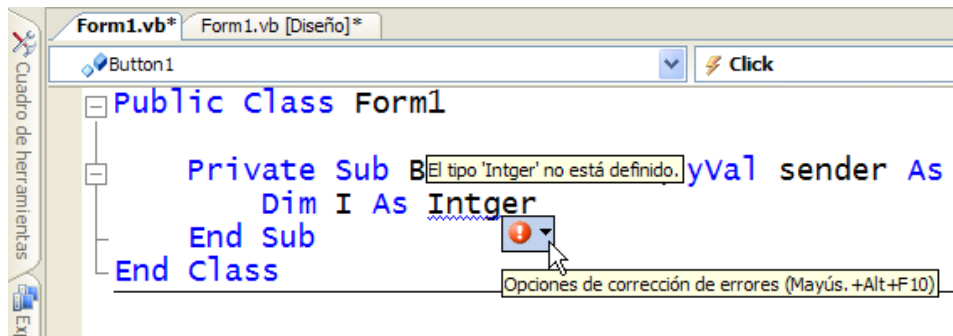
Queríamos decir *Integer*, pero con las prisas he escrito *intger*. En ese instante, Visual Studio 2008 se ha percatado de que la declaración no es correcta y antes de que me pueda dar cuenta de que lo he escrito incorrectamente, el entorno me indica el error marcándolo con un subrayado característico como se indica en la figura 6.



Subrayado de Visual Studio 2008 indicando un error en la instrucción de código escrita


Figura 6

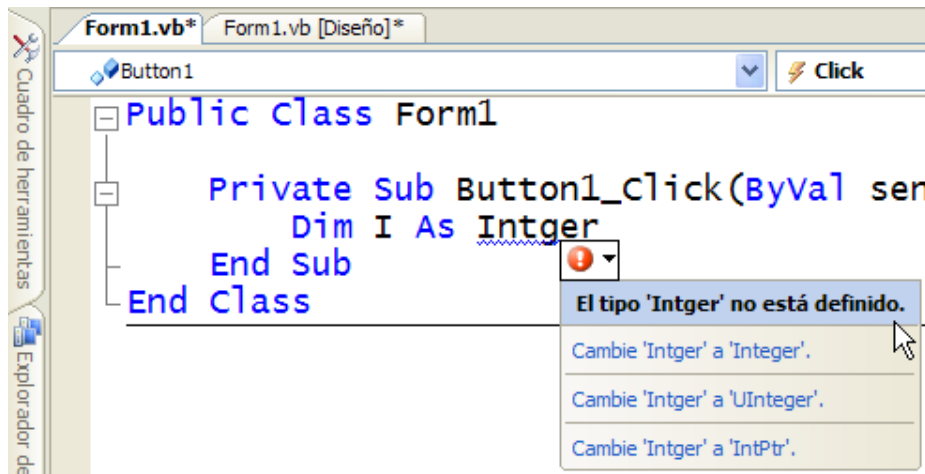
Cuando acercamos el puntero del mouse sobre el recuadro de color rojo de la declaración errónea, aparece una información adicional junto a un icono parecido a este  y una información emergente que se muestra en la figura 7.



Información sobre el detalle del error encontrado

Figura 7

Si hacemos clic sobre el icono , accederemos a una información adicional que nos muestra un conjunto de sugerencias. En nuestro caso es la primera de ellas como se indica en la figura 8, por lo que al hacer clic sobre ella, la declaración se actualiza directamente por la declaración correcta:



Opciones o sugerencias de resolución del error

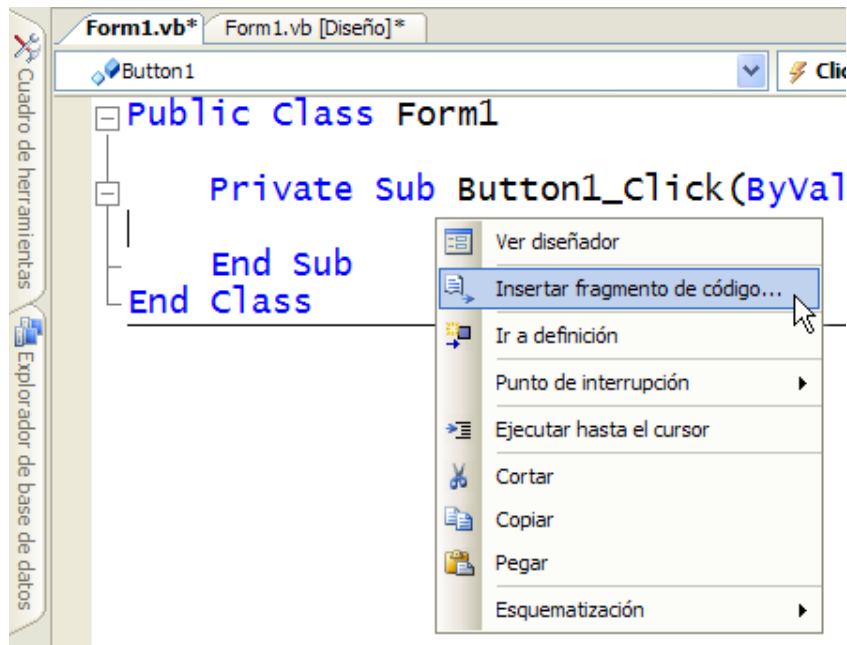
Figura 8



Generación de código rápido

Otra consideración a tener en cuenta a la hora de trabajar con código y a la hora por lo tanto, de desarrollar nuestros propios controles, componentes, clases, etc., es la ayuda que nos proporciona el entorno cuando deseamos escribir determinadas funciones rápidas de código que son bastante habituales.

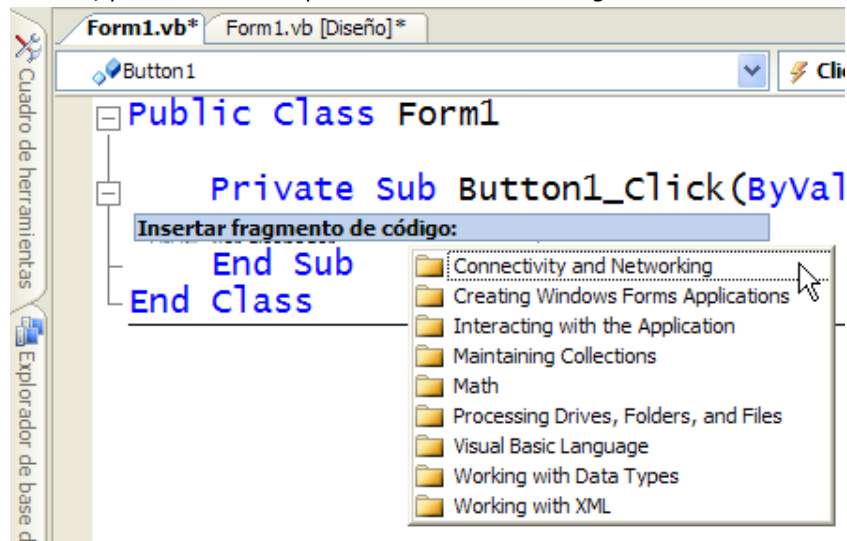
Posiciónese sobre el código de Visual Basic 2008 y haga clic con el botón secundario del mouse y seleccione la opción **Insertar fragmento de código...** como se indica en la figura 9.



Opción de recortes de código de Visual Studio 2008

Figura 9

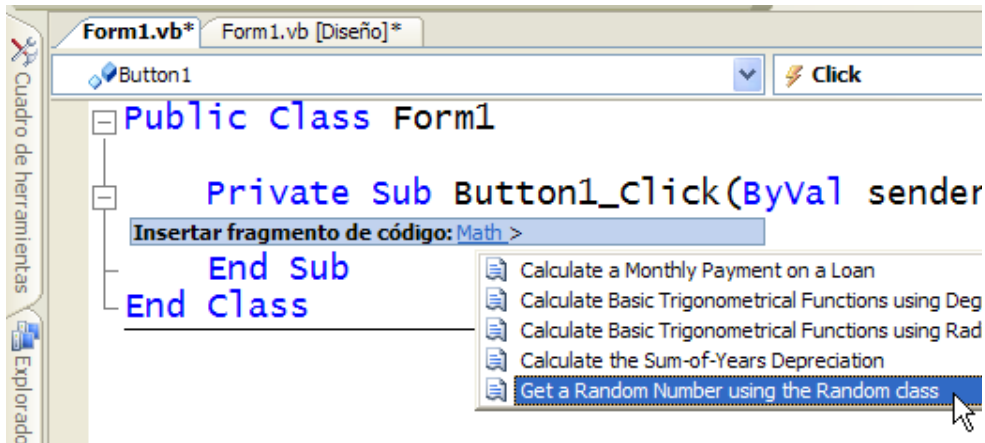
Aparecerá una ventana emergente como la que se muestra en la figura 10 dentro de la cuál, podremos seleccionar la carpeta que contiene un amplio conjunto de funciones, para insertarla rápidamente a nuestro código.



Ventana emergente de carpetas de recortes de código

Figura 10

Como ejemplo, sitúese con los cursores sobre **Math**, pulse *Enter*, seleccione a continuación *Get a Random Number using the Random class* como se indica en la figura 11 y vuelva a presionar *Enter*. El código quedará insertado a nuestro proyecto de Visual Basic 2008 como se muestra a continuación:



Selección de un recorte de código

Figura 11

```
Código

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

    Dim generator As New Random

    Dim randomValue As Integer

    ' Generates numbers between 1 and 5, inclusive.

    randomValue = generator.Next(1, 6)

End Sub
```

Estos atajos, aumentan la productividad en el desarrollo de aplicaciones de forma abismal. Tengamos en cuenta, que muchas de las operaciones que realizamos los desarrolladores, son acciones repetitivas, funciones generales o funciones habituales que reducen nuestro rendimiento y productividad y nos permite cometer errores. Con estas acciones, evitamos todo esto.

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 4: Trabajo con imágenes y gráficos


- Gráficos 3D
- Gráficos 2D
- Dibujando líneas con GDI+
- Dibujando curvas con GDI+
- Dibujando cadenas de texto con GDI+
- Otras consideraciones

Introducción

En este capítulo veremos las partes más generales en el uso y generación de imágenes y gráficos con Visual Basic 2008. Pasaremos de puntillas sobre el uso de DirectX para la generación de gráficos 3D y nos adentraremos un poco más profundamente, en el desarrollo de gráficos e imágenes 2D con GDI+.

Módulo 3 - Capítulo 4

- 1. Gráficos 3D
- 2. Gráficos 2D
- 3. Dibujando líneas con GDI+
- 4. Dibujando curvas con GDI+
- 5. Dibujando cadenas de texto con GDI+
- 6. Otras consideraciones

 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 4: Trabajo con imágenes y gráficos

Gráficos 3D

- Gráficos 2D
- Dibujando líneas con GDI+
- Dibujando curvas con GDI+
- Dibujando cadenas de texto con GDI+
- Otras consideraciones

Módulo 3 - Capítulo 4

1. Gráficos 3D

Para trabajar con imágenes y gráficos en 3D, deberemos utilizar DirectX, ya que dentro de la plataforma .NET de Microsoft, no tenemos la posibilidad de crear representaciones 3D.

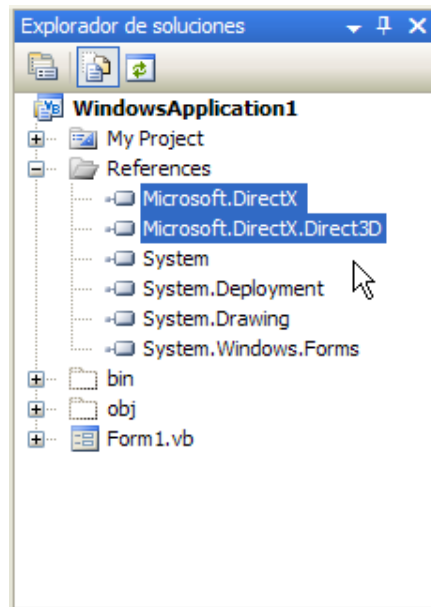
Esto significa por otra parte, que si desarrollamos una aplicación con representación 3D con DirectX, deberemos distribuir también las librerías DirectX en la máquina en la que se ejecute nuestra aplicación. Es decir, no bastará con distribuir Microsoft .NET Framework.

Pero DirectX no nos permite sólo crear gráficos e imágenes en 3D, también podemos crear imágenes en 2D, pero lo más normal en este último caso y salvo que no requeramos un uso continuado de DirectX, será utilizar en su lugar GDI+, como veremos más adelante.

Para trabajar con DirectX en Visual Basic 2008, deberemos añadir una referencia al proyecto con la librería o librerías COM de DirectX, eso si trabajamos con DirectX 8 o anterior, ya que a partir de DirectX 9, Microsoft ha proporcionado un conjunto de clases y librerías que interactúan con .NET directamente, permitiéndonos ejecutar aplicaciones .NET con DirectX administrado.

En nuestro caso, lo mejor es realizar un ejemplo, no sin antes recordar, que debería descargar Microsoft DirectX SDK 9 e instalarlo en su sistema. El SDK contiene ficheros de ayuda y documentación que le enseñará a crear aplicaciones con DirectX 9 en Visual Basic 2008.

Una vez que tenemos instalado en nuestro sistema las librerías de desarrollo de DirectX para .NET, iniciaremos una nueva aplicación Windows, añadiremos las referencias a las librerías *Microsoft.DirectX* y *Microsoft.DirectX.Direct3D* tal y como se muestra en la figura 1 y las cuáles encontraremos normalmente en el directorio `c:\windows\system32\`.



Referencias a DirectX añadidas a nuestro proyecto

Figura 1

A continuación, escribiremos el siguiente código:

```

Código

Imports Microsoft.DirectX

Imports Microsoft.DirectX.Direct3D

Public Class Form1

    Public Sub CreateDevice()

        Dim pp As New PresentParameters()

        pp.Windowed = True

        pp.SwapEffect = SwapEffect.Discard

        Dim dv As New Device(0, DeviceType.Hardware, Me,
        CreateFlags.SoftwareVertexProcessing, pp)

        Dim vertices(6) As CustomVertex.TransformedColored

        vertices(0).Position = New Vector4(Me.Width / 2.0F, 70.0F,
        0.5F, 1.0F))

        vertices(1).Position = New Vector4(Me.Width - (Me.Width /
        5.0F), Me.Height - (Me.Height / 5.0F), 0.5F, 1.0F))

        vertices(2).Position = New Vector4(Me.Width / 5.0F,
        Me.Height - (Me.Height / 5.0F), 0.5F, 1.0F))
    
```

```

        vertices(3).Position = New Vector4(Me.Width / 2.0F, 50.0F,
0.5F, 1.0F))

        vertices(4).Position = New Vector4(Me.Width - (Me.Width /
5.0F), Me.Height - (Me.Height / 5.0F), 0.5F, 1.0F))

        vertices(5).Position = New Vector4(Me.Width / 5.0F,
Me.Height - (Me.Height / 5.0F), 0.5F, 1.0F))

        dv.Clear(ClearFlags.Target, System.Drawing.Color.BlueViolet,
2.0F, 0)

        dv.BeginScene()

        dv.VertexFormat = VertexFormats.Transformed

        dv.DrawUserPrimitives(PrimitiveType.TriangleList, 2,
vertices)

        dv.EndScene()

        dv.Present()

End Sub

Protected Overrides Sub OnPaint(ByVal e As
System.Windows.Forms.PaintEventArgs)

    MyBase.OnPaint(e)

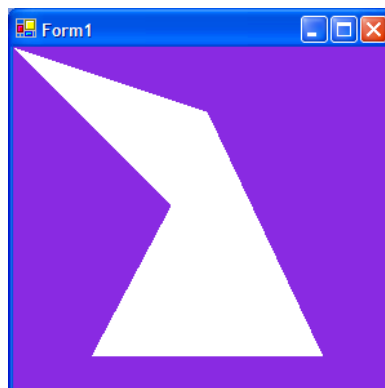
    CreateDevice()

End Sub

End Class

```

Este pequeño ejemplo demostrativo en ejecución del uso de DirectX desde nuestras aplicaciones Visual Basic 2008, es el que puede verse en la figura 2



Ejemplo de DirectX con Visual Basic 2008 en ejecución

Figura 2

Lección 4: Trabajo con imágenes y gráficos

- Gráficos 3D

Gráficos 2D

- Dibujando líneas con GDI+
- Dibujando curvas con GDI+
- Dibujando cadenas de texto con GDI+
- Otras consideraciones

Módulo 3 - Capítulo 4

2. Gráficos 2D

Las APIs de GDI+ corresponden a la evolución natural de las APIs y librerías GDI que se utilizaban en otros lenguajes de desarrollo. GDI+ no es otra cosa que un conjunto de clases desarrolladas para el entorno .NET y por esa razón, podemos entonces dibujar y crear representaciones gráficas en Visual Basic 2008.

Todas las clases GDI+, pueden ser localizadas a través del nombre de espacio **System.Drawing**. Así, podemos acceder a todas las posibilidades que nos ofrece GDI+ y las cuales veremos más adelante.

GDI+, no accede al hardware directamente, interactúa con los *drivers* de los dispositivos gráficos. Como particularidad, debemos saber que GDI+ está soportado por Win32 y Win64.

Respecto a los sistemas operativos y el soporte de estos para GDI+, Windows XP contiene la librería **gdiplus.dll** que encontraremos normalmente en `c:\windows\system32\` y la cuál nos permite trabajar con GDI+. Microsoft .NET por otra parte, nos proporciona el acceso directo a esta librería para poder desarrollar nuestras aplicaciones de forma mucho más cómoda y sencilla.

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 4: Trabajo con imágenes y gráficos

- Gráficos 3D
- Gráficos 2D
- Dibujando líneas con GDI+
- Dibujando curvas con GDI+
- Dibujando cadenas de texto con GDI+
- Otras consideraciones

Módulo 3 - Capítulo 4

3. Dibujando líneas con GDI+

Lo primero que aprenderemos a representar con GDI+ y Visual Basic 2008, son líneas muy sencillas.

Líneas simples

Cuando representamos líneas, debemos tener en cuenta varios aspectos. Una línea está representada por dos puntos. Cuando la representamos en un plano, La representación de una línea tiene dos pares de puntos (esto me recuerda a mis tiempos de estudiante con el álgebra y el cálculo). Para representar una línea por lo tanto, debemos indicar un par de puntos (x, y) cuyas coordenadas (horizontal, vertical), representa en este orden, el lugar o punto de inicio indicado por el margen superior del formulario o superficie sobre la cuál deseamos dibujar nuestra línea, y un segundo par de puntos que representan la dirección final de nuestra línea.

Un ejemplo práctico de este tipo de representación es la que se detalla en el siguiente código fuente de ejemplo:

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
```



```

MyBase.OnPaint(e)

    e.Graphics.DrawLine(New System.Drawing.Pen(Color.DarkBlue,
2), 1, 1, 50, 50)

End Sub

End Class

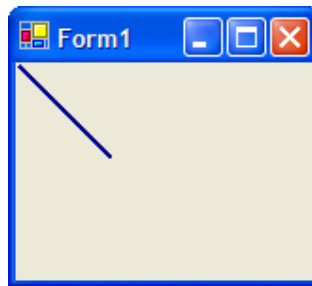
```

Atendiendo al código, observamos que hemos representado la gráfica indicando que queremos dibujar una línea *e.Graphics.DrawLine* indicando posteriormente una serie de atributos como el color del lápiz y tamaño o grosor de éste *Pen(Color.DarkBlue, 2)* y un conjunto de parámetros (x,y) que representan las coordenadas de la línea (1,1) y (50,50).

Otra representación similar sería *e.Graphics.DrawLine(Pens.DarkBlue, 1, 1, 50, 50)*, con la salvedad de que en este caso, el grosor del lápiz es el grosor por defecto, que es 1.

De todos los modos, la declaración *e.Graphics.DrawLine(New System.Drawing.Pen(Color.DarkBlue, 2), 1, 1, 50, 50)* y *e.Graphics.DrawLine(New System.Drawing.Pen(Color.DarkBlue, 2), 50, 50, 1, 1)* en el caso de la representación de líneas es igualmente compatible.

Este ejemplo de prueba en ejecución es el que se puede ver en la figura 1.



Ejemplo de dibujo con GDI+ de una línea recta en un formulario Windows

Figura 1

Líneas personalizadas

Sin embargo, la representación de líneas con GDI+ tiene diferentes particularidades que nos permiten sacar un alto grado de personalización a la hora de pintar o representar las imágenes y gráficos en nuestras aplicaciones.

La representación de líneas con GDI+, nos permite entre otras cosas, personalizar no sólo el color y el grosor de una línea, sino otras características de ésta como por ejemplo los extremos a dibujar.

Así, podemos dibujar unos extremos más o menos redondeados, puntiagudos, o personalizados.

Esto lo conseguimos hacer mediante las propiedades **StartCap** y **EndCap** de la clase **Pen** que es lo que vulgarmente he denominado como lápiz.

A estas propiedades, las podemos dar diferentes valores. Sirva el siguiente ejemplo de muestra de lo que podemos llegar a hacer.

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As
System.Windows.Forms.PaintEventArgs)

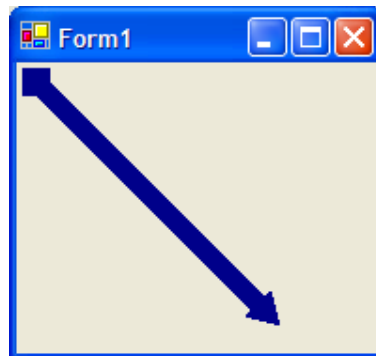
        MyBase.OnPaint(e)

        Dim Lapis As New Pen(Color.DarkBlue, 10)

        Lapis.StartCap = Drawing2D.LineCap.DiamondAnchor
        Lapis.EndCap = Drawing2D.LineCap.ArrowAnchor
        e.Graphics.DrawLine(Lapis, 10, 10, 140, 140)

    End Sub

End Class
```



Demostración de como representar diferentes extremos en una línea con GDI+
Figura 2

Trazando caminos o rutas de líneas

Otra posibilidad de GDI+ es la de crear líneas entrelazadas sin llegar a cerrarlas. Esto se hace con el método **AddLine**.

De esta manera, podemos dibujar diferentes líneas para representarlas en el marco de trabajo.

El siguiente ejemplo, nos enseña a utilizar el método **AddLine**.

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

        MyBase.OnPaint(e)

        Dim Ruta As New Drawing2D.GraphicsPath()

        Ruta.StartFigure()

        Ruta.AddLine(New PointF(10, 10), New PointF(100, 10))

        Ruta.AddLine(New PointF(10, 10), New PointF(170, 100))

        Ruta.AddLine(New PointF(170, 100), New PointF(130, 50))

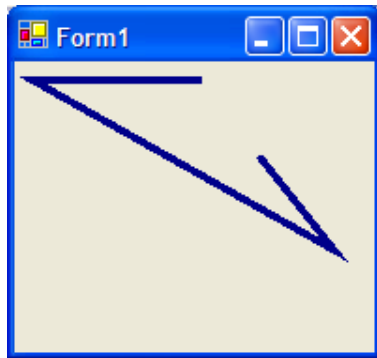
        Dim Lapis As New Pen(Color.DarkBlue, 4)

        e.Graphics.DrawPath(Lapis, Ruta)

    End Sub

End Class
```

En la figura 3 podemos ver el resultado de crear diferentes líneas en Visual Basic 2008 con GDI+.



Ejecución del ejemplo de uso del método **AddLine** con Visual Basic 2008

Figura 3

Observando el código, vemos que hemos declarado el método **StartFigure** y el método **AddLine** de la clase **GraphicsPath**.

Finalmente y para representar la gráfica correspondiente, hemos llamado al método **DrawPath**.

Líneas con texturas

Otra de las características de la representación gráfica de líneas con Visual Basic 2008 y GDI+, es la posibilidad de representar líneas aplicando a esas líneas una determinada textura.

El siguiente ejemplo, aplica una textura de la bandera de la Comunidad Económica Europea a un camino de líneas.

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

        MyBase.OnPaint(e)

        Dim Imagen As New Bitmap("c:\Flag.bmp")

        Dim Cepillo As New TextureBrush(Imagen)

        Dim TexturaPincel As New Pen(Cepillo, 20)

        Dim Ruta As New Drawing2D.GraphicsPath()

        Ruta.StartFigure()

        Ruta.AddLine(New PointF(10, 10), New PointF(180, 50))

    End Sub

End Class
```

```
Ruta.AddLine(New PointF(10, 100), New PointF(180, 140))

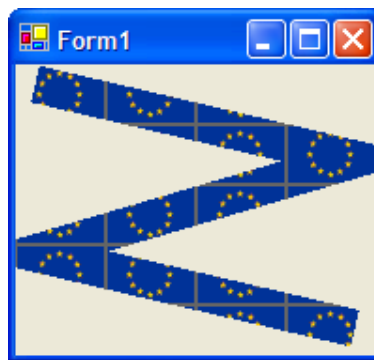
e.Graphics.DrawPath(TexturaPincel, Ruta)

End Sub

End Class
```

Como podemos observar, lo primero que hacemos es cargar una imagen que será la textura que utilizaremos, dentro de un objeto **Bitmap**, para posteriormente, preparar el trazo con su textura y tamaño.

Luego creamos una ruta o camino que marcaremos para dibujarla en el formulario Windows en nuestro caso, tal y como puede verse en la figura 4.



Ejemplo de un trazo de línea aplicando texturas

Figura 4

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 4: Trabajo con imágenes y gráficos

- Gráficos 3D
- Gráficos 2D
- Dibujando líneas con GDI+
- Dibujando curvas con GDI+
- Dibujando cadenas de texto con GDI+
- Otras consideraciones

Módulo 3 - Capítulo 4

4. Dibujando curvas con GDI+

La representación de curvas con GDI+ tiene cierta similitud con la representación de líneas.

A continuación veremos las partes más destacables en la creación de trazos curvos con Visual Basic 2008 y GDI+.

Trazando curvas simples

Lo más sencillo de todo es siempre el trazo de una línea curva genérica con Visual Basic 2008.

Esto lo conseguiremos con el siguiente código:

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

        MyBase.OnPaint(e)

        Dim Puntos As PointF() = {New PointF(10, Math.Sin(1) * 100), _
```

```

100), _
New PointF(60, Math.Sin(0) *
100), _
New PointF(110, Math.Sin(1) *
100), _
New PointF(160, Math.Sin(0) *
100), _
New PointF(210, Math.Sin(1) *
100)}

e.Graphics.DrawCurve(New Pen(Color.DarkOrange, 4), Puntos,
2.0F)

End Sub

End Class

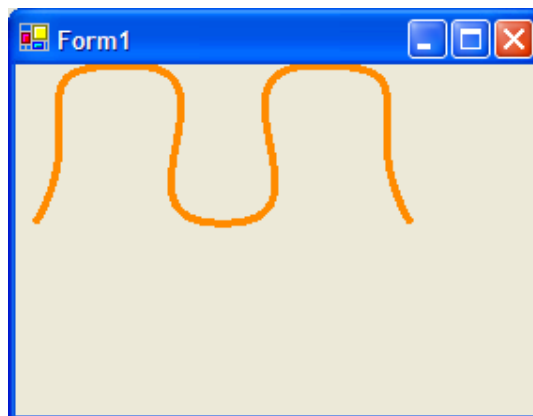
```

Si observamos este código, veremos que lo que hacemos es crear un conjunto de puntos para representar los trazos o líneas rectas que unan los puntos. Esto lo conseguimos utilizando la clase **PointF**.

Posteriormente utilizamos el método **DrawCurve** con la salvedad de que el tercer parámetro, indica la tensión de la curva.

Si este valor recibe un *0.0F*, la curva no tendrá tensión y por lo tanto, la representación será en trazos rectos, mientras que si ese valor aumenta, la tensión aumenta produciéndose el efecto curvo que deseamos conseguir.

El ejemplo anterior en ejecución es el que se muestra en la figura 1



Ejemplo en ejecución de la representación gráfica de trazos curvos

Figura 1

Curvas de Bézier

Otra posibilidad que nos ofrece GDI+ es la representación de curvas de Bézier, algo que conseguiremos gracias al método **DrawBezier** de GDI+.

La representación de las curvas de Bézier pueden hacerse mediante dos pares de puntos (x,y) o mediante cuatro coordenadas de puntos que definen la asignación de las curvas a representar.

El siguiente ejemplo nos muestra como representar una curva de Bézier con Visual Basic 2008 y GDI+.

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

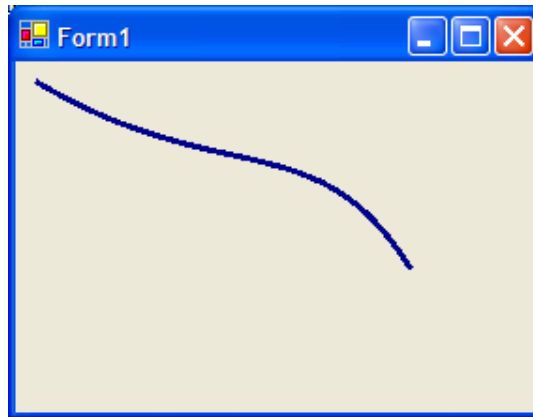
        MyBase.OnPaint(e)

        e.Graphics.DrawBezier(New Pen(Color.DarkBlue, 3), _
                               New PointF(10, 10), _
                               New PointF(110, 70), _
                               New PointF(160, 30), _
                               New PointF(210, 110))

    End Sub

End Class
```

Este código en ejecución es el que puede verse en la figura 2.



Representación de las curvas de Bézier con GDI+ en Visual Basic 2008

Figura 2

Rellenando curvas

En algunas ocasiones, nos podemos ver con la necesidad de crear curvas y de rellenar su interior para destacarlo de alguna manera. Esto es lo que veremos a continuación.

El método **AddArc** nos permite añadir una serie de puntos para representar una circunferencia.

En realidad, se representan puntos, los cuales son el punto (x,y) inicial, el ancho y el alto de la representación, el ángulo de comienzo de representación, y el ángulo final de la representación.

En ejemplo que veremos a continuación, utilizaremos también los métodos **FillPath** y **DrawPath**, para representar la curva en el formulario.

El código del ejemplo es el que se detalla a continuación:

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

        MyBase.OnPaint(e)

        Me.BackColor = Color.LightBlue

        Dim Camino As New Drawing2D.GraphicsPath()

        Camino.AddArc(50, 0, 150, 150, 0, 180)
```

```

        e.Graphics.FillPath(Brushes.White, Camino)

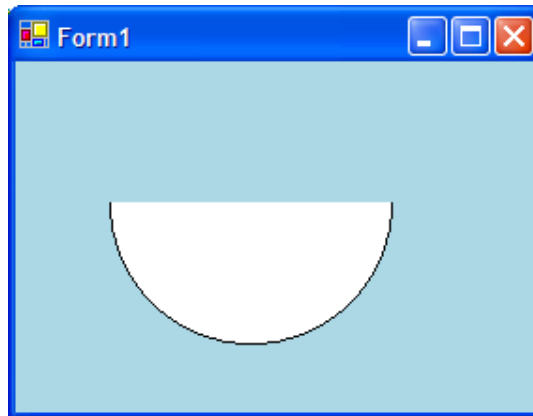
        e.Graphics.DrawPath(Pens.Black, Camino)

    End Sub

End Class

```

En la figura 3, podemos ver el ejemplo en ejecución.



Ejemplo de un dibujo de curvas cerradas rellenando su interior

Figura 3

Dibujando tartas

Otra posibilidad que nos ofrece GDI+ es la representación de las conocidas tartas. Todas estas representaciones son representaciones 2D, pero siempre se puede emular una representación 3D, superponiendo tartas una encima de otra.

El siguiente ejemplo, utiliza los métodos **FillPie** y **DrawPie** para generar un gráfico de tarta y rellenarlo de un determinado color.

El código de nuestro ejemplo, quedaría de la siguiente manera:

```

Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As
System.Windows.Forms.PaintEventArgs)

```

```

MyBase.OnPaint(e)

    e.Graphics.FillPie(Brushes.LightBlue, 50, 20, 120.0F,
120.0F, 115.0F, 180.0F)

    e.Graphics.DrawPie(Pens.DarkBlue, 50, 20, 120.0F, 120.0F,
115.0F, 180.0F)

    e.Graphics.FillPie(Brushes.White, 50, 20, 120.0F, 120.0F,
0.0F, 115.0F)

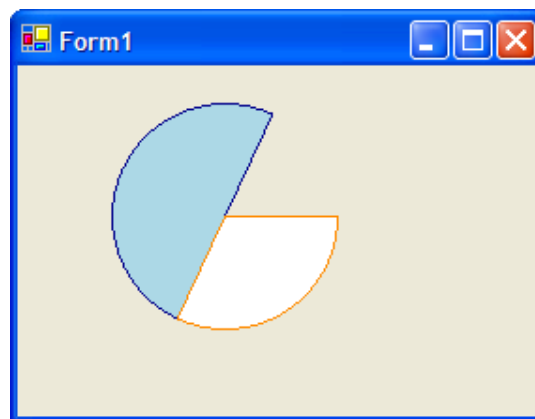
    e.Graphics.DrawPie(Pens.DarkOrange, 50, 20, 120.0F, 120.0F,
0.0F, 115.0F)

End Sub

End Class

```

Nuestro ejemplo en ejecución, es el que se muestra en la figura 4.

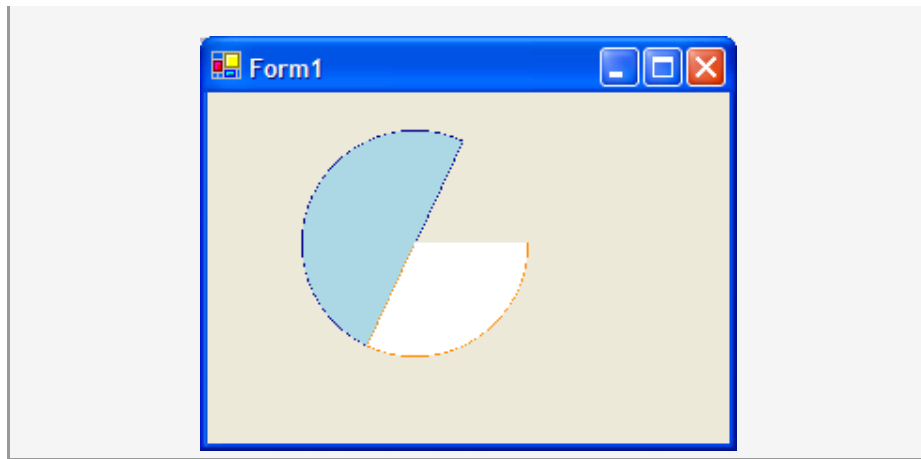


Demostración de cómo crear gráficos de tartas y como rellenar su interior

Figura 4

Consejo:

Cuando represente gráficos de tartas por ejemplo y desee rellenar una parte de esta de un color y marcar el borde en otro color, use primero el método FillPie y después el método DrawPie, en caso contrario, la representación gráfica perderá nitidez como se indica en la siguiente imagen:



» [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 4: Trabajo con imágenes y gráficos

- Gráficos 3D
- Gráficos 2D
- Dibujando líneas con GDI+
- Dibujando curvas con GDI+
- Dibujando cadenas de texto con GDI+**
- Otras consideraciones

Módulo 3 - Capítulo 4

5. Dibujando cadenas de texto con GDI+

Como hemos visto ya en el capítulo referente a la creación de nuestros propios controles, desde Visual Basic 2008, podemos utilizar GDI+ para crear cadenas de texto y manipularlas como deseemos.

En los siguientes apartados veremos como representar cadenas de texto desde Visual Basic 2008

Dibujando cadenas de texto

El método **DrawString** nos permite representar cadenas de texto de forma gráfica.

El funcionamiento en Visual Basic 2008 de estas instrucciones es realmente simple. El siguiente ejemplo, ilustra en cómo abordar un pequeño proyecto para representar una cadena de texto en pantalla.

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As
System.Windows.Forms.PaintEventArgs)

        MyBase.OnPaint(e)
```

```

        Dim MiFuente As New Font("Verdana", 24, FontStyle.Bold)

        Dim Brocha As New SolidBrush(Color.BurlyWood)

        e.Graphics.DrawString("Ejemplo GDI+", MiFuente, Brocha, 10,
10)

    End Sub

End Class

```

Este ejemplo en ejecución es el que puede verse en la figura 1.



Ejemplo de cómo insertar una cadena de texto gráficamente en un formulario

Figura 1

Dibujando cadenas de texto con textura

Otra particularidad que a estas alturas ya no lo es tanto, es la posibilidad de trabajar con texturas dentro de cadenas de texto.

Como si estuviéramos dibujando una cadena de texto, la única variación es que asignamos como brocha de dibujo, una textura determinada.

El siguiente código, aclarará suficientemente esto que comento.

```

Código

Imports System.Drawing

Public Class Form1

```

```

Protected Overrides Sub OnPaint(ByVal e As
System.Windows.Forms.PaintEventArgs)

    MyBase.OnPaint(e)

    Dim Imagen As New Bitmap("c:\Flag.bmp")

    Dim TexturaDeFondo As New TextureBrush(Imagen)

    Dim MiFuente As New Font("Arial", 30, FontStyle.Bold)

    e.Graphics.DrawString("Ejemplo GDI+", MiFuente,
TexturaDeFondo, 4, 10)

End Sub

End Class

```

El resultado de este ejemplo, es el que puede verse en la figura 2.



Demostración del efecto de añadir una textura a la representación de una cadena de texto

Figura 2

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 4: Trabajo con imágenes y gráficos

- Gráficos 3D
 - Gráficos 2D
 - Dibujando líneas con GDI+
 - Dibujando curvas con GDI+
 - Dibujando cadenas de texto con GDI+
- Otras consideraciones

Módulo 3 - Capítulo 4

6. Otras consideraciones

Uso de degradados con GDI+

GDI+ proporciona un variadísimo juego de brochas que nos permiten dibujar degradados en un formulario o control que permita el trabajo con gráficos.

El siguiente ejemplo de código, nos muestra como dibujar un degradado en un formulario Windows.

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
        MyBase.OnPaint(e)

        Dim Forma As New Rectangle(New Point(0, 0), Me.ClientSize)

        Dim Gradiente As New Drawing2D.LinearGradientBrush(Forma, _
            Color.Cyan, _
            Color.DarkBlue, _
```



```
Drawing2D.LinearGradientMode.ForwardDiagonal)  
  
    e.Graphics.FillRegion(Gradiente, New Region(Forma))  
  
End Sub  
  
End Class
```

La figura 1 representa el ejemplo anterior en ejecución



Representación gráfica de un degradado en una ventana Windows

Figura 1

Evidentemente, podemos jugar con la clase ***LinearGradientBrush*** y con la lista enumerada ***LinearGradientMode*** para dar un aspecto o un toque ligeramente diferente al degradado, como el que se indica en la figura 2



Otro ejemplo de representación degradada en un formulario Windows

Figura 2

Insertando y trabajando con imágenes con System.Drawing

Sirva como detalle general, que GDI+ nos permite también, trabajar con imágenes. Para cargar una imagen en un formulario con GDI+, utilizaremos el método ***DrawImage***.

Un ejemplo del uso de este método para cargar una imagen en un formulario es el siguiente:

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

        MyBase.OnPaint(e)

        e.Graphics.DrawImage(New Bitmap("c:\159.jpg"), 1, 1)

    End Sub

End Class
```

La figura 3 nos muestra el resultado final de insertar una imagen en el formulario Windows.

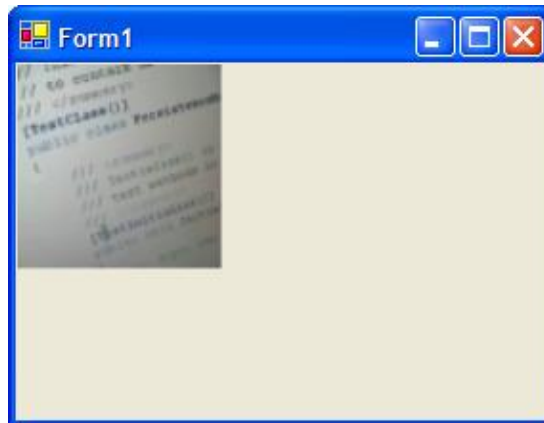


Imagen insertada con GDI+ dentro del formulario Windows

Figura 3

Aplicando transparencias a una imagen

Otra de las características que nos ofrece GDI+, es el trabajo con imágenes aplicando transparencias. Para realizar esto, deberemos indicar el color o colores que queremos utilizar para provocar en él un efecto transparente.

El siguiente ejemplo, nos muestra como hacer esto utilizando para ello el método **MakeTransparent**.

```
Código

Imports System.Drawing

Public Class Form1

    Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

        MyBase.OnPaint(e)

        Dim Imagen As New Bitmap("c:\Flag.bmp")

        e.Graphics.DrawImage(New Bitmap(Imagen), 1, 1)

        Imagen.MakeTransparent(Color.FromArgb(255, 0, 51, 153))

        e.Graphics.DrawImage(New Bitmap(Imagen), 100, 1)

    End Sub

End Class
```

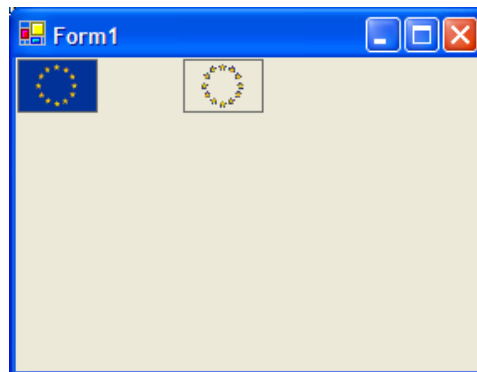


Imagen insertada con GDI+ dentro del formulario Windows

Figura 4

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 5: Despliegue de aplicaciones

- Desmitificando los ensamblados
- Desplegando con XCOPY
- GAC y Strong Names
- Creando un paquete de instalación
- Otras consideraciones

Introducción

En este capítulo, aprenderemos lo que son los ensamblados o *assemblies*, un término completamente nuevo para los desarrolladores de otros entornos distintos de .NET, y aprenderemos a desplegar o instalar nuestras aplicaciones.

Módulo 3 - Capítulo 5

- 1. Desmitificando los ensamblados
- 2. Desplegando con XCOPY
- 3. GAC y Strong Names
- 4. Creando un paquete de instalación
- 5. Otras consideraciones

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 5: Despliegue de aplicaciones

Desmitificando los ensamblados

- Desplegando con XCOPY
- GAC y Strong Names
- Creando un paquete de instalación
- Otras consideraciones

Módulo 3 - Capítulo 5

1. Desmitificando los ensamblados

Un concepto completamente nuevo para los que nunca han desarrollado con .NET, es la palabra ***Assembly***, denominada *ensamblado*.

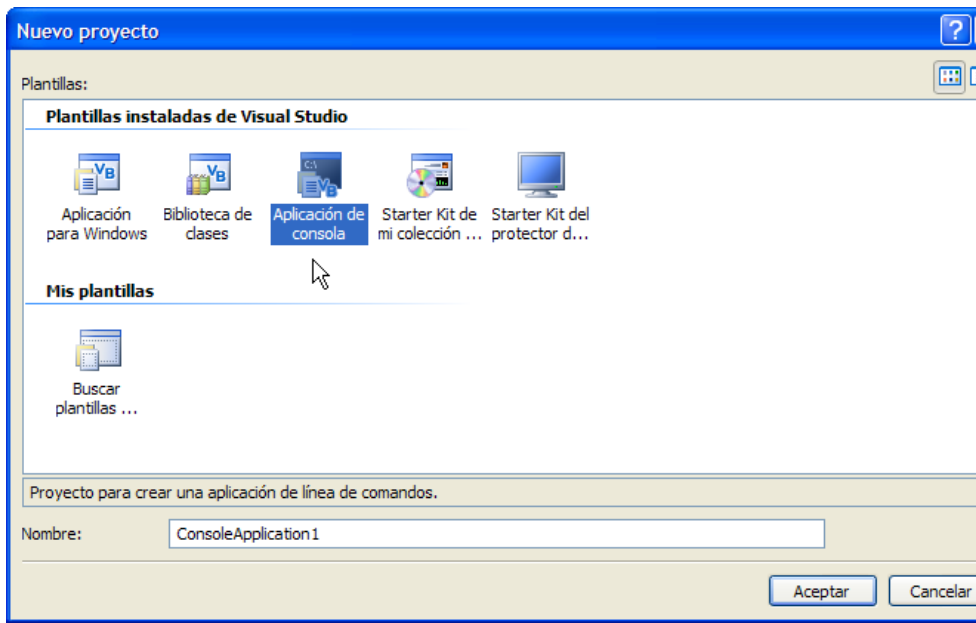
Los ensamblados, son para entenderlo muy rápidamente, como los ejecutables de una aplicación.

La única diferencia notable, es que en .NET, un proyecto o aplicación, se compila en código intermedio, el conocido como *Intermediate Language* o lenguaje intermedio, que luego interpretará el CLR o *Common Language Runtime* para ejecutarla en el sistema operativo correspondiente.

Ese código intermedio, es el ensamblado de nuestra aplicación que a su vez puede contener uno o más ficheros, y a su vez, un proyecto, puede estar contenido por uno o más ensamblados.

Por lo tanto, dentro de un ensamblado, se encierran algunas partes importantes que debemos conocer.

Lo mejor para entender bien lo que hay en un ensamblado y que contiene es que abramos Visual Studio 2008 y seleccionemos una plantilla de proyecto de tipo ***Console Application*** como se muestra en la figura 1.



Seleccionando un proyecto de tipo Console Application

Figura 1

A continuación, escribiremos un ejemplo simple de consola, para que estudiemos el resultado de éste.

El código fuente de nuestra pequeña aplicación de ejemplo, es el que se detalla a continuación:

```
Código

Module Module1

    Sub Main()

        Console.WriteLine("Ejemplo de consola")

        Console.WriteLine("")

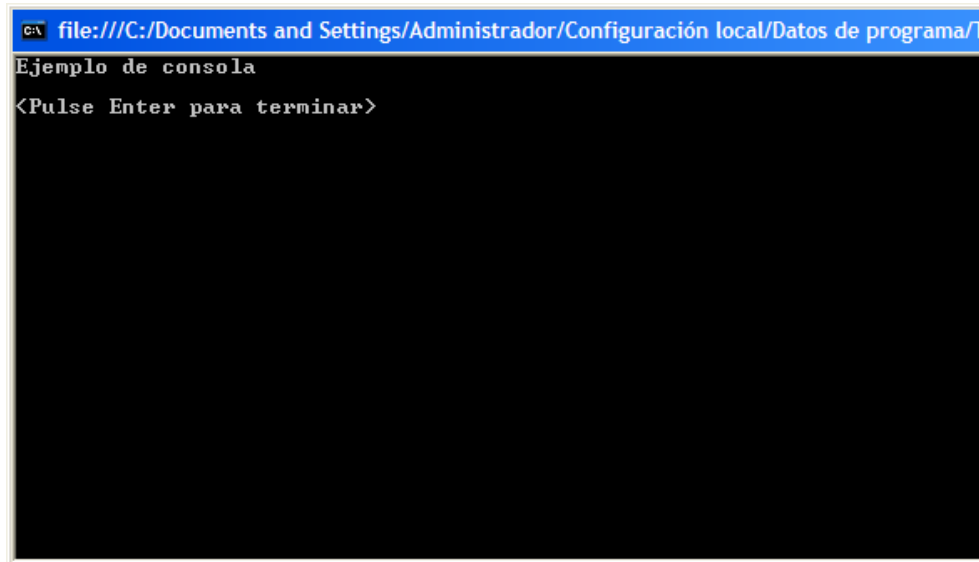
        Console.Write("<Pulse Enter para terminar>")

        Console.ReadLine()

    End Sub

End Module
```

Nuestro ejemplo de prueba en ejecución, es el que puede verse en la figura 2.



Ejecución del ejemplo de Consola

Figura 2

Ahora bien, lo que tenemos una vez compilamos nuestra aplicación de consola, no es un ejecutable como tal o como lo entenderíamos en otros compiladores, por ejemplo en Visual C++.

En Visual C++, generábamos un ejecutable nativo al sistema en el cuál compilábamos el proyecto y si ese ejecutable lo llevábamos a otra máquina con otro sistema operativo diferente a Windows, ese programa no iba a funcionar. Con .NET y en nuestro caso con Visual Basic 2008, este concepto ha cambiado. El proyecto que hemos desarrollado no se compila a un ejecutable nativo, sino que el sistema .NET lo compila a un lenguaje intermedio, que luego el **CLR** de la máquina en la cuál lanzamos nuestra aplicación, será capaz de interpretar adecuadamente.

Puede que estos conceptos le puedan desorientar un poco, pero es muy fácil de comprender.

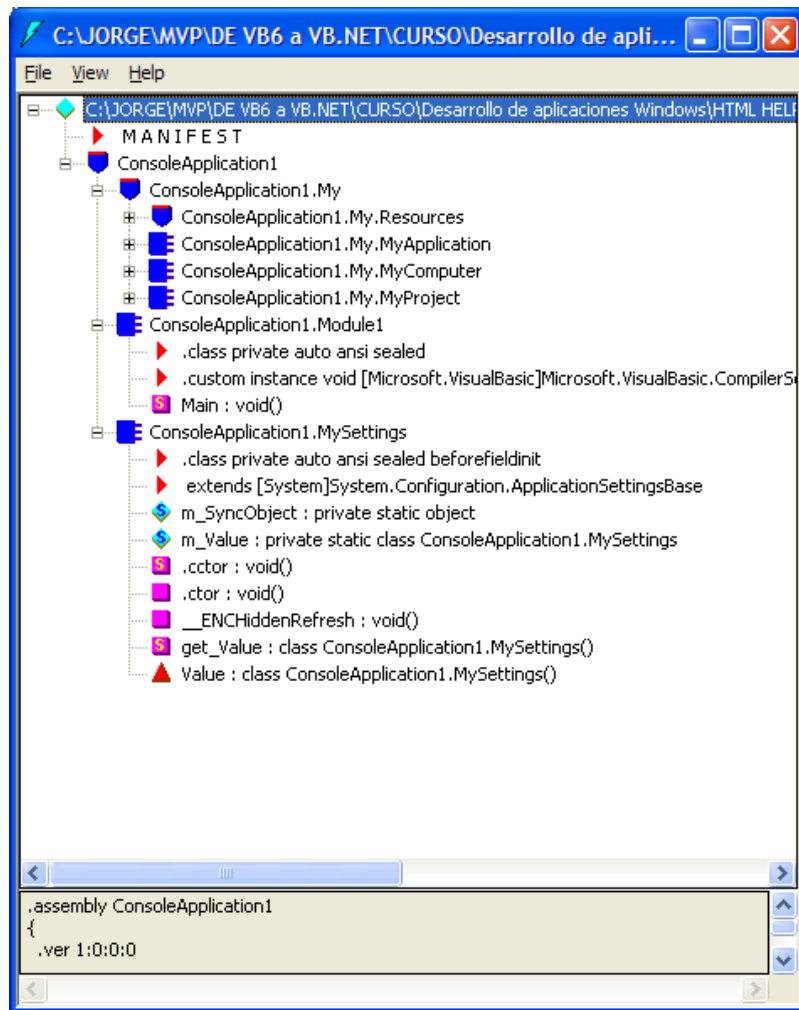
Adicionalmente a esto, lo suyo es destripar un poco lo que hay dentro del ensamblado que hemos convertido a código intermedio. Para llevar a cabo nuestra tarea, haremos uso de una herramienta externa de Microsoft y del entorno Visual Studio 2008, que nos permite analizar el ensamblado de un proyecto convertido a código intermedio.

Aclaración:

*Un ensamblado o el código intermedio, (**IL** a partir de ahora), no es ni el código fuente de nuestra aplicación ni el programa ejecutable. El **IL** se puede analizar con una herramienta de Microsoft denominada **ildasm**.*

Si está usando una versión Express de Visual Studio debe saber que esta herramienta no se incluye con estas versiones, sólo forma parte del SDK de .NET Framework que se incluye con las versiones de Visual Studio 2008.

Cuando abrimos el fichero ejecutable de nuestra aplicación con la herramienta *ildasm.exe*, observamos que esta tiene una gran cantidad de información, como se indica en la figura 3.



ildasm con el fichero ejecutable de nuestra aplicación de ejemplo abierto

Figura 3

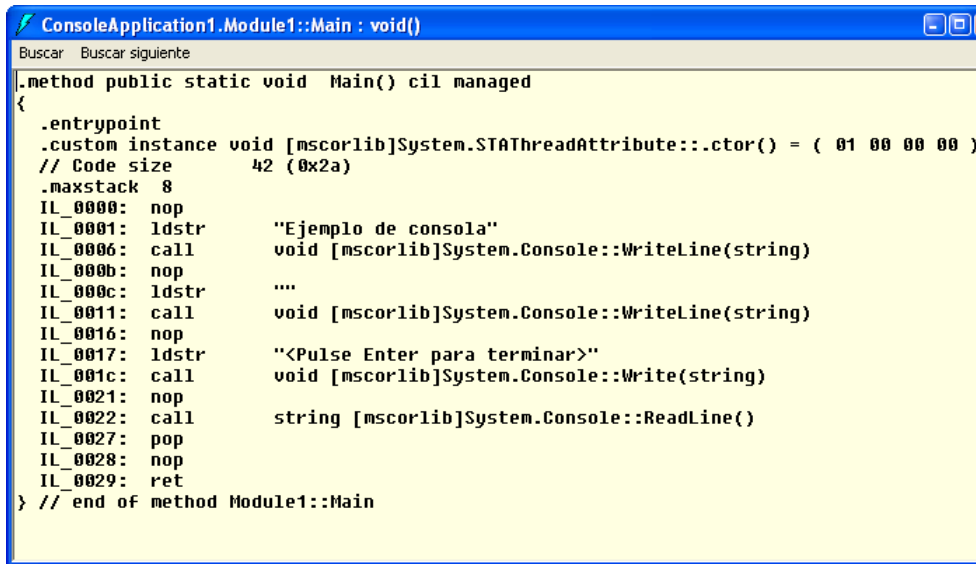
Antes de adentrarnos más en los entresijos de un ensamblado, piense en él como si fuera una colección de elementos.

Esos elementos, pueden ser recursos, tipos, funcionalidades, que todas juntas, forman nuestra aplicación.

Así, lo primero que vemos en la figura 3, es la palabra **MANIFEST**. Referencias a las librerías utilizadas que el **CLR** deberá interpretar posteriormente.

Luego encontramos otras el ámbito de la aplicación y las clases de ésta, con su método tanto estático como no estático.

No es cuestión de entrar mucho más en detalle del código **IL** que contiene un proyecto compilado en .NET como éste, pero es conveniente a verlo para entender los conceptos que estamos tratando. En la figura 4, tenemos el código **IL** del método **Main**.



```
.method public static void Main() cil managed
{
    .entrypoint
    .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )
    // Code size 42 (0x2a)
    .maxstack 8
    IL_0000: nop
    IL_0001: ldstr      "Ejemplo de consola"
    IL_0006: call       void [mscorlib]System.Console::WriteLine(string)
    IL_000b: nop
    IL_000c: ldstr      ""
    IL_0011: call       void [mscorlib]System.Console::WriteLine(string)
    IL_0016: nop
    IL_0017: ldstr      "<Pulse Enter para terminar>"
    IL_001c: call       void [mscorlib]System.Console::Write(string)
    IL_0021: nop
    IL_0022: call       string [mscorlib]System.Console::ReadLine()
    IL_0027: pop
    IL_0028: nop
    IL_0029: ret
} // end of method Module1::Main
```

El código IL del método Main de la aplicación de ejemplo

Figura 4

Los ensamblados como podemos ver, contiene más información de la que propiamente tendría un ejecutable "normal".

Un ensamblado en .NET, contiene también como hemos podido ver, datos e información que encontraríamos en cualquier tipo de librería, lo cuál representa además, toda la información necesaria del **CLR** en cualquier momento.

Con todo esto, podemos resumir por lo tanto, que un ensamblado contiene código, recursos y metadatos.

El código en **IL** es el código que será ejecutado por el **CLR**. Además, cualquier recurso (imágenes, metadatos, etc.) son accesibles en el ensamblado.

Los metadatos por su parte, contiene información sobre las clases, interfases, métodos y propiedades, que posibilitan toda la información necesaria por el **CLR** para poder ejecutar nuestra aplicación correctamente.

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 5: Despliegue de aplicaciones

- Desmitificando los ensamblados

Desplegando con XCOPY

- GAC y Strong Names
- Creando un paquete de instalación
- Otras consideraciones

Módulo 3 - Capítulo 5

2. Desplegando con XCOPY

Notas previas

Todas las aplicaciones desarrolladas con .NET están aisladas, no como ocurre con los compilados pre-.NET, de manera tal que los conflictos con las DLL se han visto reducidos enormemente, por no decir que han desaparecido. El famoso *infierno de las DLLs* que tanto hemos sufrido los desarrolladores, ha pasado ya a la historia.

También podemos usar componentes privados. Basta con copiarlos al mismo directorio en el que se encuentra nuestra aplicación ejecutable.

Además, .NET nos permite tener más de un componente versionado (diferentes versiones de un mismo componentes) dentro de un mismo ordenador, por lo que los problemas de compatibilidad estarían resueltos.

Con estos detalles, repasamos algunas de las mejoras a la hora de desplegar o instalar una aplicación desarrollada con Visual Basic 2008 en un sistema. A continuación, veremos algunas anotaciones adicionales que nos ayudarán a realizar estas y otras acciones.

XCOPY

Lo que nos ofrece **XCOPY** a los desarrolladores e ingenieros, es la posibilidad de instalar e implementar nuestra solución y proyecto a un sistema de una manera rápida, fiable y sin apenas impacto.

El método **XCOPY** para desplegar aplicaciones, pasa por alto la posibilidad de implementar los ensamblados en la **GAC**, algo que según determinadas circunstancias, resulta más que provechoso.

Hay que tener en cuenta, que si hacemos un mal uso de la **GAC**, ésta puede convertirse en un desván difícil de gestionar.

Utilice la **GAC** con criterio y si no quiere complicaciones, despliegue sus aplicaciones con **XCOPY**.

Como habrá podido ya observar, dentro de un proyecto de Visual Basic 2008, nos podemos encontrar con una extensa estructura de directorios. Esta estructura, lejos de ser una molestia, constituye las características más importantes a la hora de desplegar nuestras aplicaciones a través de **XCOPY**. Como vemos, todo en .NET tiene su sentido y tiene un porqué.

Para instalar nuestra aplicación desarrollada en Visual Basic 2008 en un sistema cliente, bastará por lo tanto, con realizar una acción similar al **XCOPY** de **DOS**, es decir, copiaremos en el ordenador cliente, los ficheros o ensamblados necesarios (ejecutables, recursos, dlls, etc.) de la estructura de nuestro proyecto.

Debemos tener en cuenta, que en otros lenguajes basados en COM, como Visual Basic 6, podíamos hacer esto igualmente, pero debíamos además registrar las DLL con aquel famosísimo comando *regsvr32* para que no hubiera problemas, aún así, algunas veces nos encontrábamos con algunos contratiempos, sin embargo, con Visual Basic 2008, esta forma de trabajar ha desaparecido y ahora el despliegue de una aplicación es mucho más sencilla.

Cuando desarrollemos una aplicación, tenga en cuenta otros recursos que utiliza en la misma.

Crystal Reports, bases de datos SQL Server, que la máquina cliente disponga de .NET Framework o de la versión mínima de MDAC necesaria, etc.

En caso contrario, la ejecución de nuestra aplicación, podría no funcionar o provocar algún tipo de excepción o error.

Por último, y aunque parezca de perogrullo, no olvide que debe tener los permisos necesarios para instalar y ejecutar los ensamblados y otras partes de Software, necesarios para ejecutar su aplicación.

📺 [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 5: Despliegue de aplicaciones

- Desmitificando los ensamblados
- Desplegando con XCOPY
- GAC y Strong Names**
- Creando un paquete de instalación
- Otras consideraciones

Módulo 3 - Capítulo 5

3. GAC y Strong Names

GAC

El **GAC** o *Global Assembly Cache* no es otra cosa que un repositorio de ensamblados globales.

Imaginemos que usted todos los días cuando llega a casa de trabajar, lo primero que hace es quitarse los zapatos y ponerse unas zapatillas cómodas para estar por casa. Lo lógico en este caso, será situar un zapatero o un pequeño armario para que ponga ahí las zapatillas, ya que todos los días, hace repetitivamente esta operación. Obviamente, las zapatillas deben de estar ahí y no en la otra punta de la casa, pero para estar en ese zapatero, deberá cumplir una serie de requisitos que usted mismo exige, por lo que no todos los zapatos o zapatillas deberían estar ahí.

El **GAC** funciona de una manera realmente semejante. En el **GAC** se incluyen aquellas librerías o ensamblados que son utilizados frecuentemente.

Si usted va a desarrollar y distribuir su propio ensamblado, convendría ponerlo en el **GAC**.

Una aplicación .NET, lo primero que hace cuando se ejecuta, es revisar los ensamblados que va a necesitar, y el primer sitio dónde va a ir a buscarlo es en el **GAC**.

Sino lo encuentra, se pondrá a buscarlo en el directorio en el que se encuentra el fichero ejecutable, pero esto repercute en el rendimiento. Si nuestras aplicaciones utilizan frecuentemente unos ensamblados, sería lógico y conveniente ponerlos en el **GAC**.

¿Y cómo se añade un ensamblado al **GAC**?

La tarea no es sencilla, ya que para añadirlo debemos realizar algunos pasos, entre los que está el crear un nombre fuerte o **Strong Name**.

Strong Names

Con los **Strong Names**, aseguramos un uso seguro de los componentes contenidos en el ensamblado.

Hay que tener en cuenta que un ensamblado declarado en la **GAC** que es llamado por varias aplicaciones, crea una única instancia.

De ahí, que crear un **Strong Name** para el ensamblado, está más que justificado.

Un **Strong Name**, nos asegura por otro lado, que el nombre de un ensamblado es único y que por lo tanto, al ser único, no puede ser utilizado por otros ensamblados.

Los **Strong Names** se generan con un par de claves, una de ellas de carácter público y otra de carácter privado, claves que se introducen en fichero de ensamblado de la aplicación, y que luego al compilar nuestra aplicación, queda registrada con ese **Strong Name**.

Para indicar un **Strong Names** a un ensamblado, debe crear primero el par de claves (clave pública y clave privada), que se generará en un fichero con extensión **snk**, y modificar posteriormente el fichero *AssemblyInfo.vb* de su proyecto. En ese archivo, debe añadir una instrucción similar a la siguiente:

```
<Assembly: AssemblyKeyFile("KeyFile.snk")>
```

» [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 5: Despliegue de aplicaciones

- Desmitificando los ensamblados
 - Desplegando con XCOPY
 - GAC y Strong Names
- Creando un paquete de instalación
- Otras consideraciones

Módulo 3 - Capítulo 5

4. Creando un paquete de instalación

Setup Project

Otra acción habitual, es que cuando desarrollemos nuestras aplicaciones, generemos diferentes dependencias con otros ensamblados o componentes, para lo cuál, la forma más sencilla de desplegar nuestra aplicación, es generando un paquete de distribución que contenga esas dependencias y relaciones.

Nota:

Si está utilizando Visual Basic 2008 Express no tendrá disponible la opción de proyecto Setup.

Con la plantilla *Setup Project* crearemos un nuevo proyecto para generar el paquete de distribución e instalación de nuestra aplicación y proyecto.

Cuando generamos el paquete de instalación, debemos tener en cuenta que se generan dos archivos que por lo general tendrán los nombres de *Setup.exe* y *Setup.msi*.

La diferencia entre ambos ficheros es que el fichero con extensión **exe** instalará *Windows Installer* si es necesario, mientras que el fichero con extensión **msi**, no instalará *Windows Installer*, por lo que si *Windows Installer* no está presente en la máquina en la que se ejecuta el programa de instalación, dará un error.

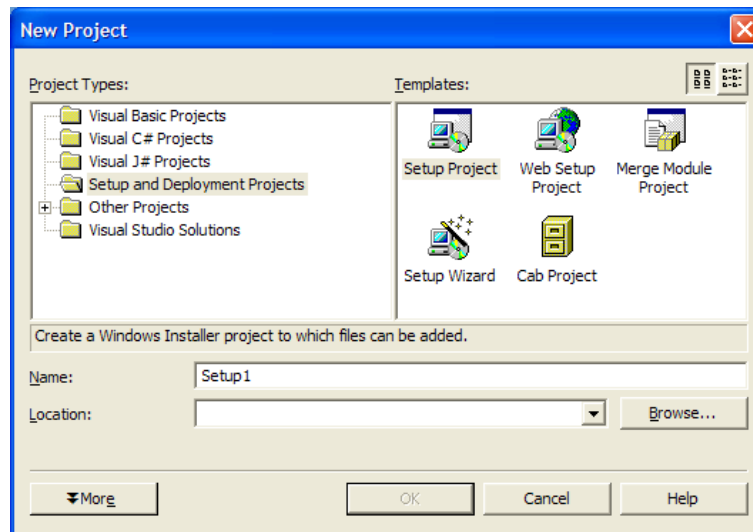
Otra consideración a tener en cuenta cuando generamos el paquete de instalación, es en el momento de la distribución, el asegurar que el sistema destino tenga el .NET Framework correspondiente.

Respecto al comportamiento de *Windows Installer* sobre aplicaciones ya instaladas que deseamos desinstalar, el entorno se comporta de manera tal, que si detecta componentes compartidos, estos no son desinstalados del sistema.

Otro comportamiento de *Windows Installer* es el que nos permite echar marcha atrás si el proceso de instalación se cancela o falla por algún motivo. *Windows Installer* dejará el sistema en el mismo estado que tenía justo antes de realizar la instalación.

Tipos de despliegues de proyectos

Cuando generamos un paquete de instalación tenemos dentro del entorno de desarrollo Visual Studio varias opciones, como se indica en la imagen 1. Así, nos podemos encontrar por lo general con diferentes tipos de despliegues de proyectos.



Tipos de despliegue de proyectos en Visual Studio

Figura 1

La plantilla **Cab Project** El fichero con extensión **CAB** es un fichero comprimido que contiene todos los ficheros y recursos necesarios para instalar nuestra aplicación. Por lo general, este tipo de ficheros son usados para descargarlos de Servidores Web.

Por lo general y en nuestro caso, crearemos aplicaciones Windows, y deberíamos entonces utilizar la plantilla **Setup Project**.

Si utilizamos la plantilla, **Merge Module Project**, crearemos un paquete instalación para componentes compartidos.

El uso de la plantilla, **Web Setup Project** por su parte, nos permitirá generar un paquete de instalación para aplicaciones basadas en Internet o aplicaciones Web.

La plantilla, **Setup Wizard** genera un asistente para generar uno de los cuatro anteriores tipos de proyectos de despliegue.

Es una forma rápida de generar el proyecto de despliegue.

[Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008

Lección 5: Despliegue de aplicaciones

- Desmitificando los ensamblados
 - Desplegando con XCOPY
 - GAC y Strong Names
 - Creando un paquete de instalación
- Otras consideraciones

Módulo 3 - Capítulo 5

5. Otras consideraciones

Setup Project

Ya lo hemos comentado por encima en los capítulos anteriores sobre el *Despliegue de aplicaciones*, pero cuando se procede a instalar y distribuir una aplicación en otros sistemas, se deben tener en cuenta diferentes aspectos que no podemos pasar por alto.

Debemos conocer en primer lugar, la naturaleza del sistema o sistemas destino dónde vamos a implantar nuestro programa.

Entendiendo y conociendo bien esto, podremos saber qué aplicaciones Software adicionales necesitaremos para abordar nuestro trabajo adecuadamente.

Entre otros, debemos tener en cuenta que el sistema destino dónde se va a ejecutar nuestra aplicación dispone de los drivers de acceso a datos adecuados, *MDAC* (versión 2.6 ó superior), y por supuesto de Microsoft .NET Framework.

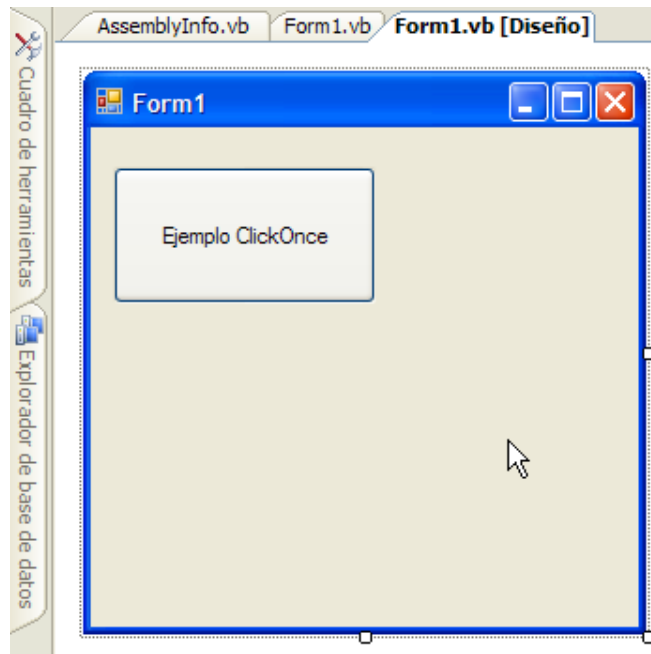
El anterior artículo, nos da un enfoque sobre qué tipo de instalación o despliegue establecer de nuestra aplicación desarrollada con Visual Basic 2008.

El concepto ClickOnce

En *Visual Studio 2008* se introduce un concepto nuevo denominado *ClickOnce*.

Este concepto representa la tecnología que permite instalar y ejecutar aplicaciones Windows desde un servidor Web con una escasa acción por parte del usuario.

Inicie un nuevo proyecto Windows. Dentro del formulario Windows inserte un control *Button* dentro del cuál, inserte un texto identificativo como el que se muestra en la figura 1.



Aplicación Windows de ejemplo para explicar el concepto de ClickOnce

Figura 1

A continuación, escriba el siguiente código fuente:

```
Código

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles Button1.Click

        MessageBox.Show("Ejemplo ClickOnce ejecutado a las:" &
vbCrLf & Date.Now.ToLongTimeString)

    End Sub

End Class
```

Una vez que ha escrito el código fuente de la aplicación, ejecútela y compruebe que funciona como se espera.

Si lo desea, abra el fichero *AssemblyInfo.vb* e indique la versión de la aplicación que desee. En mi caso he dejado la versión *1.0.0.0* que es la versión que aparece por defecto.

Después de esto, genere la aplicación como se indica en la figura 2.

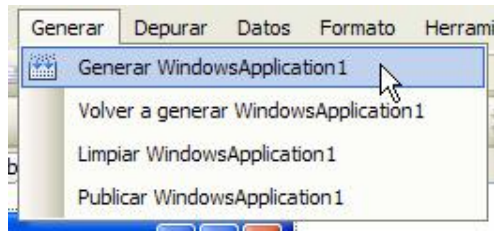


Figura 2

Por último, pulse el botón secundario del mouse sobre el proyecto y seleccione la opción **Publicar...** como se indica en la figura 3.

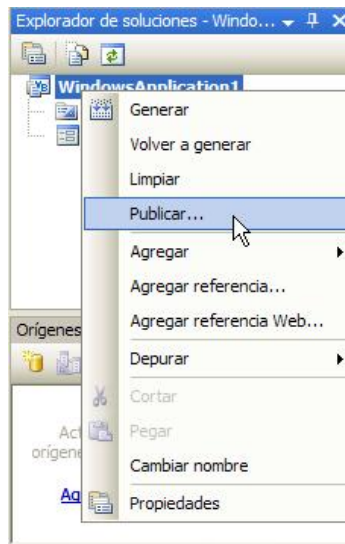


Figura 3

Al ejecutar la opción **Publicar...**, el entorno nos muestra una ventana similar a la que se muestra en la figura 4.

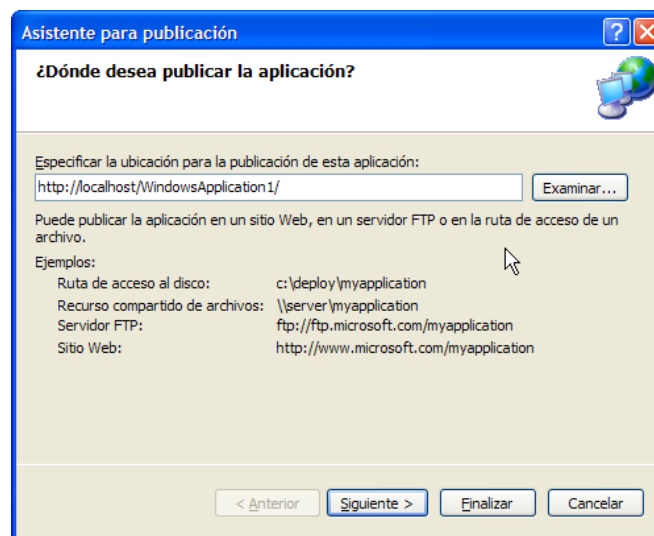
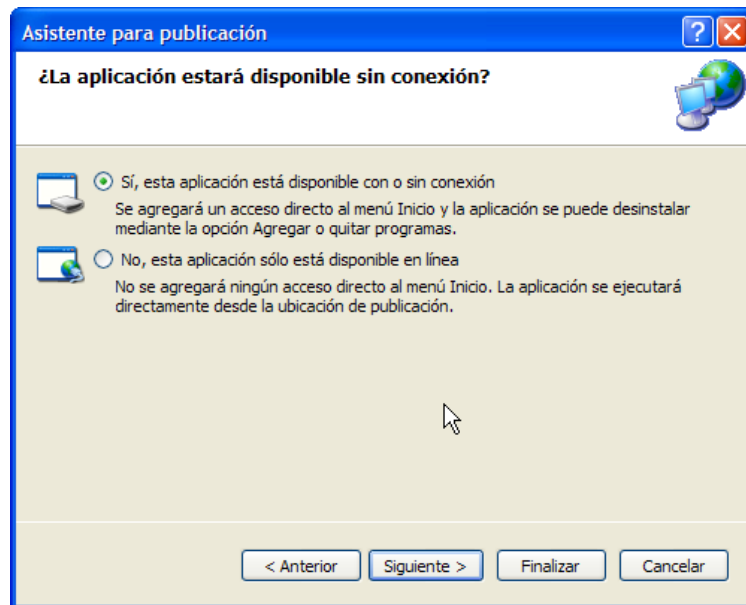


Figura 4

Seleccione una ubicación para publicar la aplicación y haga clic en el botón **Siguiente**.

Aparecerá en el asistente entonces, una ventana similar a la que se presenta en la figura 5.



Ventana del asistente para indicar dónde estará disponible la aplicación

Figura 5

Pulse el botón **Siguiente**.

Nuevamente, el asistente mostrará una última ventana similar a la que se muestra en la figura 6.

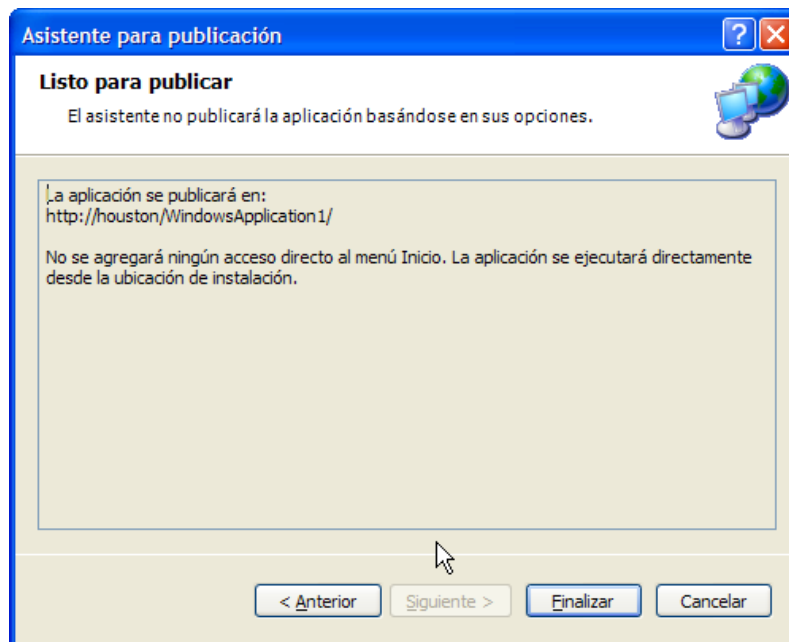
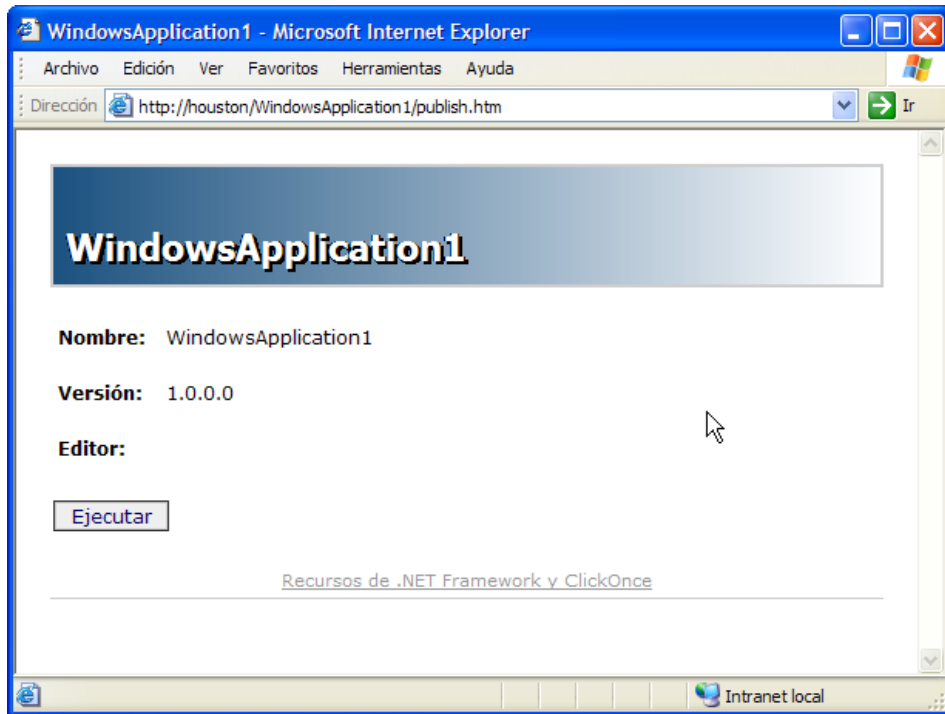


Figura 6

El asistente muestra en este caso, una información de resumen. Lo que haremos a continuación, será presionar el botón **Finalizar**.

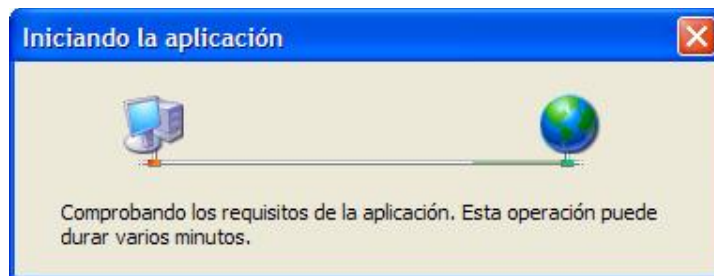
Con esta acción, nuestra aplicación está ya publicada en el servidor Web, por lo que si abrimos una ventana de nuestro explorador Web y escribimos la dirección en la cuál hemos publicado nuestra aplicación, ésta se ejecutará de forma correcta como se indica en la figura 7.



Aplicación Web del lanzamiento de la aplicación Windows a través del Servidor Web

Figura 7

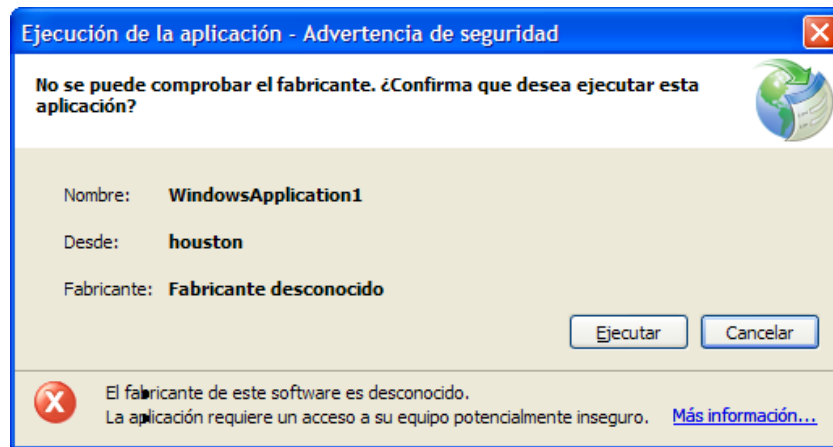
Si presionamos sobre el botón **Instalar** del navegador Web, observaremos que el Servidor Web realiza diferentes acciones de verificación. La primera acción es una acción de conexión como la que se muestra en la figura 8.



La primera acción que se realiza es una acción de conexión con el Servidor Web

Figura 8

Posteriormente, puede aparecer una ventana de seguridad como la que se indica en la figura 9, siempre y cuando no hayamos realizado un proceso de generación segura o confiable de la aplicación, como ha sido el caso.

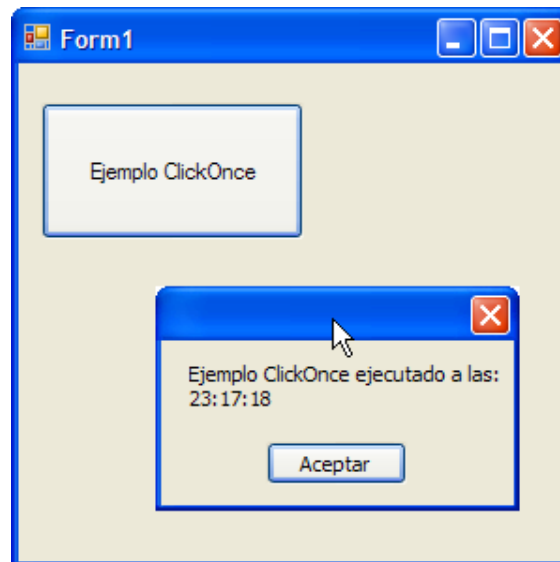


La aplicación Windows ejecutada a través del Servidor Web, debe ser confiable y segura

Figura 9

En nuestro caso, como sabemos que no hay problema en ejecutar la aplicación, presionaremos el botón **Instalar**.

Nuestra aplicación en ejecución es la que se muestra en la figura 10.



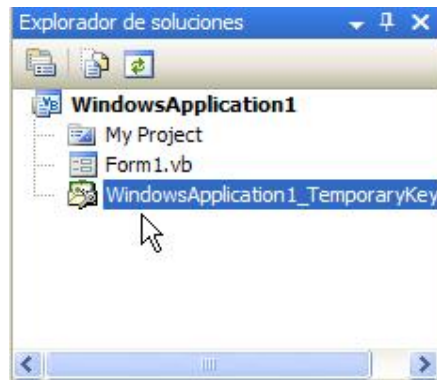
Aplicación Windows en ejecución

Figura 10

Cuando hacemos esto, siempre que ejecutemos la aplicación, el sistema detectará que aceptamos una vez su seguridad, por lo que siempre se ejecutará sin indicarnos ningún mensaje de seguridad.

Ahora bien, supongamos que decidimos modificar parte del código de nuestra aplicación y que por supuesto, cambiamos la versión de la misma.

Acuda antes a la ventana del *Explorador de soluciones* y observe que se ha añadido en la ventana un fichero de nombre *WindowsApplication1_TemporaryKey.pfx* que corresponde a una llave o clave temporal relacionada con el proyecto publicado. Esto es lo que se muestra en la figura 11.



Ventana del Explorador de soluciones con el fichero *WindowsApplication1_TemporaryKey.pfx* añadido a él

Figura 11

Vamos a actualizar nuestra aplicación y la vamos a cambiar la versión, por lo que ahora escribiremos el siguiente código:

```
Código

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

        MessageBox.Show("Ejemplo ClickOnce ejecutado a las:" & vbCrLf & _

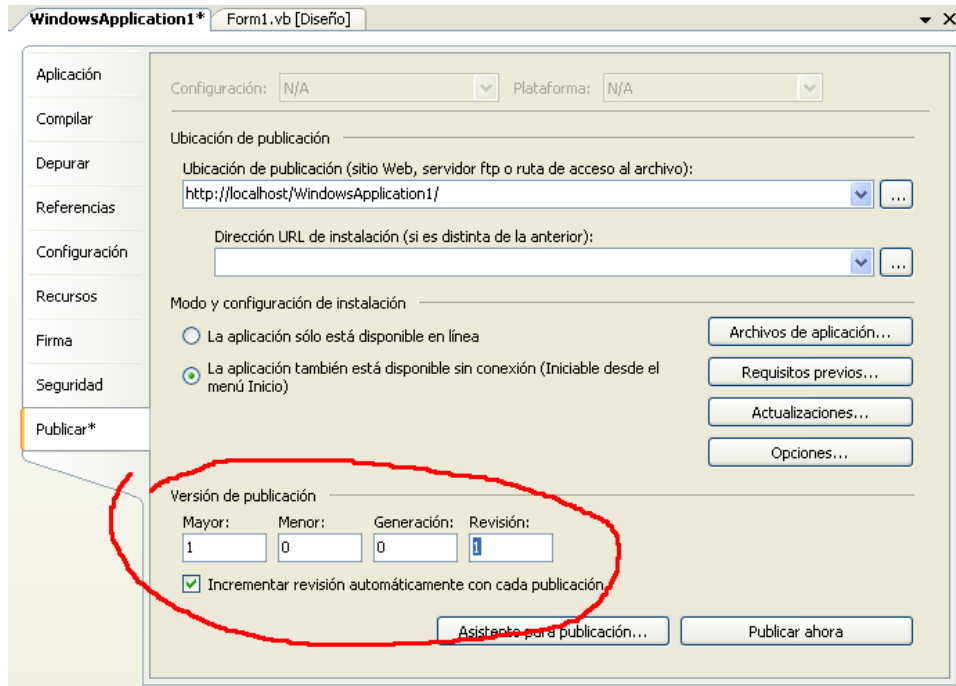
            Date.Now.ToShortDateString & vbCrLf & _

            Date.Now.ToLongTimeString)

    End Sub

End Class
```

La versión de la aplicación, no tiene relación con la versión de publicación, por ejemplo, la versión de la aplicación la he modificado a *1.1.0.0*, pero el instalador autogenera su propio número de versión, el cual podemos cambiar en las propiedades del proyecto, ficha **Publish**. En la figura 12 podemos ver la ficha **Publish** de las propiedades del proyecto.

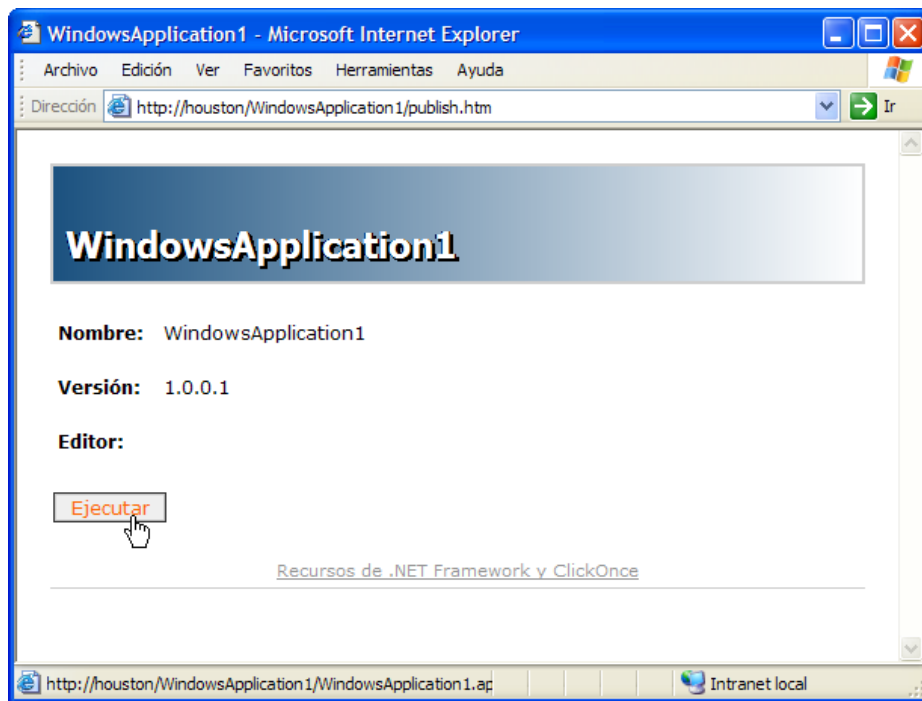


Ventana Web para ejecutar la aplicación

Figura 12

El siguiente paso que he hecho es compilar la aplicación y publicarla nuevamente.

Una vez hecho esto, acudimos a la página Web de la aplicación y presionamos nuevamente el botón **Instalar** como se indica en la figura 13.



Ventana Web para ejecutar la aplicación

Figura 13

La ejecución de la aplicación se realizará sin problemas de manera sencilla y controlada.

Como vemos, la publicación de aplicaciones Windows a través de un Servidor Web, lo que se denomina tecnología de publicación *ClickOnce*, es un proceso de publicación rápido y sencillo, que aporta grandes ventajas y que en otras versiones de .NET trae consigo algunos inconvenientes superados en esta versión de .NET.

▶ [Ver vídeo de esta lección](#) - video en Visual Studio 2005 válido para Visual Studio 2008