

Manual sobre la plataforma .Net

Manual sobre la plataforma .Net. Manual indicado para comprender el funcionamiento de la nueva plataforma multilinguaje de Microsoft y para iniciarse en la construcción de aplicaciones Web, formularios de Windows y servicios Web, entre otros.



Autores del manual

Este manual ha sido realizado por los siguientes colaboradores de DesarrolloWeb.com:

Francisco Recio y David Provencio
(13 capítulos)

Miguel Angel Alvarez
Director de DesarrolloWeb.com
<http://www.desarrolloweb.com>
(1 capítulo)

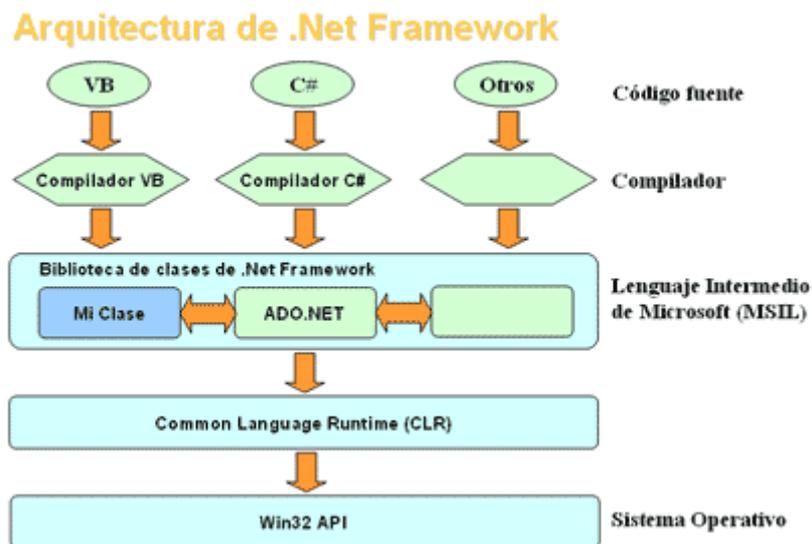
.Net Framework

Arquitectura básica de la plataforma .Net. Descripción del Framework y sus principales componentes: Lenguajes, biblioteca de clases y CLR.

La nueva tecnología de Microsoft ofrece soluciones a los problemas de programación actuales, como son la administración de código o la programación para Internet. Para aprovechar al máximo las características de .Net es necesario entender la arquitectura básica en la que esta implementada esta tecnología y así beneficiarse de todas las características que ofrece esta nueva plataforma.

El Framework de .Net es una infraestructura sobre la que se reúne todo un conjunto de lenguajes y servicios que simplifican enormemente el desarrollo de aplicaciones. Mediante esta herramienta se ofrece un entorno de ejecución altamente distribuido, que permite crear aplicaciones robustas y escalables. Los principales componentes de este entorno son:

- Lenguajes de compilación
- Biblioteca de clases de .Net
- CLR (Common Language Runtime)



Actualmente, el Framework de .Net es una plataforma no incluida en los diferentes sistemas operativos distribuidos por Microsoft, por lo que es necesaria su instalación previa a la ejecución de programas creados mediante .Net. El Framework se puede descargar gratuitamente desde la web oficial de Microsoft (ver link de descarga en los recursos del final).

.Net Framework soporta múltiples lenguajes de programación y aunque cada lenguaje tiene sus características propias, es posible desarrollar cualquier tipo de aplicación con cualquiera de estos lenguajes. Existen más de 30 lenguajes adaptados a .Net, desde los más conocidos como C# (C Sharp), Visual Basic o C++ hasta otros lenguajes menos conocidos como Perl o Cobol.

Common Language Runtime (CLR)

El CLR es el verdadero núcleo del Framework de .Net, ya que es el entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes, ampliando el conjunto de servicios que ofrece el sistema operativo estándar Win32.

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .Net en un mismo código, denominado código intermedio (MSIL, Microsoft Intermediate Language). Para generar dicho código el compilador se basa en el Common Language Specification (CLS) que determina las reglas necesarias para crear código MSIL compatible

con el CLR.

De esta forma, indistintamente de la herramienta de desarrollo utilizada y del lenguaje elegido, el código generado es siempre el mismo, ya que el MSIL es el único lenguaje que entiende directamente el CLR. Este código es transparente al desarrollo de la aplicación ya que lo genera automáticamente el compilador.

Sin embargo, el código generado en MSIL no es código máquina y por tanto no puede ejecutarse directamente. Se necesita un segundo paso en el que una herramienta denominada compilador JIT (Just-In-Time) genera el código máquina real que se ejecuta en la plataforma que tenga la computadora.

De esta forma se consigue con .Net cierta independencia de la plataforma, ya que cada plataforma puede tener su compilador JIT y crear su propio código máquina a partir del código MSIL.

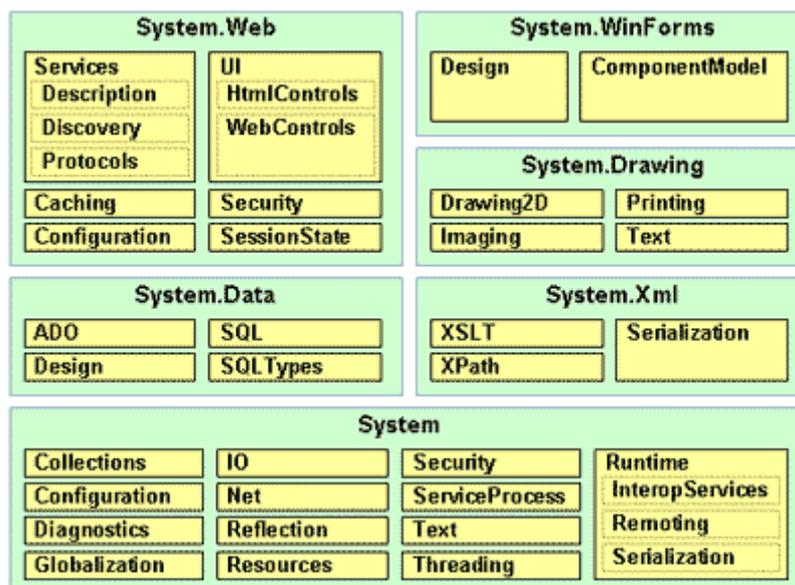
La compilación JIT la realiza el CLR a medida que se invocan los métodos en el programa y, el código ejecutable obtenido, se almacena en la memoria caché de la computadora, siendo recompilado sólo cuando se produce algún cambio en el código fuente.

Biblioteca de clases de .Net

Cuando se está programando una aplicación muchas veces se necesitan realizar acciones como manipulación de archivos, acceso a datos, conocer el estado del sistema, implementar seguridad, etc. El Framework organiza toda la funcionalidad del sistema operativo en un espacio de nombres jerárquico de forma que a la hora de programar resulta bastante sencillo encontrar lo que se necesita.

Para ello, el Framework posee un sistema de tipos universal, denominado Common Type System (CTS). Este sistema permite que el programador pueda interactuar los tipos que se incluyen en el propio Framework (biblioteca de clases de .Net) con los creados por él mismo (clases). De esta forma se aprovechan las ventajas propias de la programación orientada a objetos, como la herencia de clases predefinidas para crear nuevas clases, o el polimorfismo de clases para modificar o ampliar funcionalidades de clases ya existentes.

Biblioteca de clases de .NET Framework



La biblioteca de clases de .Net Framework incluye, entre otros, tres componentes clave:

- ASP.NET para construir aplicaciones y servicios Web.
- Windows Forms para desarrollar interfaces de usuario.
- ADO.NET para conectar las aplicaciones a bases de datos.

La forma de organizar la biblioteca de clases de .Net dentro del código es a través de los espacios de nombres (namespaces), donde cada clase está organizada en espacios de nombres según su funcionalidad. Por ejemplo, para manejar ficheros se utiliza el espacio de nombres System.IO y si lo que se quiere es obtener información de una fuente de datos se utilizará el espacio de nombres System.Data.

La principal ventaja de los espacios de nombres de .Net es que de esta forma se tiene toda la biblioteca de clases de .Net centralizada bajo el mismo espacio de nombres (System). Además, desde cualquier lenguaje se usa la misma sintaxis de invocación, ya que a todos los lenguajes se aplica la misma biblioteca de clases.

Ensamblados

Uno de los mayores problemas de las aplicaciones actuales es que en muchos casos tienen que tratar con diferentes archivos binarios (DLL's), elementos de registro, conectividad abierta a bases de datos (ODBC), etc.

Para solucionarlo el Framework de .Net maneja un nuevo concepto denominado ensamblado. Los ensamblados son ficheros con forma de EXE o DLL que contienen toda la funcionalidad de la aplicación de forma encapsulada. Por tanto la solución al problema puede ser tan fácil como copiar todos los ensamblados en el directorio de la aplicación.

Con los ensamblados ya no es necesario registrar los componentes de la aplicación. Esto se debe a que los ensamblados almacenan dentro de sí mismos toda la información necesaria en lo que se denomina el manifiesto del ensamblado. El manifiesto recoge todos los métodos y propiedades en forma de meta-datos junto con otra información descriptiva, como permisos, dependencias, etc.

Para gestionar el uso que hacen las aplicaciones de los ensamblados .Net utiliza la llamada caché global de ensamblados (GAC, Global Assembly Cache). Así, .Net Framework puede albergar en el GAC los ensamblados que puedan ser usados por varias aplicaciones e incluso distintas versiones de un mismo ensamblado, algo que no era posible con el anterior modelo COM.

Recursos:

Página oficial de .Net Framework
<http://msdn.microsoft.com/netframework/>

Descarga de .Net Framework
<http://msdn.microsoft.com/netframework/howtoget/default.aspx>

Artículo por Francisco Recio y David Provencio

Ventajas de .Net

Algunas de las ventajas e inconvenientes de la plataforma .Net.

A continuación se resumen las ventajas más importantes que proporciona .Net Framework:

- **Código administrado:** El CLR realiza un control automático del código para que este sea seguro, es decir, controla los recursos del sistema para que la aplicación se ejecute correctamente.
- **Interoperabilidad multilinguaje:** El código puede ser escrito en cualquier lenguaje compatible con .Net ya que siempre se compila en código intermedio (MSIL).
- **Compilación just-in-time:** El compilador JIT incluido en el Framework compila el código intermedio (MSIL) generando el código máquina propio de la plataforma. Se aumenta así el rendimiento de la aplicación al ser específico para cada plataforma.
- **Garbage collector:** El CLR proporciona un sistema automático de administración de memoria denominado recolector de basura (garbage collector). El CLR detecta cuándo el programa deja de utilizar la memoria y la libera.

automáticamente. De esta forma el programador no tiene por que liberar la memoria de forma explícita aunque también sea posible hacerlo manualmente (mediante el método `dispose()` liberamos el objeto para que el recolector de basura lo elimine de memoria).

- **Seguridad de acceso al código:** Se puede especificar que una pieza de código tenga permisos de lectura de archivos pero no de escritura. Es posible aplicar distintos niveles de seguridad al código, de forma que se puede ejecutar código procedente del Web sin tener que preocuparse si esto va a estropear el sistema.
- **Despliegue:** Por medio de los ensamblados resulta mucho más fácil el desarrollo de aplicaciones distribuidas y el mantenimiento de las mismas. El Framework realiza esta tarea de forma automática mejorando el rendimiento y asegurando el funcionamiento correcto de todas las aplicaciones.

¿Todo son ventajas?

Procesos como la recolección de basura de .Net o la administración de código introducen factores de sobrecarga que repercuten en la demanda de más requisitos del sistema.

El código administrado proporciona una mayor velocidad de desarrollo y mayor seguridad de que el código sea bueno. En contrapartida el consumo de recursos durante la ejecución es mucho mayor, aunque con los procesadores actuales esto cada vez es menos inconveniente.

El nivel de administración del código dependerá en gran medida del lenguaje que utilicemos para programar. Por ejemplo, mientras que Visual Basic .Net es un lenguaje totalmente administrado, C Sharp permite la administración de código de forma manual, siendo por defecto también un lenguaje administrado. Mientras que C++ es un lenguaje no administrado en el que se tiene un control mucho mayor del uso de la memoria que hace la aplicación.

Artículo por Francisco Recio y David Provencio

Visual Web Developer 2005 Express Edition

Un entorno de desarrollo gratuito para tus creaciones ASP.NET, que además ofrece componentes adicionales de fácil instalación como el .NET Framework o la base de datos SQL Server Express.

Comentamos un programa que ha presentado recientemente Microsoft para facilitar el acceso a su tecnología de desarrollo ASP.NET. El programa en concreto es Visual Web Developer Express Edition, un IDE (Entorno de desarrollo) para programar fácilmente en ASP.NET, pero que además proporciona otras herramientas útiles o imprescindibles para realizar nuestros proyectos.

En el momento de escribir este artículo, el programa se encuentra en su versión "Beta 2", que es la que hemos instalado y probado. Se puede descargar gratuitamente desde la web de Microsoft. La versión publicada gratuitamente se puede utilizar sin limitaciones durante 30 días. En adelante, si se desea continuar su uso, se debe registrar, también gratis, desde el mismo ordenador donde se ha instalado y ello nos dará acceso a nuevos beneficios, como documentación adicional.

<http://www.microsoft.com/spanish/msdn/vstudio/express/VWD/default.aspx>

La primera impresión sobre este programa es muy buena. Muchos de nosotros hemos podido tener problemas para acceder fácilmente a la nueva tecnología .NET de Microsoft y con esta herramienta finalizan en parte esas dificultades. Ello es debido a que, no sólo proporciona un programa con el que aumentar la productividad de los programadores al escribir código ASP.NET, sino que además incluye el .NET Framework, un servidor donde ejecutar las aplicaciones creadas en .NET. Por tanto, en pocos minutos podremos empezar a realizar nuestras primeras incursiones en .NET.

Instalación

Durante la instalación se eligen los componentes que se desean poner en nuestro ordenador. El propio programa de instalación se conectará a Internet para recibir aquellos componentes que deseemos. Entre los componentes opcionales se encuentran:

- Microsoft MSDN Express Library

Si necesitamos documentación sobre .NET, con la MSDN tendremos acceso a gran cantidad de información y ayuda.

- Microsoft SQL Server 2005 Express Edition

Una versión reducida de SQL Server, pero que nos servirá para nuestras tareas de desarrollo. Debemos instalar este componente porque, por poco que deseemos hacer, necesitaremos una base de datos.

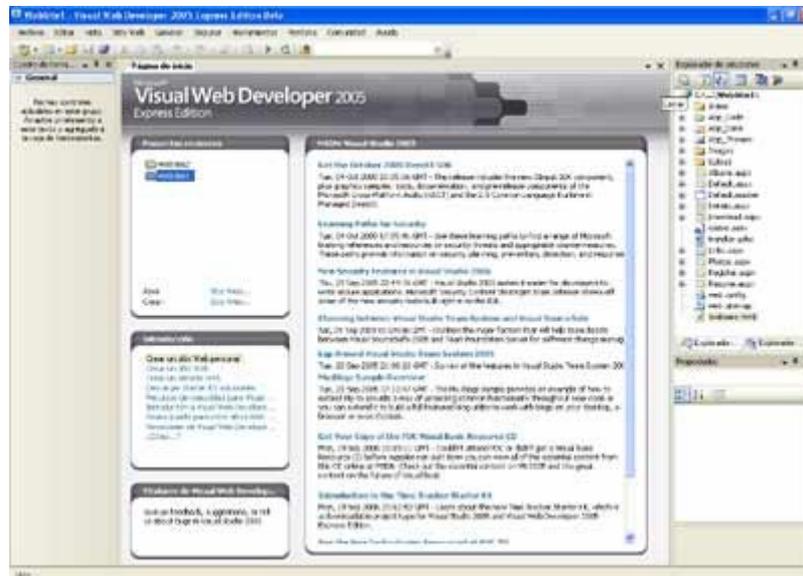


El proceso de instalación continúa solicitando el directorio donde se va a instalar el programa y además mostrando los componentes adicionales que se van a incorporar a nuestro sistema. Entre otros componentes adicionales se encuentran paquetes de idioma y el mencionado .NET Framework, que necesitaremos para ejecutar nuestras aplicaciones ASP.NET.



Una vez dentro del programa

El programa comienza mostrando una página de inicio con las acciones típicas que puede necesitar una persona que acaba de empezar a utilizar Visual Web Developer. Entre otras, podemos realizar acciones como crear un sitio personal, crear un sitio web, crear un servicio web, nuevas descargas, recursos, tutoriales para manejar el programa, etc.



Nosotros, para empezar, creamos un sitio web personal, pulsando tan sólo un botón y nos mostró en pocos segundos una pantalla de bienvenida comentando las características de la página personal. Pulsando CTRL.+F5 se puso en marcha el sitio web sin problemas. (Eso si, hay que tener instalado el SQL Server Express... y el .NET Framework, claro. Pero como dijimos, todo lo necesario se puede obtener durante la instalación.)

Trabajar en ASP.NET resulta más complicado que el otros lenguajes para hacer páginas lado del servidor, como ASP o PHP. Aunque como ventaja hay que remarcar que podrá aumentar sustancialmente nuestra productividad, dado que ofrece muchas ayudas a los desarrolladores como la programación visual. Por todo ello, sin duda, necesitaremos cierta experiencia para sacar provecho de este programa. Aunque los programadores de sistemas Microsoft tendrán mucho de su parte, ya que el entorno y las herramientas son muy similares a los de otros productos de la compañía.

Para adquirir esos conocimientos sobre .NET, si nos registramos como usuarios de Visual Web Developer Express, obtendremos acceso, entre otros, a un completo libro para aprender ASP.NET de la editorial Microsoft Press, la pena es que está en Inglés.

También podemos conocer [algo sobre .NET en nuestro manual](#).

Artículo por *Miguel Angel Alvarez*

Programación orientada a objetos, introducción

Introducción a la programación orientada a objetos, utilizada en la tecnología .Net.

La programación orientada a objetos es una evolución de la programación procedural basada en funciones. La POO nos permite agrupar secciones de código con funcionalidades comunes.

Una de las principales desventajas de la programación procedural basada en funciones es su construcción, cuando una aplicación bajo este tipo de programación crece, la modificación del código se hace muy trabajosa y difícil debido a que el cambio de una sola línea en una función, puede acarrear la modificación de muchas otras líneas de código pertenecientes a otras funciones que estén relacionadas.

Con la programación orientada a objetos se pretende agrupar el código encapsulándolo y haciéndolo independiente, de manera que una modificación debida al crecimiento de la aplicación solo afecte a unas pocas líneas.

La organización de una aplicación en POO se realiza mediante estructuras de código, también llamados objetos. Estos objetos contienen una serie de procedimientos e información destinados a resolver un grupo de tareas con un denominador común. Un procedimiento que este situado en un objeto no podrá ser usado por otro procedimiento perteneciente a otro objeto, si no es bajo una serie de reglas. Los datos que mantenga el objeto, permanecerán aislados del exterior y sólo se podrá acceder a ellos siguiendo ciertas normas.

El objetivo de POO es catalogar y diferenciar el código, en base a estructuras jerárquicas dependientes, al estilo de un árbol genealógico.

Los objetos se crean a partir de una serie de especificaciones o normas que definen como va a ser el objeto, esto es lo que en POO se conoce como una clase.

Las clases definen la estructura que van a tener los objetos que se creen a partir de ella, indicando que propiedades y métodos tendrán los objetos.

Las propiedades definen los datos o información del objeto, permitiendo modificar o consultar su estado, mientras que los **métodos son las rutinas que definen el comportamiento del objeto**. Es necesario tener muy clara cual es la diferencia entre un objeto y una clase, a este respecto podemos decir que una clase constituye la representación abstracta de algo mientras que un objeto constituye la representación concreta de lo que la clase define. Imaginemos los planos de una casa diseñados por un arquitecto, en ellos encontramos el esquema de la casa, las medidas, los materiales etc... Una vez construida la casa podremos comprobar que cumple todo lo que los planos determinaban, de esta manera podemos comparar los planos de la casa con las clases en POO, y la casa en sí con un objeto creado a partir de una clase. Se debe destacar también que con los mismos planos se pueden crear muchas casas iguales, lo mismo ocurre en POO, a partir de una clase se pueden crear muchos objetos iguales.

La creación de un objeto a partir de una clase se conoce como instanciación de un objeto.

Artículo por Francisco Recio y David Provencio

Tipos de datos en .NET

Vemos los tipos de datos para la plataforma .NET, con sus correspondencias tanto en VB.NET y C#.

Todos los lenguajes de programación que cumplen las normas de .NET tienen muchas cosas en común, una de ellas es el conjunto de tipos de datos. Hay que destacar que estos tipos de datos están implementados como clases, de manera que una variable declarada de un tipo determinado, tendrá la capacidad de usar tanto los métodos como las propiedades que pertenezcan a la clase del tipo de dato.

VB.NET

```
Dim Cadena As String
Dim Longitud As Integer
Cadena = "Datos"
Longitud = Cadena.Length()
```

C#

```
String Cadena;
Int Longitud;
Cadena = "Datos";
Longitud = Cadena.Length();
```

En el ejemplo anterior declaramos una variable de tipo String (Cadena de caracteres), y podemos ver como esta variable posee una serie de propiedades y métodos que pueden ser invocados, en este caso usamos la propiedad Length() para obtener el numero de caracteres de la variable Cadena y asignarlo a la variable Longitud, que pasaría a tener el valor 5.

En la siguiente tabla se muestra una relación de los tipos de datos de .NET Framework y su correspondencia en VB.NET y C#.

Nombre de la clase	Tipo de dato en VB.NET	Tipo de dato en C#	Descripción
Byte	Byte	Byte	Entero sin signo de 8 bit.
Sbyte	Sbyte (No nativo)	sbyte	Entero sin signo de 8bit (Tipo no acorde con el CLS)
Int16	Short	short	Entero con signo de 16 bit.
Int32	Integer	int	Entero con signo de 32 bit.
Int64	Long	long	Entero con signo de 64 bit.
UInt16	UInt16 (No nativo)	ushort	Entero sin signo de 16 bit. (Tipo no acorde con el CLS)
UInt32	UInt32 (No nativo)	uint	Entero sin signo de 32 bit. (Tipo no acorde con el CLS)
UInt64	UInt64 (No nativo)	ulong	Entero sin signo de 64 bit. (Tipo no acorde con el CLS)
Single	Single	float	Numero con coma flotante de precisión simple, de 32 bit.
Double	Double	double	Numero con coma flotante de precisión doble, de 64 bit.
Boolean	Boolean	bool	Valor logico
Char	Char	char	Carácter unicode de 16 bit.
Decimal	Decimal	decimal	Valor decimal de 96 bit.

IntPtr	IntPtr (No nativo)	--	Entero con signo cuyo tamaño depende de la plataforma: 32 bit en plataformas de 32 bit y 64 bit en plataformas de 64 bit. (Tipo no acorde con el CLS)
UIntPtr	UIntPtr (No nativo)	--	Entero sin signo cuyo tamaño depende de la plataforma: 32 bit en plataformas de 32 bit y 64 bit en plataformas de 64 bit. (Tipo no acorde con el CLS)
String	String	string	Cadena de caracteres.

Según el modo en el que se almacenan y manipulan estos tipos de datos se pueden dividir en dos categorías.

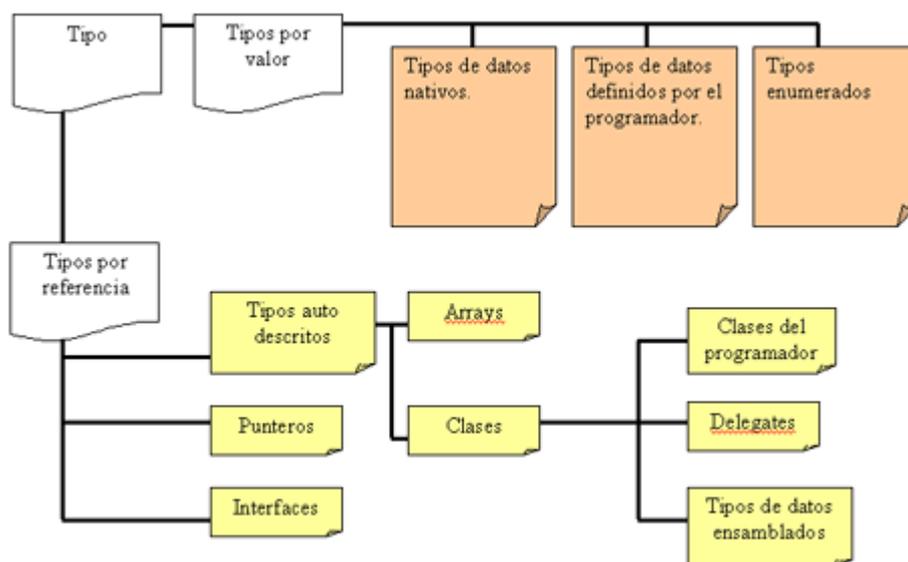
Artículo por *Francisco Recio* y *David Provencio*

Tipos de datos por valor y por referencia

Podemos almacenar variables por valor y por referencia. Explicamos lo que significa cada uno y las maneras que .NET realiza este almacenamiento.

Tipos por valor: los tipos por valor almacenan datos a los que se puede acceder de forma directa, a su vez dentro de esta categoría encontramos mas subcategorías como los tipos nativos de .NET, los tipos de datos creados por el programador y los enumerados. Los tipos por valor no pueden tener valores nulos.

Tipos por referencia: Los tipos creados por referencia almacenan la dirección de memoria en la que se encuentra un dato determinado de manera que usaremos esa dirección de memoria para acceder de forma indirecta al dato. Los tipos por referencia se dividen en varios subgrupos como son las clases propias de la plataforma, interfaces, clases creadas por el programador, etc.



Cuando ejecutamos una aplicación es necesario que los datos se sitúen en la memoria del ordenador, la cual esta divide en dos partes, una llamada Stack, de pequeño tamaño pero de un acceso muy rápido y otra llamada Heap que cuenta con un

mayor tamaño pero con una velocidad de acceso inferior.

Cuando creamos tipos por valor, el valor de la variable de este tipo se almacena en el Stack, si asignamos una variable de estas características a otra, se crea una copia en el Stack. Al destruir un tipo por valor, se destruye también el valor que se guardo en el Stack.

Cuando creamos un tipo por referencia, en realidad lo que guardamos en el Heap es una dirección de memoria que apunta a un valor, pero no al valor en si mismo. Si asignamos una variable que contiene un tipo por referencia a otra variable, se dice que ambas se refieren al mismo valor. Los tipos por referencia si pueden contener valores nulos.

Artículo por Francisco Recio y David Provencio

Creación de clases en VB.Net y C#

Explicación de la creación de clases y objetos en los dos principales lenguajes de la plataforma .Net.

Para explicar la creación de clases usaremos un ejemplo sencillo basado en una clase principal "bicicleta".

Para crear una clase debemos definir sus propiedades y sus métodos, en este ejemplo se usan los siguientes términos como propiedades de la clase bicicleta: Modelo, Precio, NumeroDeVelocidades y Velocidad; como métodos de la clase se usan: Acelerar(km) , Frenar() y ConsultarVelocidad().

VB.NET

```
Public Class Bicicleta
    Public Modelo as String
    Public Precio as Double
    Public NumeroDeVelocidades as Integer
    Private Velocidad as Integer

    Public Sub Acelerar(ByVal km As Integer)
        Velocidad = Velocidad + km
    End Sub

    Public Sub Frenar()
        If Velocidad > 0 Then
            Velocidad = Velocidad -1
        End If
    End Sub

    Public Function ConsultarVelocidad() As Integer
        Return Velocidad
    End Function
End Class
```

C#

```
Class Bicicleta
{
    public string Modelo;
    public double Precio;
    public int NumeroDeVelocidades
    private int Velocidad

    public void Acelerar(int km)
    {
        Velocidad = Velocidad + km;
    }
}
```

```
    }  
  
    public void Frenar()  
    {  
        if (Velocidad > 0)  
        {  
            Velocidad = Velocidad - 1;  
        }  
    }  
  
    public int ConsultarVelocidad()  
    {  
        return Velocidad;  
    }  
}
```

Nuestra clase bicicleta consta de varias propiedades y métodos, las palabras Private y Public definen la accesibilidad de las propiedades, funciones o subrutinas. La definición de una propiedad o método de tipo privado indica que sólo podrá ser usada dentro del código de la misma clase, si creásemos un objeto de tipo bicicleta, las especificaciones de la clase no nos permitirían acceder a la propiedad velocidad para consultarla o modificarla, ya que esta definida como privada. En cambio se pueden usar las subrutinas Acelerar() y Frenar() ya que son de tipo Public, y desde dentro de ellas se interactúa con las propiedades privadas, con esto conseguimos encapsular el código y hacer accesible solo aquello que queramos.

Una vez está construida la clase ya se pueden instanciar objetos de la misma.

VB.NET

```
Dim objBicicleta as Bicicleta = New Bicicleta  
  
Dim VelocidadActual as Integer  
  
objBicicleta.Modelo = "Montaña"  
objBicicleta.Precio = 200  
objBicicleta.NumeroDeVelocidades = 21  
  
objBicicleta.Acelerar(5)  
objBicicleta.Frenar()  
  
VelocidadActual = objBicicleta.ConsultarVelocidad
```

C#

```
Bicicleta objBicicleta = new Bicicleta();  
  
int VelocidadActual;  
  
objBicicleta.Modelo = "Montaña";  
objBicicleta.Precio = 200;  
objBicicleta.NumeroDeVelocidades = 21;  
  
objBicicleta.Acelerar(5);  
objBicicleta.Frenar();  
  
VelocidadActual = objBicicleta.ConsultarVelocidad();
```

Tras la creación del objeto objBicicleta a partir de la clase, se pueden modificar los valores de las propiedades de tipo Public, y llamar a los métodos de tipo Public.

En el ejemplo se llama a los métodos Acelerar(5), pasándole el numero de km que queremos acelerar a través del parámetro "km" que está definido en la subrutina.

Luego se llama al método Frenar() que decrementa en una unidad el valor de la propiedad Velocidad.

Por último se usa la función ConsultarVelocidad(), que retorna el valor de la propiedad Velocidad para introducirlo en la variable VelocidadActual.

Artículo por Francisco Recio y David Provencio

Aplicaciones de Consola

Realizamos nuestros primeros programas en .NET, utilizando la consola, que utiliza un formato de salida y entrada de datos en modo texto.

Se puede definir una aplicación de consola como aquella que se ejecuta en una ventana de MS-DOS, es decir, en línea de comandos.

Lo más común dentro del desarrollo bajo la plataforma .Net es la creación de aplicaciones Web o aplicaciones Windows sin embargo la mejor forma de sentar unas bases firmes acerca de la programación orientada a objetos es comenzar construyendo aplicaciones sencillas de consola.

Nota: Para los ejemplos descritos en este artículo usaremos el entorno de desarrollo facilitado por Micosoft, Visual Studio.Net.

El primer ejemplo de aplicación de consola es un sencillo programa que pide al usuario 2 números y le pregunta si desea sumarlos o restarlos.

Antes de comenzar a desarrollar la aplicación se ha de conocer la clase principal que interactua con la consola de MS-DOS, la clase Console.

Mediante esta clase se consigue mostrar información en la pantalla así como capturar la información que introduzca el usuario, cabe destacar que los métodos de la clase Console son de tipo Shared, esto significa que no es necesario crear un objeto a partir de la clase para invocar a sus métodos, es posible hacerlo indicando el nombre de la clase seguido de un punto y el nombre del método.

El método WriteLine()

Este método es el que se usa para mostrar texto en la consola, el método escribe en la pantalla el valor que le pasemos como parámetro.

El parámetro que recibe el método puede ser de varios tipos, ya sea una cadena de caracteres, un número entero, una línea en blanco, etc...

VB.NET

```

Module Module1
Sub Main()
    'Escribimos una cadena de caracteres.
    Console.WriteLine("Escribiendo una línea en la consola")
    'Escribimos un numero entero
    Console.WriteLine(23)
    'Escribimos una comparación lógica
    Console.WriteLine(3 > 1)
    Console.ReadLine()
End Sub
End Module
    
```

C#

```

using System;

namespace ConsoleApplication2
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            //Escribimos una cadena de caracteres.
            Console.WriteLine("Escribiendo una línea en la consola");
            //Escribimos un numero entero
            Console.WriteLine(23);
            //Escribimos una comparación lógica
            Console.WriteLine(3 > 1);
            Console.ReadLine();
        }
    }
}
    
```

Es importante destacar que este método añade automáticamente el salto de carro al final de la línea, esto significa que la siguiente llamada a `Console.WriteLine()` escribe en la siguiente línea.

La última línea en la que realizamos una llamada al método `ReadLine()` se utiliza para evitar que la pantalla se cierre automáticamente.

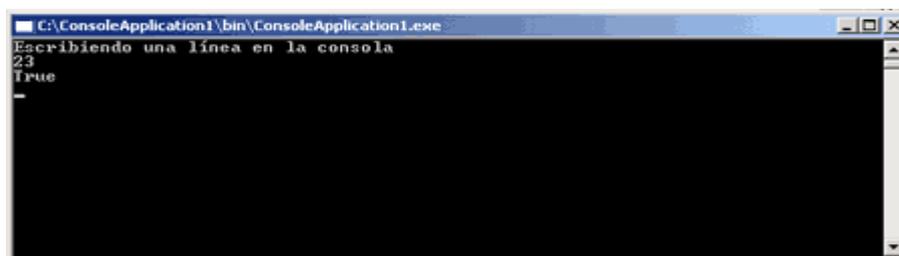


Fig1: Ejemplo del método `WriteLine()`

El método ReadLine()

Este método se usa para recoger la información que el usuario introduce cuando la aplicación así lo requiera. Cuando invocamos al método `Console.ReadLine()` el sistema queda en espera hasta que el usuario pulsa la tecla Intro.

Si se asigna la llamada a `Console.ReadLine()` a una variable se consigue capturar el dato introducido por el usuario, para después poder operar con él.

VB.NET

```
'Declaramos una variable de tipo cadena de caracteres
Dim cadena As String
'Mostramos un mensaje al usuario
Console.WriteLine("Por favor, introduzca su nombre:")
'Capturamos el dato introducido por el usuario
cadena = Console.ReadLine()
'Operamos con el dato
cadena = "El nombre introducido es: " & cadena
'Mostramos la cadena
Console.WriteLine(cadena)
Console.ReadLine()
```

C#

```
//Declaramos una variable de tipo cadena de caracteres
string cadena;
//Mostramos un mensaje al usuario
Console.WriteLine("Por favor, introduzca su nombre:");
//Capturamos el dato introducido por el usuario
cadena = Console.ReadLine();
//Operamos con el dato
cadena = "El nombre introducido es: " + cadena;
//Mostramos la cadena
Console.WriteLine(cadena);
Console.ReadLine();
```

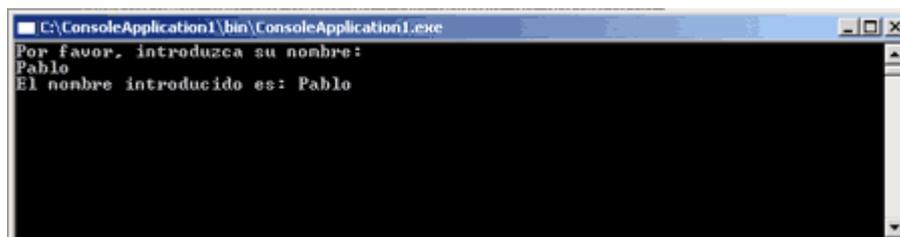


Fig2: WriteLine() y ReadLine()

El método Read()

Este es otro método que permite capturar información que proviene del usuario. La diferencia con el anterior es que `Read()` no espera a que el usuario pulse intro para capturar el dato introducido, sino que lo hace tras la pulsación de cualquier tecla, capturando el valor de la tecla pulsada en código ASCII.

Artículo por *Francisco Recio* y *David Provencio*

Ejemplo de aplicación de consola

Una vez que hemos aprendido a usar la clase consola, vamos a realizar una aplicación de consola.

Ahora que se conoce un poco mejor la clase Console, se dará comienzo a la aplicación, los lenguajes usados para este ejemplo son Visual Basic.Net y C#.

Lo primero que se debe hacer después de arrancar Visual Studio.Net, es escoger la opción "Aplicación de consola" (Fig1), tras este paso Visual Studio genera las siguientes líneas:

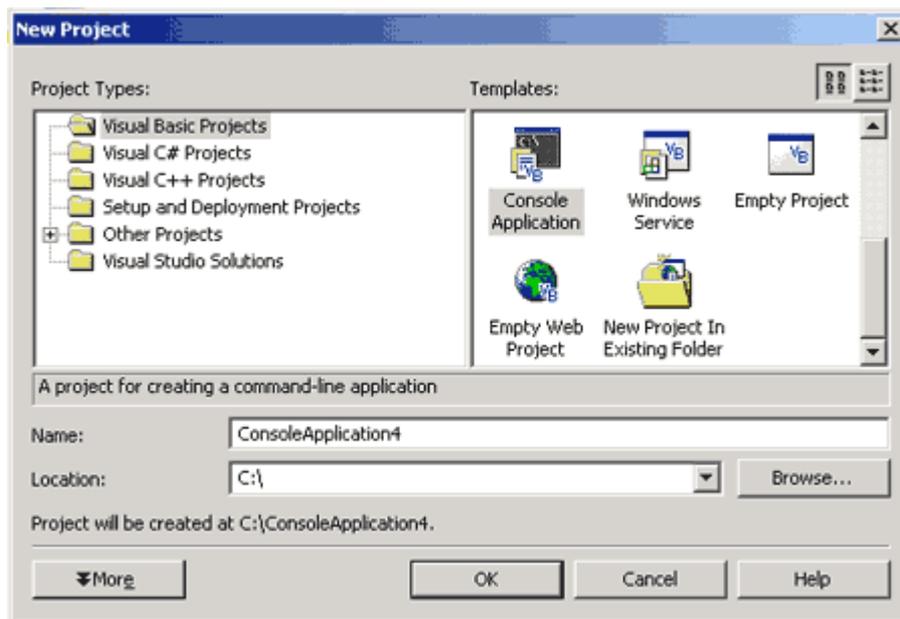


Fig3: Creación de una aplicación de consola.

VB.NET

```
Module Module1
    Sub Main()

        End Sub
End Module
```

C#

```
using System;

namespace ConsoleApplication3
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            //
        }
    }
}
```

Dentro del procedimiento Main(), se introduce el código que se quiere ejecutar. Lo primero que hay que hacer es declarar las variables que se van a usar, para este ejemplo se usan 2 variables de tipo entero para recoger los valores de los números que introduzca el usuario:

VB.NET

```
Module Module1
    Sub Main()
        Dim Numero1 As Integer
        Dim NUmero2 As Integer
    End Sub
End Module
```

C#

```
using System;

namespace ConsoleApplication3
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            int Numero1;
            int Numero2;
        }
    }
}
```

```
}
```

Una vez están declaradas las variables, se solicitan ambos números al usuario y se introducen sus valores en las dos variables, a continuación se pide que se seleccione una de las opciones posibles, sumar o restar.

VB.NET

```
Module Module1
    Sub Main()
        Dim Numero1 As Integer
        Dim Numero2 As Integer

        Console.WriteLine("Introduzca el primer número")
        Numero1 = Console.ReadLine()

        Console.WriteLine("Introduzca el segundo número")
        Numero2 = Console.ReadLine()

        Console.WriteLine("Escoja una opción: 1 - Sumar / 2 - Restar")
        If (Console.ReadLine = 1) Then
            Console.WriteLine("El resultado de la suma es: " & Numero1 + Numero2)
            Console.ReadLine()
        ElseIf (Console.ReadLine = 2) Then
            Console.WriteLine("El resultado de la resta es: " & Numero1 - Numero2)
            Console.ReadLine()
        Else
            Console.WriteLine("Opción Incorrecta")
        End If
    End Sub
End Module
```

C#

```
using System;

namespace ConsoleApplication2
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            int Numero1;
            int Numero2;
            int opcion;

            Console.WriteLine("Introduzca el primer número");
            Numero1 = Int32.Parse(Console.ReadLine());

            Console.WriteLine("Introduzca el segundo número");
            Numero2 = Int32.Parse(Console.ReadLine());

            Console.WriteLine("Escoja una opción: 1 - Sumar / 2 - Restar");
            opcion = Int32.Parse(Console.ReadLine());
            if (opcion == 1)
            {
                Console.WriteLine("El resultado de la suma es: " + (Numero1 + Numero2));
            }
            else if (opcion == 2)
            {
                Console.WriteLine("El resultado de la resta es: " + (Numero1 - Numero2));
            }
            else
            {
            }
        }
    }
}
```

```
        {
            Console.WriteLine("Opción Incorrecta");
        }
        Console.ReadLine();
    }
}
```

Siguiendo el flujo de la aplicación se puede ver que después de que el usuario selecciona una de las 2 opciones, se muestra el resultado de la operación por pantalla.

Si lo que se desea es encapsular el código en la medida de lo posible se pueden construir 2 funciones que realicen las operaciones de sumar y restar y que escriben el resultado en la pantalla, el código quedaría de la siguiente manera:

VB.NET

```
Module Module1
    Sub Main()
        Dim Numero1 As Integer
        Dim Numero2 As Integer

        Console.WriteLine("Introduzca el primer número")
        Numero1 = Console.ReadLine()

        Console.WriteLine("Introduzca el segundo número")
        Numero2 = Console.ReadLine()

        Console.WriteLine("Escoja una opción: 1 - Sumar / 2 - Restar")
        If (Console.ReadLine = 1) Then
            Sumar(Numero1, Numero2)
        ElseIf (Console.ReadLine = 2) Then
            Restar(Numero1, Numero2)
        Else
            Console.WriteLine("Opción Incorrecta")
        End If
    End Sub

    Sub Sumar(ByVal Numero1, ByVal Numero2)
        Console.WriteLine("El resultado de la suma es: " & Numero1 + Numero2)
        Console.ReadLine()
    End Sub

    Sub Restar(ByVal Numero1, ByVal Numero2)
        Console.WriteLine("El resultado de la resta es: " & Numero1 - Numero2)
        Console.ReadLine()
    End Sub
End Module
```

C#

```
using System;

namespace ConsoleApplication2
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            int Numero1;
            int Numero2;
            int opcion;
        }
    }
}
```

```

Console.WriteLine("Introduzca el primer número");
Numero1 = Int32.Parse(Console.ReadLine());

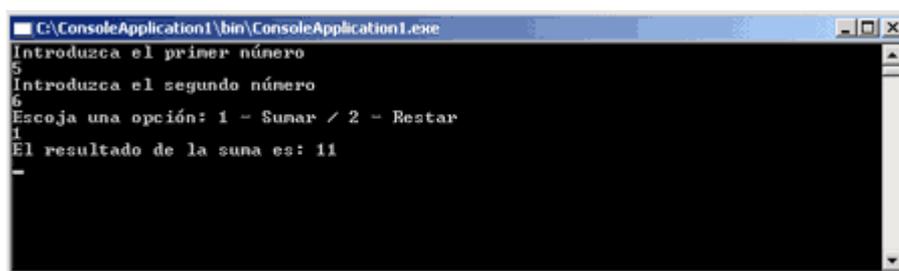
Console.WriteLine("Introduzca el segundo número");
Numero2 =Int32.Parse(Console.ReadLine());

Console.WriteLine("Escoja una opción: 1 - Sumar / 2 - Restar");
opcion = Int32.Parse(Console.ReadLine());
if (opcion == 1)
{
    Sumar(Numero1, Numero2);
}
else if (opcion == 2)
{
    Restar(Numero1, Numero2);
}
else
{
    Console.WriteLine("Opción Incorrecta");
}
Console.ReadLine();
}
static void Sumar (int Numero1, int Numero2)
{
    Console.WriteLine("El resultado de la suma es: " + (Numero1 + Numero2));
    Console.ReadLine();
}

static void Restar (int Numero1, int Numero2)
{
    Console.WriteLine("El resultado de la resta es: " + (Numero1 - Numero2));
    Console.ReadLine();
}
}
    
```

De esta manera se consigue encapsular funcionalidades dentro de la aplicación, asignando las tareas de Sumar y Restar a dos subrutinas, la principal ventaja es que una vez hayamos asegurado que ambas subrutinas funcionan, podremos olvidarnos de ellas y continuar con el desarrollo de la aplicación.

El resultado de la ejecución de cualquiera de los 2 códigos anteriores es el siguiente:



```

C:\ConsoleApplication1\bin\ConsoleApplication1.exe
Introduzca el primer número
5
Introduzca el segundo número
6
Escoja una opción: 1 - Sumar / 2 - Restar
1
El resultado de la suma es: 11
    
```

Fig4: Resultado del ejemplo.

Artículo por *Francisco Recio* y *David Provencio*

Creación de una aplicación Windows

Introducción a la creación en .NET de aplicaciones en entorno de ventanas, llamadas generalmente aplicaciones Windows.

La creación de aplicaciones Windows ha resultado siempre una tarea compleja debido a la dificultad de tener que crear una interfaz gráfica que interactúe con el usuario. Los *Formularios de Windows* (Windows Forms) de .Net permiten la creación de aplicaciones de interfaz gráfica de forma sencilla. .Net proporciona un amplio conjunto de controles como botones, cajas de texto, etiquetas, etc. que, unidos a la completa biblioteca de clases de .Net, hace posible el desarrollo de aplicaciones en poco tiempo.

En los siguientes ejemplos se ha usado Visual Studio.Net, no obstante, es posible crear aplicaciones Windows con un simple editor de texto y una herramienta de compilación compatible con el CLR de .Net Framework. Visual Studio.Net admite diseñar la aplicación de forma visual, permitiendo en cada momento acceder al código generado y sirviendo además como herramienta de compilación y depuración.

Para comenzar una nueva aplicación, se ejecuta Visual Studio y se selecciona Nuevo Proyecto, donde aparecen los distintos tipos de aplicaciones que se pueden realizar con cada lenguaje, seleccionando en este caso Aplicación para Windows. Una vez introducido el nombre de la aplicación y la ruta donde se ubicará se pulsa Aceptar.

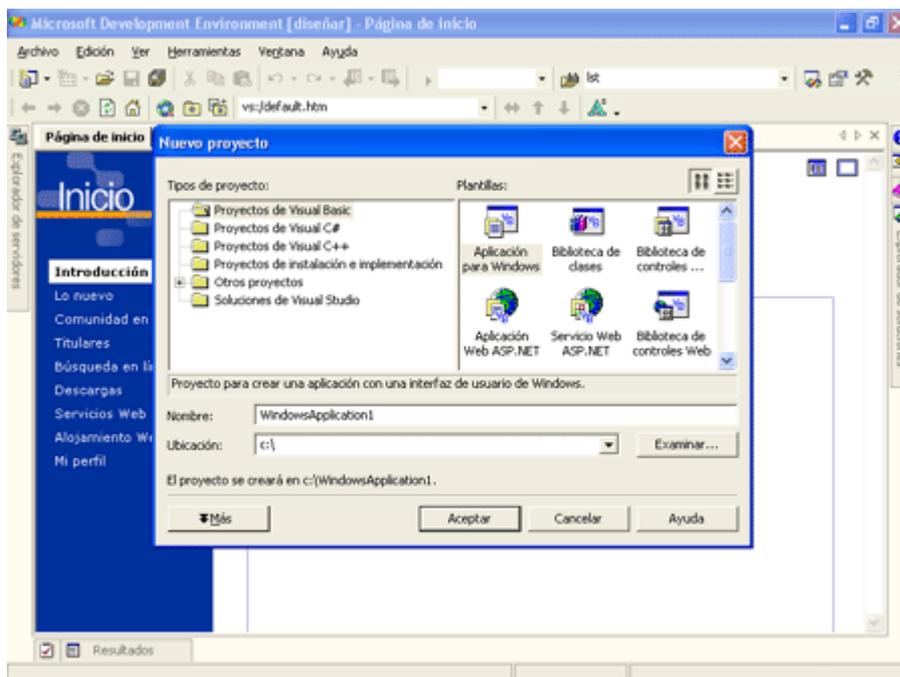


Fig 1. Ventana de creación de nuevo proyecto.

En los siguientes ejemplos se muestra como crear una aplicación basada en formularios en los dos lenguajes más comunes, tanto para Visual Basic .Net como para C#, dejando al lector la elección del lenguaje que le resulte más cercano.

Después de haber creado el proyecto, se dispone de una aplicación completa que puede ser ejecutada. Esto se puede realizar en el menú Depurar pulsando en el elemento Iniciar (Tecla F5) lo que ejecutará directamente la aplicación dentro de Visual Studio.Net. Véase el código creado hasta ahora:

VB.NET

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    + " Código generado por el Diseñador de Windows Forms "

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'escriba aquí su código.
    End Sub
End Class
```

Si se ha optado por C# se puede comprobar que el código es muy similar, aunque algo más extenso ya que la inicialización por defecto esta situada fuera de la región del código que va generando automáticamente el diseñador.

En el código generado, el formulario 'Form1' es una clase que proviene mediante la palabra clave inherits (heredar) del espacio de nombres System.Windows.Forms.Form perteneciente a la biblioteca de clases de .Net. Las aplicaciones desarrolladas con .Net utilizan la biblioteca de clases incluida en el Framework de .Net que proporcionan un conjunto de funcionalidades prefabricadas que facilitan el desarrollo. Esta biblioteca de clases está organizada en espacios de nombres dependiendo de su funcionalidad.

Todos los lenguajes incluidos en .Net están orientados a objetos, siguiendo esta metodología el formulario 'Form1' se declara como una clase. Como se verá más adelante esto facilita el acceso a los métodos y propiedades del formulario y de los controles que se incluyan al tratar a cada elemento como objetos independientes.

Artículo por Francisco Recio y David Provencio

Inserción de controles

Explicación sobre el proceso de insertar controles en una aplicación Windows. Los controles son elementos que podemos colocar en una aplicación para interactuar con el usuario o visualizar los resultados del programa.

Los controles simplifican la creación del interfaz facilitando además la interacción ordenada del usuario con la aplicación y la visualización de los resultados. Dentro de la región denominada "Código generado por el Diseñador de Windows Forms", el diseñador crea automáticamente el código correspondiente a cada control según se van añadiendo estos desde la pantalla de diseño visual. Por tanto, para insertar un nuevo control basta con arrastrarlo desde el cuadro de herramientas al formulario de la aplicación.

Arrastrando un control de tipo botón a nuestro formulario y analizando el código generado se observa lo siguiente:

VB.NET

```
Me.Button1 = New System.Windows.Forms.Button()  
,  
'Button1  
,  
Me.Button1.Location = New System.Drawing.Point(184, 64)  
Me.Button1.Name = "Button1"  
Me.Button1.TabIndex = 0  
Me.Button1.Text = "Button1"
```

C#

```
this.button1 = new System.Windows.Forms.Button();  
//  
// button1  
//  
this.button1.Location = new System.Drawing.Point(184, 88);  
this.button1.Name = "button1";  
this.button1.TabIndex = 0;  
this.button1.Text = "button1";
```

Al introducir un control de tipo botón se genera en el código un objeto llamado 'button1' perteneciente a la clase System.Windows.Forms.Button y se establecen las propiedades por defecto para ese objeto. Estas propiedades se pueden modificar desde la vista de diseño pulsando sobre el control 'button1' y seleccionado el panel de propiedades (Tecla F4).

Otra forma de cambiar las propiedades de un control es modificando el código generado automáticamente por el diseñador. A continuación se muestra un sencillo ejemplo donde se que modifica el literal de texto que aparece sobre el botón 'button1':

VB.NET

```
'load de la clase Form1  
Button1.Text = "Pulsa aquí"
```

C#

```
//load de la clase Form1  
button1.Text="Pulsa aquí";
```

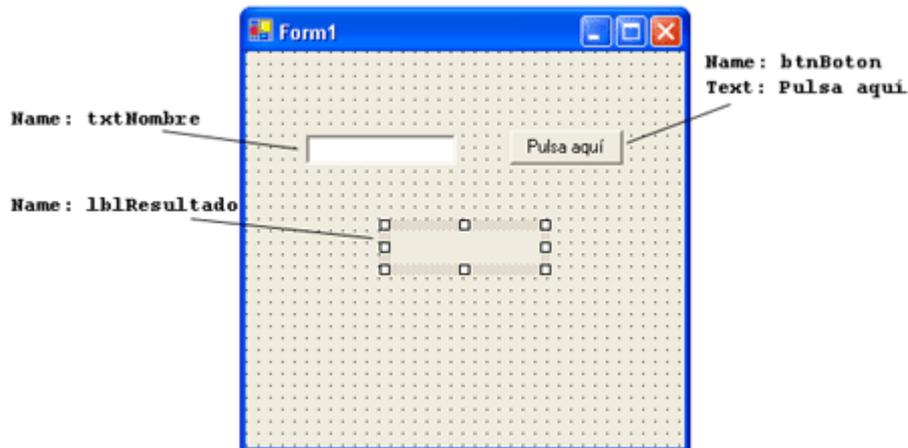
Artículo por *Francisco Recio* y *David Provencio*

Uso de eventos en .Net

Los eventos son llamadas al código que se producen cuando el usuario realiza una acción. Aprendemos a utilizarlos en .NET

Los eventos son llamadas al código que se producen cuando el usuario realiza una acción como, por ejemplo, pulsar un botón o seleccionar un elemento de una lista. Los eventos responden a la acción del usuario sobre un control ejecutando una función situada en el código.

A continuación se muestra un sencillo ejemplo en el que se introducen una serie de controles en la vista de diseño modificando las propiedades tal como aparecen en la imagen.



Al pulsar dos veces sobre el botón 'btnBoton' se crea automáticamente en el código del formulario el evento que se corresponde con la pulsación del botón. La función creada se denomina por defecto 'btnBoton_Click' y responde al evento btnBoton.Click que se indica por medio de la palabra clave handles (manejador). Seguidamente se muestra el código en el evento que interactúa con el resto de controles:

VB.NET

```
Private Sub btnBoton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnBoton.Click
    lblResultado.Text = "Hola " & txtNombre.Text
End Sub
```

C#

```
private void btnBoton_Click(object sender, System.EventArgs e)
{
    lblResultado.Text = "Hola " + txtNombre.Text;
}
```

Cada control tiene su propia colección de eventos. Para crear un nuevo evento se utiliza el menú desplegable de clases y su correspondiente menú de métodos situados en la parte superior del código.



Artículo por Francisco Recio y David Provencio

Interacción entre objetos y controles.

Las aplicaciones .NET utilizan objetos que podemos programar nosotros mismos. Aprendemos a trabajar conjuntamente con los controles de las aplicaciones Windows y los objetos que podamos haber programado.

La programación orientada a objetos proporciona un mejor ordenamiento y claridad del código. La forma de programar consiste en dividir el código en clases de objetos que posteriormente pueden ser reutilizados. Para más información se recomienda leer el artículo de 'Programación orientada a objetos, introducción' incluido en esta sección.

Al arrancar la aplicación, el punto de inicio del código se sitúa por defecto en el evento 'Load' de la clase 'Form1', si bien se puede cambiar dentro de las propiedades del proyecto desde el explorador de soluciones. El evento 'Load' es una subrutina que no devuelve ningún valor. En ella se pueden instanciar objetos de otras clases para ser utilizados posteriormente. El siguiente ejemplo muestra como crear una clase propia y la instanciación de un objeto de dicha clase:

VB.NET

```
Public Class MiClaseSumar

    Public resultado As Integer

    Public Function sumar(ByVal NumSuma1 As Integer, ByVal NumSuma2 As Integer) As Integer

        Return NumSuma1 + NumSuma2

    End Function

End Class
```

C#

```
public class MiClaseSumar
{
    public int resultado;

    public int sumar(int NumSuma1, int NumSuma2)
    {
        return NumSuma1 + NumSuma2;
    }
}
```

En el evento load de Form1 se crea un objeto de la clase y se llama al método sumar.

VB.NET

```
'load de la clase Form1
Me.Text = " Mi primera Aplicación con Windows Forms"

Dim objetoSuma As New MiClaseSumar()
objetoSuma.resultado = objetoSuma.sumar (10, 5) 'resultado = 15

lblResultado.Text = objetoSuma.resultado
```

C#

```
//load de la clase Form1
this.Text = " Mi primera Aplicación con Windows Forms";

MiClaseSumar objetoSuma = new WindowsApplication2.MiClaseSumar();
objetoSuma.resultado = objetoSuma.sumar(10,5); //resultado = 15

lblResultado.Text = objetoSuma.resultado.ToString();
```

Se puede observar como en C# es necesario convertir previamente el resultado. Esto se debe a que el resultado de la suma es de tipo numérico y no se puede asignar directamente al texto de la etiqueta, por lo que se debe convertir previamente a tipo texto. Esto no es necesario en Visual Basic .Net, donde la conversión de tipos no es obligatoria, aunque es recomendable para obtener un código más óptimo.

Con la programación orientada a objetos resulta más fácil programar aplicaciones ya que el código está mejor estructurado y resulta más sencillo de leer y modificar. Además, Visual Studio .Net facilita en gran medida la creación de aplicaciones con interfaz gráfico al generar automáticamente el código encargado de visualizar formularios y controles.

Recursos:

101 Visual Basic and C# Code Samples

<http://www.microsoft.com/downloads/details.aspx?FamilyID=08e3d5f8-033d-420b-a3b1-3074505c03f3&DisplayLang=en>

Artículo por Francisco Recio y David Provencio

Configuración de una aplicación de ASP.NET

Una aplicación de ASP.NET tiene opciones de configuración y administración. Aquí se verá como configurar y ejecutar una aplicación web de ASP.NET a partir de servidores web como Internet Information Server.

Existen diversos elementos de configuración y administración en una aplicación Web de ASP.Net. Este tipo de aplicaciones se compone de un conjunto de ficheros almacenados en una carpeta dentro del servidor Web.

Para ejecutar una aplicación Web de ASP.Net se necesita que el servidor Web sea compatible con ASP.Net. En este caso se va a utilizar IIS 5.0 (Internet Information Server) como servidor Web. El IIS es un componente de Windows incluido en las versiones profesionales de Windows 2000 y XP. Si no se tiene este componente, se debe proceder a su instalación mediante el icono de 'Agregar o quitar programas' en el panel de control y seleccionando 'Agregar o quitar componentes de Windows' en donde aparecerá el IIS para su instalación. El acceso al IIS se realiza mediante el icono de 'Servicios de Internet Information Server' situado en las 'Herramientas administrativas' dentro del panel de control.

El servidor Web IIS permite administrar las aplicaciones Web y comunicarse con los navegadores cliente mediante

protocolo http (protocolo de transferencia de hipertexto). El IIS también ofrece otros servicios de protocolo, como transferencia de archivos (FTP), servicio de correo electrónico (SMTP) y servicio de noticias (NNTP).

Con el clásico ASP 3.0 era suficiente con tener el IIS instalado en el servidor Web, ya que era el IIS el que directamente interpretaba el código ASP y enviaba la respuesta al cliente. Sin embargo, en ASP.Net se necesita que el servidor Web tenga instalado .Net Framework para poder procesar código de ASP.Net, como ocurre con cualquier otra aplicación de .Net. Es importante decir que los navegadores cliente que accedan a la aplicación Web no necesitan tener instalado IIS ni tampoco .Net Framework ya que es el servidor Web el que tiene que saber interpretar el código de ASP.Net.

Cuando se solicita una página de tipo .aspx (página de ASP.Net) el servidor Web de IIS envía la solicitud a .Net Framework que es quien realmente procesa la petición de la página. De esta forma, las aplicaciones Web de ASP.Net se benefician de todas las ventajas de ejecución de código en .Net Framework, ya que el código es compilado y ejecutado por .Net Framework y devuelto al IIS para que éste a su vez lo envíe al cliente.

Con ASP.Net también es posible tener código de ASP 3.0 dentro de páginas de ASP.Net, con la ventaja de que el código de ASP 3.0 también se compila junto con el código de ASP.Net aumentando el rendimiento del servidor Web.

A continuación, se muestra un ejemplo de los distintos ficheros que pueden existir en una aplicación Web de ASP.Net.



Fig. Aplicación Web de ASP.Net

Una vez creada la aplicación, el código de servidor se ensambla en un fichero .dll situado en la carpeta Bin de la aplicación Web. Por tanto, una vez realizada la compilación, los ficheros de código (.vb ó .cs) ya no son necesarios para la ejecución de la aplicación ya que están ensamblados en la dll y es aconsejable quitarlos del servidor para que no se pueda acceder desde el exterior a su contenido.

En resumen, para que funcione una aplicación Web de ASP.Net se debe tener en el Servidor Web lo siguiente:

- Tener instalado IIS 5.0 ó superior en el servidor Web y configurar un directorio virtual asociado a la aplicación Web.
 - Tener instalado en el servidor Web .Net Framework.
 - Los archivos .aspx correspondientes a las páginas Web.
- Un archivo de ensamblado (DLL) situado en la carpeta Bin de la aplicación Web, que contiene el código de servidor que necesitan las páginas aspx.
- Un archivo llamado Global.asax que sirve para el control general de la aplicación durante su ejecución.
- Un archivo llamado Web.config donde se establece la configuración de la aplicación. Aunque este fichero es opcional se necesita cuando se quieren establecer parámetros de configuración que no sean los de por defecto.
 - De manera adicional también puede aparecer en la carpeta Web otro tipo de archivos como:
 - Archivos .aspx (controles personalizados de usuario de ASP.Net)
 - Archivos .asmx (servicios Web XML de ASP.Net).
 - Páginas .htm ó .html (páginas Web estáticas)
 - Páginas .asp (páginas activas de servidor)
 - Archivos .css (hojas de estilo CSS, Cascade Style Sheet).
 - Documentos, imágenes, etc...

Para terminar, se va a crear una aplicación Web de tipo ASP.Net y a instalarla en un servidor Web con IIS. El primer paso es crear la aplicación Web, para ello se entra en Visual Studio .Net y en el menú 'Archivo' se selecciona 'Nuevo proyecto'. Aquí se debe elegir uno de los lenguajes disponibles y seleccionar 'Aplicación Web ASP.Net'.



Fig. Creación de una aplicación Web de ASP.Net

De forma automática, al crear un nuevo proyecto Web, Visual Studio .Net crea un directorio virtual en el IIS y lo asocia con la aplicación Web. Si se ha instalado IIS con la configuración por defecto, el sitio Web predeterminado (localhost) será 'c:\inetpub\wwwroot'.

En el caso de que se tuviera una aplicación Web de ASP.Net ya creada y se desee instalar en un servidor Web, se debe copiar la carpeta con la aplicación en el servidor Web y asociarla manualmente a un directorio virtual. Para ello, dentro de IIS se selecciona el elemento de 'Sitio Web predeterminado' y pulsando con el botón derecho se selecciona la opción: 'Nuevo' > 'Directorio virtual' donde mediante un asistente se asocia la carpeta de la aplicación Web a un directorio virtual en el servidor.



Fig. Creación de un directorio virtual en IIS

Para probar que la aplicación Web funciona correctamente se debe compilar primero en Visual Studio .Net y posteriormente acceder a la aplicación mediante el navegador:

`http://[Nombre_del_servidor]/[directorio_virtual]/[página]`

Por ejemplo, `http://localhost/MiWeb/webform1.aspx`

Artículo por Francisco Recio y David Provencio