

Introducción a la plataforma .NET

En este capítulo veremos los conceptos básicos de .NET y sus componentes.

Analizaremos la importancia del CLR y las bibliotecas base dentro de la plataforma, y veremos cómo organizar el código a través de los Namespaces.

Aprenderemos cómo navegar por IDE y a configurar sus principales características.

.Net Framework	14
CLR	14
Base Class Library	15
MSIL y lenguajes de programación	15
Namespaces	16
Distribución	17
Visual Studio	19
El IDE	21
Principales menús del IDE	26
Configuración del entorno	26
Usando Visual Studio	28
Soluciones y proyectos	28
Resumen	31
Actividades	32

.NET FRAMEWORK

La plataforma .Net es la propuesta de Microsoft para el desarrollo de aplicaciones completamente orientadas a objetos, seguras, sencillas de instalar y multiplataforma; no es un lenguaje de programación: es un conjunto de tecnologías de software que permite el desarrollo de aplicaciones de escritorio (**Windows**), Web (**ASP.Net**), bibliotecas de código (componentes reutilizables: **DLL**), dispositivos móviles, controles de usuario, acceso a datos, reportes y mucho más, que se ejecutan bajo el **.Net Framework**. Es completamente extensible: podemos **heredar** de sus clases base y modificar su comportamiento o **implementar** sus interfaces para desarrollar clases completamente nuevas.

El entorno de programación de la plataforma .Net, **Visual Studio**, es uno de los más potentes en cuanto a facilidad de uso y proporciona una gran cantidad de herramientas para mejorar la productividad del programador. Es independiente del lenguaje de programación, por lo que podemos realizar componentes en un lenguaje y utilizarlo en otro, sin complicaciones ni pérdida de rendimiento, debido a su sistema de tipos comunes y a su arquitectura de lenguaje intermedio (**MSIL**).

Si bien el .NET es bastante amplio, básicamente podemos dividirlo en las siguientes partes que lo componen:

CLR

El **CLR** o *Common Language Runtime* es el entorno que administra la ejecución de código y que proporciona los servicios necesarios para el desarrollo de las aplicaciones. El CLR proporciona todos los elementos comunes a los lenguajes de programación de la plataforma .NET. Dichos lenguajes exponen estas funcionalidades a través de su propia sintaxis. No hay conflicto en utilizar componentes realizados en diferentes lenguajes ya que utilizan una base común independiente de éstos. El CLR expone lo que se denomina el *Common Type System* (*Sistema de Tipos Comunes* o **CTS**); es decir, que proporciona la especificación de los tipos de datos para utilizar en **todos** los lenguajes de programación y por lo tanto son comunes a todos: un tipo de dato **string** es el mismo en Visual Basic que en C#. Esta característica elimina la incompatibilidad existente entre los lenguajes de programación, ya que –en este caso– es el Framework el que proporciona las especificaciones, y no los lenguajes en sí mismos.

Todo el código generado sobre la base de CLR es lo que se denomina **Managed Code** (*Código Manejado o Administrado*). La creación de Código Manejado permite al CLR proporcionar ciertas funcionalidades útiles a todos los lenguajes, y es lo que hace que .NET sea tan poderoso:

- **Administración de memoria inteligente:** a través del **Garbage Collector** se libera al programador de la tarea de liberar memoria.
- **Aislamiento de aplicaciones:** si una aplicación deja de funcionar, no afecta a otra en ejecución ni al sistema operativo.
- **Seguridad de ejecución de los componentes:** basa la ejecución de éstos sobre la base de su metadata, que le indica cómo debe ejecutarse el componente, qué versión utilizar y bajo qué contexto de seguridad, entre otras cosas.
- **Generación de código nativo:** la ejecución de los componentes se realiza con un compilador Just In Time (**JIT**), que traduce los componentes al código nativo según la CPU en la que se encuentre y los ejecuta en su entorno.

Base Class Library

.Net proporciona de base un conjunto de clases que incluyen la mayoría de las funcionalidades que los programadores realizan de manera cotidiana y, además, un conjunto de clases específicas, de acuerdo con la tecnología que se va a utilizar en el desarrollo de las aplicaciones. Entre estas clases podemos mencionar la implementación de todos los tipos de datos (si bien el CLR da la especificación, éstos están implementados en las bibliotecas base), clases para la administración de colecciones y de estructuras de datos, clases para manejo del sistema de archivos y de entrada/salida (I/O). También proporciona las definiciones y clases correspondientes a la creación, al manejo y al uso de XML, acceso a datos (ADO.NET), aplicaciones de escritorio (Windows Forms), aplicaciones Web (ASP.NET), etc.

MSIL y lenguajes de programación

Los lenguajes de programación de .Net hacen uso de las especificaciones del CLR y de las bibliotecas base para el desarrollo de las aplicaciones. Cuando éstas se compilan, generan lo que se denomina un **Assembly** (o ensamblado). Este Assembly puede ser un ejecutable (**EXE**) o un componente (**DLL**). Los ensamblados se autodescriben y proporcionan información para su ejecución en el entorno del CLR (especificaciones de seguridad, versión, componentes referen-



CÓDIGO MSIL

Todos los lenguajes .NET generan código MSIL, por lo tanto, no existen problemas de interoperabilidad entre los componentes generados por cada uno de ellos. El que se encarga de la ejecución de los componentes siempre es el CLR, y no el compilador específico de cada lenguaje.

ciados, recursos utilizados, etc.). Estos Assemblies generados por los compiladores de los lenguajes de programación no generan código de máquina, sino que lo hacen en un código intermedio, llamado MSIL. Esto permite la portabilidad del código .Net a cualquier plataforma y asegurar un óptimo rendimiento según el entorno en el que se ejecuten ya que, como vimos, es el CLR el que se encarga de “traducir” el Assembly al código nativo y proporcionar el entorno de ejecución de éstos.

Cuando trabajemos en .Net, el resultado de nuestro trabajo siempre serán componentes en MSIL.

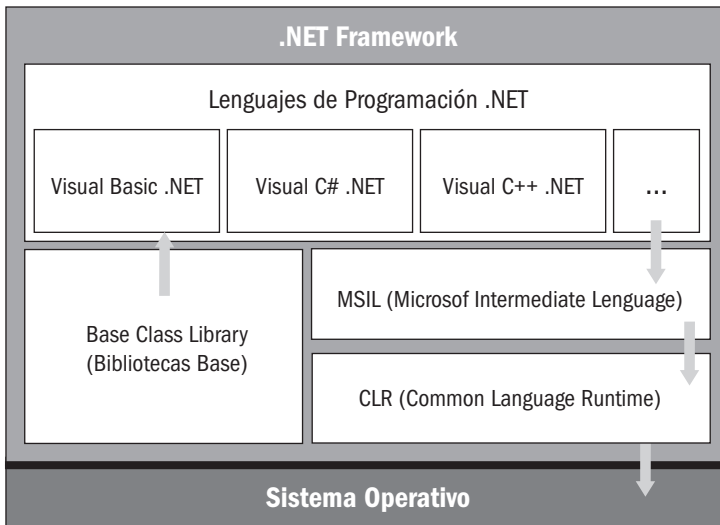


Figura 1. Esquema general del funcionamiento de los componentes en .Net: los lenguajes de programación hacen uso de las bibliotecas base y generan Assemblies en MSIL. Cuando se ejecutan, lo hacen a través del CLR, que asegura su aislamiento del resto del sistema y los traduce mediante el JIT a código nativo para su ejecución en el sistema operativo.

Namespaces

Como vimos, .Net nos proporciona, en las bibliotecas base, muchas clases preconstruidas. Dada la gran cantidad de éstas, es inevitable tener varias con el mismo nombre. Los Namespaces son la clave en la organización de las bibliotecas base dentro de la plataforma y es la manera en que .NET utiliza para navegarlas e identificarlas.

Los **Namespaces** o Espacios de Nombre son calificadores de clases. Sirven para evitar ambigüedades y proporcionan una manera inteligente de organizar nuestro código dentro de la plataforma .NET. Supongamos que tenemos una clase Cliente que representa a un cliente en nuestra capa de negocios (hablaremos sobre esto más adelante) y una clase Cliente en nuestra capa de acceso a datos. Cuando creamos un objeto del tipo Cliente, .Net no puede determinar si queremos un Cliente de nego-

cios o un Cliente de acceso a datos. Para evitar esta ambigüedad, tenemos que **calificar** nuestra clase, indicándole cuál debemos utilizar:

Negocios.Cliente

o

AccesoADatos.Cliente

De esta manera aseguramos la correcta creación del objeto esperado y tenemos un agrupamiento lógico. Todas las clases de la biblioteca base están organizadas de esta manera. Por ejemplo, toda la definición de los tipos de datos y de clases base de la mayoría de los objetos en .Net están bajo el Namespace **System**; los de utilidad para creación de aplicaciones Windows, bajo el de **System.Windows.Forms**; los de utilidad para la creación de aplicaciones Web, bajo el de **System.Web**, etc.

El conocimiento de los Namespaces de .Net es de fundamental importancia para aprovechar toda la funcionalidad que nos proporciona, y así no tener que escribir código que ya es provisto por .Net.

Debido a que es un calificador lógico, se aconseja también ponerle el nombre de nuestro Namespace base o raíz a nuestros Assemblies para tener visualmente una idea rápida de lo que contiene. Así, por ejemplo, como dijimos, el **namespace System** tiene su contraparte en System.dll; System.Windows.Forms la tiene en su System.Windows.Forms.dll; System.Web en System.Web.dll; etc.

Distribución

Debido a que toda la funcionalidad principal está dada por el .NET Framework, es requisito para que funcionen nuestras aplicaciones que esté instalado en la PC del usuario. El .NET Framework es gratuito y puede descargarse del sitio Web de Microsoft. Se instala una vez, y cualquier aplicación realizada en .NET puede hacer uso de él. Esto es una ventaja ya que, de estar instalado el Framework, sólo tenemos que distribuir únicamente nuestros componentes (assemblies) y ejecutables para que funcione nuestra aplicación.

Hasta el momento se han liberado 3 versiones del .NET Framework:

- **.NET 1.0:** liberado en el 2002, fue la primera versión de .Net como release final. Antes de ese año ya se estaba trabajando con las versiones beta de éste. Fue la base para las sucesivas versiones,
- **.NET 1.1:** liberado en el 2003, presentó significativas mejoras en lo que respecta a seguridad dentro de la plataforma. Solucionó diferentes errores encontrados en la versión 1.1 y mejoró sustancialmente debido al feedback obtenido durante el uso de la primera versión. Además presentó una nueva generación de proyectos orientados a la creación de aplicaciones para dispositivos móviles.

- **.NET 2.0:** liberado en el 2005, representa un gran salto cualitativo en lo referente a productividad. Se agregaron miles de clases nuevas para mejorar la productividad de los desarrolladores y se unificaron pequeñas diferencias en lo referente a las implementaciones en los distintos lenguajes, entre otras cosas. Es el Framework más completo hasta el momento y es el que usaremos para el desarrollo de lo ejemplos de este libro.

Cuando se instala .Net Framework, lo hace en la carpeta:

C:\Windows\Microsoft.NET\Framework

Siendo **C:** la unidad del sistema y **Windows** el directorio de instalación de Windows (en caso de Windows 2000 éste es WinNT).

Dentro hay un subdirectorio por cada versión del Framework instalado:

- v1.0.3705: .Net Framework 1.0
- v1.1.4322: .Net Framework 1.1
- v2.0.50727: .Net Framework 2.0

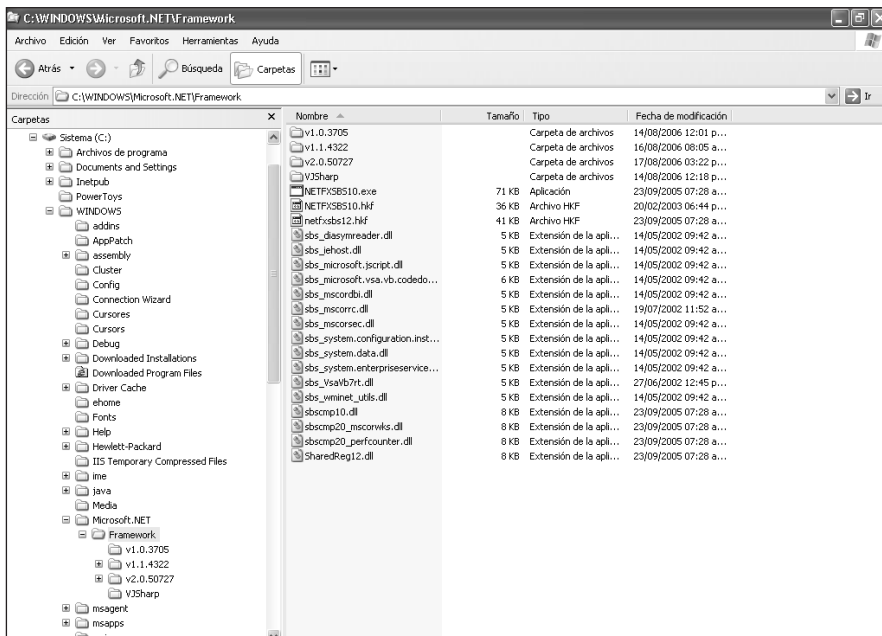


Figura 2. El .Net Framework instalado. Dentro de la carpeta Framework, se visualizan sus versiones instaladas.

Todos los .NET Framework pueden coexistir ya que cada uno tiene su propio CLR y bibliotecas base para funcionar. Esto significa que podemos tener instalados, en

sión. Se echaban de menos la facilidad de uso y las ventajas que proporcionaba el *viejo* Visual Basic.

Con el advenimiento del .NET 2.0 Microsoft intentó solucionar este inconveniente, proporcionando diferentes “sabores” de Visual Studio, adecuado a las necesidades y a los conocimientos de cada sector:

- **Visual Studio 2005 Express Edition:** esta versión es para los que recién se inician en .Net. Es gratuita y orientada a lenguajes específicos (cada lenguaje de .NET tiene su versión Express). En el caso de Visual Basic, el nombre del producto es Visual Basic 2005 Express Edition. Presenta un IDE simplificado y proporciona lo necesario para quienes recién comienzan con .NET con Visual Basic. Dado que son versiones limitadas, sólo permiten desarrollar aplicaciones de escritorio. Si se desea programar aplicaciones Web, debe instalarse por separado el Visual Web Developer Express Edition.
- **Visual Studio 2005 Standard Edition:** esta versión presenta un IDE mejorado con respecto a las versiones Express y permite el desarrollo en conjunto de aplicaciones Web y Windows en un mismo entorno. Además, incorpora toda la documentación del producto y permite la integración con herramientas de control de código fuente. También se agregan los proyectos de desarrollo de aplicaciones para dispositivos móviles.
- **Visual Studio 2005 Professional Edition:** es la versión adecuada para quien desarrolla todo el tiempo. Presenta muchas mejoras en el IDE, proporciona además soporte de depuración remoto, la posibilidad de generar aplicaciones de 64 bits, acceso completo a los servicios del sistema y a las bases de datos. Proporciona Crystal Reports para la generación de reportes y de proyectos de instalación más complejos basados en Windows Installer.
- **Visual Studio 2005 Tools for Office:** es similar a la Professional Edition, pero incluyen como base un conjunto de herramientas para trabajar con Microsoft Office.
- **Visual Studio 2005 Team System:** esta versión está dirigida para trabajar en conjunto con los diferentes equipos de desarrollo en una empresa. Así hay una sub versión para arquitectos de software, una para desarrolladores y una para quienes realizan el testing del software. También hay una que combina todas estas últimas en una gran suite. Presenta herramientas para el control de código fuente y para la administración de proyectos de software. Además presenta un entorno de colaboración para todos los miembros del equipo de desarrollo.

El IDE

El IDE del Visual Basic presenta muchas ventanas. Aun cuando es una versión Express, hay bastantes. Cada ventana tiene una funcionalidad específica y es importante conocer cada una de ellas para sacarle el máximo provecho al entorno.

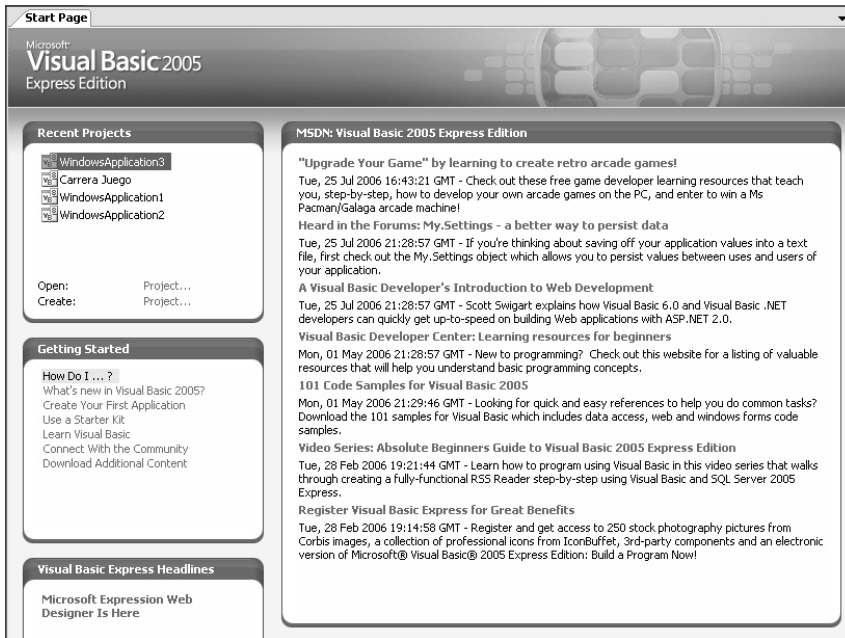


Figura 4. Vista de Visual Basic Express Edition con la configuración por defecto.

El IDE, básicamente, está compuesto por 5 secciones:

- **La sección superior:** aquí se encuentran las Toolbars o barras de herramientas que nos dan acceso a los comandos más comunes. Según la tarea que estemos realizando, esta sección se va a ir poblando de diferentes Toolbars y habilitará o no los comandos según sea necesario.
- La sección inferior: por lo general, están las ventanas de errores (**Error List**), la de tareas (**Tasks**) y la de salida (**Output**).
- **La sección izquierda:** presenta la **ToolBox** o caja de herramientas. Contiene todos los controles y componentes que podemos utilizar cuando realizamos las aplicaciones. Según sea el tipo de aplicación (Windows o Web), se llenará con los controles y componentes apropiados para cada desarrollo. También se encuentra el DataBase Explorer o explorador de bases de datos, que nos da acceso a los orígenes de datos para utilizar en nuestra aplicación.



VERSIONES UTILIZADAS

En el desarrollo de los ejemplos y de las prácticas de este libro, utilizaremos las versiones Visual Studio 2005 Express Edition y Visual Web Developer Express Edition.

- **La sección derecha:** presenta el **Solution Explorer** (Explorador de Soluciones) y la **Properties Window** (Ventana de Propiedades). Ambas son importantes, ya que el Solution Explorer presenta todos los proyectos y archivos con que estamos trabajando; y la Properties Window, las propiedades de todos los elementos del entorno. Cada objeto del proyecto es configurable a través de esta ventana y, también, las características de los controles y formularios, entre otras cosas.
- **La sección central:** es el área de trabajo. Aquí aparecen todos los documentos con los que estemos trabajando, tanto en **vista de código** como en **vista de diseño**. El IDE es un entorno multi-documento y se accede a cada uno ellos a través de las solapas con su nombre.

Ésta es la configuración normal. El IDE es completamente personalizable y puede ser adecuado al uso y a las preferencias de cada programador.

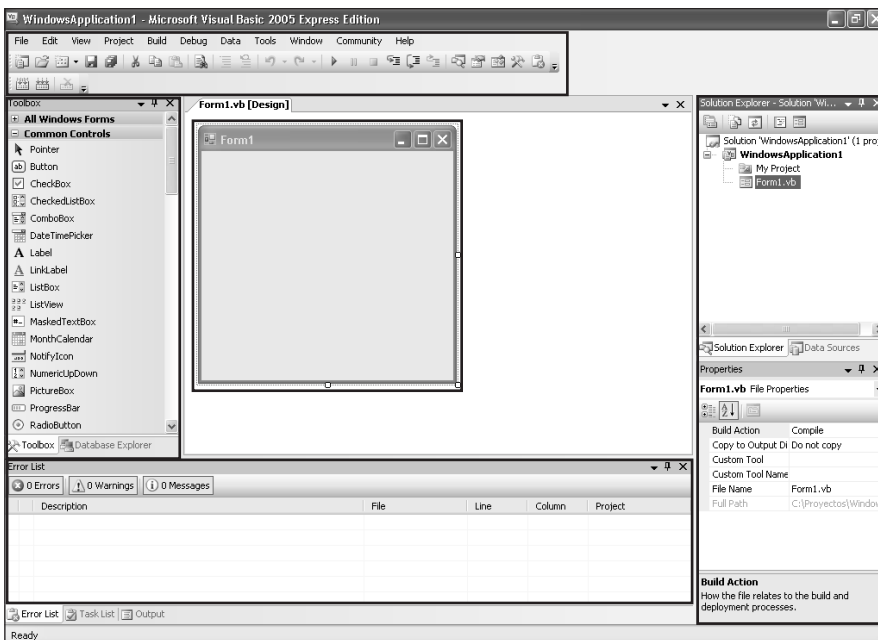


Figura 5. Vista del IDE de Visual Basic Express Edition con las ventanas habilitadas para un proyecto de aplicación Windows.

Como en el IDE no hay mucho espacio, éste presenta algunas características para ahorrar lugar, sobre todo cuando tenemos un monitor chico y resoluciones bajas. Una de ellas es el **AutoDock**, que permite mover las ventanas por el entorno y, cuando las ponemos cerca de un borde, se muestra un área donde es posible reubicarla. Otra característica es el **AutoHide**, que permite a las ventanas ocultarse en las secciones donde están ubicadas y aparecer cuando el mouse se posiciona sobre ellas. Si cerramos o perdemos alguna, basta con ir al menú **View**, y la podremos activar nuevamente.

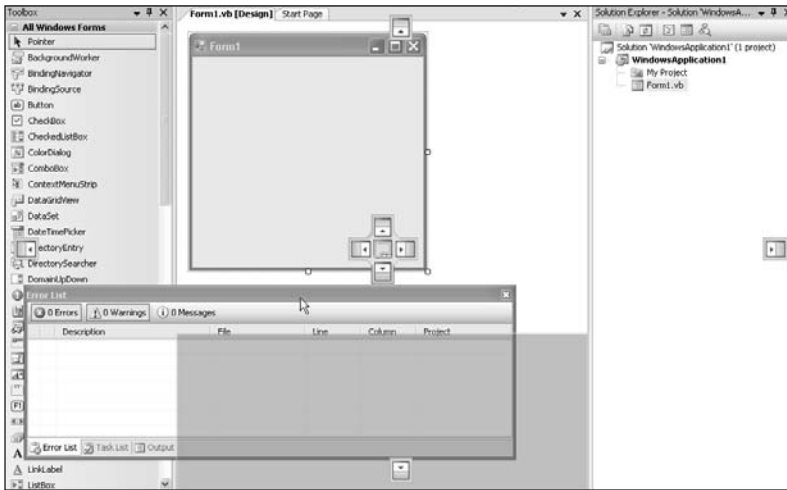


Figura 6. AutoDock en funcionamiento. A medida que movemos la ventana, aparecen en los bordes del IDE las áreas donde es posible reubicarla.

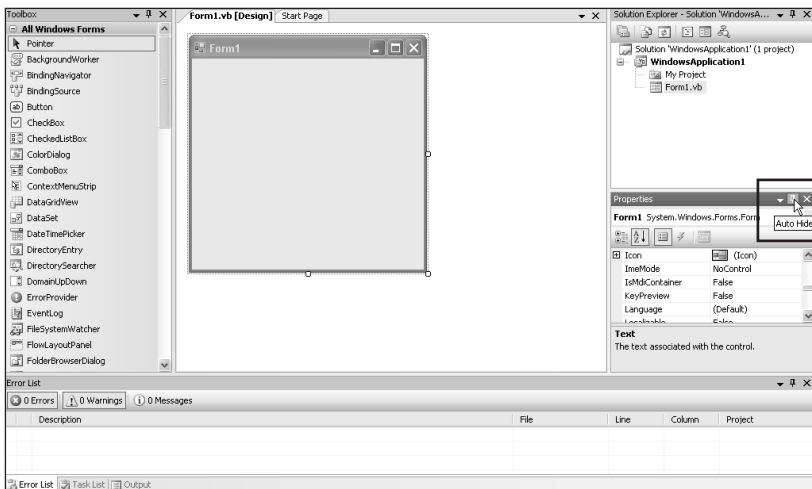


Figura 7. AutoHide en funcionamiento. Cuando ocultamos la ventana con el Pin, ésta presenta una solapa que indica dónde se encuentra.

III CONFIGURACIÓN DE VENTANAS

Es posible volver a la disposición original del entorno desde el Menú **Windows** y seleccionar la opción **Reset Windows Layout**. Esto es en especial útil, si modificamos constantemente de lugar las barras de herramientas para trabajar de manera más cómoda.

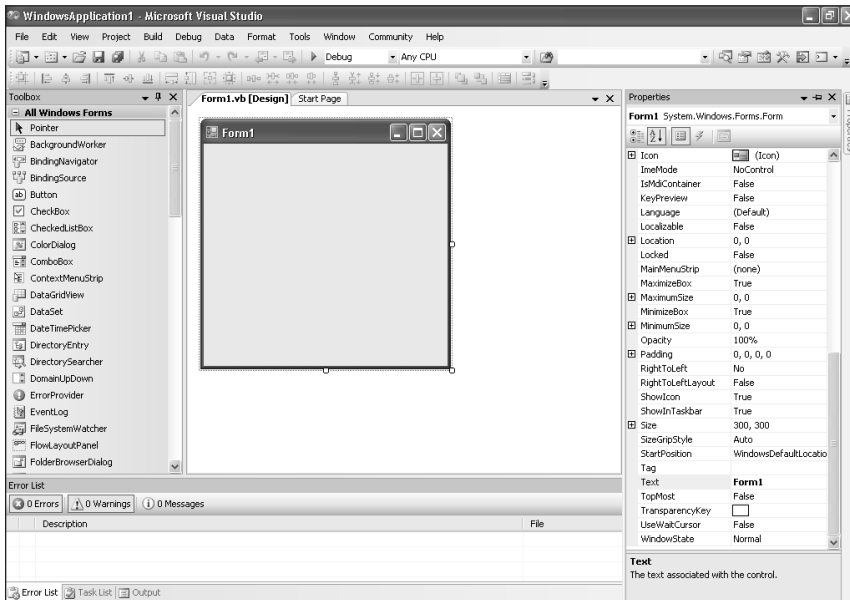


Figura 8. AutoHide en funcionamiento. Cuando movemos el mouse por encima de la solapa de la ventana, ésta aparece nuevamente.

Principales menús del IDE

El IDE presenta un conjunto de **menús básicos**. Muchos de ellos siguen el estándar de cualquier aplicación Windows. Así podemos encontrar el menú **File**, **Edit**, **Windows**, etc., con opciones y funcionamientos similares a los de cualquier programa. El resto está orientado a los diferentes proyectos y a tareas específicas con los distintos menús. A continuación se escriben los principales menús y sus funcionalidades:

MENÚ	DESCRIPCIÓN
File	proporciona las opciones para la apertura y grabación de proyectos y soluciones
Edit	proporciona los comandos para la edición de texto, tales como Cut, Copy y Paste
View	permite el acceso a las ventanas del IDE
Project	permite agregar diferentes ítems al proyecto
Build	permite la compilación del proyecto
Debug	permite el seguimiento del programa para la determinación de errores en éste.
Data	configura las conexiones a los orígenes de datos
Tools	contiene un conjunto de opciones para la configuración del entorno y de los proyectos
Windows	permite la selección y la organización de las ventanas dentro del IDE
Community	permite el acceso de los recursos en línea sobre Visual Basic
Help	accede a la ayuda integrada del sistema

Tabla 1. Principales menús del entorno. A través de ellos podemos acceder a las diferentes funcionalidades del IDE, para el desarrollo de nuestras aplicaciones.

Configuración del entorno

Como dijimos, el entorno es completamente configurable a nuestro gusto. La opciones generales aplicables a las operaciones que realicemos dentro del IDE y por cada proyecto en particular lo podemos configurar del menú **Tools/Options**.

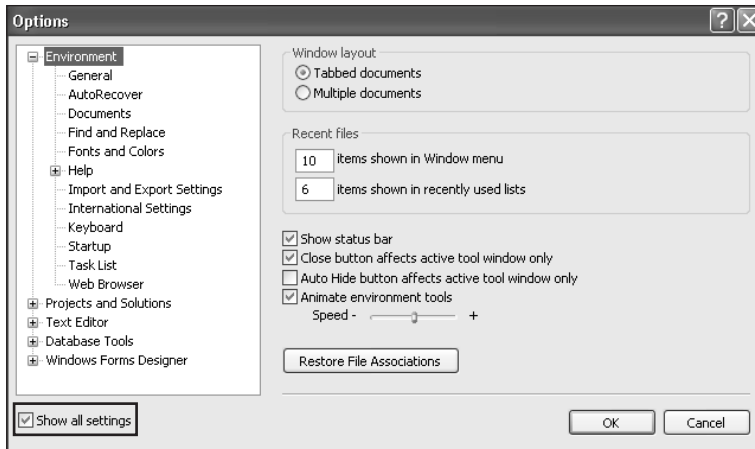


Figura 9. Ventana de Opciones. Desde esta ventana podemos configurar como debe comportarse el entorno para trabajar más cómodos. Para poder configurar todas las opciones, debemos marcar el check **Show All Settings**.

Antes de comenzar a trabajar, es conveniente configurar las opciones presentes en el menú **Projects and Solutions**. Aquí le indicaremos al Visual Studio dónde guardar por defecto los proyectos y soluciones que creemos y que debe mostrar cuando trabajemos con ellos. Todas estas opciones no vienen habilitadas por defecto, por lo tanto, es conveniente revisarlas antes de comenzar.

Las principales opciones para configurar son:

- **Visual Studio projects locations:** aquí le indicamos al Visual Studio dónde debe guardar por defecto los proyectos y soluciones.
- **Always show Error List if build finishes with errors:** muestra la ventana **Error List**, si la compilación termina con errores.

EXTENSIONES DE ARCHIVOS

Las soluciones se reconocen porque llevan la extensión **.sln** y los proyectos, la extensión **.vbproj**. Esta nomenclatura se viene utilizando desde la versión 2002 del entorno.

- **Track Active Item in Solution Explorer:** sincroniza automáticamente la selección del documento activo con el ítem que lo representa en el **Solution Explorer**.
- **Show advanced build configurations:** nos permite seleccionar las opciones para compilar en modo **Debug** o **Release**.
- **Always show solution:** muestra como elemento raíz, la solución en el **Solution Explorer**. Es conveniente habilitarla para acceder a ciertas características globales a la solución.
- **Save new projects when created:** graba los proyectos apenas es creado y arma la estructura de directorios correspondientes. Es útil para acostumbrarnos a trabajar ordenadamente y a tener en consideración los nombres que les asignamos a los proyectos.
- **Warn user when the Project location is not trusted:** avisa al usuario si el lugar desde donde está localizado el proyecto no es confiable.
- **Show Output window when build starts:** muestra la ventana de **Output**, informando el progreso de la compilación y lo que va sucediendo, mientras ejecuta esta tarea.

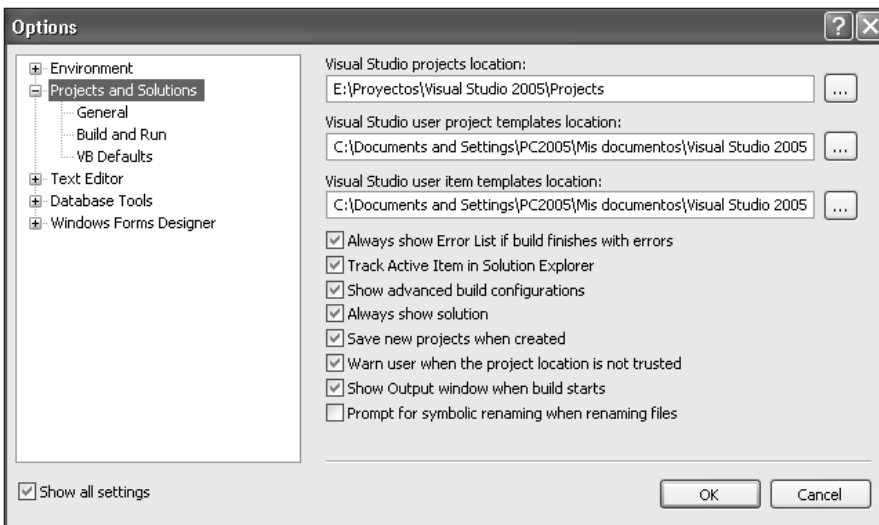


Figura 10. Ventana de **Opciones** con las opciones del menú **Projects and Solutions** configuradas.

USANDO VISUAL STUDIO

El Visual Studio está diseñado para darle al desarrollador la mayor flexibilidad posible para trabajar. No obstante, para sacarle el máximo provecho, debemos comprender primero cómo se organiza nuestro código dentro del entorno, es decir, comprender las diferencias entre **Soluciones** y **Proyectos**.

Soluciones y proyectos

Dentro de Visual Studio, es posible organizar nuestro trabajo en Soluciones y Proyectos. Una Solución es un agrupador de proyectos que representa nuestro sistema en su totalidad. Cuando realizamos nuestros sistemas, éstos están divididos en componentes de código, interfaces de usuario (controles de usuario, ejecutables, páginas Web), componentes de configuración, componentes de acceso a datos, etc. Cada uno de ellos tiene una funcionalidad específica y el conjunto forma nuestro sistema. Dentro del Visual Studio, podemos trabajar con cada uno de ellos, sin necesidad de abrir un entorno diferente. Eso es lo que hace la solución: los agrupa y nos permite trabajar con todos, como si fueran una unidad.

Por otro lado, cada proyecto representa una parte del sistema, es decir, un componente de éste. Así, si necesitamos crear la interfaz de usuario Windows, crearemos un proyecto **Windows Application**. Si necesitamos un componente de código, usaremos un proyecto **Class Library**. Por lo tanto, siempre trabajaremos con proyectos agrupados en soluciones. Si elegimos crear un proyecto sin especificar la solución a la que pertenece, el Visual Studio crea una solución por defecto con el nombre del proyecto.

Para crear un proyecto o una solución, vamos al menú **File** y seleccionamos **New Project**. Esta opción nos permite seleccionar el tipo de proyecto y su ubicación.

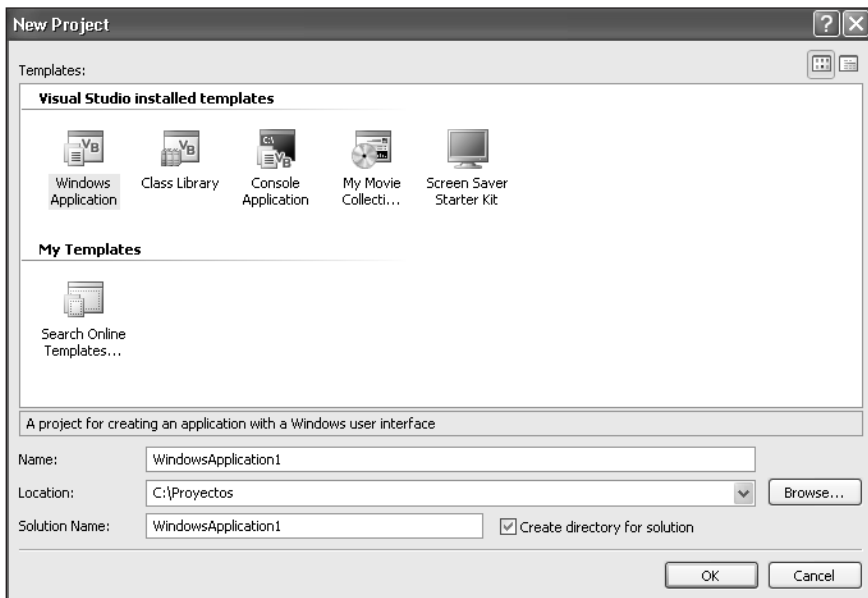


Figura 11. Ventana de Selección de Proyectos. Desde aquí seleccionamos los proyectos para crear dentro de la solución.

Los proyectos disponibles dentro de la versión **Express** son:

- **Windows Application:** crea aplicaciones basadas en formularios para Windows.
- **Class Library:** crea componentes de código. Permiten encapsular clases y rutinas para hacer nuestro código reutilizable.
- **Console Application:** crea aplicaciones de consola o ventana de comandos.

También contiene ejemplos de aplicaciones funcionales, como **My Movie Collection** y **Screen Saver Starter Kit**, que muestran lo que se puede realizar con estas versiones. Dentro de esta ventana, una vez seleccionado el proyecto, debemos darle el nombre. El campo **Name** sirve para ello. Es muy importante este nombre ya que con él se va a crear una estructura de directorios y a su vez será el **Root Namespace** de la aplicación. **Location** indica dónde ubicaremos el proyecto y **Solution Name**, el nombre que tendrá la solución.

Una cosa importante para tener en cuenta es que el resultado de la compilación de los proyectos es un **Assembly**. Por lo tanto, si es un EXE o una DLL, éste será determinado por el tipo de proyecto elegido.

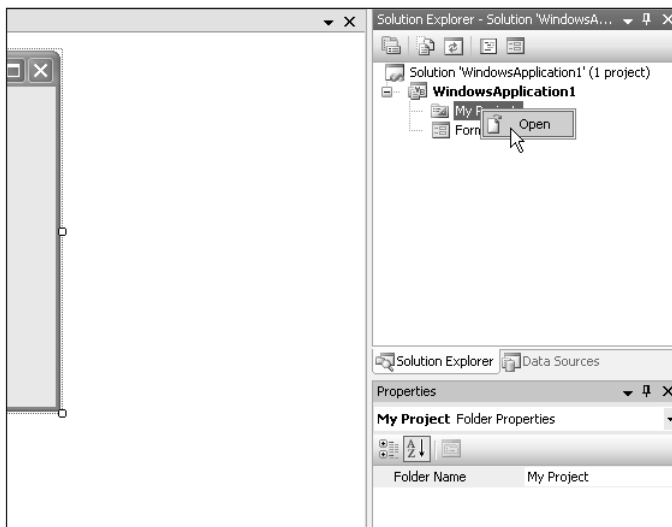


Figura 12. Apertura de la ventana de *Propiedades del proyecto*. Se debe seleccionar la opción *Open* en el menú contextual de la carpeta *My Project*.

SOLUCIONES Y DIRECTORIOS

Es conveniente crear una solución en blanco. Esto permite organizar mejor nuestra estructura de directorios y agregarle los proyectos necesarios. Como cada proyecto crea su propio directorio, éstos quedarán bajo el de la solución. De esta manera podemos mover o copiar todo el conjunto de proyectos con solo mover el directorio de la solución.

Una vez seleccionado el proyecto con el que queremos trabajar, antes de empezar a escribir código, una buena práctica es establecer ciertos parámetros del proyecto para que las clases y todo lo que escribamos tengan un orden coherente. Algunas de las cosas que modificaremos serán el nombre del **Assembly** que generará el proyecto, el **Root Namespace** bajo el que se organizará el código y la versión del **Assembly**, entre otras cosas. Estas propiedades y algunas otras se encuentran en la ventana de Propiedades del proyecto.

Para acceder a ella, haremos doble clic en la carpeta **My Project** en el **Solution Explorer** o clic con el botón derecho del mouse en la misma carpeta y, seleccionaremos la opción **Open** del menú contextual.

Una vez abierta, la ventana de Propiedades aparece en el área de trabajo. Esta ventana tiene las solapas a la izquierda y presenta las diferentes opciones de configuración del proyecto. Ellas son:

- **Application:** Esta solapa permite modificar el nombre del assembly al compilarlo, el ícono de la aplicación, el **Root Namespace**, cómo debe iniciarse éste, la información del assembly y la habilitación de los eventos de la aplicación, entre otras cosas.
- **Compile:** esta solapa indica las opciones de compilación que tendrá en cuenta el compilador de Visual Basic para realizar su tarea.
- **Debug:** permite configurar elementos para utilizar durante la depuración del programa, como ser líneas de comando, directorios de inicio, etc.
- **References:** muestra las referencias o enlaces hacia otros componentes que están siendo utilizados o consumidos por la aplicación.
- **Resources:** esta solapa permite agregar recursos con información de texto o binaria embebida dentro de la aplicación.
- **Settings:** permite administrar los diferentes valores de configuración que va a utilizar el programa en tiempo de ejecución. Estos valores se almacenarán en el archivo de configuración del mismo.
- **Signing:** permite darle al **Assembly** un **Strong Name** (*nombre seguro*) para que pueda ser utilizado, entre otras cosas, en el **Global Assembly Cache (GAC)**, el cual es un repositorio común de **Assemblies**.



SOLUTION EXPLORER Y ARCHIVOS OCULTOS

El entorno del Visual Studio en esta versión 2005 oculta muchas de las opciones y archivos que, en versiones previas, estaban al alcance de la mano. Para los desarrolladores experimentados o para quienes deseen modificar directamente los archivos y otras opciones, pueden acceder a ellas, presionando el botón **Show All Files** en la toolbar del **Solution Explorer**.

- **Security:** permite configurar la seguridad que se va a utilizar, si se usa la característica de **ClickOnce** para la distribución de la aplicación.
- **Publish:** permite publicar la aplicación, utilizando la tecnología **ClickOnce**.

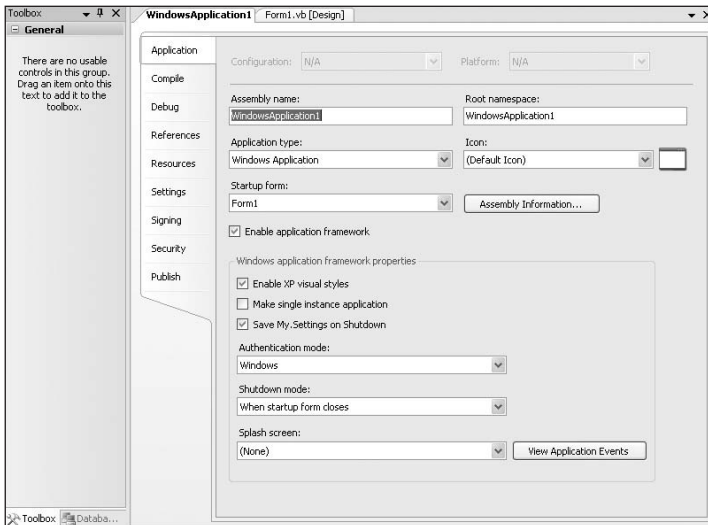


Figura 13. Ventana de propiedades del proyecto con la solapa **Application** seleccionada. Desde este lugar se configuran los elementos básicos del proyecto.

El campo **Assembly name** permite modificar el nombre del Assembly que tendrá cuando se compile; el campo **Root Namespace**, el Namespace raíz bajo el que se organizará nuestro código y el botón **Assembly Information**, que permite modificar la información de versión y lo relativo al desarrollo del Assembly.

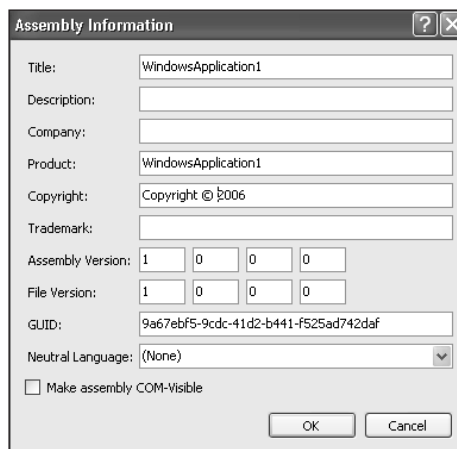


Figura 14. Ventana de edición de la información del **Assembly**. Presionando el botón **Assembly Information** se muestra esta ventana y nos permite configurar, entre otras cosas, la versión de nuestro componente.

El IDE de desarrollo de .NET es muy poderoso y presenta muchas facilidades al desarrollador. No obstante a veces puede resultar intimidante por la cantidad de opciones que presenta. La mejor manera de aprovecharlo es haciendo proyectos simples e ir acostumbrándonos al entorno de a poco. Si recién empezamos a trabajar con .NET, la opción más adecuada es utilizar la versión Express de Visual Studio y, en la medida que lo necesitemos, migrar luego a Visual Studio 2005 Standard Edition o Visual Studio 2005 Professional Edition.

RESUMEN

En este capítulo hemos aprendido qué es .NET y cómo está compuesto. Hemos visto la importancia del CLR y las bibliotecas base dentro del Framework para el desarrollo de las aplicaciones .NET y cómo organizar nuestro código con los Namespaces. También hemos aprendido a navegar por IDE y a configurar sus principales propiedades para poder trabajar lo más cómodamente posible y hemos echado un vistazo a las principales propiedades que vamos a configurar en los proyectos.



ACTIVIDADES

1 ¿Cómo está compuesto el .Net Framework?

2 ¿En qué directorio se instala el .NET Framework en la PC?

3 ¿Qué mejoras presenta el .NET 2.0 respecto de las versiones anteriores?

4 ¿Qué función cumple el CLR?

5 ¿Qué es el MSIL?

6 ¿Qué es un Namespace y para qué sirve?

7 ¿Qué es un Assembly?

8 ¿Qué es el IDE?

9 ¿Dónde se configuran las propiedades de un proyecto?

10 ¿En qué lugar se modifica la versión del Assembly?

11 ¿Dónde configuramos los directorios por defecto de los proyectos y de las soluciones que creamos?

12 ¿Qué tipos de proyectos tenemos la posibilidad de desarrollar con la versión Express de Visual Basic 2005?
