

El lenguaje C# y la plataforma .NET

En este primer capítulo estudiaremos qué nos ofrece este nuevo lenguaje y por qué habría de interesarnos.

Realizaremos una comparación con lenguajes similares y analizaremos cuál es la arquitectura de la plataforma sobre la que está construido.

¿Otro lenguaje nuevo?	16
.NET	19
El framework .NET	19
El lenguaje C#	20
Características fundamentales del lenguaje	21
C# contra Visual Basic 6.0	21
C# contra C++	22
C# contra Java	23
El entorno de desarrollo	
Visual Studio .NET	24
Nuestra primera aplicación con Visual Studio .NET	28
Resumen	33
Actividades	34

¿OTRO LENGUAJE NUEVO?

C# irrumpe en el mercado como un lenguaje muy bien diseñado y con muchas virtudes en una industria plagada de soluciones y herramientas de programación para todos los gustos. ¿Cuáles son, entonces, los motivos por los cuales deberíamos optar por C#?

- C# es un lenguaje moderno y orientado a objetos, con una sintaxis muy similar a la de C++ y Java. Combina la alta productividad de **Visual Basic** con el poder y la flexibilidad de C++.
- La misma aplicación que se ejecuta bajo **Windows** podría funcionar en un dispositivo móvil de tipo **PDA**. Con C#/NET no nos atamos a ninguna plataforma en particular.
- Se puede crear una gran variedad de aplicaciones en C#: aplicaciones de consola, aplicaciones para Windows con ventanas y controles, aplicaciones para la Web, etc.
- C# gestiona automáticamente la memoria, y de este modo evita los problemas de programación tan típicos en lenguajes como C o C++.
- Mediante la plataforma .NET desde la cual se ejecuta es posible interactuar con otros componentes realizados en otros lenguajes .NET de manera muy sencilla.
- También es posible interactuar con componentes no gestionados fuera de la plataforma .NET. Por ello, puede ser integrado con facilidad en sistemas ya creados.
- Desde C# podremos acceder a una librería de clases muy completa y muy bien diseñada, que nos permitirá disminuir en gran medida los tiempos de desarrollo.

Pero ¿dónde quedan los demás lenguajes? ¿Qué motivó a Microsoft a desarrollar la plataforma .NET?

Durante algún tiempo, cuando la programación en plataforma PC/Windows se popularizó, los caminos más comunes eran, principalmente:

- **Visual Basic:** un lenguaje fácil de aprender pero con muchos defectos. Gran parte de esas deficiencias es fruto de su afanoso objetivo por ser sencillo para el programador novato. Es un lenguaje orientado a objetos *light*. Posee algunas de las características más populares de la POO implementadas, pero muchas de ellas (las que realmente extrañaremos en proyectos complejos) permanecen ausentes, como la herencia, los métodos virtuales, la sobrecarga de operadores, etc. Claro que VB también posee muchas virtudes. Realizar una aplicación Windows nunca había sido tan fácil, y si ciertas tareas se encuentran fuera del alcance del lenguaje, es posible realizar un componente en, por ejemplo, C++ y utilizarlo desde VB sin inconvenientes.
- **Visual C++:** es ideal para crear componentes, librerías y drivers, pero la productividad desciende abruptamente cuando se trata de aplicaciones con formularios complejos. Es que, para esto, **Visual C++** se basa en **MFC** (*Microsoft Foundation*

Classes, un grupo de clases que encapsulan el API de Win32 y agregan algunas funcionalidades para facilitar la creación de aplicaciones). Diseñar formularios con MFC es una tarea poco grata; muchas de las propiedades de los controles de esta librería deberán ser fijadas en tiempo de ejecución, lo que aumentará la cantidad de código que deberá poseer nuestra aplicación.

- **Delphi y C++ Builder:** basados en el lenguaje de programación **Pascal** y **C++**, respectivamente, y ambos haciendo uso de una librería llamada **VCL** (*Visual Component Library*), permiten la creación de formularios complejos de una manera bastante sencilla, y poseen la capacidad de fijar casi la totalidad de las propiedades necesarias en tiempo de diseño y con base en dos de los lenguajes más exitosos de todos los tiempos.

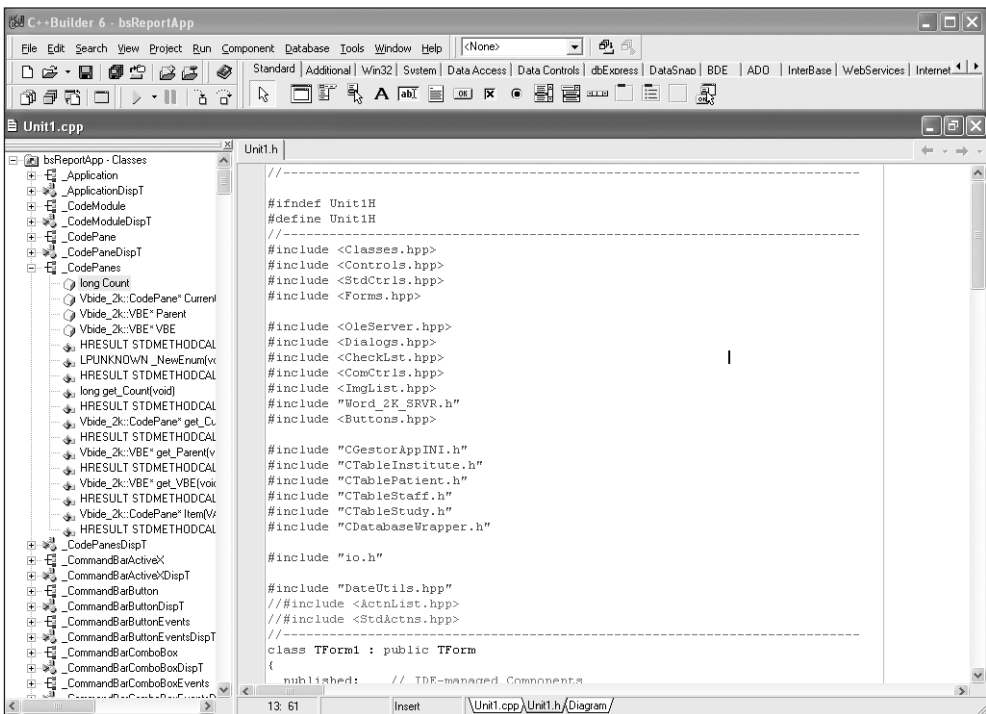


Figura 1. El entorno de programación Borland C++ Builder 6.0.



ONWEB.TECTIMES.COM

En el sitio onweb.tectimes.com encontrará contenido adicional vinculado a los temas tratados en este manual, incluyendo el código fuente de los ejemplos que se desarrollarán en este libro, listo bajar y probar, así como una completa guía de sitios web recomendados.

Pero no todo es color de rosa para estos entornos. Las aplicaciones son cada día más complejas y los programadores requieren, cada vez en mayor medida, un buen soporte a la colaboración entre componentes. Sería deseable poder emplear desde **Delphi** una librería C++ ya existente, de una manera sencilla, sin tener que estar utilizando herramientas de conversión.

En el caso de C++ Builder, hasta existen inconvenientes para acceder a librerías creadas en Visual C++ debido a que utilizan formatos de librerías distintos (**OMF** en el caso de C++ Builder y **COFF** en el caso de Visual C++), y esto es malo dado que Visual C++ es el entorno más popular (por lejos) para la creación de librerías para Windows, y muchísimos recursos que podemos encontrar en la Red fueron creados con este entorno.

Si deseamos, por ejemplo, crear una aplicación que utilice DirectX desde C++ Builder, tendremos que buscar en la Red las librerías **.LIB** correspondientes en formato OMF (existe una herramienta provista por el entorno para la conversión de formato COFF a OMF, pero la importación de librerías complejas, como lo es DirectX, no es para nada trivial).

Finalmente, en los últimos años ha surgido una gran variedad de plataformas móviles muy poderosas: PDAs, teléfonos celulares, Tablet PCs, etc. Estos dispositivos no son compatibles con binarios creados para microprocesadores Intel x86, ya que poseen su propia familia de microprocesadores con su propio paquete de herramientas para la construcción de aplicaciones nativas.

Ni Delphi ni C++ Builder se encuentran presentes en otras plataformas más allá de Windows (y Linux, si tomamos en consideración a Kylix, que es un entorno de desarrollo creado por Borland, muy similar a Delphi y a C++Builder). Visual Basic y Visual C++ se hallan en versiones especiales para algunos dispositivos móviles, pero el subconjunto de funciones del API de Win32 que están en dichos dispositivos hace que la migración de una aplicación de una plataforma se convierta en una tarea bastante engorrosa.



BORLAND KYLIX

Kylix es un entorno de programación para sistemas Linux muy similar a Delphi y a C++ Builder. Con él es posible crear aplicaciones con ventanas y controles (botones, cuadros de texto, cuadros de lista, etc.) y compilar en Windows y en Linux. Más info: www.borland.com/kylix.

.NET

.NET es un conjunto de tecnologías construidas a partir de una estrategia de Microsoft para la cual ha destinado gran parte de su presupuesto de investigación. Esta estrategia surge, a su vez, del nuevo mapa de necesidades y requerimientos que Microsoft advirtió que se suscitarían en el futuro.

.NET son aplicaciones de servidor (SQL Server 2000, Exchange 2000, etc.), es un entorno de desarrollo (Visual Studio .NET), son componentes clave que se integran al sistema operativo, son servicios, y es, finalmente, una plataforma de desarrollo denominada **framework .NET**.

El framework .NET

Es una infraestructura de desarrollo que está compuesta por diversos recursos, entre los cuales se destaca el más importante, que es una máquina virtual conocida como **CLR** (*Common Language Runtime*), sobre la cual se ejecutan las aplicaciones. De este modo, nuestros programas ya no poseerán código nativo de ningún microprocesador en particular, sino instrucciones **MSIL** (*Microsoft Intermediate Language*) que serán traducidas a código nativo en el momento de su ejecución (por medio de un compilador *Just In Time*).

El framework también define una librería base de clases, **BCL** (*Base Class Library*), a la cual puede acceder cualquier desde lenguaje desarrollado para la plataforma.

Por encima de la infraestructura se ubicará un conjunto de reglas básicas que debe implementar un lenguaje para poder ser parte de la familia .NET. Esta especificación es conocida como **CLS** (*Common Language Specification*).

Finalmente, se encuentra el conjunto de lenguajes que cumplan con la especificación CLS, como el C#, el VB.NET, Managed C++, etc.



.NET FRAMEWORK SDK

El desarrollo del framework .NET no se detiene. Actualmente la versión 2.0 se encuentra en etapa beta. Es posible acceder a más información acerca de esta nueva versión en el sitio oficial: <http://msdn.microsoft.com/netframework>.

Es posible crear recursos en cualquiera de estos lenguajes que hagan uso de recursos escritos en otros; de hecho, es posible crear una clase en C# que herede de una clase creada en Managed C++.

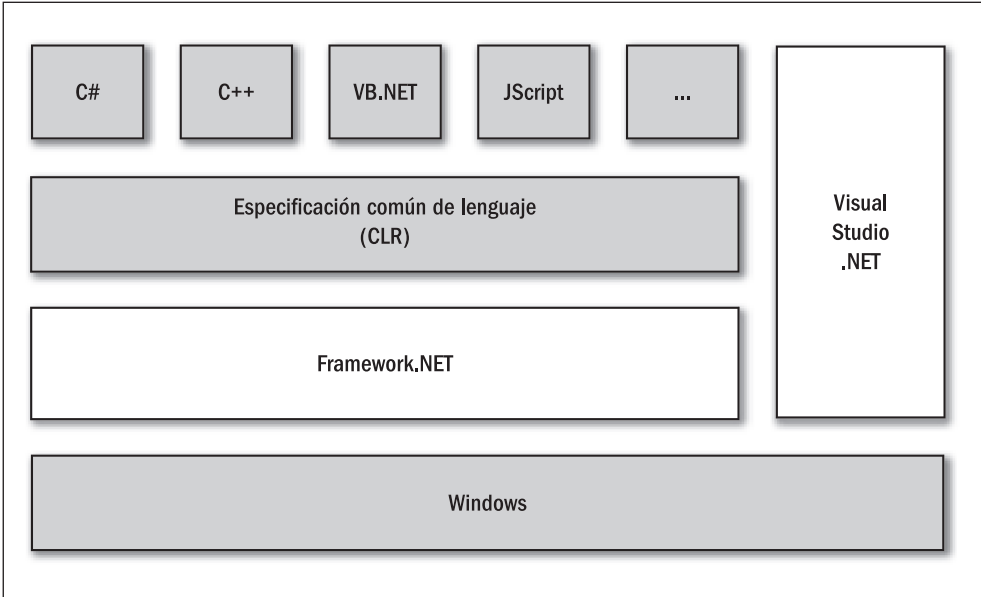


Figura 2. Arquitectura de la infraestructura de desarrollo.

EL LENGUAJE C#

C# es un lenguaje que cumple con la especificación CLS. El código que crearemos con él será traducido a instrucciones **MSIL** para entonces ser traducido, justo antes de su ejecución, a instrucciones nativas que correspondan a la plataforma concreta sobre la cual estemos trabajando.

Cabe destacar que el compilador JIT (*Just In Time*) traduce el código MSIL a código nativo no de manera monolítica, sino por métodos, módulos y componentes. Por lo tanto, a grandes rasgos: código que no sea ejecutado no será compilado.

El código MSIL generado a partir de la compilación de código C# es idéntico al código MSIL generado a partir de cualquier otro lenguaje CLS. Esto podría abrir el interrogante de ¿por qué programar en C# en lugar de hacerlo en VB.NET o en Managed C++ o, incluso, en Delphi .NET? Esta pregunta podría responderse con otra: ¿por qué programar en C++ en lugar de hacerlo en C o Pascal, o en cualquier otro lenguaje compilado, si todos generan el mismo código Intel x86?

Cada lenguaje posee sus características que lo tornan ideal para ciertos usos; además, presenta diversos grados de expresividad que pueden permitir implementar el mismo algoritmo de maneras diversas, por lo que un modo puede resultar más eficiente que otro.

Características fundamentales del lenguaje

C# es un lenguaje moderno y altamente expresivo que se ajusta al paradigma de programación orientada a objetos. Su sintaxis es similar a C++ y Java. El lenguaje fue desarrollado en gran parte por **Anders Hejlsberg** (creador del mítico compilador **Turbo Pascal**¹ y uno de los diseñadores líder del lenguaje de programación Delphi).

En C# no existe el concepto de función global o variable fuera de una clase u objeto. Por su buen apego a la POO, es posible sobrecargar métodos y operadores. Soporta definición de interfaces. Ninguna clase puede poseer más de un padre (no se permite la herencia múltiple), pero sí puede suscribir un contrato con diversas interfaces. Soporta la definición de estructuras, pero, a diferencia de C++, aquí no son tan parecidas a las clases y poseen ciertas restricciones que veremos luego.

Permite además la declaración de propiedades, eventos y atributos (que son construcciones declarativas).

Por último, una característica distintiva cada vez más presente en lenguajes modernos es la gestión automática de memoria y el uso de referencias en lugar de punteros. Gracias a esta gestión automática de memoria ya no tendremos que preocuparnos por la existencia de *memory leaks* (zonas de memoria que permanecen reservadas pero ya no son utilizadas debido a errores de programación).

A continuación realizaremos una breve comparación de C# con los lenguajes más populares del momento:

C# contra Visual Basic 6.0

Si usted es un programador Visual Basic y está evaluando “moverse” a C#, no lo dude un segundo. Existen muchísimas razones para tomar esta decisión, y aquí exponemos algunas de ellas. En primer lugar, uno de los motivos por los cuales Visual Basic era atractivo era su productividad. Sí, a pesar de algunos de sus inconvenientes para crear aplicaciones sencillas, no existía mejor solución que Visual Basic; ningún

¹ No confundir con el autor del lenguaje Pascal, que fue Niklaus Wirth. Anders Hejlsberg fue el creador de un compilador llamado “Turbo Pascal”, que fue muy popular en sus tiempos por su velocidad y bajo costo.

otro lenguaje podía competir en velocidad de desarrollo con él. Pues bien, C# le ha quitado la corona; la misma aplicación que usted puede realizar en Visual Basic podrá crearla en C# en, al menos, el mismo tiempo, e incluso más rápidamente. Por otro lado, gracias a las nuevas características del lenguaje, el diseño de sus aplicaciones podrá ser más elegante y simple, y de este modo podrá *manejar* la complejidad de una manera más natural.

En segundo lugar, seamos sinceros: los programadores Visual Basic nunca fueron vistos como programadores reales, aunque muchísimos profesionales sufran de esta etiqueta de manera injusta. Los sueldos de un programador Visual Basic son mucho más bajos que los de un programador C++ o Java. C#, en cambio, es visto como un lenguaje más profesional. Claro que éste es un punto un tanto controvertido, y podríamos discutir las razones del porqué de esta situación, pero, dejando excepciones a un lado, es una realidad de mercado.

Por último, el pasaje de Visual Basic 6.0 a C# hasta podría ser considerado lógico para quien sea un dominador del VB; es como comprarse un automóvil más lujoso o mudarse a una casa más grande: es el paso evolutivo natural.

Probablemente, lo que usted estará preguntándose respecto al pasaje de Visual Basic 6.0 a C# es el natural ¿para qué existe Visual Basic .NET? Bueno, tal vez Visual Basic .NET nunca debió haber existido; es “demasiado” distinto de Visual Basic 6.0 para ser considerado una nueva versión del lenguaje y, a fin de cuentas, sigue siendo Basic. Tal vez la pregunta que podríamos hacernos sería: si es que vamos a tener que aprender un lenguaje nuevo, ¿por qué no aprender el mejor y más popular lenguaje de la plataforma?

C# contra C++

Admito que C++ es mi lenguaje preferido; trabajé durante muchos años con él e, incluso, escribí un libro de C++ (*C++ Programación Orientada a Objetos*, de esta misma editorial), pero probé C# y fue como probar ambrosía.



VISUAL BASIC .NET VS. C#

En el enlace de referencia es posible encontrar una comparativa entre Visual Basic .NET y C# escrita por Mario Félix Guerrero.

Enlace: www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_2128.asp.

Hoy día, en mi trabajo diario, complemento de manera natural ambos lenguajes; no fue difícil ingresar en el mundo C# desde C++. Lo que debo admitir es que luego de estar días trabajando con C#, volver a C++ es un cambio un tanto abrupto, y me descubro preguntando una y otra vez por qué no se encontrará disponible tal o cual característica de C# en C++.

Los detractores de C++ lo acusan de ser un lenguaje híbrido, un C con clases, como muchas veces es definido. El problema de esta falta de decisión en cuanto al diseño del lenguaje es que permite que convivan clases con funciones y variables globales, lo cual podría ser tentador para el programador y podría empobrecer el diseño de una aplicación construida de este modo. Claro que, por otro lado, hay quienes gustan de ver el vaso medio lleno y sostienen que esta característica de “hibridez” es positiva, pues deja al programador la posibilidad de tomar lo mejor de los dos mundos.

C# es, en alguna medida, una evolución de C++, ya que toma prestada su sintaxis y mejora muchos de sus aspectos, como el hecho de poseer una librería de clases unificada y ser un lenguaje orientado a objetos puro (adiós hibridez). Esto no quiere decir que lo reemplace en todos sus usos, claro. C++ seguirá teniendo su segmento de utilización en donde es ideal, pero C# invade día tras día áreas donde antes C++ era único conquistador.

C# contra Java

Muchos sostienen que C# es una copia de Java. Si tomamos ciertos trozos de código (convenientemente) escritos en alguno de estos lenguajes, podría ser imposible determinar si es C# o es Java.

Lo cierto es que parte de la filosofía empleada por ambos lenguajes es la misma. Ambos se ejecutan sobre una máquina virtual, y esta característica los convierte en lenguajes potencialmente multiplataforma; ambos poseen especificaciones de subconjuntos de recursos de lenguaje para implementaciones en diversos dispositivos (por ejemplo, móviles), y ambos tienen una librería de clases con muchas características en común.



SIMILITUDES ENTRE C#, C++ Y JAVA

En el enlace de referencia podrán encontrar una breve comparativa entre estos lenguajes. Enlace: www.microsoft.com/spanish/MSDN/estudiantes/desarrollo/lenguajes/c-sharp.asp.

Sin embargo, C# presenta la ventaja de integrarse mejor con aplicaciones nativas de la plataforma sobre la cual estemos trabajando. Claro que, si accedemos a recursos nativos, perderá la característica de ser multiplataforma, pero esta característica no siempre es deseada. Podríamos tener la necesidad de invocar métodos de librerías nativas creadas en C++ para Windows de una manera sencilla y eficiente.

Luego podríamos discutir en muchos puntos cuál es mejor que cuál; en la Red, los foros de discusión sobre programación se encuentran plagados de peleas de este tipo. La realidad es que hoy día C# es más fuerte en plataformas basadas en Windows, mientras que Java es más fuerte en una gran diversidad de plataformas menos populares (celulares, tarjetas inteligentes, etc.) y es el candidato ideal si hoy desea construir una aplicación que deba ejecutarse sin cambios en Linux y Windows, aunque esta realidad se encuentre próxima a cambiar.

EL ENTORNO DE DESARROLLO VISUAL STUDIO .NET

En el desarrollo de este libro trabajaremos con Visual Studio .NET 2003, que es la última versión de producción del entorno en el momento de escribir estas líneas. Existe una versión “2005 Express” en estado beta que también podría utilizar el lector.

Por otro lado, el libro se basa principalmente en la versión 1.1 del framework, que también es la última versión de producción a la fecha de publicación de esta obra. Quienes no posean el entorno Microsoft Visual Studio pueden utilizar **Mono** en ambientes **Windows**, **GNU/Linux** y **Mac Os X**.

El entorno Visual Studio .NET 2003 es una excelente herramienta de desarrollo. Con ella podremos crear soluciones que, a su vez, podrán contener uno o más proyectos de diversos lenguajes en función de los paquetes que tengamos instalados; en

III VISUAL STUDIO 2005 EDICIÓN EXPRESS

Es posible descargar la versión beta de este entorno desde <http://msdn.microsoft.com/vs2005>. Seguramente, en el transcurso del año, el entorno saldrá en su versión definitiva y ya no será posible descargar la versión beta de manera gratuita. De todos modos, Microsoft planea ofrecer las versiones express del entorno de desarrollo a un costo alcanzable por cualquier desarrollador.

principio dispondremos de C#, Visual Basic .NET y C++. También es posible crear aplicaciones C# utilizando el framework .NET SDK, que es un conjunto de herramientas que nos permiten compilar código fuente C# para crear aplicaciones. Visual Studio .NET hace uso del framework .NET SDK y funciona como *front end* para evitar que nosotros tengamos que interactuar con herramientas en comando de línea. El framework .NET SDK puede ser descargado desde el siguiente enlace: <http://msdn.microsoft.com/netframework>.

Sin embargo, Visual Studio .NET es mucho más que un simple *front end*. Algunas de las tareas que podremos realizar con él son las siguientes:

- Navegar fácilmente por las clases por medio del visor de clases.
- Navegar por los archivos de nuestros proyectos por medio del explorador de soluciones.
- Entender más rápidamente el código escrito gracias a que el editor colorea las palabras reservadas y los tipos de datos conocidos.
- Organizar múltiples proyectos y editar fácilmente sus propiedades.
- Configurar el entorno para que ejecute herramientas externas (como precompiladores).
- Depurar nuestros proyectos fácilmente y consultar valores de objetos de modo interactivo, así como realizar depuraciones remotas desde otras computadoras.
- Acceder a facilidades de búsqueda y reemplazo por hoja de código fuente activo y en archivos.
- Editar recursos (bitmaps, iconos, archivos binarios, etc.) por medio de herramientas integradas, y navegar por ellos por medio del visor de recursos.
- Agregar tareas por realizar haciendo uso de la lista de tareas pendientes, que además inserta automáticamente tareas a partir de comentarios prefijados.
- Generar documentación en formato HTML por medio de una herramienta provista con el entorno. Esta documentación es generada a partir del código fuente y comentarios con formatos específicos que escribiremos en él.
- Colapsar y expandir trozos de código para mejorar la legibilidad de nuestras fuentes.
- Posibilidad de integrar herramientas al entorno por medio de un sistema de plug-ins.



¿QUÉ ES MONO?

Desde el año 2001, la empresa **Ximian** (adquirida luego por **Novell**) comenzó a desarrollar el proyecto **Mono**, que es una plataforma de desarrollo **Open Source** basada en el framework .NET. Con Mono es posible escribir aplicaciones en C# (o VB.NET) y ejecutarlas no sólo en Windows, sino también en **GNU/Linux** y **Mac Os X**. Se puede descargar desde www.mono-project.com.

Haciendo uso del entorno de programación Visual Studio .NET crearemos proyectos. Un proyecto contendrá, básicamente, hojas de código fuente en C# (luego veremos que también podrá contener otros recursos).

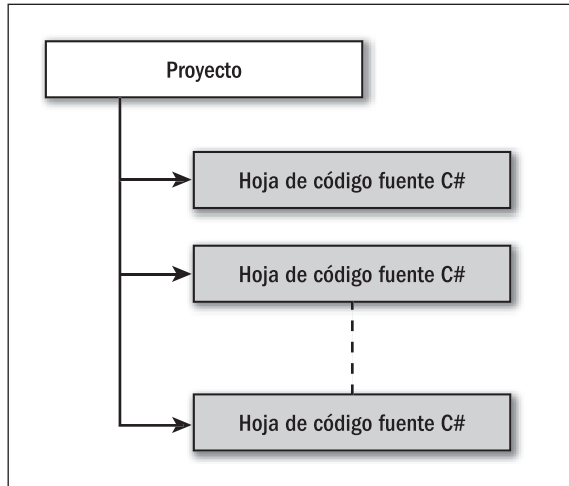


Figura 3. Organización de un proyecto.

Un proyecto posee ciertas propiedades, las cuales indican de qué manera deberán compilarse las fuentes y los recursos que incluye. La idea de “proyecto” como organización de fuentes se encuentra muy extendida, y casi todo entorno de programación la maneja. Usualmente, un proyecto construido tendrá como salida una aplicación ejecutable (archivo **.EXE**) o una librería (archivo **.DLL**). Sin embargo, es posible (y, de hecho, muy común) que el sistema en el cual estemos trabajando se encuentre integrado por más de un componente; por ejemplo, una aplicación y una librería de enlace dinámico que sea utilizada por la aplicación. En este caso, no es sólo un proyecto el que deberemos crear, sino dos.

Ahora bien, si necesitaríamos crear dos proyectos siendo uno dependiente del otro, ¿qué deberíamos hacer? Podríamos crear un proyecto en una instancia de Visual Studio y otro proyecto en otra instancia, pero ésta no es una muy buena idea, ya

III ¿QUÉ ES UNA DLL?

Una DLL (*Dynamic Link Library*) es un conjunto de funciones y/o clases que pueden ser accedidas y utilizadas por otros programas en tiempo de ejecución. Estas librerías pueden ser creadas desde C# o desde otros lenguajes.

que si modificamos la librería y nos olvidamos de reconstruirla, nuestra aplicación seguirá utilizando una versión vieja de ésta.

Afortunadamente, Visual Studio .NET soporta más de un proyecto abierto de manera simultánea en el mismo espacio de trabajo. Estos proyectos pueden relacionarse entre sí por medio de dependencias; de modo que si reconstruimos la aplicación, el entorno automáticamente reconstruirá todos los proyectos dependientes que se hayan modificado (por ejemplo, la librería).

Visual Studio denomina **solución** a esta agrupación de proyectos (aunque en ediciones anteriores del entorno se las llamaba **espacios de trabajo**). Todo proyecto debe estar contenido en una solución; por lo tanto, cuando creamos un proyecto nuevo, Visual Studio nos crea automáticamente una solución que lo contiene.

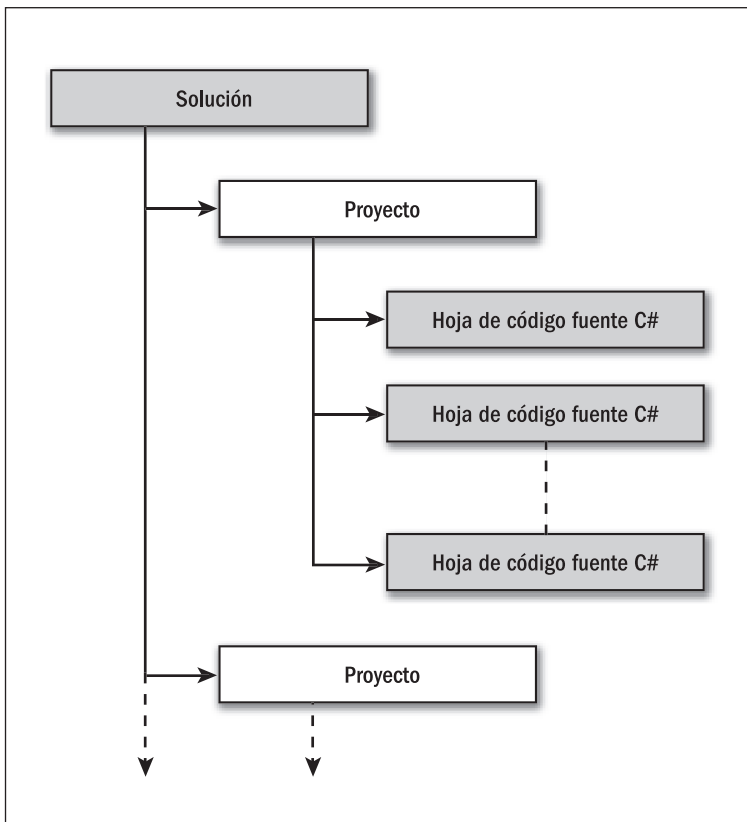


Figura 4. Organización de una solución.

Luego, cuando construimos la solución se construyen todos los proyectos (que pueden estar relacionados directamente o no), es decir que se procesan todos los elementos de cada proyecto para generar los archivos de salida que correspondan.

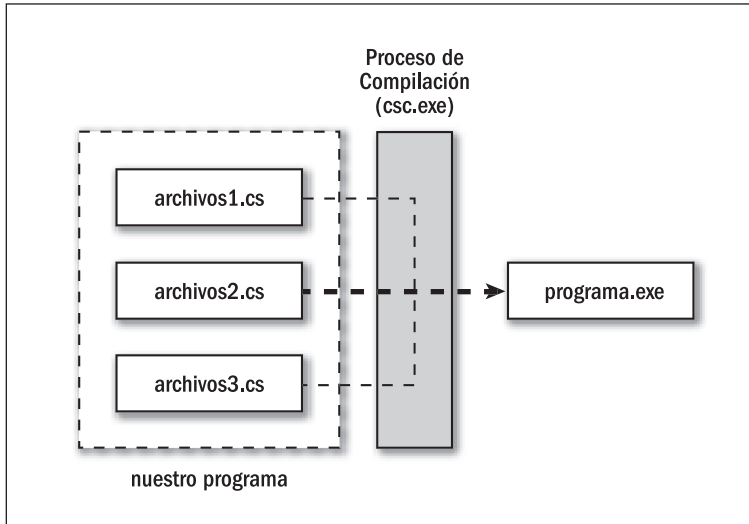


Figura 5. Compilación de un proyecto.

Nuestra primera aplicación con Visual Studio .NET

Pero veamos cómo crear nuestra primera aplicación C# con él. En primer lugar, cabe destacar que el entorno es sumamente configurable y que la organización de los paneles, así como la pantalla de inicio, puede variar en función de cómo la configuremos. Si es la primera vez que ejecuta la aplicación o si no ha modificado las opciones de inicio predeterminadas, debería encontrarse con la “página de inicio” que se muestra en la siguiente figura:

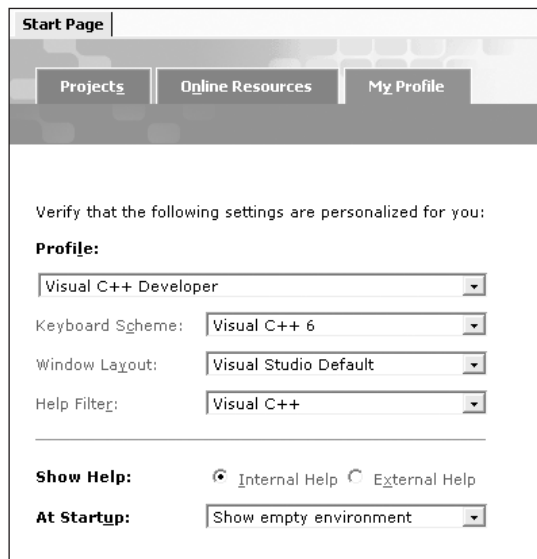


Figura 6. La página de inicio.

En lo personal, me desagrada bastante dicha “página”, por lo que lo primero que hago es fijar las opciones como indica precisamente la figura mencionada, es decir:

Profile: Visual C++ Developer

At Startup: Show empty envioment

Cambiar el perfil permite que se modifique la ubicación predeterminada de los paneles en donde se encuentran las herramientas (como el visor de clases, el explorador de soluciones, etc.). En lo personal, me he acostumbrado a tener el panel de visor de clases a la izquierda, como estaba en el viejo y querido Visual C++ 6.0, pero cada quien podrá hacer lo que le plazca, ya que esta configuración no nos impedirá modificar ni crear ningún tipo de proyecto u opción en particular.

Fijar el inicio de aplicación en **Show empty envioment** (*Mostrar un entorno vacío*) también es una cuestión de gustos, y lo que especifica es que cada vez que iniciemos la aplicación veremos lo que muestra la siguiente figura:

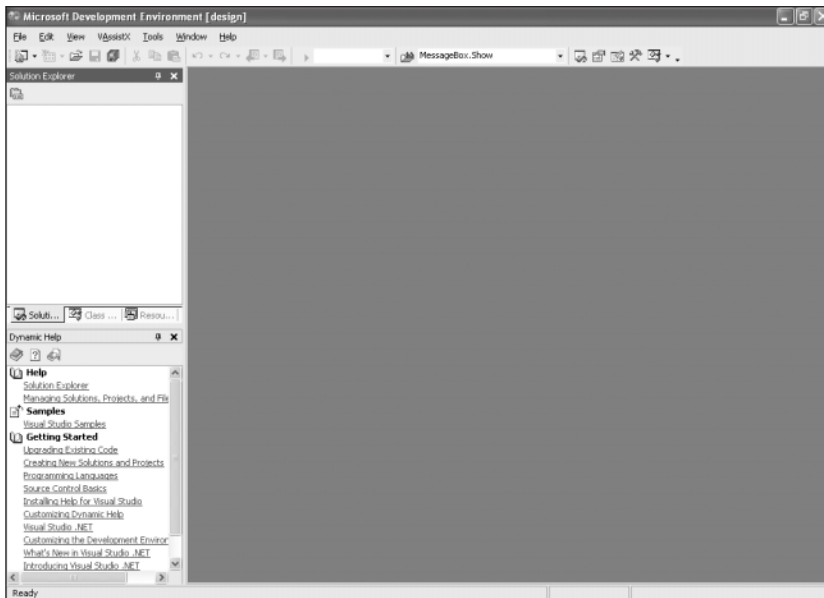
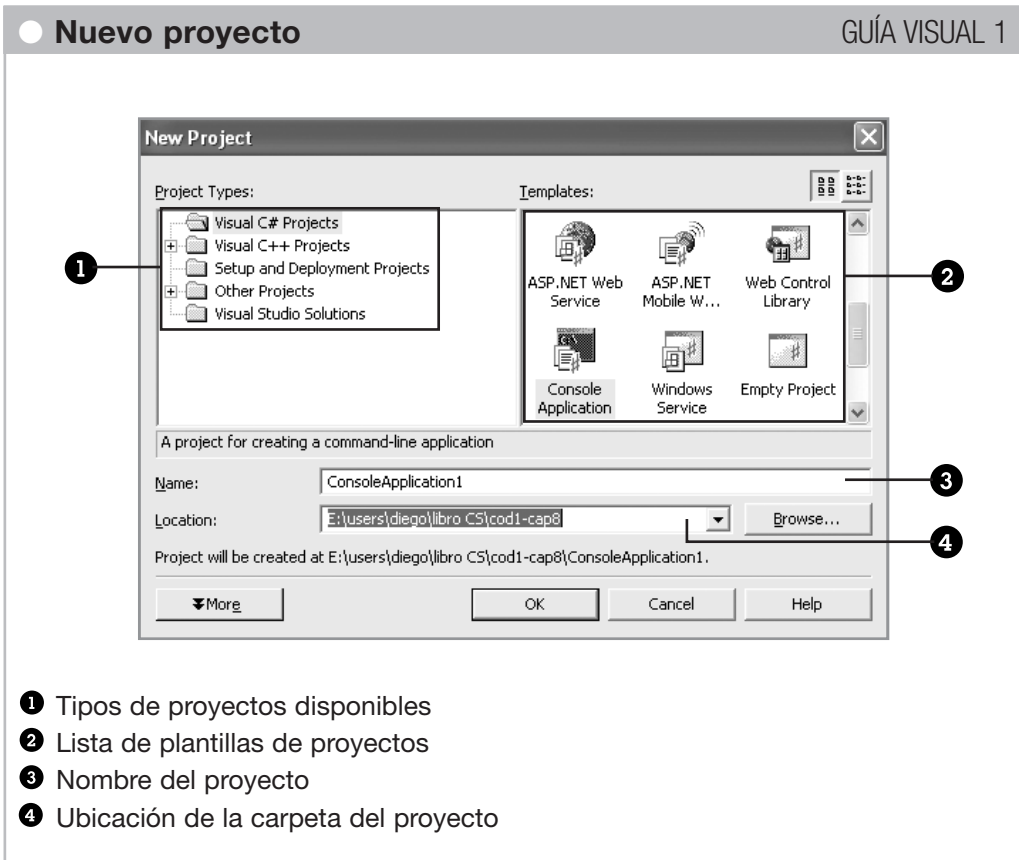


Figura 7. Visual Studio, comenzando con el entorno vacío.

Bueno, ahora sí, vayamos a la acción. Teniendo el entorno ya abierto, lo primero que deberemos hacer será “Crear un nuevo proyecto”. Si estamos en la “Página de Inicio”, deberemos seleccionar la pestaña **Projects**, y luego, presionar el botón **New Project** que se encuentra en el extremo inferior de la página. Si estamos en un entorno vacío, deberemos acceder al menú **File**, desde allí seleccionar **New** y, finalmente, la opción **Project**. Hecho esto, deberá aparecer la ventana de la **Guía Visual 1**.



Seleccionaremos siempre que el tipo de proyecto sea **Visual C# Projects**, luego especificaremos la plantilla del tipo de proyecto que deseamos crear. Lo que hace la plantilla del proyecto es fijar opciones predeterminadas al modo de compilación de nuestro proyecto, que finalmente especificará los parámetros que se enviarán al compilador C# cuando construyamos el proyecto para generar un archivo ejecutable (con instrucciones MSIL).

Para comenzar con algo sencillo, seleccionaremos la plantilla **Aplicación de consola** (en inglés, **Console Application**), modificaremos el nombre del proyecto y su ubicación, si es que no nos gustan las opciones que nos sugiere el entorno, y finalmente presionaremos el botón **OK**.

Hecho esto, el entorno creará un proyecto C# que poseerá dos archivos del tipo **.CS** (por C#, que se pronuncia **C Sharp** y usualmente se abrevia **CS**). Los **.CS** son archivos con código fuente C#; no serán distribuidos con nuestra aplicación, solamente los utilizaremos para crearla.

Si por medio del explorador de soluciones miramos los nombres de los archivos que componen nuestro proyecto, veremos estos dos archivos **.CS**, que se llaman:

```
AssemblyInfo.cs
Class1.cs
```

- **AssemblyInfo**: posee información de nuestra aplicación (nombre del producto, versión, empresa, etc.). Este recurso compilado es situado dentro de nuestra aplicación y se conoce como *assembly*. Luego explicaremos en detalle qué es un *assembly*.
- **Class1**: archivo de código fuente que posee la única clase de la aplicación y el punto de entrada al programa.

No importa lo que esté escrito en dicho archivo, nosotros cambiaremos el programa y escribiremos lo siguiente:

```
class Clase1
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Nuestro primer programa");
    }
}
```

Analicemos el código anterior. En primer lugar vemos la palabra **class** seguida de un identificador de clase (en este caso, **Clase1**); luego existe un bloque de código encerrado entre llaves.

Quienes tengan algunos mínimos conocimientos de lenguaje C o C++ sabrán que en todo programa debe existir una función llamada **main**, que hace de punto de entrada al programa, es decir, que es la función que debe ejecutarse para iniciar la aplicación.

Habíamos mencionado que en C# todo programa debe estar compuesto de clases, es decir, que no pueden existir funciones globales. Por lo tanto, al menos una clase deberá poseer un método llamado **Main** (esta vez con la “M” en mayúscula), que además deberá ser “estático” (luego veremos qué significa esto).

Dentro del método **Main** se encontrará el código que será ejecutado cuando nuestro programa sea inicializado. En nuestro caso, dicho código será solamente:

```
System.Console.WriteLine("Nuestro primer programa");
```

WriteLine es un método de la clase **Console** que escribe un texto en la salida estándar. Dicho texto se especifica entre comillas (comillas dobles).

¡Bien!, ahora podremos compilar nuestro programa ingresando en el menú **Build**, opción **Build Solution**.

Si hacemos esto, podremos ver que en el panel llamado **Output** (salida) se observa el progreso de la construcción de nuestra aplicación. Si existe algún error, también se mostrará allí como información, y podremos hacer doble clic sobre él para que el entorno nos lleve automáticamente a la línea donde se encontró la falla.

```
— Build started: Project: ConsoleApplication1, Configuration:
  Debug .NET —
Preparing resources...
Updating references...
Performing main compilation...
Build complete - 0 errors, 0 warnings
Building satellite assemblies...
————— Done —————
Build: 1 succeeded, 0 failed, 0 skipped
```

Si hemos tipeado todo bien, deberemos ver lo que indica el cuadro anterior, es decir, la notificación de que la aplicación ha sido construida satisfactoriamente.

Luego, podremos ingresar nuevamente en el menú **Build** y seleccionar la opción **Start** (comienzo) o **Start without debugging** (comienzo sin depurar). Lo bueno de esta última opción es que no cierra la ventana donde está nuestro programa cuando éste termina, sino que nos solicita que ingresemos una tecla para continuar.

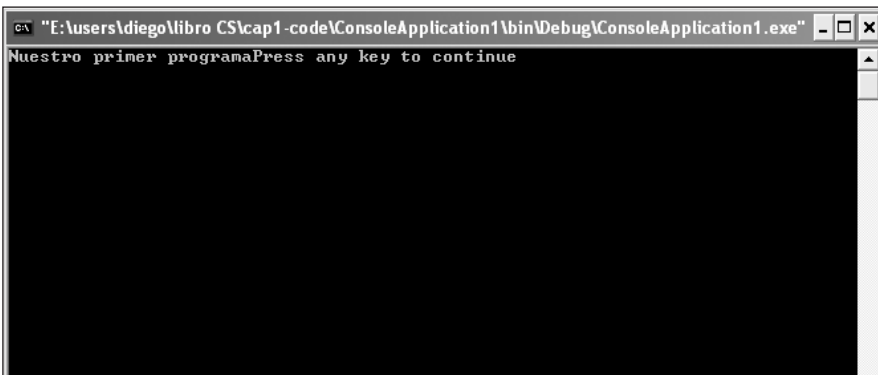


Figura 8. Nuestro primer programa C#.

Claro que si no tenemos Visual Studio a mano, pero hemos instalado el framework .NET, podremos realizar la compilación de modo manual (o para expresarnos mejor: utilizando el compilador desde comando de línea). Al fin y al cabo, habíamos comentado que Visual Studio, entre muchas otras cosas, es un *front end* al compilador C# (y otros compiladores y herramientas). El compilador de C# es una aplicación llamada **csc.exe**, que se encuentra en el directorio de **Windows**, dentro de la carpeta **Microsoft.NET/Framework**, y allí, dentro de la versión del framework que tengamos instalado (por ejemplo, **v1.1.4322**).

Teniendo un archivo llamado, por ejemplo, **programa.cs**, podremos realizar una compilación en comando de línea escribiendo:

```
csc programa.cs
```

Esto nos dejará como salida, en el directorio donde nos encontremos y si no hubo errores en la compilación, un archivo llamado “programa.exe”. Si ejecuta la aplicación **csc.exe** con el parámetro **/?**, se listarán los switches válidos del compilador.

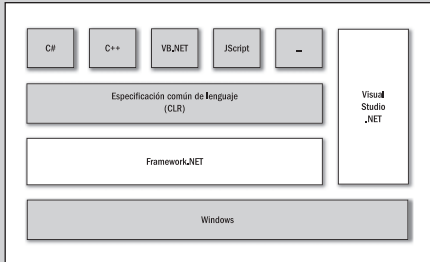
RESUMEN

Está bien, nuestra primera aplicación no ha sido muy excitante, pero al menos hemos dado el primer paso y eso no es poco. También es cierto que algunas cuestiones han quedado sin mucha explicación: ¿qué es una clase?, ¿qué es un método estático?, ¿qué es System? No nos desespereemos; a medida que avancemos en el libro, iremos tratando todos estos temas de manera detallada. Ya hemos visto el conejo blanco, ahora sólo deberemos seguirlo.



TEST DE AUTOEVALUACIÓN

1 ¿Qué características diferencian al lenguaje C# de C++?



2 ¿Qué es el código MSIL?

3 ¿Qué es la BCL y para que se utiliza?

4 ¿Qué es el CLR?

5 ¿En qué plataformas se pueden ejecutar los programas creados en C#?

6 ¿Es posible compilar un programa C# sin hacer uso del Visual Studio .NET?