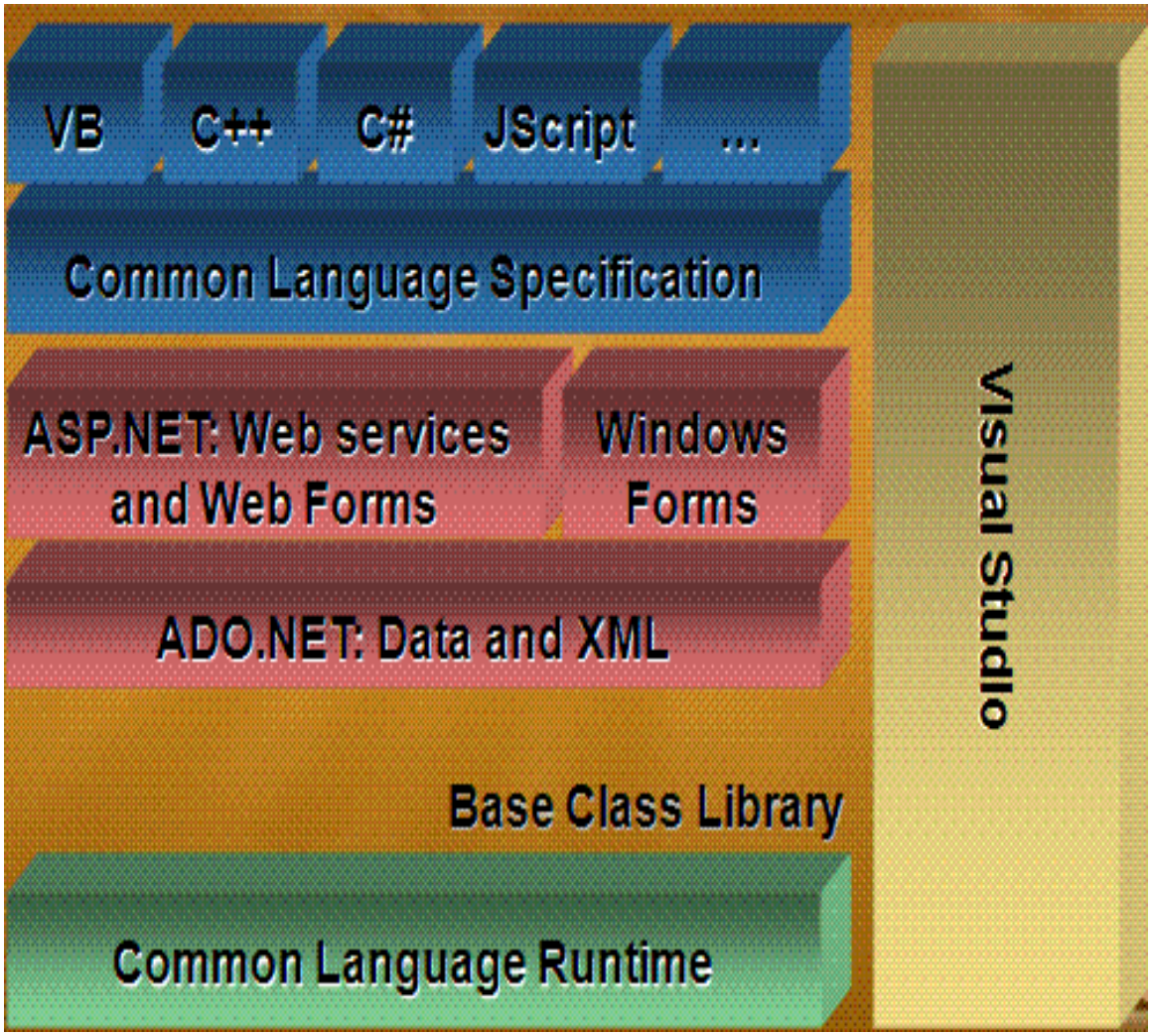
MASSIMO UBERTINI



VISUAL STUDIO

WWW.UBERTINI.IT

MODULO 1

.NET FRAMEWORK

La piattaforma .NET
Traduzione
Ambiente di sviluppo
Applicazioni
Guida in linea
Offuscatore
Distribuzione applicazioni
Progetto di modello
Mono
C/C++
MASM
MSIL

LA PIATTAFORMA .NET

CARATTERISTICHE

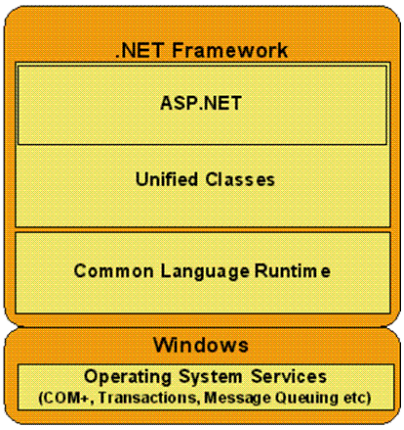
Modello di servizi web **XML** (*eXtensible Markup Language*).
Architettura componentizzata e accessibile a qualunque piattaforma via web service.
Interoperabilità e interlinguaggio.
Utilizzo di architetture fortemente e debolmente accoppiate.
Standard riconosciuti per linguaggi di programmazione e infrastruttura.
Nuovo modello di sicurezza.
Sistema di sviluppo unificato e standard, scalabile.
Pronto per dispositivi non strettamente informatici: cellulari,TV.
Elevato supporto per le interfacce grafiche lato client.

COM (*Component Object Model)*

Microsoft ha introdotto .NET con il suo nuovo paradigma di programmazione, per porre rimedio ad una serie di limiti e inadeguatezze insiti nello standard COM.
Il COM è una tecnologia specifica della famiglia di sistemi operativi Windows che abilita la comunicazione fra i componenti software del sistema, favorisce lo sviluppo di componenti riusabili, permette di creare applicazioni connettendo fra loro insiemi di componenti, si tratta di una famiglia di tecnologie che include COM+, DCOM, ActiveX.
Le applicazioni COM sono afflitte dal problema del controllo di versione, i progettisti non prevedevano la necessità d'installare più versioni sullo stesso sistema, infatti ogni nuova versione di una componente COM sovrascrive quella preesistente.
Questa sostituzione non dovrebbe comportare conseguenze negative in quanto si suppone che ogni nuova versione comprenda tutte le funzionalità delle precedenti: in pratica, però, ciò non è accaduto.
Ultimo difetto è nel **deployment**, la consegna al cliente, con relativa installazione e messa in funzione, di un'applicazione: occorre installare tutte le componenti necessarie nelle directory di sistema, registrarle nel registro di sistema, configurarle.
All'interno della piattaforma Windows è difficile far interagire parti di codice scritte con linguaggi differenti poiché ogni linguaggio possiede le proprie convenzioni.
Ad esempio, nel codice Visual Basic è possibile invocare una **DLL** (*Dynamic Link Library*) del C++, ma occorre fare attenzione agli interi privi di segno che non sono riconosciuti.
A causa di questo problema spesso si opta per la scelta di un unico linguaggio perdendo così i vantaggi che ogni linguaggio presenta.
Le **ASP** (*Active Server Page*) utilizzano solo linguaggi come VBScript e JScript.
Il codice ASP è unito a quello dell'interfaccia utente, ciò rende impossibile l'aggiornamento

delle interfacce utente a programmatori che non sono sviluppatori esperti.
Il debugging di un'applicazione ASP è un vero e proprio incubo poiché occorre essere molto abili e utilizzare molte istruzioni aggiuntive per scovare un errore.
È virtualmente impossibile riutilizzare il codice ASP su larga scala: l'unico modo è utilizzare i file d'*include*.
La maggior parte dei siti web in ASP memorizzano le informazioni sui singoli client connessi all'interno di variabili *Session* in quanto la programmazione ASP è priva di stato, in altre parole i valori non sono conservati tra successive richieste al server.

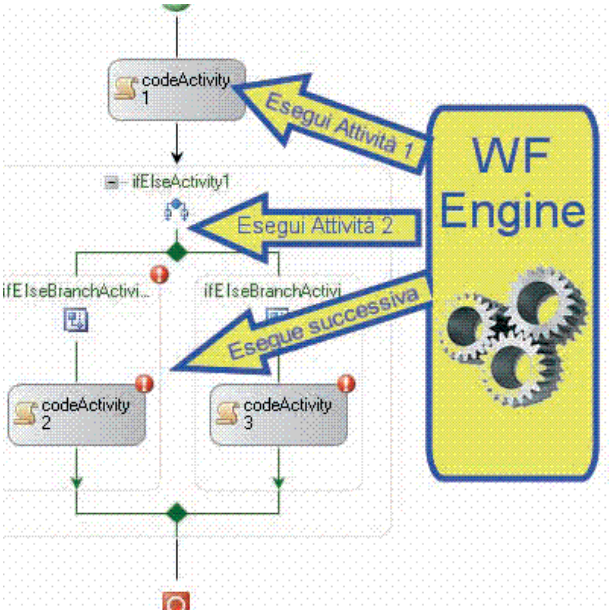
Soluzione
Il .NET Framework è stato progettato attorno al concetto di ereditarietà ciò che invece non è stato per COM, tutti gli oggetti del .NET Framework costituiscono una gerarchia con un'unica radice, la classe *System.Object*, da cui derivano tutte le altre classi.
Queste forniscono funzionalità praticamente in tutte le aree possibili e immaginabili, interfaccia utente, accesso ai dati, programmazione per Internet.
Forte rispetto degli standard ed è per questo che, a fondamento di tutta questa visione, si ritrovano protocolli standard di Internet: **SOAP**, (*Simple Object Access Protocol*), **HTTP** (*Hyper Text Transfer Protocol***)**, **SMTP** (*Simple Mail Transfer Protocol***)**, XML, **FTP** (*File Transfer Protocol***)**.
Programmare sotto .NET significa estendere una di queste classi.



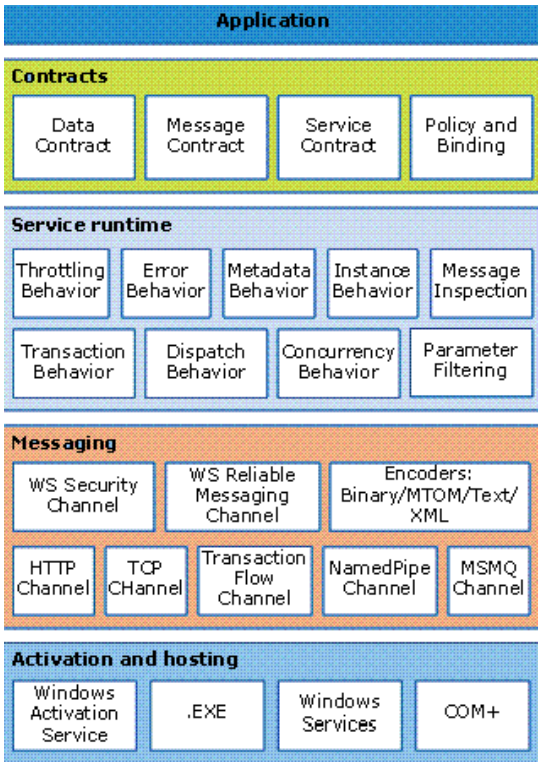
ARCHITETTURA

L'architetto di .NET è Anders Hejlsberg (Copenaghen, 1961) che ha realizzato in Borland TurboPascal e Delphi e in Microsoft Visual J++ (1997).

WF (*Windows Workflow Foundation*)
Un workflow è un concetto semplice: una sequenza di attività eseguite in un dato ordine.
WF fornisce una tecnologia comune alle applicazioni che hanno la necessità d'implementare tale logica.
Dato che esistono e possono esistere decine, centinaia o migliaia di diversi workflow con altrettanti diversi requisiti, realizzare una tecnologia come WF, che possa gestirli tutti, ha richiesto un certo sforzo di astrazione.
La soluzione è stata quella di considerare il concetto più generale possibile di workflow, in tal modo un workflow di WF è un insieme di attività eseguite da un motore.
Ogni attività è rappresentata da una classe che contiene il codice necessario ad eseguirla.
Le attività possono così essere riutilizzate in workflow diversi.
Visual Studio con le estensioni per WF consente di creare un workflow in maniera grafica e generare un corrispondente codice in un nuovo linguaggio di markup, **XOML** (*Extensible Object Markup Language*), oppure, se si preferisce direttamente in codice sorgente C#.



WCF (*Windows Communication Foundation*)
Qualunque sia il tipo di applicazione da sviluppare, alcune volte c'è la necessità di stabilire una comunicazione fra diverse applicazioni.
Microsoft introduce un'interfaccia di programmazione comune per ogni tipo di comunicazione, in pratica è un run-time e un insieme di **API** (*Application Programming Interface*)per creare sistemi che spediscono messaggi fra servizi e client, il messaggio.



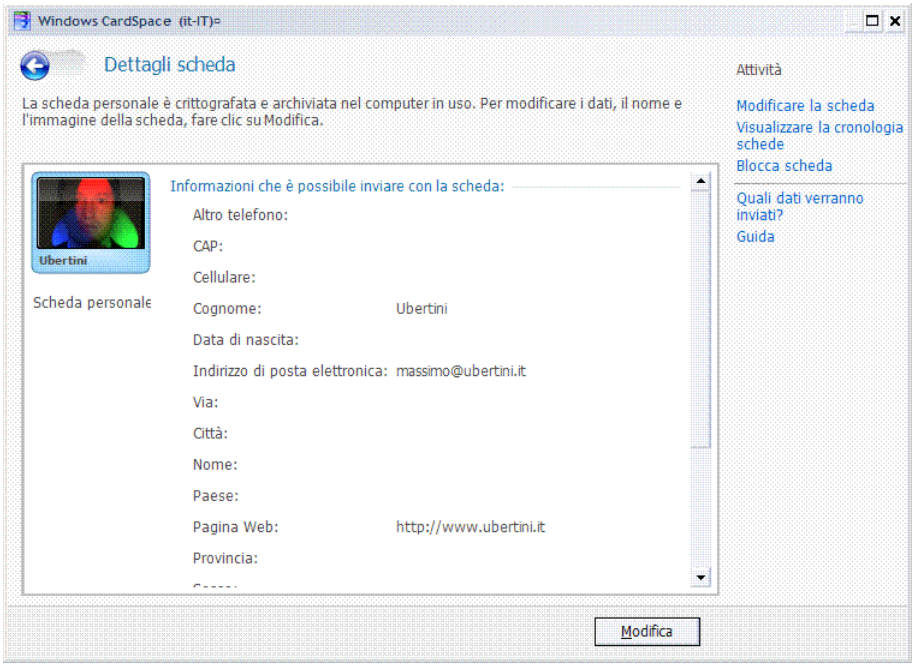
I ***Contracts*** definiscono i vari aspetti del sistema di messaggistica, come si può comunicare con esso, in altre parole quali operazioni si possono chiamare e quali parametri possono essere passati.
Lo strato ***Service Runtime*** contiene i diversi behaviour, in pratica definisce il comportamento del servizio nelle varie situazioni possibili durante il suo ciclo di vita, quanti messaggi possono essere processati, cosa succede in caso di errore, come e quando i messaggi sono processati.
Lo strato di ***Messaging*** è composto da canali, sono i componenti che processano i

messaggi, esistono due tipologie di canali.
1. Trasporto si occupano della lettura e della scrittura dei messaggi sulla rete.
2. Protocollo: si occupano dell'interpretazione dei protocolli di comunicazione.

Lo strato ***Activation and hosting*** supporta diversi tipi di comunicazione a messaggi, per default tutte del tipo Request-Reply, ma con la possibilità di utilizzarle in diverse maniere. Ad esempio, un client deve solo invocare un'operazione remota e non gli importa del risultato, quindi basta una semplice comunicazione OneWay, altre volte è necessaria invece un'infrastruttura in cui il servizio possa invocare una funzione del chiamante e si ha l'operazione Callback, o ancora possono essere necessarie comunicazioni con sottoscrizioni di eventi o di tipo publish-subscribe.

WCS (*Windows CardSpace*)

Quando gli utenti di un'applicazione, sia essa web o Windows, accedono ad essa, si ha un trasferimento della loro identità digitale, che può essere costituita dal loro nome utente, dalla password o di altre informazioni sensibili, con conseguenti problemi di sicurezza.
WCS si occupa del trattamento delle diverse identità digitali degli utenti, è un sistema per creare delle relazioni fra un sito web e l'utente.
WCS fornisce un modo per permettere a tali servizi di richiedere le informazioni sull'utente, all'utente di essere sicuro dell'identità del sito, di gestire le informazioni per mezzo di card e d'inviarle a tali servizi, senza la necessità di tenere a mente decine di nomi utenti, di password, di codici di accesso.
Basta creare una nuova Card, dare un nome associabile ad un dato servizio e si potrà utilizzare questa per accedere al servizio stesso ogni volta che ce n'è bisogno.



WPF (*Windows Presentation Foundation*)

Qualunque sia la complessità di un'applicazione, sia essa un'applicazione basata su Workflow, su Servizi, quello che l'utente vede è l'interfaccia utente.
WPF è la tecnologia destinata alla creazione d'interface grafiche evolute e coerenti anche in diversi ambienti e rende più facile la vita agli sviluppatori perché offre una varietà di componenti grafici, con supporto per video, animazioni, grafica 2D e 3D.
L'innovazione principale è forse quella di poter usare grafica vettoriale e creare contenuti più facilmente fruibili dagli utenti, su diverse piattaforme.
Per esempio, il problema del ridimensionamento di una finestra su monitor con diverse risoluzioni: con grafica vettoriale il problema non si pone.
Tutto questo è realizzabile grazie ad un nuovo linguaggio di markup, chiamato **XAML**

(*Extensible Application Markup Lan*guage), derivato da XML, con cui sono costruiti e definiti i building block delle interfacce grafiche.

CLR (*COMMON LANGUAGE RUNTIME*)

È l'implementazione Microsoft di un insieme di specifiche note come **CLI** (*Common Language Infrastructure*) che sono state standardizzate da **ECMA** (*European Computer Manufacturers Association*) ECMA-334, ECMA-335 nel dicembre del 2001 per creare un'architettura aperta.
Esistono già altre implementazioni di CLI.

- ✓ SSCLI Microsoft per Windows, FreeBSD e Macintosh.
- ✓ Mono per Linux.
- ✓ DotGNU.
- ✓ INTEL **OCL** (*Open CLI Library*).

Motore di esecuzione ad elevate prestazioni.

- ✓ Gestione della memoria e **GC** (*Garbage Collector*).
- ✓ Gestione dei thread e dei servizi del sistema operativo.
- ✓ Gestione della sicurezza.
- ✓ Gestione automatica delle dipendenze da altre componenti.
- ✓ Compilazione **JIT** (*Just In Time*) di tutto il codice.

Strumento di sviluppo.

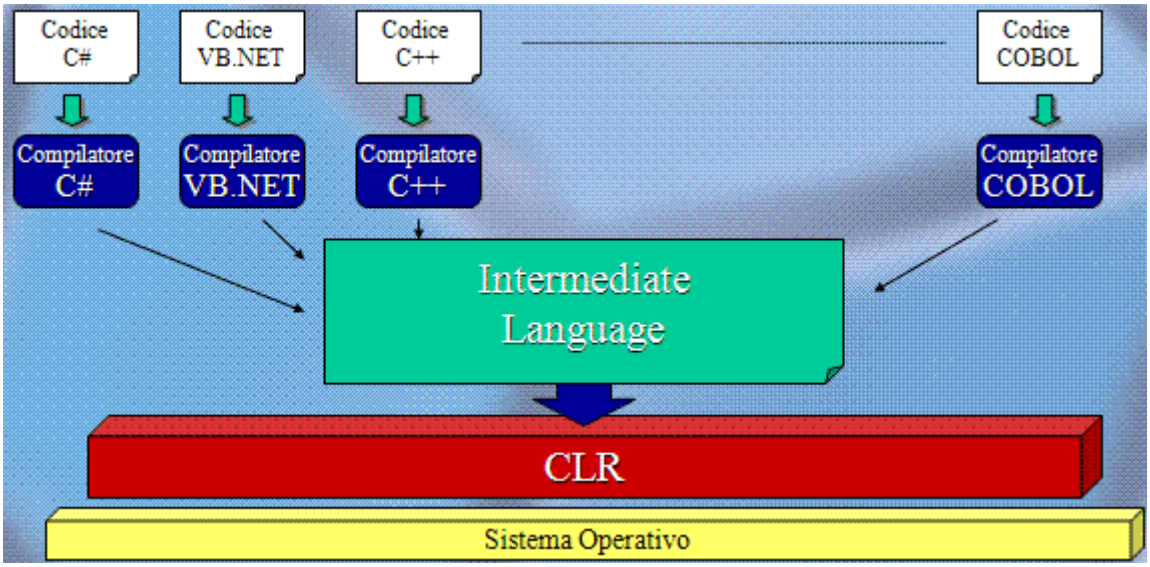
- ✓ Controllo sui tipi.
- ✓ Gestione delle eccezioni interlinguaggio.
- ✓ Accesso facilitato a servizi avanzati.
- ✓ Ambiente di debug unificato per tutti i linguaggi conformi.
- ✓ Linguaggi Microsoft disponibili: C/C++, C#, VB.NET, F#, Axum.

Nel caso di Java il binario si chiama bytecode e l'interprete è l'**JVM** (*Java Virtual Machine*), mentre nel caso .NET il binario si chiama assembly e l'interprete è il CLR.

Managed Code, codice gestito.
Tutto il codice aderente alle specifiche del CLR del quale può sfruttare i servizi perché è codice "sicuro".

Unmanaged Code, codice non gestito.
Tutto il resto, codice "insicuro" perché scavalca il CLR.

Il CLR è composto da cinque componenti.



1. CTS (*COMMON TYPE SYSTEM*)

Sistema di tipi unificato ed interlinguaggio.
Un insieme standard di tipi di dato e di regole necessarie per la realizzazione di nuovi tipi.
Tutto è un oggetto, due categorie di tipi.

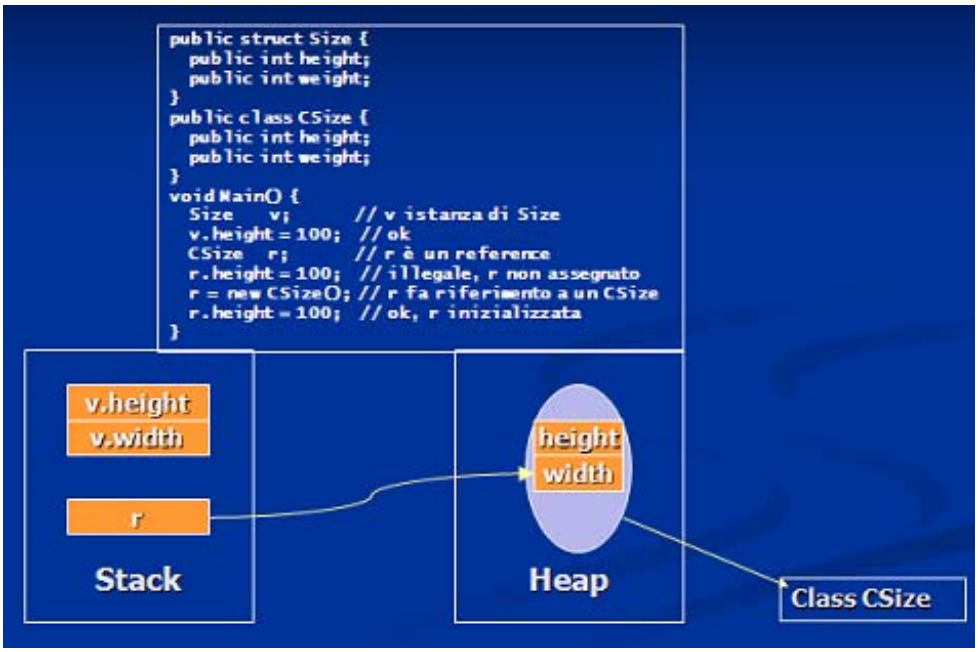
1. **RT** (*Reference Type*).

Contengono solo un reference, un pointer, ad un oggetto.
La copia di un tipo *reference* implica la duplicazione del solo *reference*.
Le modifiche su due *reference* modificheranno l'oggetto cui puntano.
Il *reference* che non referenzia nessuna istanza vale *null*.
I tipi *reference* sono tutte le classi, le stringhe, gli *objects*.
Risiedono in memoria in un'area chiamata **managed heap**.

2. **VT** (*Value Type*).

Contengono i dati direttamente, ereditano le caratteristiche da *System.ValueType.*
Una copia di un VT implica la copia dei dati in esso contenuti.
Le modificazioni hanno effetto solo su un'istanza: in pratica risiedono in memoria in un'area chiamata **stack**.
Deve sempre assumere un valore *null* non direttamente ammesso.
Conversione: tutti i VT possono essere visti come tipi reference (Boxing e Unboxing).
Tra i VT ci sono: *int*, *byte*, *enum*, *struct* e tutti i tipi base.

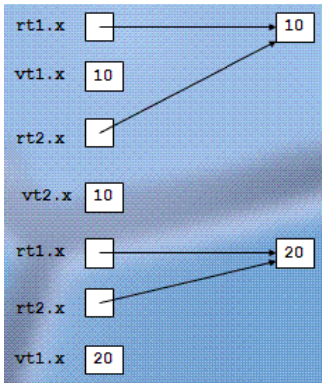
RT e VT in memoria



Esempio.

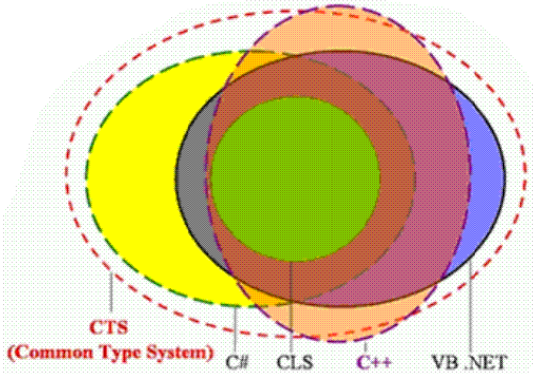
```
using System;
// è un tipo Reference (a causa del 'class')
class  RefType { public int x; }
// è un tipo Value (a causa del 'struct')
struct ValType { public int x; }
class MyClass
{
   static void Main(string[] args)
   {      RefType rt1 = new RefType();  // Allocato nell'heap
           ValType vt1 = new ValType();  // Allocato nello stack
          rt1.x = 10;              // Cambia il riferimento a cui punta
```

```
            vt1.x = 10;                 // Cambiato il valore nello stack
            RefType rt2 = rt1;          // Copia il solo puntatore
            ValType vt2 = vt1;          // Alloca nello stack e copia i membri
            rt1.x = 20;
            vt1.x = 20;
            Console.WriteLine("rt1.x = {0}, rt2.x = {1}", rt1.x, rt2.x);
            Console.WriteLine("vt1.x = {0}, vt2.x = {1}", vt1.x, vt2.x);
            Console.ReadKey();
        }
}
rt1.x = 20, rt2.x = 20
vt1.x = 20, vt2.x = 10
```



2. CLS (*COMMON LANGUAGE SPECIFICATION*)

Definisce un sotto insieme del CTS al quale tutti i fornitori di librerie di classi e progettisti di linguaggi che puntano al CLR, devono aderire.
Se un componente scritto in un linguaggio, ad esempio C#, dovrà essere utilizzato da un altro linguaggio, ad esempio VB.NET, allora chi scrive il componente dovrà aderire ai tipi e alle strutture definite dal CLS.
Ad esempio, il tipo *Int32* è compatibile con il CLS e i linguaggi e gli strumenti possono aspettarsi che altri linguaggi e strumenti conformi al CLS sappiano come utilizzarlo correttamente.
CLS Framework: una libreria costituita da codice aderente al CLS.
CLS Consumer: un linguaggio o tool di sviluppo progettato per accedere a tutte le caratteristiche fornite dai CLS Framework, ma non necessariamente in grado di produrne di nuove..
CLS Extender: superset del CLS Consumer.
Le CLS definiscono le specifiche comuni a tutti i linguaggi .NET in modo che gli assembly prodotti siano usabili da tutti i linguaggi: vantaggio è la compatibilità binaria.
CTS definisce come devono essere costruiti e definiti i tipi: visibilità, contenuto in termini di proprietà, metodi.



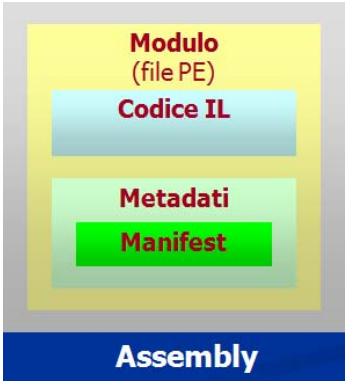
Ciò che nessuno di questi linguaggi è autorizzato a fare è implementare un proprio insieme di tipi, perché il concetto di compatibilità è stato ridefinito nel .NET Framework e non è più un concetto di compatibilità binario come avveniva in COM, ma è un concetto diverso, è un concetto di compatibilità a livello di tipo, nessuno di questi linguaggi implementa un proprio insieme di tipi, ma tutti quando hanno bisogno di una stringa, di una struttura, di una classe, chiedono al .NET Framework.
Il vantaggio è che la rappresentazione in memoria di questa entità è la stessa per tutti i linguaggi e ciò consente d'implementare tecniche estremamente sofisticate.
Visual Studio è volutamente rappresentato come un'entità separata rispetto al .NET Framework in quanto è uno strumento di produttività sopra ai servizi del .NET Framework, una sorta di grande wizard sui servizi messi a disposizione dal .NET Framework.

3. IL (*INTERMEDIATE LANGUAGE*)

Standard ECMA del 1997, tutti i compilatori che aderiscono alla struttura del CLR devono generare una rappresentazione intermedia del codice, indipendente dalla CPU.
Il run-time utilizza questo linguaggio intermedio per generare codice nativo oppure è eseguito al volo mediante la compilazione JIT.
Presenta similitudini con linguaggi ad alto livello ma anche con il linguaggio assembly.
Istruzioni di caricamento, memorizzazione e inizializzazione dei dati.
Istruzioni per richiamare metodi da oggetti.
Istruzioni aritmetiche e logiche.
Istruzioni per la gestione di eccezioni di tipo *Try .. Catch*.
Operazioni sui registri, ma indipendente dalla piattaforma.
Permette al CLR controlli durante la compilazione.

- ✓ Codice type safe.
- ✓ Puntatori corretti.
- ✓ Conversioni corrette.

Di fatto rappresenta il linguaggio a livello più basso e l'unico "eseguibile" dal CLR.
Un compilatore conforme al CLS produce due file.
Codice IL: rappresenta l'applicazione vera e propria.
Metadati: descrivono i tipi specifici appartenenti al **CLT** (*Common Language Types*) utilizzati nel codice, comprendente la definizione di ogni tipo, le signature per ogni membro del tipo, i membri ai quali il codice fa riferimento e gli altri dati che il run-time usa durante l'esecuzione, permettono componenti autodescrittivi.
IL e metadati sono alla fine contenuti in uno o più file **PE** (*Portable Executable*) nella forma tradizionale: EXE se è un'applicazione, DLL se è una libreria.



Metadati
Sono organizzati in tabelle, in cui fondamentalmente è descritto ciò che il codice definisce e cui fa riferimento, in pratica sono le informazioni sui tipi di un assembly.
Generati automaticamente dai compilatori.

Estendibili da terze parti.
Descrizione di un assembly
Identità: nome, versione, cultura [,pubblic key], tipi esportati, assembly da cui dipende.
Descrizione dei tipi
Nome, visibilità, classe base, interfacce implementate.
Attributi custom
Definiti dall'utente definiti dal compilatore.
La chiave per un modello di programmazione più semplice.
Generati automaticamente: memorizzati con il codice nel file eseguibile (.DLL o .EXE); utilizza il formato COFF esistente; memorizzati in formato binario.
Convertibili in/da type library COM.
Formato binario rappresentabile con XML schema **XSD** (*XML Schema Definition*).
Serializzazione e deserializzazione oggetti a run-time in XML.
L'assembly elenca al proprio interno tutti i tipi esportati ed importati relativi all'assembly stesso, ciò grazie ad una nuova definizione del concetto di metadata.
I metadati, vale a dire le informazioni per il corretto byning ad una componente COM, sono sparsi sul sistema, in parte sono all'interno del registro e in parte all'interno della stessa componente COM.
Nel mondo del .NET Framework tutto ciò cambia e durante la fase di compilazione è raccolta una quantità d'informazioni molto ampia che è portata in un formato binario ed inserita in un'opportuna area di un'assembly detta manifest che è, all'interno degli assembly, il contenitore dei metadati.
Cosa contengono i metadati?

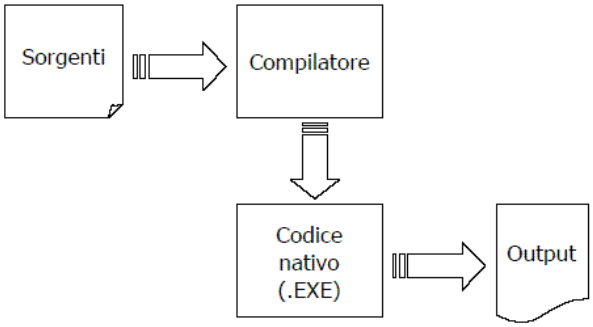
1. Descrizione dell'unità di deployment (assembly)
   1.1. Identità: nome, versione, lingua [, chiave pubblica]
   1.2. Elenco dei tipi esportati
   1.3. Elenco delle dipendenze da altri assembly
   1.4. Le impostazioni di sicurezza necessarie all'esecuzione
2. Descrizione dei tipi
   2.1. Nome, visibilità, classi di base, interfacce implementate
   2.2. Membri (metodi, campi, proprietà, eventi, tipi annidati)
3. Attributi personalizzati
   3.1. Definiti dall'utente
   3.2. Definiti dal compilatore
   3.3. Definiti dal framework

I metadati contengono informazioni importanti ed è comunque possibile estendere la quantità di informazioni presenti negli stessi, creando degli attributi personalizzati, quindi delle informazioni specifiche per il corretto funzionamento di una particolare applicazione e anche queste informazioni possono essere portate all'interno di un'assembly.

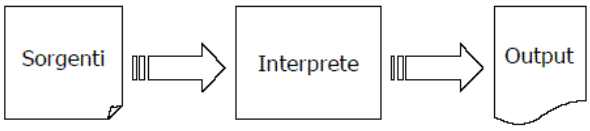
4. JIT COMPILER

Poiché all'interno del mondo del .NET Framework nulla gira in interpretato, ma tutto è compilato, è evidente che questa rappresentazione non dipendente dalla CPU, dovrà essere soggetta ad una fase di compilazione JIT, prima di poter essere mandata in esecuzione.

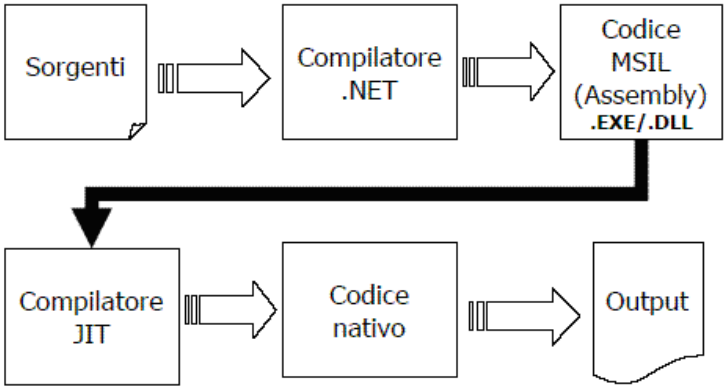
Codice nativo



Codice interpretato

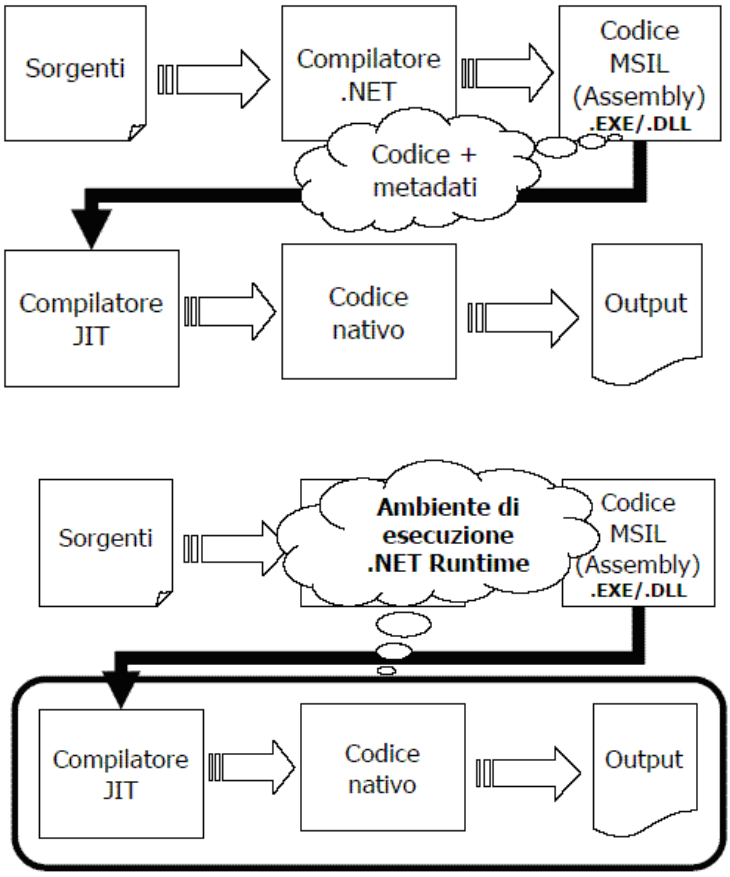


Codice MSIL



Compilatore al volo basato sul concetto JIT.
Non tutto l'IL di un PE è eseguito durante un'applicazione, solo la parte necessaria è compilata un istante prima della sua esecuzione.
Il codice compilato è memorizzato per successive esecuzioni.
Tutto il codice .NET è compilato JIT, anche linguaggi di scripting come VBScript e JavaScript.
Solo il codice usato sarà compilato.
Minore occupazione di memoria.
Facile rimozione del codice inutilizzato da tempo.
Controlli sull'IL in fase di compilazione.
Dati per la compilazione contenuti nello stesso file del codice: metadati.
Compilazione ottimizzante perché conosce lo stato preciso dell'ambiente di esecuzione.
L'overhead è una lieve differenza che, nella maggior parte dei casi, non sarà rilevata.
Quando è caricata una classe, il caricatore aggiunge uno stub a ogni metodo della classe.

La prima volta che è chiamato il metodo, il codice stub cede il controllo al compilatore JIT, che compila MSIL nel codice nativo.
Lo stub è quindi modificato per puntare al codice nativo appena creato, affinché le chiamate successive passino direttamente al codice nativo.



Ciò che va in esecuzione è sempre il codice nativo, fa eccezione a questa regola soltanto il Visual C++ che è ancora in grado di produrre un codice non gestito, quindi un codice nativo della piattaforma che si sta indirizzando e che è anche in grado di realizzare componenti che invece sono assembly.NET simili a quelli prodotti da altri linguaggi, quindi assembly gestiti, assoggettati alla fase JIT prima di poter essere eseguiti.

Motori JIT

Motore	Descrizione	Dove si trova
JIT		Attuale implementazione.
OptiJIT	Codice più ottimizzato.	Non implementato.
FastJIT	Esecuzione JIT più veloce.	.NET Compact Framework.
Native (Pre-JIT)	Compilazione preventiva, assembly compilato e salvato.	NGEN.EXE

5. VES (*VIRTUAL EXECUTION SYSTEM)*

È l'ambiente di esecuzione, la macchina virtuale, del CLR.
VES carica, realizza i link ed esegue le applicazioni scritte per il CLR contenute nei file PE.
Il VES adempie le sue funzioni di loader utilizzando le informazioni contenute nei metadati ed utilizza late binding per integrare moduli compilati separatamente, che possono essere anche scritti in linguaggi differenti.
Il VES fornisce servizi durante l'esecuzione dei codici, che includono la gestione automatica della memoria, supporto per debugging, sandbox per la sicurezza analoghe a

Java e l'interoperabilità con il codice non gestito come ad esempio componenti COM.

AD (*APPLICATION DOMAIN*)

Sono i processi leggeri del CLR e possono essere immaginati come una fusione della sandbox di Java e del modello a thread.
"Leggeri" perché più AD sono eseguiti in un unico processo Win32, ma con meccanismi di sicurezza ed isolamento.
Controllo di sicurezza in fase di compilazione.
Ogni applicazione può avere AD multipli associata con essa e ognuno di questi ha un file di configurazione contenente i permessi di sicurezza.
Nonostante più AD possano essere eseguiti in un unico processo, nessuna chiamata diretta è permessa tra metodi di oggetti presenti in differenti AD.
In alternativa un meccanismo di tipo proxy è utilizzato per isolare lo spazio dei codici.

ASSEMBLY

È una collezione di funzionalità sviluppate e distribuite come una singola unità applicativa, uno o più file.
In pratica è una raccolta di codice compilato.
Completamente autodescrittivo grazie al suo manifesto.
Installazione di tipo XCOPY.
Il manifesto è un metadato che ha i seguenti compiti.

- ✓ Stabilisce l'identità dell'assembly in termini di nome, versione, livello di condivisione tra applicazioni diverse, firma digitale.
- ✓ Definisce quali file costituiscono l'implementazione dell'assembly.
- ✓ Specifica le dipendenze in fase di compilazione da altri assembly.
- ✓ Specifica i tipi e le risorse che costituiscono l'assembly, inclusi quelli che sono esportati dall'assembly.
- ✓ Specifica l'insieme dei permessi necessari al corretto funzionamento dell'assembly.

Il manifesto è parte indissolubile dell'assembly ed è compreso nello stesso file.
È il CLR che si preoccupa che le dipendenze espresse nel manifesto siano verificate ed eventualmente si occupa di "ripararle".
Il run-time è in grado di eseguire due versioni diverse della stessa componente side-by-side.
Il run-time è in grado di rendere disponibile due versioni diverse della stessa libreria.
Nessuna registrazione è necessaria.
Il CLR fornisce anche API utilizzate dai motori di scripting che creano assembly dinamici durante l'esecuzione degli script; questi assembly sono eseguiti direttamente senza essere salvati su disco.
Global Assembly Cache: memoria per gli assembly "sicuri", gestione riservata agli amministratori, eseguiti fuori dalla sandbox, maggiori privilegi di accesso alle risorse.
Downloaded Assembly Cache: memoria per gli assembly transitori e/o "insicuri", assembly esterni, ad esempio scaricati dalla rete, eseguiti nella sandbox più lenti e con minor accesso alle risorse.

.NET FRAMEWORK E VISUAL STUDIO

Visual Studio è lo strumento di produttività sui servizi messi a disposizione dal .NET Framework e le applicazioni che emergono possono essere componenti Assembly.NET, web service.
Tecnologia alla base di tutti i nuovi prodotti: i linguaggi devono interagire tra loro.
C# è considerato il linguaggio migliore per realizzare applicazioni .NET, è un linguaggio che riassume in sé le idee migliori di molti linguaggi esistenti,quali C++ e Java.
VB.NET presenta alcuni vantaggi rispetto a C#, ad esempio la gestione degli errori è molto più flessibile; si può utilizzare il vecchio sistema basato sull'istruzione *On Error*, sia la

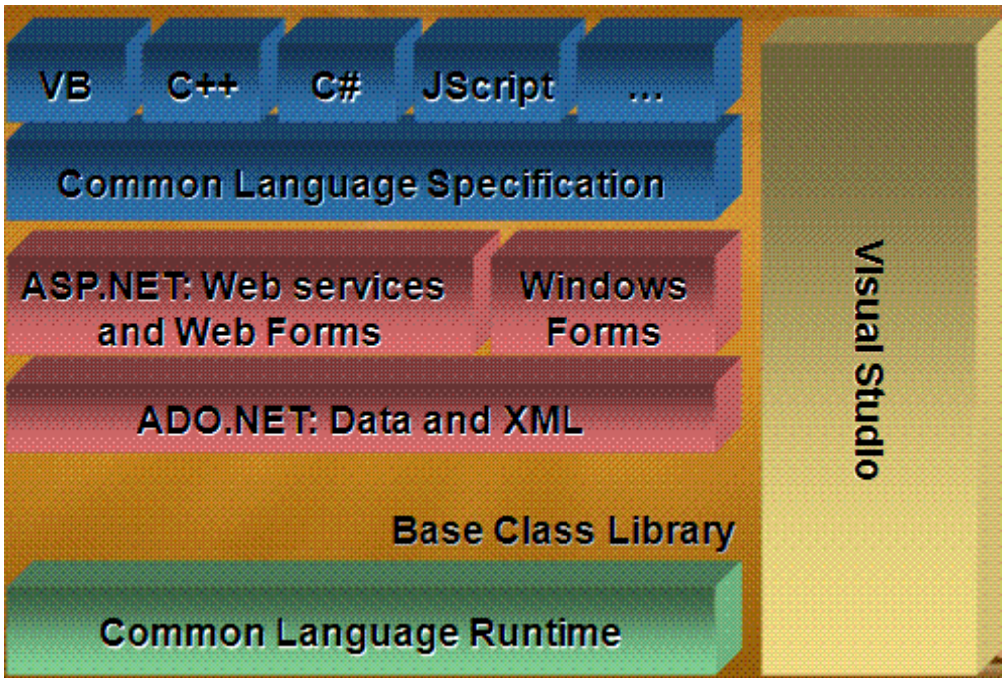
nuova gestione strutturata delle eccezioni.
ASP.NET rappresenta la parte più interessante del .NET Framework, o, comunque, il motivo principale per cui tutti gli sviluppatori Internet dovrebbero prendere in considerazione il passaggio a questa piattaforma, comprende due tecnologie.

1. Le Web Form: utilizzate per applicazioni Internet dotate d'interfaccia utente e sostituiscono le applicazioni ASP.
2. I Web Service: sono applicazioni Internet prive di interfaccia utente.

ASP.NET mette a disposizione una gamma completa di funzionalità per il debugging.
Consente di promuovere la separazione tra interfaccia utente, ossia il codice **XHTML** (*eXtensible HyperText Markup Language*) e il codice che consente all'applicazione di funzionare, scritto in C#, VB.NET o qualsiasi altro linguaggio .NET.

Grazie al concetto di code behind si può suddividere una pagina ASP.NET in due file distinti, uno contenente il codice **HTML** (*HyperText Markup Language*) e i controlli e l'altro il codice sorgente.

Gestisce una versione più flessibile dell'oggetto *Session*: le variabili di questo tipo possono essere mantenute sulla macchina che ospita **IIS** (*Internet Information Server*) o su una macchina diversa appartenente alla stessa rete.
Standard **ISO** (*International Standard Organization*).
Esiste una libreria di classi che riespone con una metafora ad oggetti, tutti i servizi del CLR, i servizi del sistema operativo sottostante e dell'Application Server Microsoft.
Il .NET Framework include anche ADO.NET tecnologia per l'accesso ai dati adatta in scenari disconnessi e ha un forte supporto per l'XML.
ASP.NET, una componente innovativa perché offre un supporto nativo per il web service.
Le Web Forms permettono di realizzare applicazioni ASP.NET con una metafora di programmazione indistinguibile da quella utilizzata dai programmatori VB che realizzano applicazioni WIN32.
Le Windows Forms, una tecnologia che permette di realizzare applicazioni per il sistema operativo sottostante in particolare per Win32, ma anche per i servizi Windows e di applicazioni dalla console.



DEPLOYMENT

Versioni side-by-side dei componenti condivisi: determinata in fase di compilazione e

policy amministrativa in fase di esecuzione.
Policy di sicurezza evidence-based: basata sul codice e sull'utente; origine del codice (locazione); publisher (chiave pubblica).
Le applicazioni .NET usano i servizi del Framework: non hanno accesso diretto al sistema operativo e non dipendono dalla CPU.
È difficile effettuare sviluppo omogeneo, molto tempo è dedicato a far comunicare i vari "strati", allora serve un salto qualitativo per semplificare lo scenario.
In futuro le applicazioni e i driver saranno compilati in IL.
Modello di programmazione **OOP** (*Object Oriented Programming*) comune: niente più accessi alle DLL.

Aggiornamenti facilitati: non si conservano le impostazioni di un'applicazione nel registro perché gli assembly sono installabili e funzionanti con un copia e incolla dei file necessari.

Codice ottimizzato per la piattaforma di esecuzione e sicuro in quanto gestito.
Integrazione dei linguaggi di programmazione: CTS comune.
Gestione automatica della memoria: GC.
I vantaggi dell'utilizzo del Framework sono legati non solo all'intera gamma di tecnologie presenti nel CLR, ma anche ad una ridefinizione di alcuni concetti importanti che hanno segnato la fine di quel problema che è noto come "inferno delle DLL".
Per poter utilizzare un'assembly .NET, non è necessario modificare il registro, anche perché il concetto stesso di pubblicazione è stato ridefinito, quindi si arriva ad un concetto d'installazione dell'applicazione del .NET che non è distinguibile da quello dell'installazione di un'applicazione **DOS** (*Disk Operating System*), ovvero creazione di una directory e copia al suo interno di tutti i componenti dell'applicazione.
Di default nel mondo .NET, la registrazione di una componente avviene sempre in modalità isolata o side-by-side, ovvero un'applicazione cercherà sempre all'interno della propria directory o di opportune sotto directory le componenti alle quali fare il byning. Questo è vero indipendentemente dal tipo di applicazione realizzata e la rimozione di questa applicazione è evidentemente la cancellazione della directory.
Ciò è possibile perché, grazie al nuovo concetto di metadati, le informazioni sono sempre e comunque reperibili in maniera facile, veloce e soprattutto isolata.

APPLICAZIONI

Sono unità configurabili: uno o più assembly; file e dati dipendenti dall'applicazione.
Gli assembly sono localizzati in base a seguenti parametri.
- ✓ Il loro nome logico.
- ✓ L'applicazione che li carica.

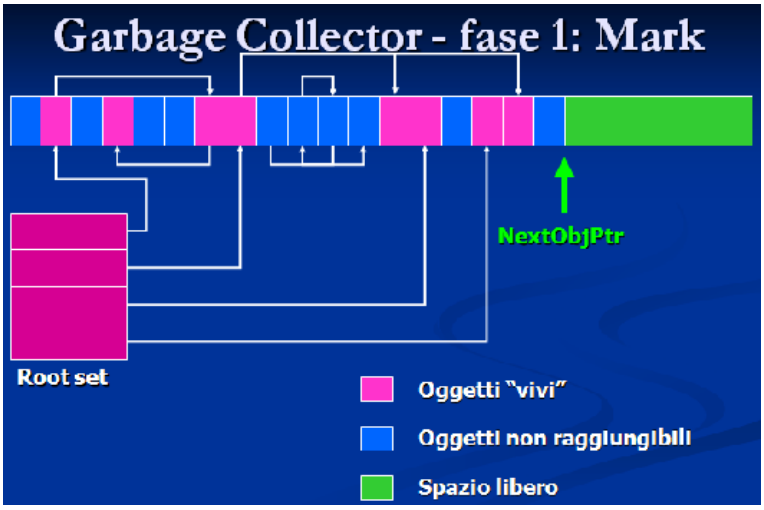
Le applicazioni possono avere versioni private degli assembly: da preferire rispetto a quelle condivise; la politica di versioning può essere per applicazione.
Le applicazioni sono costituite da uno o più assembly localizzati in base ad informazioni che dipendono prevalentemente dal loro nome e anche dall'applicazione che li carica che ha al suo interno, nella propria area manifest, le informazioni per il corretto byning a determinati assembly.
È importante ricordare che di default l'installazione degli assembly è sempre side-by-side, quindi in modalità isolata, non condivisa.

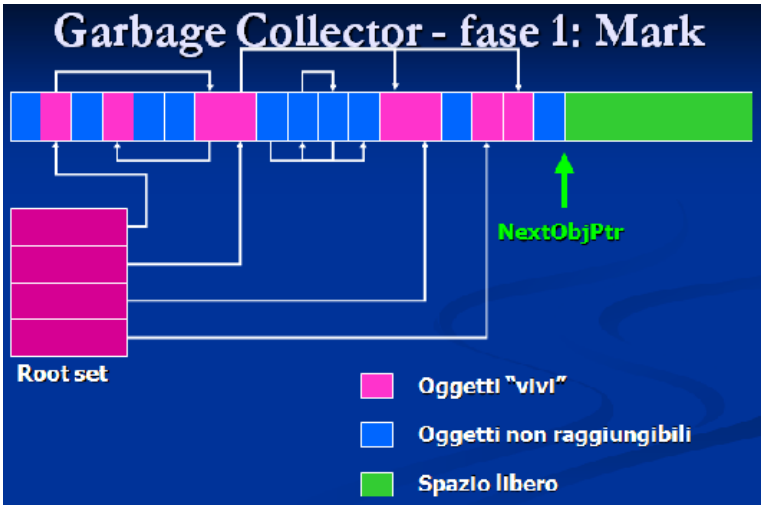
GC (*GARBAGE COLLECTOR*)

Il .NET Framework fornisce una gestione automatica della memoria, usando un meccanismo chiamato GC.
Un'applicazione non deve liberare esplicitamente la memoria che ha allocato.
Il CLR scopre, quando un'applicazione non sta più usando un blocco di memoria ed automaticamente lo ricicla.

Quando tutte le variabili puntatore hanno terminato il loro ciclo di vita o sono state esplicitamente impostate a *Nothing*, l'oggetto è sottoposto ad un processo conosciuto come GC e la memoria occupata nello heap dall'oggetto stesso è liberata.
Questa è la ragione per cui i tipi *value* possono considerarsi generalmente più **veloci** dei tipi *reference*: non essendo allocati nel managed heap, non sono sottoposti al GC.
La memoria allocata per gli oggetti .NET non è rilasciata immediatamente dopo che tutte le variabili che puntano all'oggetto sono state distrutte; ciò avviene perché il GC è eseguito solo, quando l'heap esaurisce la memoria.
Questo fenomeno è anche conosciuto come **distruzione (o finalizzazione) non deterministica**.

Algoritmo Mark-and-Compact





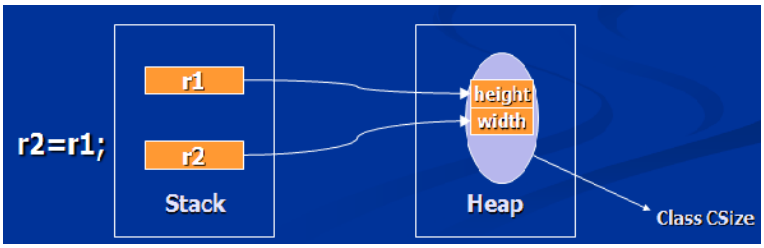
GC e distruzione deterministica
In alcuni casi serve un comportamento di finalizzazione deterministica.
- ✓ Riferimenti a oggetti non gestiti.
- ✓ Utilizzo di risorse che devono essere rilasciate appena termina il loro utilizzo.

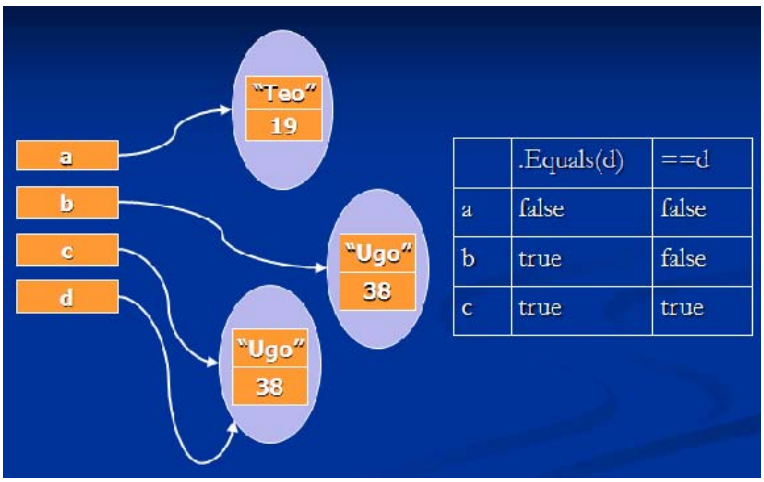
Non si possono usare i finalizzatori, che non sono richiamabili direttamente.
Implementare l'interfaccia *IDisposable*.

Equivalenza e identità

Il confronto tra oggetti può essere il seguente.

- ✓ Di equivalenza, *Object.Equals*: oggetti con stesso tipo e uguale contenuto.
- ✓ D'identità, *Object.ReferenceEquals*: stessa istanza o entrambi null, ==: dipende dal tipo (*ReferenceEquals* o altro), *Object.GetHashCode*: rappresentazione univoca istanza.



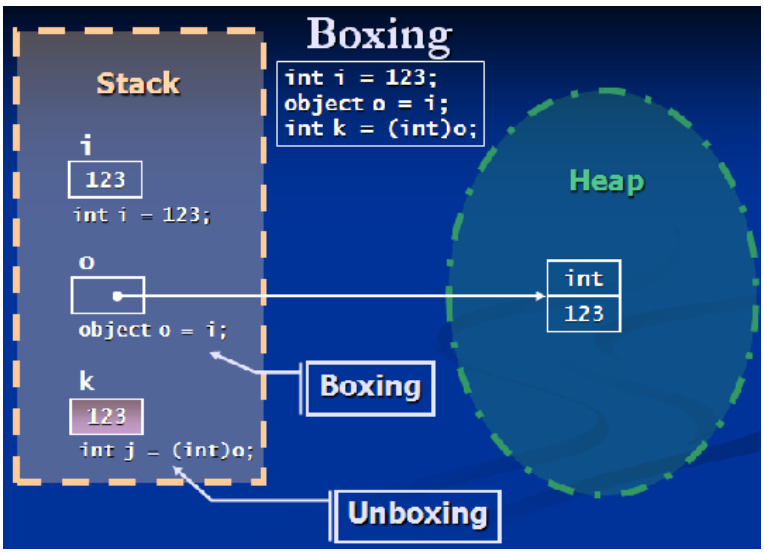


Boxing - Unboxing

È la conversione di un VT in un RT e viceversa, operazione dispendiosa in termini di prestazioni e memoria.

I VT si possono sottoporre a boxing per supportare le funzionalità tipiche degli oggetti, un tipo *value* boxed è un clone indipendente.

Un tipo value boxed può tornare ad essere *value* mediante unboxing.



Ogni variabile di qualsiasi tipo può essere trattata come oggetto, questo significa che una variabile allocata sullo stack può essere spostata nello heap con un'operazione di boxing, e viceversa, dallo heap allo stack con l'operazione di unboxing.

TRADUZIONE

C# 2010

1. JIT: il codice sorgente

```
// hello.cs
using System;
class hello
{
   static void Main(string[] args)
   {
      Console.Clear();
      Console.WriteLine("Ciao, mondo in .NET");
      Console.ReadKey();
   }
}
```

C# è un linguaggio che non tiene conto della posizione delle parole nel codice: un'espressione può essere divisa su più linee di testo senza aggiungere caratteri particolari e deve terminare con un punto e virgola.
In C#, come in Java, non esistono variabili o funzioni globali, tutto deve essere contenuto in una classe: per questo motivo si dichiara una classe che si chiama *hello* che contiene un solo metodo statico *Main*, è il metodo di avvio dell'applicazione e, come in Java, deve essere *static* per essere globale rispetto al codice della classe e quindi poter essere invocato indipendentemente da una specifica istanza della classe.
Main utilizza i metodi statici della classe *Console* per scrivere con il metodo *WriteLine* una stringa e tramite la stessa classe legge un input dalla tastiera con il metodo *ReadLine*.
La classe *Console* fa parte del .NET Framework e per utilizzarla non basta scrivere il suo nome: il compilatore non la troverebbe.
Nel Framework ogni classe e in generale ogni tipo, fa parte di uno spazio di nomi, o *namespace*, quindi per utilizzare una classe si deve dire al compilatore che nell'applicazione si fa riferimento ad un certo spazio di nomi.
La classe *Console* si trova nel namespace *System*, quindi la prima linea del programma, *using System*, permette di utilizzare la classe *Console* all'interno di *Main*.

2. JIT: traduzione in MSIL

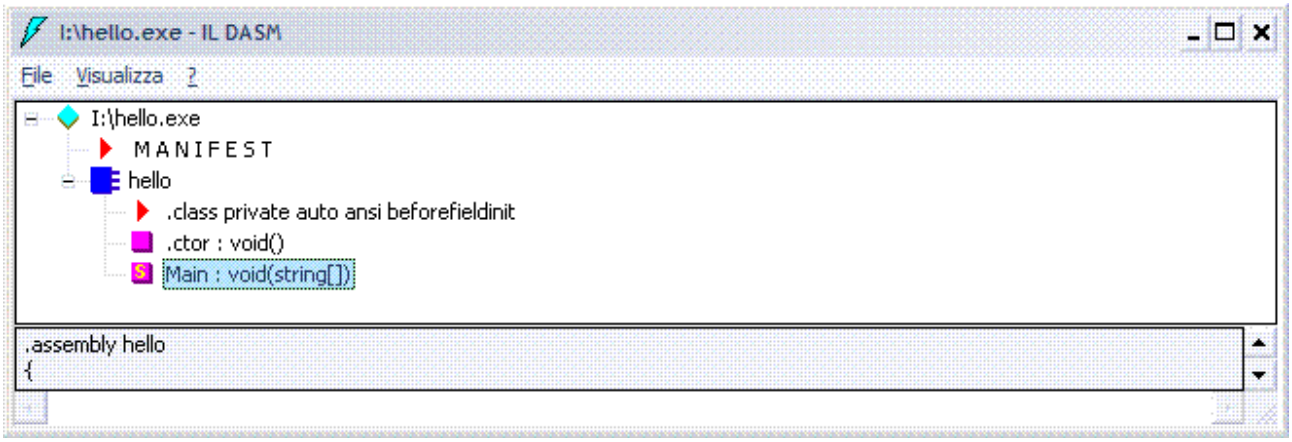
Il primo passo verso l'esecuzione dell'applicazione è la traduzione in IL, il compilatore può fermarsi qui, infatti tutte le macchine virtuali compatibili con .NET possono caricare ed eseguire il codice intermedio, che è il codice di una macchina astratta, come il byte code dell'JVM.
Il file eseguibile, compilato in IL, è costituito da due parti.
1. La prima è il codice MSIL, utilizzato per generare il codice nativo.
2. La seconda è rappresentata dai metadati.

Con un tool in dotazione con l'**SDK** (*Software Development Kit*) è possibile disassemblare il file ottenuto dalla compilazione e si ottiene il seguente output.

Start/Tutti i programmi/Microsoft Visual Studio 2010/
Microsoft Windows SDK Tools/ IL Disassembler

Bisogna sottolineare che il codice MSIL è un'implementazione di IL, ma non è quello

standard.
I metadati, inoltre, sono aggiunti dal compilatore, dal Framework e volendo dall'utente stesso.



Significato dei vari simboli.

Symbol	Meaning
	More info
	Namespace
	Class
	Interface
	Value Class
	Enum
	Method
	Static method
	Field
	Static field
	Event
	Property
	Manifest or a class info item

Doppio clic sul metodo *Main* si ottiene una finestra che contiene il codice seguente.

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  // Code size       25 (0x19)
  .maxstack  8
  IL_0000:  nop
  IL_0001:  call       void [mscorlib]System.Console::Clear()
  IL_0006:  nop
  IL_0007:  ldstr      "Ciao, mondo in .NET"
  IL_000c:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_0011:  nop
  IL_0012:  call valuetype [mscorlib]System.ConsoleKeyInfo
[mscorlib]System.Console::ReadKey()
  IL_0017:  pop
  IL_0018:  ret
} // end of method hello::Main
```

Tutto ciò che inizia con un "." in IL è una direttiva, in questo caso s'indica un metodo, tutto ciò, invece, che non è preceduto da punti, sono le istruzioni da eseguire.

.entrypoint
La direttiva *.entrypoint* indica l'entrypoint del codice, è una direttiva che in un assembly può esser usata una sola volta, se nello stesso assembly si usa più volte errore.

.maxstack 8
IL è stack based significa che all'interno di questo metodo non sono caricati sullo stack, virtual stack ovviamente, mai di più di 8 valori.

IL_0000: nop

IL_0001: call void [mscorlib]System.Console::Clear()

IL_0007: ldstr "Ciao, mondo in .NET"
Push sullo stack la stringa, in realtà push ad un reference alla stringa che si trova nel metadata dell'eseguibile.

IL_000c: call void [mscorlib]System.Console::WriteLine(string)
IL_0012: call valuetype [mscorlib]System.ConsoleKeyInfo [mscorlib]System.Console::ReadKey()
È chiamata la funzione, fornisce anche *namespace* e classe.

IL_0017: pop
Rimuove il valore corrente in cima allo stack, è il valore di ritorno di *ReadKey* che in ogni caso non si mette in nessuna variabile, quindi basta toglierlo con pop senza usare *stloc*.

IL_0018: ret
Conclude il metodo.

3. JIT: identificazione dei blocchi
Identificare i punti d'ingresso e di uscita dei blocchi.

Block 0
Block 1
Block 2
Block 3

Il primo passo per la traduzione dell'IL in codice nativo è l'identificazione dei blocchi di codice, questo lavoro può essere eseguito in due modi.
1. Dall'interprete di byte code che è il cuore della macchina virtuale .NET.
2. Da compilare al volo, si parla di compilazione JIT.

Eseguito a priori dal compilatore, si parla di compilazione **AOT** (*Ahead Of Time*).

4. JIT: raggruppamento del codice
Il codice di ogni blocco è raggruppato in alberi e foreste per rappresentare la struttura dei blocchi e la struttura interna del codice, per esempio loop e nidificazioni.

Block 0:
(…)
Block 1.
(…)
Block 2:

(…)
Block 3:
(…)

5. JIT: emissione del codice nativo
È la traduzione del codice intermedio in una rappresentazione naturale per la CPU INTEL a bordo del PC.
Il disassembler fornisce gli opcode delle varie istruzioni, nonché l'indirizzo da dove inizia il metodo in questione.
Fare clic sul menu ***Visualizza/Mostra byte***, riaprire il codice del *main*.

```
.method private hidebysig static void  Main(string[] args) cil managed
// SIG: 00 01 01 1D 0E
{
  .entrypoint
  // Method begins at RVA 0x2050
  // Code size       25 (0x19)
  .maxstack  8
  IL_0000:  /* 00   |                */ nop
  IL_0001:  /* 28   | (0A)000011       */ call       void [mscorlib]System.Console::Clear()
  IL_0006:  /* 00   |                */ nop
  IL_0007:  /* 72   | (70)000001       */ ldstr      "Ciao, mondo in .NET"
  IL_000c:  /* 28   | (0A)000012       */ call void [mscorlib]System.Console::WriteLine(string)
  IL_0011:  /* 00   |                */ nop
  IL_0012:  /* 28   | (0A)000013       */ call       valuetype [mscorlib]System.ConsoleKeyInfo
[mscorlib]System.Console::ReadKey()
  IL_0017:  /* 26   |                */ pop
  IL_0018:  /* 2A   |                */ ret
} // end of method hello::Main
```

La direttiva *.entrypoint* indica l'entrypoint del codice, il metodo inizia a RVA 2050h.
I metodi sono delimitati dalle parentesi graffe, ciò nonostante è opportuno concludere il metodo con *ret*.
L'attributo *hidebysig* nasconde il metodo a classi derivate.
Se si clicca sul triangolino rosso della classe si vede quanto segue.

```
.class private auto ansi beforefieldinit hello
       extends [mscorlib]System.Object
{
} // end of class hello
```

Da qui si vede che la classe è un'estensione della classe *System.Object*.
Ciò comporta l'esistenza dovuta del costruttore *ctor*.
Questo potrebbe essere l’ultimo passo, oppure potrebbe ancora passare un ottimizzatore in grado di sostituire determinate sequenze d’istruzioni con altre equivalenti, ma più veloci.

INSTRUCTION SET E OPCODE

Instruction	Opcode (Hex)	Short Description
add	58	Adds two values and pushes the result onto the evaluation stack.
add.ovf	D6	Adds two integers, performs an overflow check, and pushes the result onto the evaluation stack.
add.ovf.un	D7	Adds two unsigned integer values, performs an overflow check, and pushes the result onto the evaluation stack.
and	5F	Computes the bitwise AND of two values and pushes the result onto the evalution stack.
arglist	FE 00	Returns an unmanaged pointer to the argument list of the current method.
beq	3B	Transfers control to a target instruction if two values are equal.
beq.s	2E	Transfers control to a target instruction (short form) if two values are equal.
bge	3C	Transfers control to a target instruction if the first value is greater than or equal to the second value.
bge.s	2F	Transfers control to a target instruction (short form) if the first value is greater than or equal to the second value.
bge.un	41	Transfers control to a target instruction if the the first value is greather than the second value, when comparing unsigned integer values or unordered float values.
bge.un.s	34	Transfers control to a target instruction (short form) if the first value is greather than the second value, when comparing unsigned integer values or unordered float values.
bgt	3D	Transfers control to a target instruction if the first value is greater than the second value.
bgt.s	30	Transfers control to a target instruction (short form) if the first value is greater than the second value.
bgt.un	42	Transfers control to a target instruction if the first value is greater than the second value, when comparing unsigned integer values or unordered float values.
bgt.un.s	35	Transfers control to a target instruction (short form) if the first value is greater than the second value, when comparing unsigned integer values or unordered float values.
ble	3E	Transfers control to a target instruction if the first value is less than or equal to the second value.
ble.s	31	Transfers control to a target instruction (short form) if the first value is less than or equal to the second value.
ble.un	43	Transfers control to a target instruction if the first value is less than or equal to the second value, when comparing unsigned integer values or unordered float values.
ble.un.s	36	Transfers control to a target instruction (short form) if the first value is less than or equal to the second value, when comparing unsigned integer values or unordered float values.
blt	3F	Transfers control to a target instruction if the first value is less than the second value.
blt.s	32	Transfers control to a target instruction (short form) if the first value is less than the second value.

blt.un	44	Transfers control to a target instruction if the first value is less than the second value, when comparing unsigned integer values or unordered float values.
blt.un.s	37	Transfers control to a target instruction (short form) if the first value is less than the second value, when comparing unsigned integer values or unordered float values.
bne.un	40	Transfers control to a target instruction when two unsigned integer values or unordered float values are not equal.
bne.un.s	33	Transfers control to a target instruction (short form) when two unsigned integer values or unordered float values are not equal.
box	8C	Converts a value type to an object reference (type O).
br	38	Unconditionally transfers control to a target instruction.
break	01	Signals the Common Language Infrastructure (CLI) to inform the debugger that a break point has been tripped.
brfalse	39	Transfers control to a target instruction if value is false, a null reference (Nothing in Visual Basic), or zero.
brfalse.s	2C	Transfers control to a target instruction if *value* is false, a null reference, or zero.
brtrue	3A	Transfers control to a target instruction if *value* is true, not null, or non-zero.
brtrue.s	2D	Transfers control to a target instruction (short form) if *value* is true, not null, or non-zero.
br.s	2B	Unconditionally transfers control to a target instruction (short form).
call	28	Calls the method indicated by the passed method descriptor.
calli	29	Calls the method indicated on the evaluation stack (as a pointer to an entry point) with arguments described by a calling convention.
callvirt	6F	Calls a late-bound method on an object, pushing the return value onto the evaluation stack.
castclass	74	Attempts to cast an object passed by reference to the specified class.
ceq	FE 01	Compares two values. If they are equal, the integer value 1 **(**int32) is pushed onto the evaluation stack; otherwise 0 (int32) is pushed onto the evaluation stack.
cgt	FE 02	Compares two values. If the first value is greater than the second, the integer value 1 (int32) is pushed onto the evaluation stack; otherwise 0 (int32) is pushed onto the evaluation stack.
cgt.un	FE 03	Compares two unsigned or unordered values. If the first value is greater than the second, the integer value 1 (int32) is pushed onto the evaluation stack; otherwise 0 (int32) is pushed onto the evaluation stack.
ckfinite	C3	Throws ArithmeticException if value is not a finite number.
clt	FE 04	Compares two values. If the first value is less than the second, the integer value 1 (int32) is pushed onto the evaluation stack; otherwise 0 (int32) is pushed onto the evaluation stack.
clt.un	FE 05	Compares the unsigned or unordered values *value1* and *value2*. If *value1* is less than *value2*, then the integer value 1 (int32) is pushed onto the evaluation stack; otherwise 0 (int32) is pushed onto the evaluation stack.

conv.i	D3	Converts the value on top of the evaluation stack to natural int.
conv.i1	67	Converts the value on top of the evaluation stack to int8, then extends (pads) it to int32.
conv.i2	68	Converts the value on top of the evaluation stack to int16, then extends (pads) it to int32.
conv.i4	69	Converts the value on top of the evaluation stack to int32.
conv.i8	6A	Converts the value on top of the evaluation stack to int64.
conv.ovf.i	D4	Converts the signed value on top of the evaluation stack to signed natural int, throwing OverflowException on overflow.
conv.ovf.i1	B3	Converts the signed value on top of the evaluation stack to signed int8 and extends it to int32, throwing OverflowException on overflow.
conv.ovf.i1.un	82	Converts the unsigned value on top of the evaluation stack to signed int8 and extends it to int32, throwing OverflowException on overflow.
conv.ovf.i2	B5	Converts the signed value on top of the evaluation stack to signed int16 and extending it to int32, throwing OverflowException on overflow.
conv.ovf.i2.un	83	Converts the unsigned value on top of the evaluation stack to signed int16 and extends it to int32, throwing OverflowException on overflow.
conv.ovf.i4	B7	Converts the signed value on top of the evaluation tack to signed int32, throwing OverflowException on overflow.
conv.ovf.i4.un	84	Converts the unsigned value on top of the evaluation stack to signed int32, throwing OverflowException on overflow.
conv.ovf.i8	B9	Converts the signed value on top of the evaluation stack to signed int64, throwing OverflowException on overflow.
conv.ovf.i8.un	85	Converts the unsigned value on top of the evaluation stack to signed int64, throwing OverflowException on overflow.
conv.ovf.i.un	8A	Converts the unsigned value on top of the evaluation stack to signed natural int, throwing OverflowException on overflow.
conv.ovf.u	D5	Converts the signed value on top of the evaluation stack to unsigned natural int, throwing OverflowException on overflow.
conv.ovf.u1	B4	Converts the signed value on top of the evaluation stack to unsigned int8 and extends it to int32, throwing OverflowException on overflow.
conv.ovf.u1.un	86	Converts the unsigned value on top of the evaluation stack to unsigned int8 and extends it to int32, throwing OverflowException on overflow.
conv.ovf.u2	B6	Converts the signed value on top of the evaluation stack to unsigned int16 and extends it to int32, throwing OverflowException on overflow.
conv.ovf.u2.un	87	Converts the unsigned value on top of the evaluation stack to unsigned int16 and extends it to int32, throwing OverflowException on overflow.
conv.ovf.u4	B8	Converts the signed value on top of the evaluation stack to unsigned int32, throwing OverflowException on overflow.
conv.ovf.u4.un	88	Converts the unsigned value on top of the evaluation stack to unsigned int32, throwing OverflowException on overflow.
conv.ovf.u8	BA	Converts the signed value on top of the evaluation stack to unsigned int64, throwing OverflowException on overflow.

conv.ovf.u8.un	89	Converts the unsigned value on top of the evaluation stack to unsigned int64, throwing OverflowException on overflow.
conv.ovf.u.un	8B	Converts the unsigned value on top of the evaluation stack to unsigned natural int, throwing OverflowException on overflow.
conv.r4	6B	Converts the value on top of the evaluation stack to float32.
conv.r8	6C	Converts the value on top of the evaluation stack to float64.
conv.r.un	76	Converts the unsigned integer value on top of the evaluation stack to float32.
conv.u	E0	Converts the value on top of the evaluation stack to unsigned natural int, and extends it to natural int.
conv.u1	D2	Converts the value on top of the evaluation stack to unsigned int8, and extends it to int32.
conv.u2	D1	Converts the value on top of the evaluation stack to unsigned int16, and extends it to int32.
conv.u4	6D	Converts the value on top of the evaluation stack to unsigned int32, and extends it to int32.
conv.u8	6E	Converts the value on top of the evaluation stack to unsigned int64, and extends it to int64.
cpblk	FE 17	Copies a specified number bytes from a source address to a destination address.
cpobj	70	Copies the value type located at the address of an object (type &, * or natural int) to the address of the destination object (type &, * or natural int).
div	5B	Divides two values and pushes the result as a floating-point (type F) or quotient (type int32) onto the evaluation stack.
div.un	5C	Divides two unsigned integer values and pushes the result (int32) onto the evaluation stack.
dup	25	Copies the current topmost value on the evaluation stack, and then pushes the copy onto the evaluation stack.
endfilter	FE 11	Transfers control from the filter clause of an exception back to the Common Language Infrastructure (CLI) exception handler.
endfinally	DC	Transfers control from the fault or finally clause of an exception block back to the Common Language Infrastructure (CLI) exception handler.
initblk	FE 18	Initializes a specified block of memory at a specific address to a given size and initial value.
initobj	FE 15	Initializes all the fields of the object at a specific address to a null reference or a 0 of the appropriate primitive type.
isinst	75	Tests whether an object reference (type O) is an instance of a particular class.
jmp	27	Exits current method and jumps to specified method.
ldarg	FE 09	Loads an argument (referenced by a specified index value) onto the stack.
ldarga	FE 0A	Load an argument address onto the evaluation stack.
ldarga.s	0F	Load an argument address, in short form, onto the evaluation stack.
ldarg.0	02	Loads the argument at index 0 onto the evaluation stack.
ldarg.1	03	Loads the argument at index 1 onto the evaluation stack.
ldarg.2	04	Loads the argument at index 2 onto the evaluation stack.
ldarg.3	05	Loads the argument at index 3 onto the evaluation stack.
ldarg.s	0E	Loads the argument (referenced by a specified short form index) onto the evaluation stack.

ldc.i4	20	Pushes a supplied value of type int32 onto the evaluation stack as an int32.
ldc.i4.0	16	Pushes the integer value of 0 onto the evaluation stack as an int32.
ldc.i4.1	17	Pushes the integer value of 1 onto the evaluation stack as an int32.
ldc.i4.2	18	Pushes the integer value of 2 onto the evaluation stack as an int32.
ldc.i4.3	19	Pushes the integer value of 3 onto the evaluation stack as an int32.
ldc.i4.4	1A	Pushes the integer value of 4 onto the evaluation stack as an int32.
ldc.i4.5	1B	Pushes the integer value of 5 onto the evaluation stack as an int32.
ldc.i4.6	1C	Pushes the integer value of 6 onto the evaluation stack as an int32.
ldc.i4.7	1D	Pushes the integer value of 7 onto the evaluation stack as an int32.
ldc.i4.8	1E	Pushes the integer value of 8 onto the evaluation stack as an int32.
ldc.i4.m1	15	Pushes the integer value of -1 onto the evaluation stack as an int32.
ldc.i4.s	1F	Pushes the supplied int8 value onto the evaluation stack as an int32, short form.
ldc.i8	21	Pushes a supplied value of type int64 onto the evaluation stack as an int64.
ldc.r4	22	Pushes a supplied value of type float32 onto the evaluation stack as type F (float).
ldc.r8	23	Pushes a supplied value of type float64 onto the evaluation stack as type F (float).
ldelema	8F	Loads the address of the array element at a specified array index onto the top of the evaluation stack as type & (managed pointer).
ldelem.i	97	Loads the element with type natural int at a specified array index onto the top of the evaluation stack as a natural int.
ldelem.i1	90	Loads the element with type int8 at a specified array index onto the top of the evaluation stack as an int32.
ldelem.i2	92	Loads the element with type int16 at a specified array index onto the top of the evaluation stack as an int32.
ldelem.i4	94	Loads the element with type int32 at a specified array index onto the top of the evaluation stack as an int32.
ldelem.i8	96	Loads the element with type int64 at a specified array index onto the top of the evaluation stack as an int64.
ldelem.r4	98	Loads the element with type float32 at a specified array index onto the top of the evaluation stack as type F (float).
ldelem.r8	99	Loads the element with type float64 at a specified array index onto the top of the evaluation stack as type F (float).
ldelem.ref	9A	Loads the element containing an object reference at a specified array index onto the top of the evaluation stack as type O (object reference).
ldelem.u1	91	Loads the element with type unsigned int8 at a specified array index onto the top of the evaluation stack as an int32.

ldelem.u2	93	Loads the element with type unsigned int16 at a specified array index onto the top of the evaluation stack as an int32.
ldelem.u4	95	Loads the element with type unsigned int32 at a specified array index onto the top of the evaluation stack as an int32.
ldfld	7B	Finds the value of a field in the object whose reference is currently on the evaluation stack.
ldflda	7C	Finds the address of a field in the object whose reference is currently on the evaluation stack.
ldftn	FE 06	Pushes an unmanaged pointer (type natural int) to the native code implementing a specific method onto the evaluation stack.
ldind.i	4D	Loads a value of type natural int as a natural int onto the evaluation stack indirectly.
ldind.i1	46	Loads a value of type int8 as an int32 onto the evaluation stack indirectly.
ldind.i2	48	Loads a value of type int16 as an int32 onto the evaluation stack indirectly.
ldind.i4	4A	Loads a value of type int32 as an int32 onto the evaluation stack indirectly.
ldind.i8	4C	Loads a value of type int64 as an int64 onto the evaluation stack indirectly.
ldind.r4	4E	Loads a value of type float32 as a type F (float) onto the evaluation stack indirectly.
ldind.r8	4F	Loads a value of type float64 as a type F (float) onto the evaluation stack indirectly.
ldind.ref	50	Loads an object reference as a type O (object reference) onto the evaluation stack indirectly.
ldind.u1	47	Loads a value of type unsigned int8 as an int32 onto the evaluation stack indirectly.
ldind.u2	49	Loads a value of type unsigned int16 as an int32 onto the evaluation stack indirectly.
ldind.u4	4B	Loads a value of type unsigned int32 as an int32 onto the evaluation stack indirectly.
ldlen	8E	Pushes the number of elements of a zero-based, one-dimensional array onto the evaluation stack.
ldloc	FE 0C	Loads the local variable at a specific index onto the evaluation stack.
ldloca	FE 0D	Loads the address of the local variable at a specific index onto the evaluation stack.
ldloca.s	12	Loads the address of the local variable at a specific index onto the evaluation stack, short form.
ldloc.0	06	Loads the local variable at index 0 onto the evaluation stack.
ldloc.1	07	Loads the local variable at index 1 onto the evaluation stack.
ldloc.2	08	Loads the local variable at index 2 onto the evaluation stack.
ldloc.3	09	Loads the local variable at index 3 onto the evaluation stack.
ldloc.s	11	Loads the local variable at a specific index onto the evaluation stack, short form.
ldnull	14	Pushes a null reference (type O) onto the evaluation stack.
ldobj	71	Copies the value type object pointed to by an address to the top of the evaluation stack.
ldsfld	7E	Pushes the value of a static field onto the evaluation stack.
ldsflda	7F	Pushes the address of a static field onto the evaluation stack.

ldstr	72	Pushes a new object reference to a string literal stored in the metadata.
ldtoken	D0	Converts a metadata token to its runtime representation, pushing it onto the evaluation stack.
ldvirtftn	FE 07	Pushes an unmanaged pointer (type natural int) to the native code implementing a particular virtual method associated with a specified object onto the evaluation stack.
leave	DD	Exits a protected region of code, unconditionally tranferring control to a specific target instruction.
leave.s	DE	Exits a protected region of code, unconditionally tranferring control to a target instruction (short form).
localloc	FE 0F	Allocates a certain number of bytes from the local dynamic memory pool and pushes the address (a transient pointer, type *****) of the first allocated byte onto the evaluation stack.
mkrefany	C6	Pushes a typed reference to an instance of a specific type onto the evaluation stack.
mul	5A	Multiplies two values and pushes the result on the evaluation stack.
mul.ovf	D8	Multiplies two integer values, performs an overflow check, and pushes the result onto the evaluation stack.
mul.ovf.un	D9	Multiplies two unsigned integer values, performs an overflow check, and pushes the result onto the evaluation stack.
neg	65	Negates a value and pushes the result onto the evaluation stack.
newarr	8D	Pushes an object reference to a new zero-based, one-dimensional array whose elements are of a specific type onto the evaluation stack.
newobj	73	Creates a new object or a new instance of a value type, pushing an object reference (type O) onto the evaluation stack.
nop	00	Fills space if opcodes are patched. No meaningful operation is performed although a processing cycle can be consumed.
not	66	Computes the bitwise complement of the integer value on top of the stack and pushes the result onto the evaluation stack as the same type.
or	60	Compute the bitwise complement of the two integer values on top of the stack and pushes the result onto the evaluation stack.
pop	26	Removes the value currently on top of the evaluation stack.
refanytype	FE 1D	Retrieves the type token embedded in a typed reference.
refanyval	C2	Retrieves the address (type &) embedded in a typed reference.
rem	5D	Divides two values and pushes the remainder onto the evaluation stack.
rem.un	5E	Divides two unsigned values and pushes the remainder onto the evaluation stack.
ret	2A	Returns from the current method, pushing a return value (if present) from the caller's evaluation stack onto the callee's evaluation stack.
rethrow	FE 1A	Rethrows the current exception.
shl	62	Shifts an integer value to the left (in zeroes) by a specified number of bits, pushing the result onto the evaluation stack.
shr	63	Shifts an integer value (in sign) to the right by a specified number of bits, pushing the result onto the evaluation stack.

shr.un	64	Shifts an unsigned integer value (in zeroes) to the right by a specified number of bits, pushing the result onto the evaluation stack.
sizeof	FE 1C	Pushes the size, in bytes, of a supplied value type onto the evaluation stack.
starg	FE 0B	Stores the value on top of the evaluation stack in the argument slot at a specified index.
starg.s	10	Stores the value on top of the evaluation stack in the argument slot at a specified index, short form.
stelem.i	9B	Replaces the array element at a given index with the natural int value on the evaluation stack.
stelem.i1	9C	Replaces the array element at a given index with the int8 value on the evaluation stack.
stelem.i2	9D	Replaces the array element at a given index with the int16 value on the evaluation stack.
stelem.i4	9E	Replaces the array element at a given index with the int32 value on the evaluation stack.
stelem.i8	9F	Replaces the array element at a given index with the int64 value on the evaluation stack.
stelem.r4	A0	Replaces the array element at a given index with the float32 value on the evaluation stack.
stelem.r8	A1	Replaces the array element at a given index with the float64 value on the evaluation stack.
stelem.ref	A2	Replaces the array element at a given index with the object ref value (type O) on the evaluation stack.
stfld	7D	Replaces the value stored in the field of an object reference or pointer with a new value.
stind.i	DF	Stores a value of type natural int at a supplied address.
stind.i1	52	Stores a value of type int8 at a supplied address.
stind.i2	53	Stores a value of type int16 at a supplied address.
stind.i4	54	Stores a value of type int32 at a supplied address.
stind.i8	55	Stores a value of type int64 at a supplied address.
stind.r4	56	Stores a value of type float32 at a supplied address.
stind.r8	57	Stores a value of type float64 at a supplied address.
stind.ref	51	Stores a object reference value at a supplied address.
stloc	FE 0E	Pops the current value from the top of the evaluation stack and stores it in a the local variable list at a specified index.
stloc.0	0A	Pops the current value from the top of the evaluation stack and stores it in a the local variable list at index 0.
stloc.1	0B	Pops the current value from the top of the evaluation stack and stores it in a the local variable list at index 1.
stloc.2	0C	Pops the current value from the top of the evaluation stack and stores it in a the local variable list at index 2.
stloc.3	0D	Pops the current value from the top of the evaluation stack and stores it in a the local variable list at index 3.
stloc.s	13	Pops the current value from the top of the evaluation stack and stores it in a the local variable list at *index* (short form).
stobj	81	Copies a value of a specified type from the evaluation stack into a supplied memory address.
stsfld	80	Replaces the value of a static field with a value from the evaluation stack.

sub	59	Subtracts one value from another and pushes the result onto the evaluation stack.
sub.ovf	DA	Subtracts one integer value from another, performs an overflow check, and pushes the result onto the evaluation stack.
sub.ovf.un	DB	Subtracts one unsigned integer value from another, performs an overflow check, and pushes the result onto the evaluation stack.
switch	45	Implements a jump table.
tail.	FE 14	Performs a postfixed method call instruction such that the current method's stack frame is removed before the actual call instruction is executed.
throw	7A	Throws the exception object currently on the evaluation stack.
unaligned.	FE 12	Indicates that an address currently atop the evaluation stack might not be aligned to the natural size of the immediately following ldind, stind, ldfld, stfld, ldobj, stobj, initblk, or cpblk instruction.
unbox	79	Converts the boxed representation of a value type to its unboxed form.
volatile.	FE 13	Specifies that an address currently atop the evaluation stack might be volatile, and the results of reading that location cannot be cached or that multiple stores to that location cannot be suppressed.
xor	61	Computes the bitwise XOR of the top two values on the evaluation stack, pushing the result onto the evaluation stack.

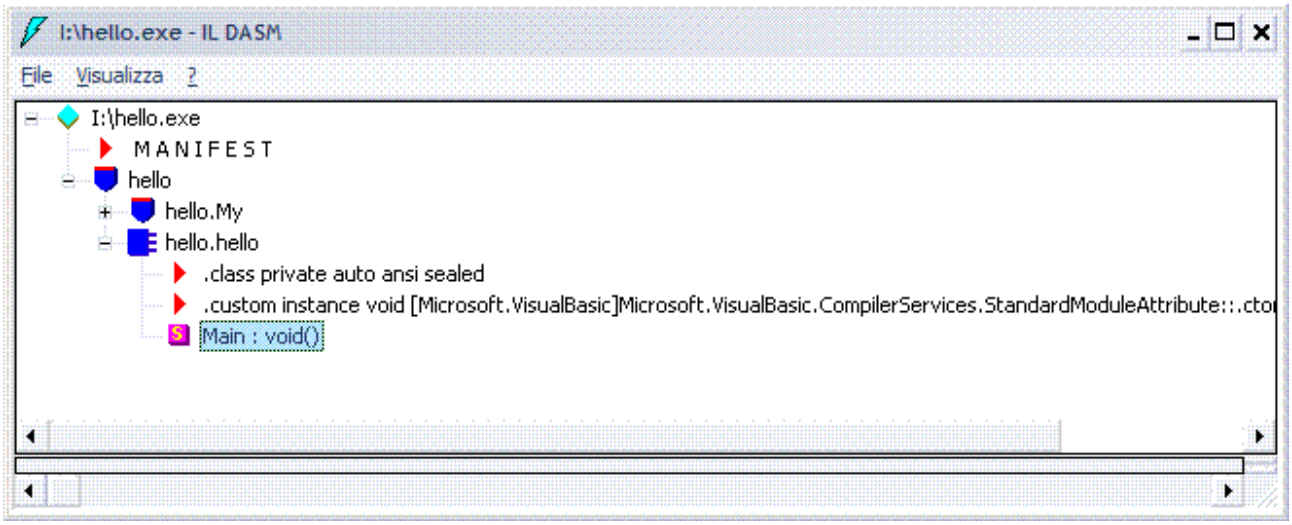
Schema di gruppi principali d’istruzioni.

- ✓ *bgt, ble, bne, br* sono i jump.
- ✓ *conv* servono per convertire grandezze diverse di valori, tipo da byte a qword.
- ✓ *ldc, ldarg, ldelem* corrispondono ai push.
- ✓ *stloc, starg, stelem* corrispondo ai pop.

1. JIT: il codice sorgente

```
' hello.vb
Module hello
    Sub Main()
        Console.Clear()
        Console.WriteLine("Ciao, mondo in .NET")
        Console.ReadKey()
    End Sub
End Module
```

2. JIT: traduzione in MSIL



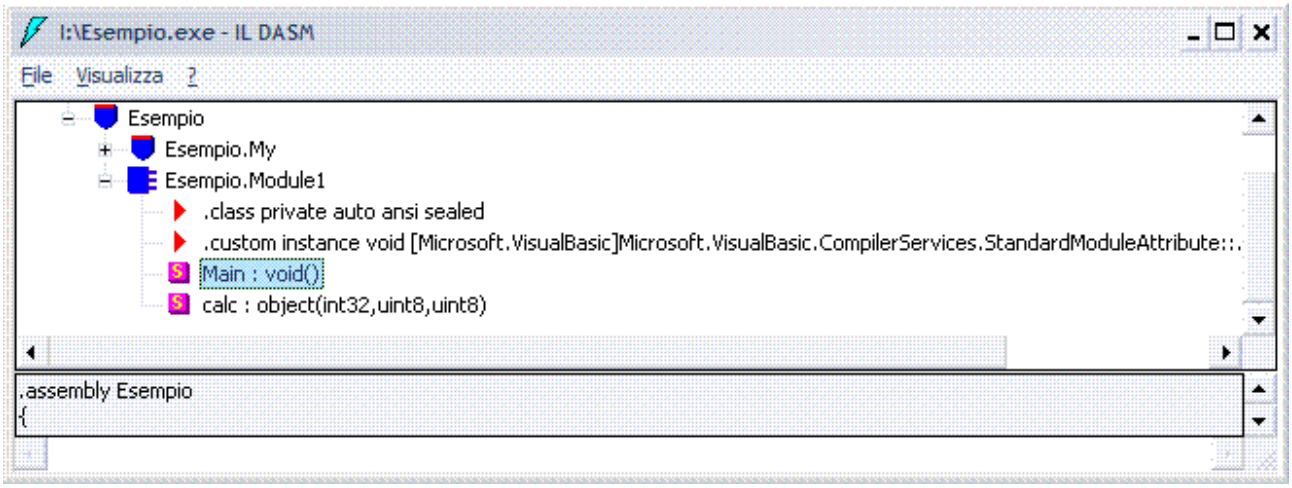
Doppio clic sul metodo *Main* si ottiene una finestra che contiene il codice seguente.

```
.method public static void  Main() cil managed
{
  .entrypoint
  .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )
  // Code size       26 (0x1a)
  .maxstack  8
  IL_0000:  nop
  IL_0001:  call       void [mscorlib]System.Console::Clear()
  IL_0006:  nop
  IL_0007:  ldstr      "Ciao, mondo in .NET"
  IL_000c:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_0011:  nop
  IL_0012:  call       valuetype [mscorlib]System.ConsoleKeyInfo
[mscorlib]System.Console::ReadKey()
  IL_0017:  pop
  IL_0018:  nop
  IL_0019:  ret
} // end of method hello::Main
```

Esempio, la funzione *Calc* ha tre parametri, un intero *n* e due byte *a* e *b*, i bit dell'intero sono shiftati di un numero pari a (*a* + *b*), immettendo 65535 come numero originale e

shiftando di 8 bit si ottiene 255.

```
Module Module1
    Sub Main()
        Dim r As Integer
        Console.Clear()
        r = calc(65535, 3, 5)
        Console.WriteLine("Risultato: {0}", r)
        Console.WriteLine()
        Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione")
        Console.ReadKey()
    End Sub
    Function calc(ByVal n As Integer, ByVal a As Byte, ByVal b As Byte)
        Return (n >> (a + b))
    End Function
End Module
```



Doppio clic sul metodo *Main* si ottiene una finestra che contiene il codice seguente.

```
.method public static void  Main() cil managed
{
  .entrypoint
  .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )
  // Code size       67 (0x43)
  .maxstack  3
  .locals init (int32 V_0)
  IL_0000:  nop
  IL_0001:  call       void [mscorlib]System.Console::Clear()
  IL_0006:  nop
  IL_0007:  ldc.i4     0xffff
  IL_000c:  ldc.i4.3
  IL_000d:  ldc.i4.5
  IL_000e:  call       object Esempio.Module1::calc(int32,uint8, uint8)
  IL_0013:  call       int32
[Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.Conversions::ToInteger(obj
```

ect)
* IL_0018: stloc.0*
* IL_0019: ldstr   "Risultato: {0}"*
* IL_001e: ldloc.0*
* IL_001f: box   [mscorlib]System.Int32*
* IL_0024: call   void [mscorlib]System.Console::WriteLine(string, object)*
* IL_0029: nop*
* IL_002a: call   void [mscorlib]System.Console::WriteLine()*
* IL_002f: nop*
* IL_0030: ldstr   "Premere un tasto qualsiasi per chiudere l'applicazione"*
* IL_0035: call   void [mscorlib]System.Console::WriteLine(string)*
* IL_003a: nop*
* IL_003b: call   valuetype [mscorlib]System.ConsoleKeyInfo*
[mscorlib]System.Console::ReadKey()
* IL_0040: pop*
* IL_0041: nop*
* IL_0042: ret*
} // end of method Module1::Main

.maxstack 3
.locals init (int32 V_0)
IL è stack based *.maxstack* dice che all'interno di questo metodo non sono caricati sullo stack, virtual stack ovviamente, mai di più di 3 valori; la lista e l'inizializzazione delle variabili locali del metodo è effettuata tramite *.locals init*, in questo caso ne abbiamo una sola (*r* che è un *Int32*).

IL_0007: ldc.i4   0xffff
IL_000c: ldc.i4.3
IL_000d: ldc.i4.5
La prima istruzione push un *Int32* sull'evaluation stack, sarebbe il 65535, a seguire vi sono le istruzioni *ldc.i4.3* e *ldc.i4.5* che pushano i valori 3 e 5, se il numero è piccolo, tra zero e otto, è possibile usare *ldc.i4* seguito da un punto e il numero da pushare; lo stack nell'IL non è come quello standard, ovvero **LIFO** (*Last In First Out*), nell'IL i parametri sono pushati nell'ordine che la funzione prevede.

IL_000e: call   object Esempio.Module1::calc(int32,uint8, uint8)
IL_0013: call   int32
[Microsoft.VisualBasic]Microsoft.VisualBasic.CompilerServices.Conversions::ToInteger(obj
ect)
È chiamata la funzione, l'ILDASM fornisce anche *namespace* e classe del metodo chiamato.

IL_0018: stloc.0
Questa istruzione fa il pop del valore corrente sulla cima dello stack e lo mette nella lista delle variabili locali all'index 0; in pratica mette il valore di ritorno della funzione che sta in cima allo stack dentro a *r,* variabile locale.

IL_0019: ldstr   "Risultato: {0}"
Push sullo stack la stringa Risultato; in pratica push un reference alla stringa che si trova nel metadata dell'eseguibile.

IL_001e: ldloc.0
Push la variabile locale a index 0 *(r*) sullo stack.

IL_0024: call void [mscorlib]System.Console::WriteLine(string, object)
Chiama l'overload per stringhe della funzione *Write*.

IL_003b: call valuetype [mscorlib]System.ConsoleKeyInfo [mscorlib]System.Console::ReadKey()
Chiama la *ReadKey*.

IL_0040: pop
Rimuove il valore corrente in cima allo stack, sarebbe il valore di ritorno di *ReadKey* che in ogni caso non si mette in nessuna variabile, quindi basta toglierlo di mezzo con pop senza usare *stloc*; segue il *ret* che conclude il metodo.

Doppio clic sulla funzione *Calc* si ottiene una finestra che contiene il codice seguente.

```
.method public static object  calc(int32 n, uint8 a, uint8 b) cil managed
{
  // Code size       20 (0x14)
  .maxstack  3
  .locals init (object V_0)
  IL_0000:  nop
  IL_0001:  ldarg.0
  IL_0002:  ldarg.1
  IL_0003:  ldarg.2
  IL_0004:  add
  IL_0005:  conv.ovf.u1.un
  IL_0006:  ldc.i4.s   31
  IL_0008:  and
  IL_0009:  shr
  IL_000a:  box        [mscorlib]System.Int32
  IL_000f:  stloc.0
  IL_0010:  br.s       IL_0012
  IL_0012:  ldloc.0
  IL_0013:  ret
} // end of method Module1::calc
```

IL_0001: ldarg.0
IL_0002: ldarg.1
IL_0003: ldarg.2
Caricano sullo stack i tre argomenti passati al metodo, il numero che segue *ldarg* specifica la posizione dell'argomento, il primo è *n* (0) seguono *a* e *b*.

IL_0004: add
Somma i due valori in cima allo stack, in questo caso *a* e *b*, e push il risultato in cima allo stack; dopo questa operazione lo stack ha questo aspetto.
n
Risultato della somma *a* + *b* ; cima dello stack.

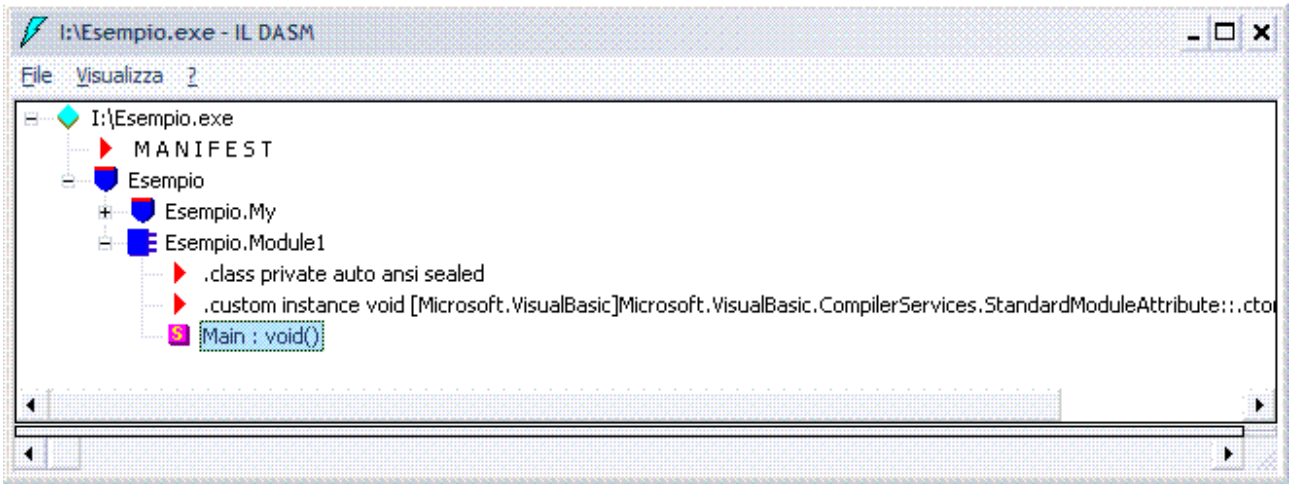
IL_0006: ldc.i4.s 31
Push sullo stack in forma di *Int32* il valore di 8 bit che segue l'istruzione, in questo caso 31, lo stack adesso ha questo aspetto.
n
Risultato della somma *a* + *b*
Numbero 31 ; cima dello stack

IL_0008: and
And è effettuato tra la somma di *a* e *b* e il numero 31; in pratica questo *And* serve ad assicurarsi che il numero non superi 31; l'*And* fa in modo che si possa usare lo shift anche come rotate, se per esempio si volesse shiftare di 32 posizioni, il sistema fa l'*And* con 31 e torna zero, quindi shifta di zero, dato che con le rotate shiftando di 32 posizioni si ritrova il numero non modificato in mano, ad ogni modo, essendo il numero 8, esso resta come è e si ritrova con questo stack.
n
Numbero 8 ; cima dello stack

IL_0009: shr
Shifta a destra *n* di otto posizioni e push il risultato sullo stack, in conclusione il valore di ritorno di un metodo si trova in cima allo stack e non in un registro.
È ovvio non si hanno registri.

Esempio, inserire il valore 1955 per registrare l'applicazione.

```
Module Module1
    Sub Main()
        Dim b As Integer
        Console.Clear()
        Console.Write("Inserire il numero di serie per la registrazione: ")
        b = Convert.ToInt32(Console.ReadLine())
        If (b Xor 666) = 1337 Then
            Console.WriteLine("Grazie per la registrazione")
        Else
            Console.WriteLine("Numero seriale non valido")
        End If
        Console.WriteLine()
        Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione")
        Console.ReadKey()
    End Sub
End Module
```



Doppio clic sul metodo *Main* si ottiene una finestra che contiene il codice seguente.

```
.method public static void  Main() cil managed
{
  .entrypoint
  .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )
  // Code size       98 (0x62)
  .maxstack  2
  .locals init (int32 V_0,
           bool V_1)
  IL_0000:  nop
  IL_0001:  call       void [mscorlib]System.Console::Clear()
  IL_0006:  nop
  IL_0007:  ldstr      "Inserire il numero di serie per la registrazione: "
  IL_000c:  call       void [mscorlib]System.Console::Write(string)
  IL_0011:  nop
  IL_0012:  call       string [mscorlib]System.Console::ReadLine()
  IL_0017:  call       int32 [mscorlib]System.Convert::ToInt32(string)
  IL_001c:  stloc.0
  IL_001d:  ldloc.0
  IL_001e:  ldc.i4     0x29a
  IL_0023:  xor
  IL_0024:  ldc.i4     0x539
  IL_0029:  ceq
  IL_002b:  stloc.1
  IL_002c:  ldloc.1
  IL_002d:  brfalse.s  IL_003c
  IL_002f:  ldstr      "Grazie per la registrazione"
  IL_0034:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_0039:  nop
  IL_003a:  br.s       IL_0048
  IL_003c:  nop
  IL_003d:  ldstr      "Numero seriale non valido"
  IL_0042:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_0047:  nop
  IL_0048:  nop
  IL_0049:  call       void [mscorlib]System.Console::WriteLine()
  IL_004e:  nop
  IL_004f:  ldstr      "Premere un tasto qualsiasi per chiudere l'applicaz"
  + "ione"
  IL_0054:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_0059:  nop
  IL_005a:  call       valuetype [mscorlib]System.ConsoleKeyInfo
[mscorlib]System.Console::ReadKey()
  IL_005f:  pop
  IL_0060:  nop
  IL_0061:  ret
} // end of method Module1::Main
```

IL_0012: call   string [mscorlib]System.Console::ReadLine()
Input stringa.

IL_0017: call   int32 [mscorlib]System.Convert::ToInt32(string)
Converte la stringa in un intero a 32bit.

IL_001c: stloc.0
IL_001d: ldloc.0
Prende la stringa, valore di ritorno della funzione *ReadLine* e la push sullo stack.

IL_001e: ldc.i4 0x29a
Push sullo stack 29AH, 666D.

IL_0023: xor
Fa lo xor tra 666 e il numero in input, il risultato è sullo stack.

IL_0024: ldc.i4 0x539
Pusha sullo stack il valore 539H, 1337D.

IL_002b: stloc.1
IL_002c: ldloc.1
Prende il valore di ritorno, ovvero il numero convertito e lo push sullo stack.

IL_002d: brfalse.s IL_003c
Salta all'istruzione all'offset *IL_003c* se i due valori sullo stack sono diversi, se il salto non è eseguito, ovvero se i valori coincidono, si ha questo il codice seguente.
IL_002f: ldstr "Grazie per la registrazione"
IL_0034: call void [mscorlib]System.Console::WriteLine(string)
IL_0039: nop
IL_003a: br.s IL_0048
br.s è un jmp che porta al termine dell'applicazione, l'applicazione è registrata.
Se invece i valori non coincidono il salto è esegue questa parte di codice.
IL_003c: nop
IL_003d: ldstr "Numero seriale non valido"
IL_0042: call void [mscorlib]System.Console::WriteLine(string)

Per modificare le istruzioni bisogna conoscere gli opcode delle istruzioni e gli indirizzi, è sufficiente andare sul menu dell'ILDASM e fare clic su ***Visualizza/Mostra byte***.

```
.method public static void  Main() cil managed
// SIG: 00 00 01
{
  .entrypoint
  .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )
  // Method begins at RVA 0x27e0
  // Code size       98 (0x62)
  .maxstack  2
  .locals init (int32 V_0, bool V_1)
  IL_0000:  /* 00   |                  */ nop
  IL_0001:  /* 28   | (0A)00003B       */ call       void [mscorlib]System.Console::Clear()
  IL_0006:  /* 00   |                  */ nop
  IL_0007:  /* 72   | (70)000001       */ ldstr   "Inserire il numero di serie per la registrazione: "
  IL_000c:  /* 28   | (0A)00003C       */ call       void [mscorlib]System.Console::Write(string)
  IL_0011:  /* 00   |                  */ nop
  IL_0012:  /* 28   | (0A)00003D       */ call       string [mscorlib]System.Console::ReadLine()
  IL_0017:  /* 28   | (0A)00003E       */ call  int32 [mscorlib]System.Convert::ToInt32(string)
  IL_001c:  /* 0A   |                  */ stloc.0
  IL_001d:  /* 06   |                  */ ldloc.0
  IL_001e:  /* 20   | 9A020000         */ ldc.i4     0x29a
```

```
  IL_0023:  /* 61   |                  */ xor
  IL_0024:  /* 20   | 39050000         */ ldc.i4     0x539
  IL_0029:  /* FE01 |                  */ ceq
  IL_002b:  /* 0B   |                  */ stloc.1
  IL_002c:  /* 07   |                  */ ldloc.1
  IL_002d:  /* 2C   | 0D               */ brfalse.s  IL_003c
  IL_002f:  /* 72   | (70)000067       */ ldstr      "Grazie per la registrazione"
  IL_0034:  /* 28   | (0A)00003F       */ call void [mscorlib]System.Console::WriteLine(string)
  IL_0039:  /* 00   |                  */ nop
  IL_003a:  /* 2B   | 0C               */ br.s       IL_0048
  IL_003c:  /* 00   |                  */ nop
  IL_003d:  /* 72   | (70)00009F       */ ldstr      "Numero seriale non valido"
  IL_0042:  /* 28   | (0A)00003F       */ call void [mscorlib]System.Console::WriteLine(string)
  IL_0047:  /* 00   |                  */ nop
  IL_0048:  /* 00   |                  */ nop
  IL_0049:  /* 28   | (0A)000040       */ call       void [mscorlib]System.Console::WriteLine()
  IL_004e:  /* 00   |                  */ nop
  IL_004f:  /* 72   | (70)0000D3       */ ldstr  "Premere un tasto qualsiasi per chiudere
l'applicazione"
  IL_0054:  /* 28   | (0A)00003F       */ call void [mscorlib]System.Console::WriteLine(string)
  IL_0059:  /* 00   |                  */ nop
  IL_005a:  /* 28   | (0A)000041       */ call       valuetype [mscorlib]System.ConsoleKeyInfo
[mscorlib]System.Console::ReadKey()
  IL_005f:  /* 26   |                  */ pop
  IL_0060:  /* 00   |                  */ nop
  IL_0061:  /* 2A   |                  */ ret
} // end of method Module1::Main
```

Il salto condizionale è il seguente.

```
IL_002d:  /* 2C   | 0D               */ brfalse.s  IL_003c
```

Si può invertire, oppure convertire in salto incondizionale.
In IL non ha molto senso invertire il salto dato che gli opcode hanno la stessa lunghezza, quindi è meglio renderlo incondizionale.
Basta sostituire l'opcode 2CH con 2BH, però in IL le cose sono un po' diverse dall'assembly e si devono fare alcune considerazioni in più che riguardano lo stack.
L'istruzione CMP non fa uso di alcuno stack, ma in IL i valori da confrontare sono sullo stack, quindi se si trasforma un salto condizionale in uno incondizionale che ovviamente non prenderà nessun valore dallo stack, bisogna assicurarsi di non impegnare lo stack.

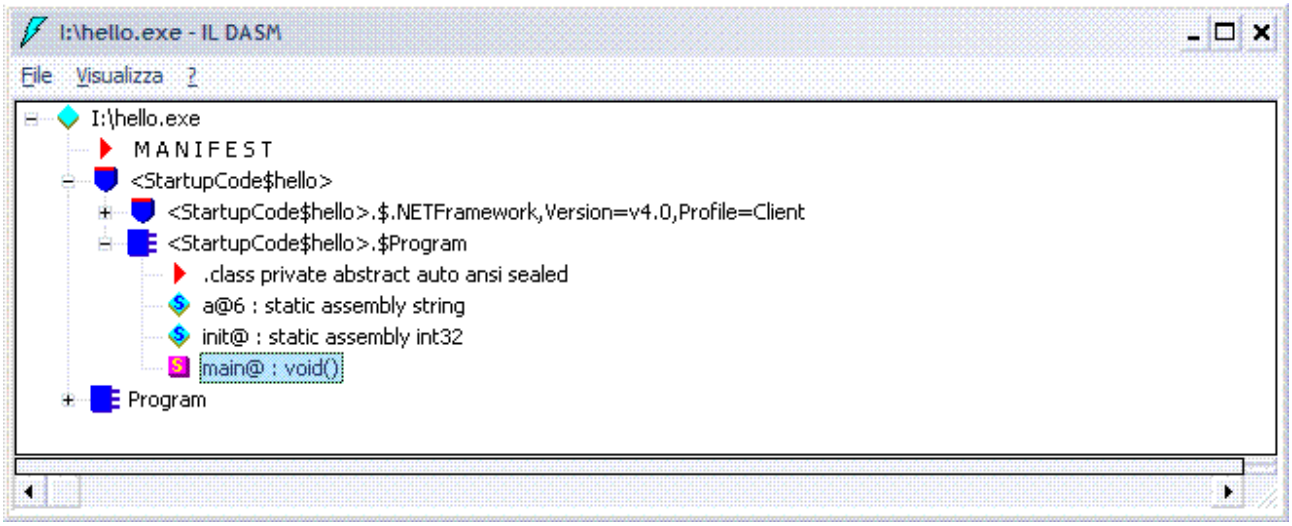
```
IL_0024:  /* 20   | 39050000         */ ldc.i4     0x539
IL_0029:  /* FE01 |                  */ ceq
IL_002b:  /* 0B   |                  */ stloc.1
IL_002c:  /* 07   |                  */ ldloc.1
IL_002d:  /* 2C   | 0D               */ brfalse.s  IL_003c
```

È necessario patchare col *nop* (00H) i due *ld* e poi rimpiazzare 2CH con 2BH.

1. JIT: il codice sorgente

```
// hello.fs
#light
open System
Console.Clear()
Console.WriteLine("Ciao, mondo in .NET")
let a= Console.ReadKey()
```

2. JIT: traduzione in MSIL



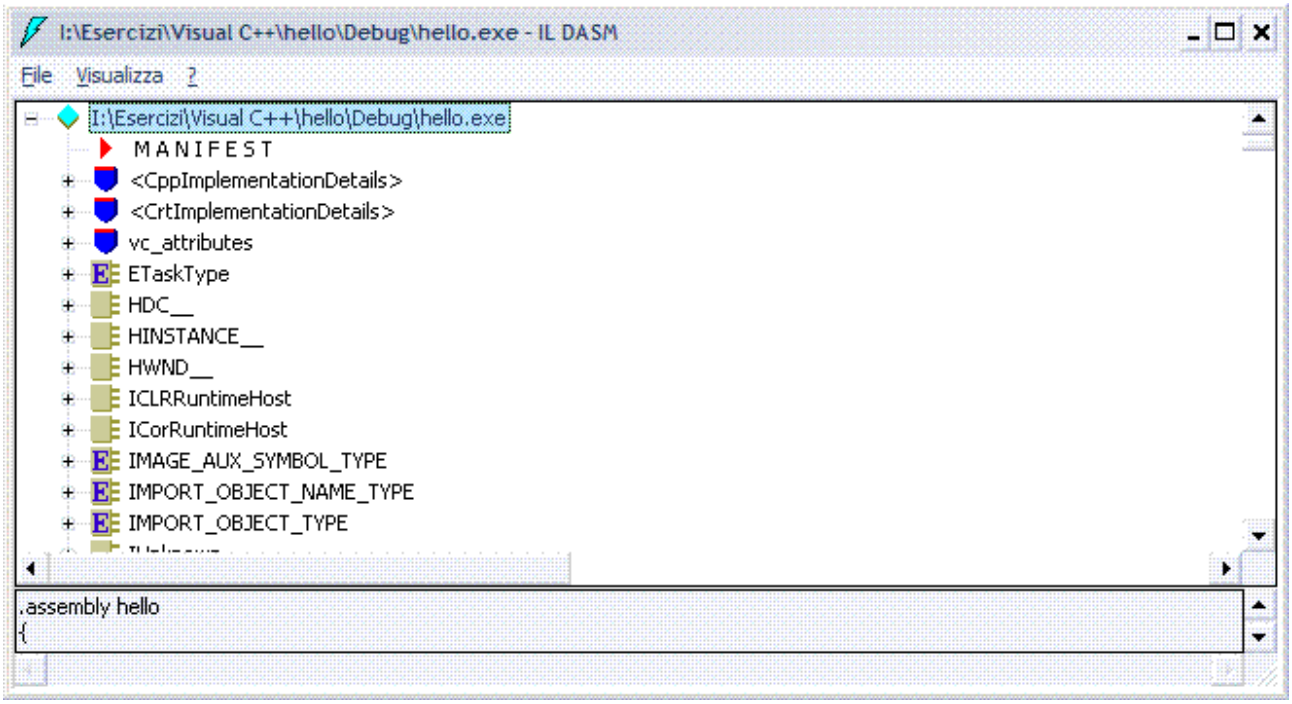
Doppio clic sul metodo *Main* si ottiene una finestra che contiene il codice seguente.

```
.method public static void  main@() cil managed
{
  .entrypoint
  // Code size       29 (0x1d)
  .maxstack  4
  .locals init (valuetype [mscorlib]System.ConsoleKeyInfo V_0)
  IL_0000:  nop
  IL_0001:  call       void [mscorlib]System.Console::Clear()
  IL_0006:  ldstr      "Ciao, mondo in .NET"
  IL_000b:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_0010:  call       valuetype [mscorlib]System.ConsoleKeyInfo
[mscorlib]System.Console::ReadKey()
  IL_0015:  dup
  IL_0016:  stsfld     valuetype [mscorlib]System.ConsoleKeyInfo
'<StartupCode$hello>'.$Program::a@6
  IL_001b:  stloc.0
  IL_001c:  ret
} // end of method $Program::main@
```

1. JIT: il codice sorgente

```
/* hello.c */
#include <stdio.h>
int main (void)
{       printf ("Ciao, mondo in .NET");
        getchar();return(0);
}
```

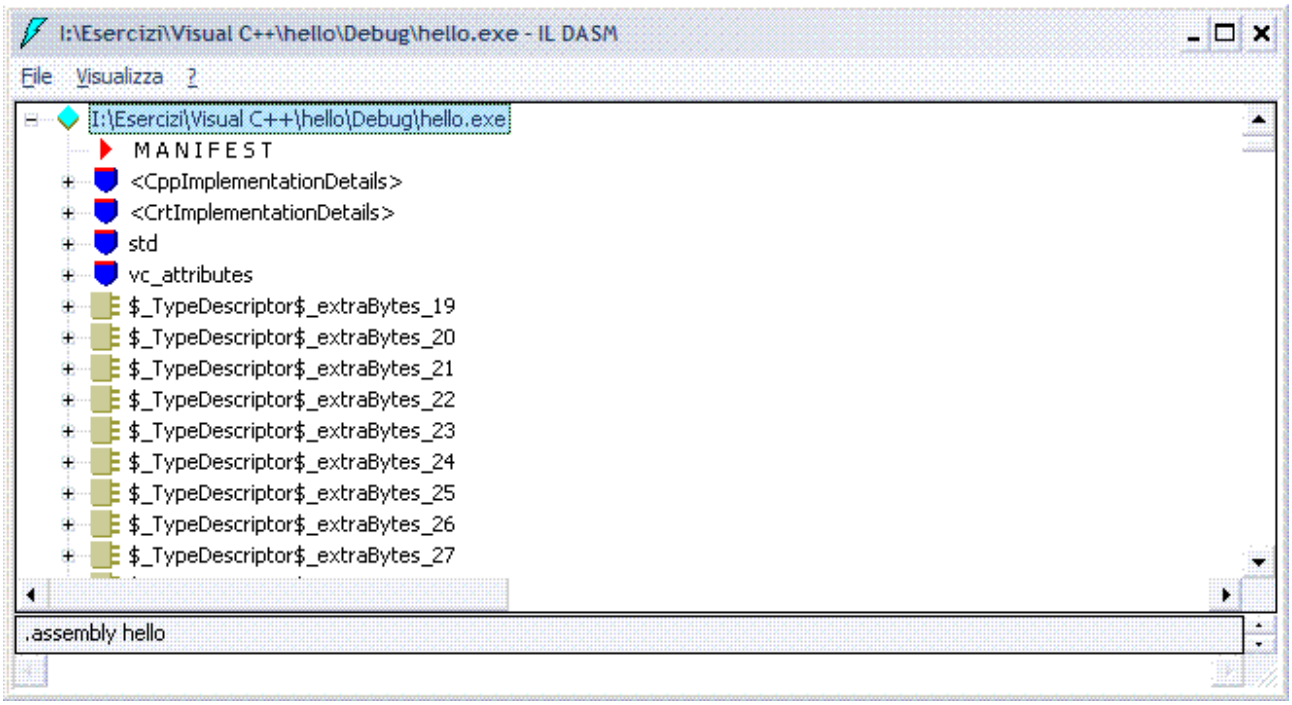
2. JIT: traduzione in MSIL



1. JIT: il codice sorgente

```
// hello.cpp
#include <iostream>
using namespace std;
int main(void)
{       cout<<"Ciao, mondo in .NET";
        getchar();return(0);
}
```

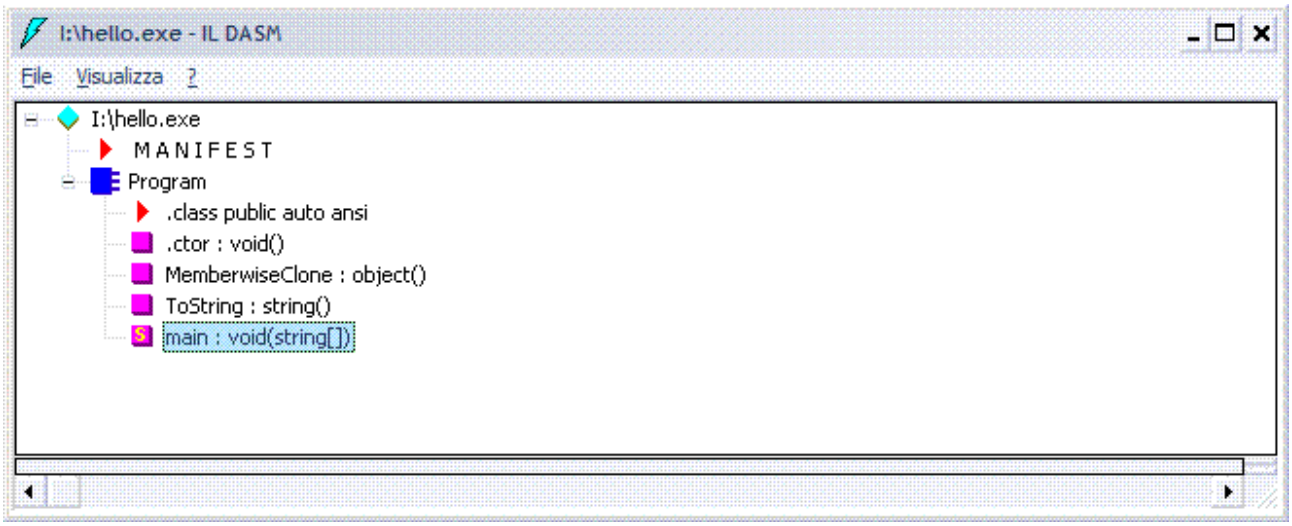
2. JIT: traduzione in MSIL



1. JIT: il codice sorgente

```
// hello.jsl
import System.*;
public class Program
{       public static void main(String[] args)
        {       Console.Clear();
                Console.WriteLine("Ciao, mondo in .NET");
                Console.ReadKey();
        }
}
```

2. JIT: traduzione in MSIL



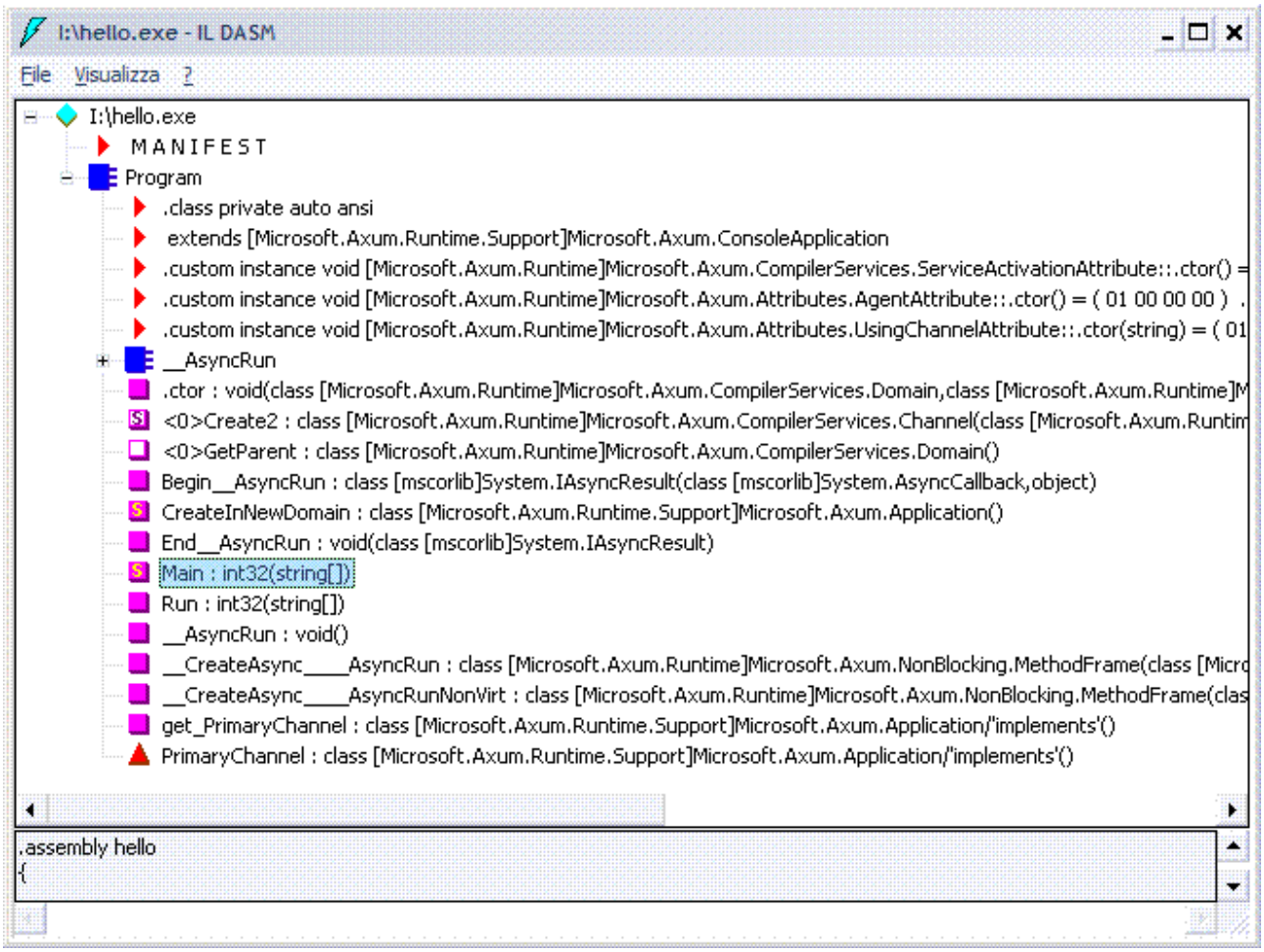
Doppio clic sul metodo *Main* si ottiene una finestra che contiene il codice seguente.

```
.method public hidebysig static void  main(string[] args) cil managed
{
  .entrypoint
  // Code size       39 (0x27)
  .maxstack  1
  IL_0000:  ldtoken    [vjslib]com.ms.vjsharp.lang.ObjectImpl
  IL_0005:  call       void
[mscorlib]System.Runtime.CompilerServices.RuntimeHelpers::RunClassConstructor(value
type [mscorlib]System.RuntimeTypeHandle)
  IL_000a:  call       void [mscorlib]System.Console::Clear()
  IL_000f:  ldstr      "Ciao, mondo in .NET"
  IL_0014:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_0019:  call       valuetype [mscorlib]System.ConsoleKeyInfo
[mscorlib]System.Console::ReadKey()
  IL_001e:  pop
  IL_001f:  call       void [vjslib]com.ms.vjsharp.util.Utilities::cleanupAfterMainReturns()
  IL_0024:  br.s       IL_0026
  IL_0026:  ret
} // end of method Program::main
```

1. JIT: il codice sorgente

```
// hello.ax
using System;
agent Program : Microsoft.Axum.ConsoleApplication
{ override int Run(String[] args)
        { Console.Clear();
         Console.WriteLine("Ciao, mondo in .NET");
        Console.ReadKey();
        }
}
```

2. JIT: traduzione in MSIL



Doppio clic sul metodo *Main* si ottiene una finestra che contiene il codice seguente.

```
.method public static int32  Main(string[] args) cil managed
{
  .entrypoint
 .custom instance void [mscorlib]System.Diagnostics.DebuggerHiddenAttribute::.ctor() =
(01 00 00 00 )
  // Code size       57 (0x39)
  .maxstack  4
  .locals init ([0] int32 V_0,
```

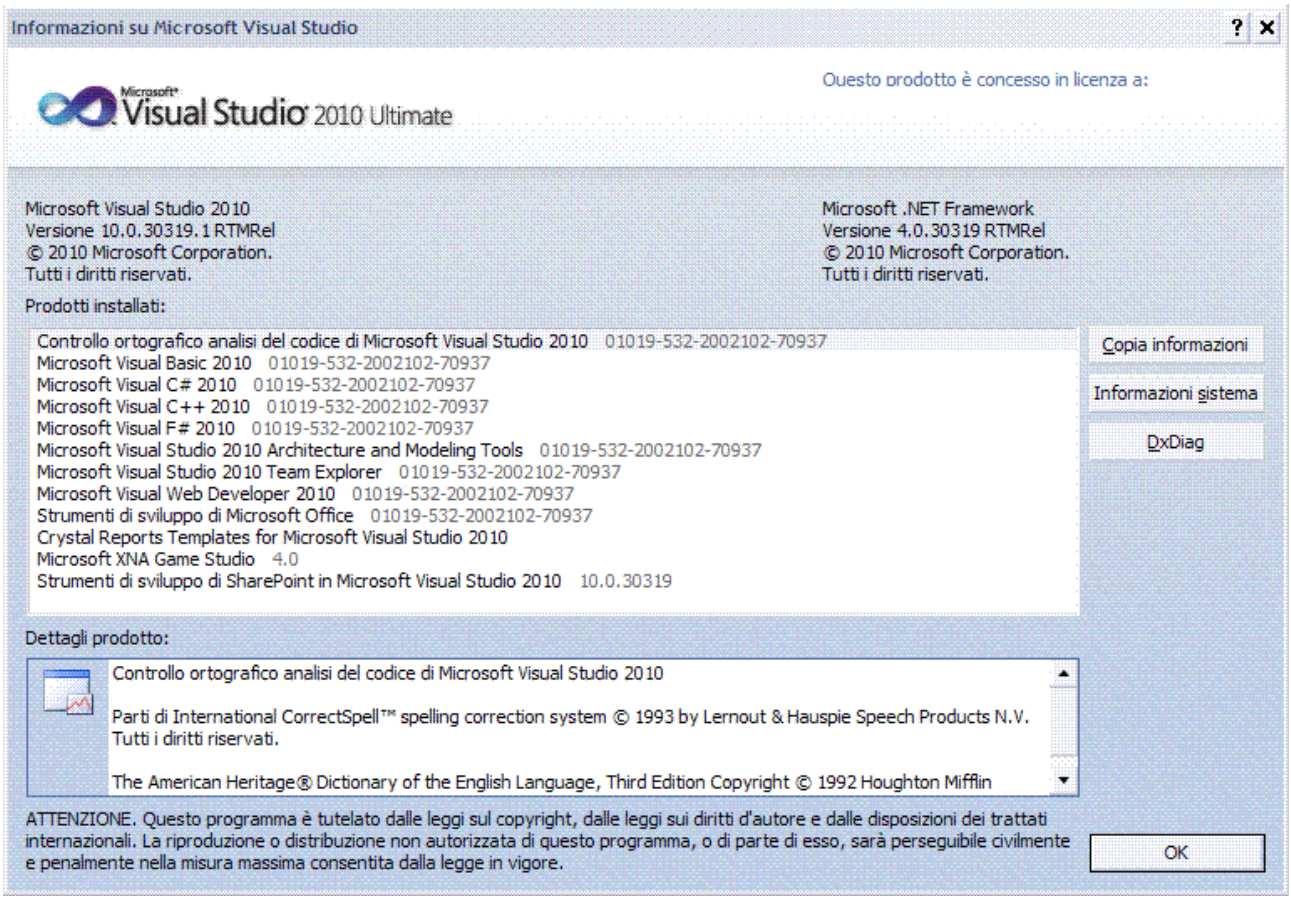
```
        [1] class [Microsoft.Axum.Runtime.Support]Microsoft.Axum.Application channel)
  IL_0000:  call       void
[Microsoft.Axum.Runtime]Microsoft.Axum.CompilerServices.Agent::InitializeApplication()
  IL_0005:  call       class [Microsoft.Axum.Runtime.Support]Microsoft.Axum.Application
Program::CreateInNewDomain()
  IL_000a:  stloc.1
  IL_000b:  ldloc.1
  IL_000c:  call       instance class
[Microsoft.Axum.Runtime]System.Concurrency.Messaging.IInteractionTarget`1<string[]>
[Microsoft.Axum.Runtime.Support]Microsoft.Axum.Application::get_CommandLine()
  IL_0011:  ldarg      args
  IL_0015:  call       void
[Microsoft.Axum.Runtime]Microsoft.Axum.CompilerServices.Agent::asend<string[]>(class
[Microsoft.Axum.Runtime]System.Concurrency.Messaging.IInteractionTarget`1<!!0>,
                                                                                !!0)
  IL_001a:  ldloc.1
  IL_001b:  call       instance class
[Microsoft.Axum.Runtime]System.Concurrency.Messaging.IInteractionSource`1<valuetype
[Microsoft.Axum.Runtime]Microsoft.Axum.Signal>
[Microsoft.Axum.Runtime.Support]Microsoft.Axum.Application::get_Done()
  IL_0020:  ldloc.1
  IL_0021:  call       instance class
[Microsoft.Axum.Runtime]System.Concurrency.Messaging.IInteractionSource`1<int32>
[Microsoft.Axum.Runtime.Support]Microsoft.Axum.Application::get_ExitCode()
  IL_0026:  ldloc.1
  IL_0027:  call       instance class
[Microsoft.Axum.Runtime]System.Concurrency.Messaging.IInteractionSource`1<class
[Microsoft.Axum.Runtime]Microsoft.Axum.Fault>
[Microsoft.Axum.Runtime]Microsoft.Axum.CompilerServices.Network::get_Faults()
  IL_002c:  call       int32
[Microsoft.Axum.Runtime]Microsoft.Axum.CompilerServices.Agent::WaitForCompletionOr
Fault(class
[Microsoft.Axum.Runtime]System.Concurrency.Messaging.IInteractionSource`1<valuetype
[Microsoft.Axum.Runtime]Microsoft.Axum.Signal>,
                                                                                class
[Microsoft.Axum.Runtime]System.Concurrency.Messaging.IInteractionSource`1<int32>,
                                                                                class
[Microsoft.Axum.Runtime]System.Concurrency.Messaging.IInteractionSource`1<class
[Microsoft.Axum.Runtime]Microsoft.Axum.Fault>)
  IL_0031:  stloc.0
  IL_0032:  br         IL_0037
  IL_0037:  ldloc.0
  IL_0038:  ret
} // end of method Program::Main
```

AMBIENTE DI SVILUPPO

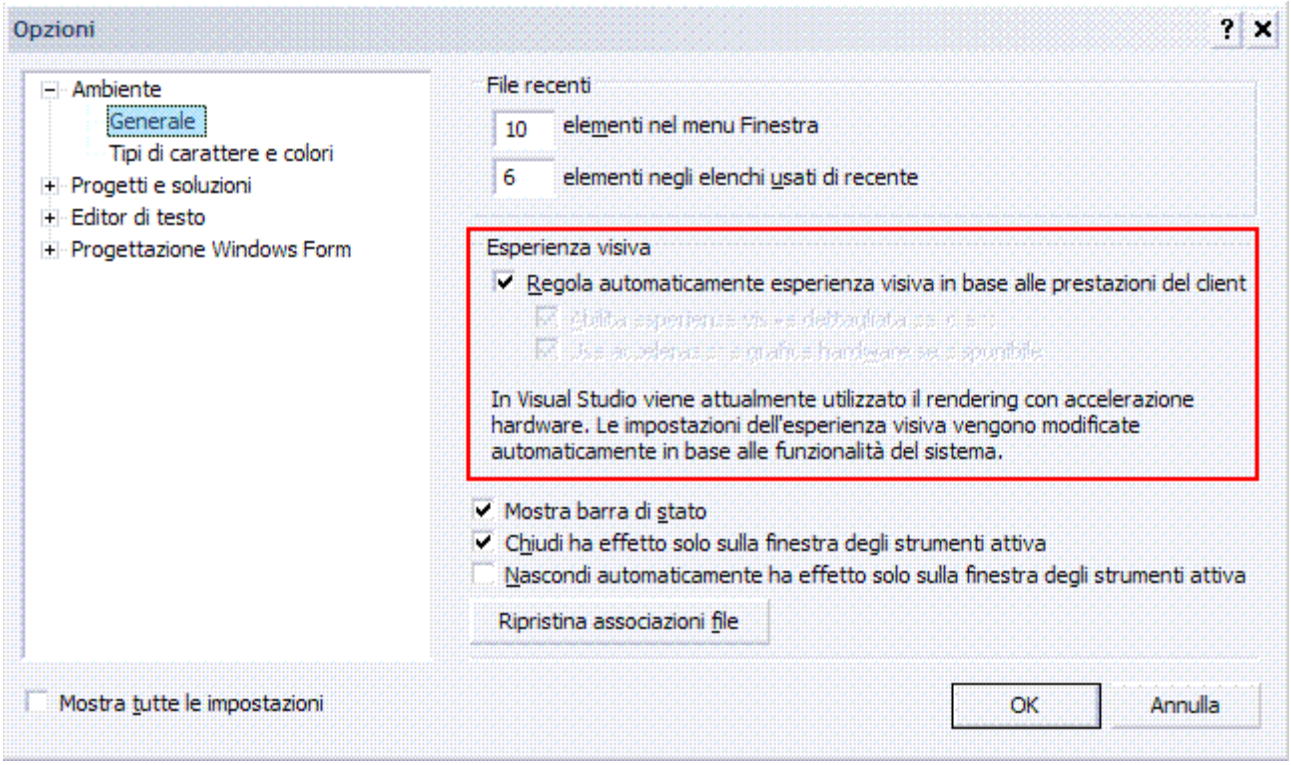
INTRODUZIONE

Il nuovo ambiente di sviluppo di Microsoft, denominato **MDE** (*Microsoft Development Environment*), è un ambiente integrato che racchiude gli editor per tutti i linguaggi di programmazione supportati dalla piattaforma .NET.
È possibile realizzare applicazioni utilizzando più linguaggi all'interno della stessa soluzione.
I diversi linguaggi di programmazione condividono anche la stessa organizzazione dei file sorgenti.

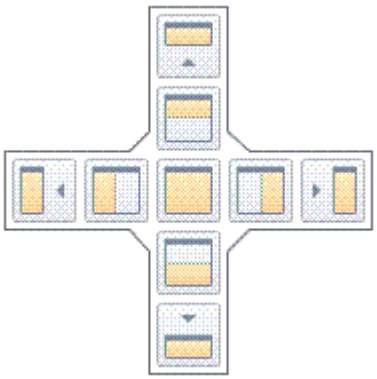
Start/Tutti i programmi/Microsoft Visual Studio 2010/Microsoft Visual Studio 2010



È possibile personalizzare l'ambiente di sviluppo tramite macro scritte con **VBA** (*Visual Basic for Applications*), utilizzando un editor analogo a quello disponibile in Office.
I sorgenti di una stessa applicazione sono raggruppati in progetti.
Avviando per la prima volta Visual Studio, saranno visualizzate nella parte centrale della finestra una serie di opzioni utili per personalizzare l'ambiente di sviluppo; esse saranno comunque accessibili anche in futuro all'interno della finestra ***Strumenti/Opzioni...***
Per esempio, la possibilità di usare l'accelerazione hardware, opzione che è automaticamente abilitata.
Funzionalità di zoom del codice, tasto CTRL e rotellina del mouse.



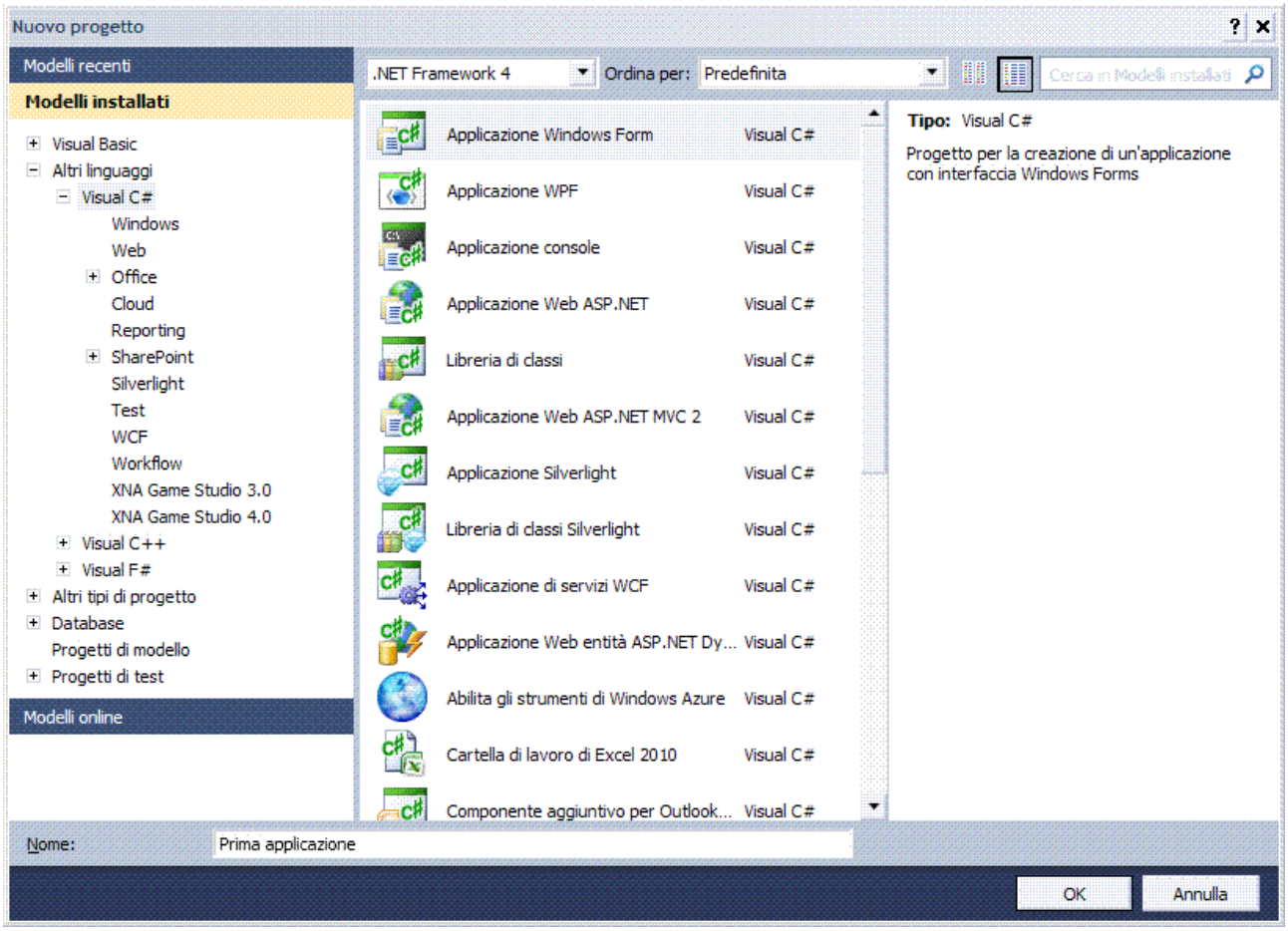
Possibilità di modificare la disposizione di finestre e pannelli grazie ai gadget per l'ancoraggio che appaiono quando si trascinano le finestre.
Lo strumento appare come una croce che indica quale posizione avrà la finestra rispetto al pannello sottostante.
Ad esempio, se si rilascia il tasto del mouse sul simbolo al centro della croce, si sovrappone la finestra trascinata a quella sottostante e si crea così un pannello con più schede, per passare da una finestra all'altra si usano le relative linguette.



Una volta personalizzato l'ambiente di sviluppo, c'è la possibilità di conservare e riapplicare tutte le preferenze, grazie alle funzionalità del seguente menu ***Strumenti/Importa/Esporta impostazioni…***, che permettono anche di effettuare il reset di tutte le impostazioni.
Questo consente di utilizzare impostazioni diverse a seconda del progetto su cui si sta lavorando.
C'è il supporto a chi ha più di un monitor, è possibile estrarre dal layout anche le finestre di codice e spostarle tra i diversi schermi.
La possibilità di visualizzare contemporaneamente più finestre di codice, designer, permette una visione d'insieme senza precedenti.

Per creare un nuovo progetto, fare clic su ***File/Nuovo/Progetto… (CTRL+N)***

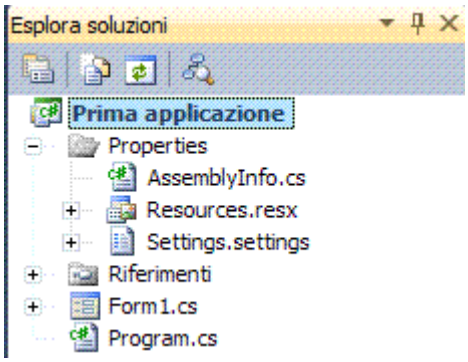
È visualizzata la finestra di dialogo seguente, nella parte superiore è possibile scegliere il .NET Framework che si vuole utilizzare, versioni 2.0, 3.0, 3.5, 4.0; in base alla versione scelta, si troveranno nella finestra di dialogo solamente i progetti compatibili.
È possibile scegliere un progetto da un elenco di modelli online.



Nella finestra di dialogo ***Nuovo Progetto*** selezionare un linguaggio a scelta dall'elenco, per esempio ***Visual C#***, quindi selezionare il tipo di applicazione, per esempio ***Applicazione Windows Forms***.
Immettere il nome del progetto nel campo ***Nome:***, per esempio ***Prima applicazione***, quindi per creare il progetto, premere ***OK***.
L'ambiente di lavoro è costituito dai seguenti componenti.

Esplora soluzioni

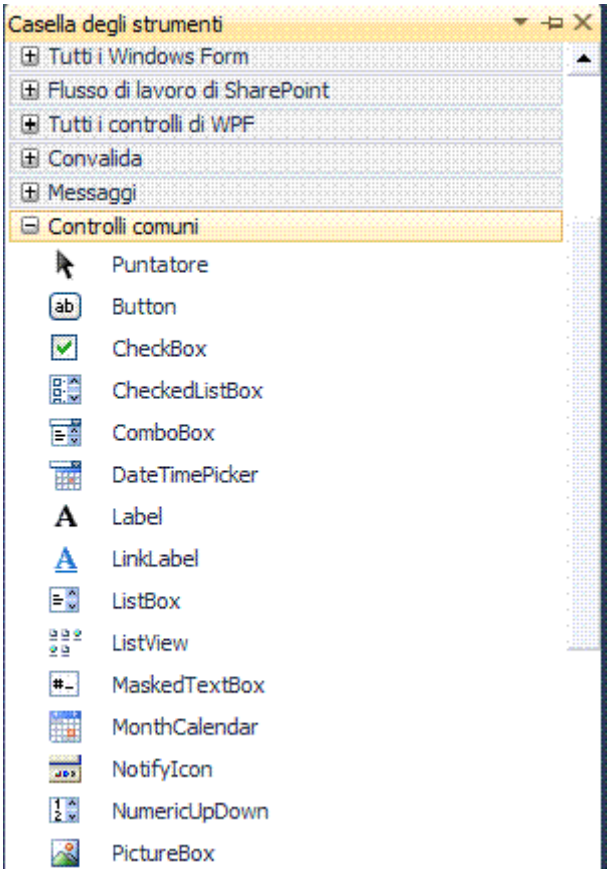
Fare clic su ***Visualizza/Esplora soluzioni (CTRL+ALT+L)***.
Presenta l'organizzazione gerarchica dei componenti del progetto con nome *Prima applicazione*: finestre, codice e moduli.



Casella degli strumenti

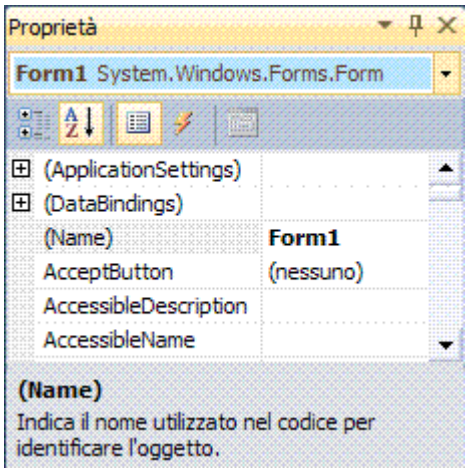
I controlli della casella degli strumenti rappresentano le classi.
Quando si crea un controllo, è creata una copia o istanza della classe di controllo.
I controlli sono strumenti, quali caselle, pulsanti ed etichette disegnate su un form per ricevere input o visualizzare output e consentono, inoltre, di migliorare l'aspetto dei form.
La casella degli strumenti contiene un insieme di tools che è possibile usare per disegnare, muovere, o ridimensionare gli oggetti nel form.
Si può scegliere lo strumento voluto semplicemente cliccando su di esso.

Fare clic su ***Visualizza/Casella degli strumenti (CTRL+ALT+X)***  .
La casella degli strumenti contiene l'elenco dei componenti utilizzabili che possono essere inseriti all'interno del form e prendono il nome di controlli.

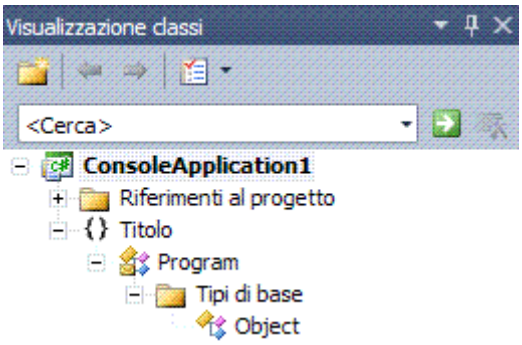


Proprietà

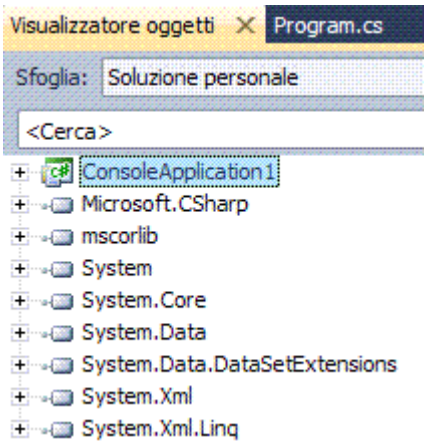
Fare clic su ***Visualizza/Finestra Proprietà (F4)***.
Mostra le proprietà dell'oggetto selezionato.



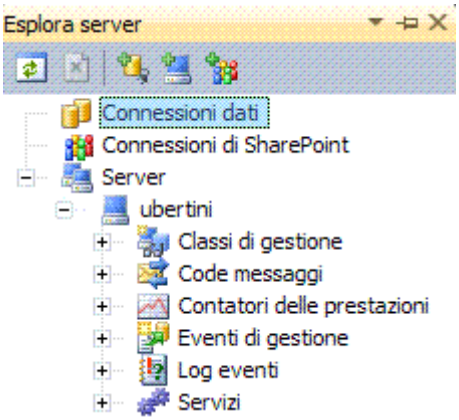
Visualizzazione classi
Fare clic sul menu ***Visualizza/Visualizzazione classi (CTRL+MAIUSC+C)***.



Visualizzatore di oggetti
Fare clic sul menu ***Visualizza/Visualizzatore oggetti (F2)***.
La finestra consente di esaminare e trovare spazi dei nomi, classi, strutture, interfacce, tipi, enumerazioni, sono inoltre disponibili informazioni su membri, proprietà, metodi, eventi, variabili, costanti ed elementi di enumerazioni di svariati componenti.



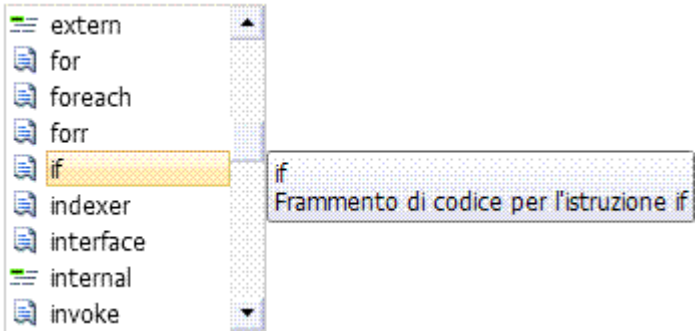
Esplora server
Fare clic sul menu ***Visualizza/Esplora server (CTRL+ALT+S)***.
La finestra consente permette di accedere a fonti di dati.



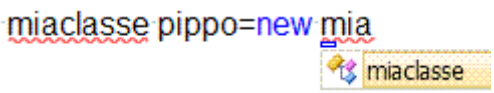
IntelliSense
Permette d'inserire del codice li dove è prevedibile che sia inserito.
Semplifica il completamento automatico d'istruzioni di codice evitando di digitare parole intere, si attiva non appena s'inizia a digitare nell'editor.
Dispone di numerose funzioni quali ad esempio un elenco a discesa di membri delle classi e delle strutture dei *namespace*, ciò offre due importanti vantaggi.

1. Non si deve ricordare tutti i membri disponibili per la classe, poiché è sufficiente scorrere l'elenco e trovare il membro che si deve utilizzare.
2. Si prevengono gli errori di sintassi, poiché non si deve digitare il nome del membro e non si rischia di commettere errori di digitazione.

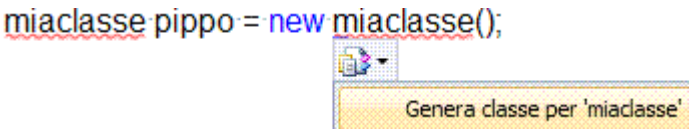
Per selezionare il membro desiderato, si deve premere il tasto TAB o INVIO, oppure fare doppio clic sul membro stesso.



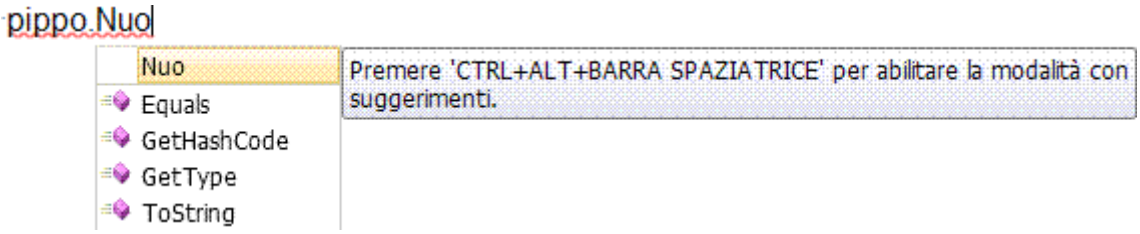
Se si tenta di utilizzare una classe chiamata *miaclasse* che non esiste nel progetto.



L'intellisense la propone come se fosse in realtà già stata dichiarata.
Al termine della digitazione, nel menu contestuale si può scegliere di generare la nuova classe, d'altra parte se questo è quello che il programmatore ha digitato, probabilmente è quello che vuole.



Ora che la nuova classe è stata creata, se si scrive *pippo*, l'intellisense propone gli unici metodi attualmente disponibili per la classe, quelli ereditati da *System.Object*.
Premendo ***CTRL+ALT+BARRA SPAZIATRICE***, però, l'intellisense presenta un'interfaccia leggermente differente, in cui la prima opzione lista quello che si sta digitando.

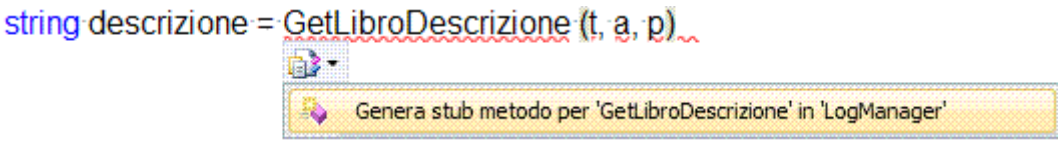


Se ad esempio si scrive una chiamata a un metodo che non esiste ancora nel *namespace* corrente: metodo *GetLibroDescrizione*.

```
string t = "";
```

```
string a = "";
decimal p = 0;
string descrizione = GetLibroDescrizione (t, a, p)
```

L'editor automaticamente individua l'assenza del metodo nella classe e quindi mostra grazie all'intellisense una sottolineatura.



Selezionando questa voce di menu sarà creato lo scheletro di un metodo che avrà la stessa firma di quello scritto.
Automaticamente saranno riconosciuti i tipi dei parametri e di conseguenza saranno incluse nella firma le corrette definizioni.

```
private static string GetLibroDescrizione(string t,string a,decimal p)
{ throw new System.NotImplementedException(); }
```

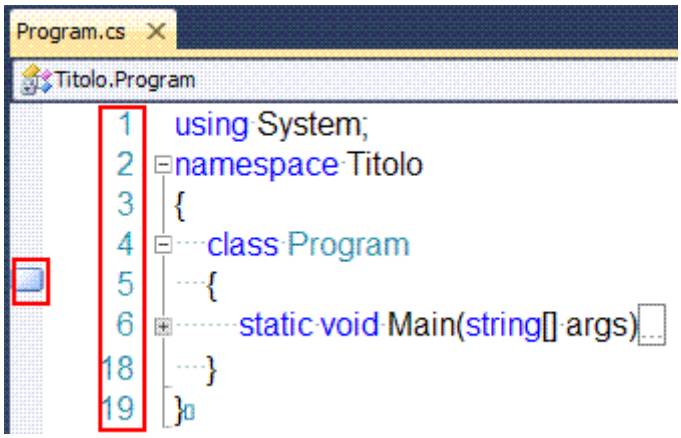
Ovvero lo scheletro del metodo *GetLibroDescrizione* con i tre parametri già tipizzati, non resta quindi altro da fare che riempire il metodo con il codice opportuno.

A capo automatico
Fare clic su ***Modifica/Avanzate/A capo automatico***.
Tutte le righe di codice che superano la lunghezza dell'editor sono automaticamente mandate a capo, per evitare la necessità di scorrere lateralmente la finestra dell'editor.

Segnalibri
Fare clic su ***Modifica/Segnalibri/Attiva/Disattiva segnalibro (CTRL+K+T)*** .

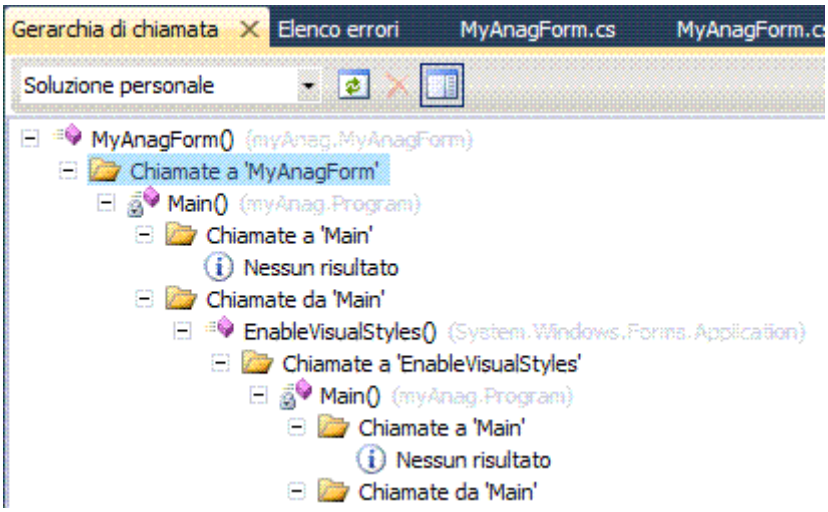
Numeri di riga
Fare clic su ***Strumenti/Opzioni…/Editor di testo/C#/Editor/Numeri di riga***.
I numeri di riga sono aggiunti al codice.

Editor
Le varie parti del codice possono essere compresse ed espanse facendo clic rispettivamente sul simbolo - (meno) e + (più) che è automaticamente visualizzato a destra delle dichiarazioni di un metodo, di un insieme di righe di commento.

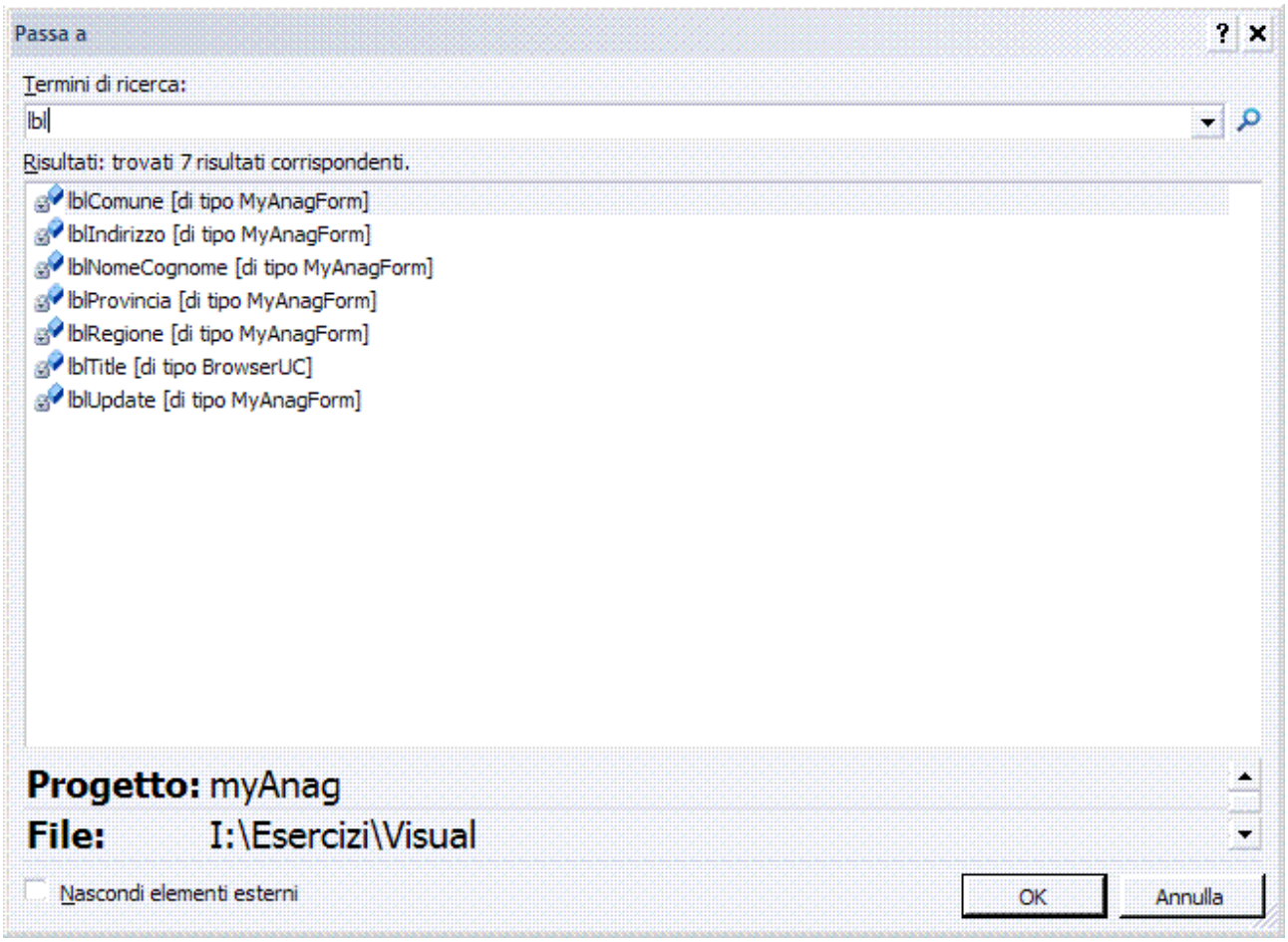


Fare clic con il tasto destro e selezionare ***Visualizza gerarchia di chiamata (CTRL+K+T)***

si vedono tutti i punti nel codice dove quel particolare oggetto è chiamato, in questo modo si può effettuare una navigazione molto profonda nel codice e comprendere meglio il pattern di utilizzo di una particolare funzione o proprietà.
In pratica è un grafo padre/figlio: il grafo rappresenta la gerarchia delle chiamate all'interno del sorgente.



Premendo ***CTRL+,*** si apre la finestra ***Passa a***, nella quale si può digitare parte del nome di una classe, metodo o proprietà e lasciare l'editor mostri tutte le corrispondenze.

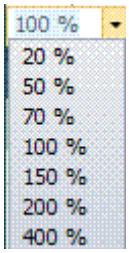


Ogni volta che si posiziona il cursore su di una variabile o oggetto, Visual Studio evidenzia nel codice tutte le volte che quella variabile è utilizzata.

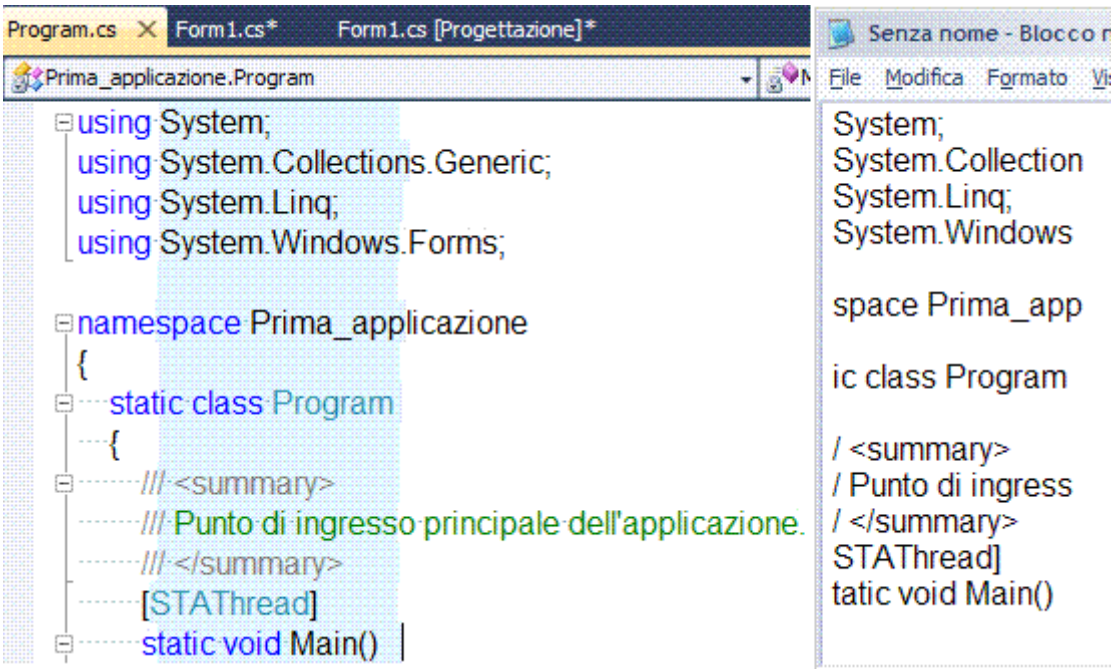
```
public static void carica()
{   int riga, colonna;
    for (riga = 0; riga < R; riga++)
        for (colonna = 0; colonna < C; colonna++)
        {   Console.WriteLine("Inserire l'elemento della riga {0} e colonna {1}: ", riga, colonna);
            mat[riga, colonna] = Convert.ToInt32(Console.ReadLine());
```

Quindi possibilità di navigare un tipo definito rispetto al suo utilizzo; in pratica premendo CTRL+SHIFT+FRECCIA SU/GIÙ è possibile scorrere tutti i riferimenti alla classe su cui si è posizionati.
È stata anche semplificata la ricerca tramite, ***Trova tutti i riferimenti (ALT+F2)***, attivabile tramite il menu contestuale sul nome di una classe.

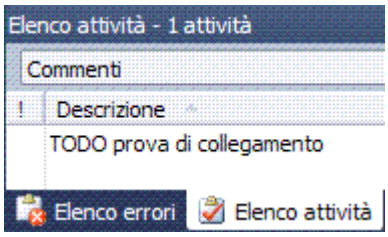
È possibile effettuare lo zoom del codice tenendo premuto il tasto ***CTRL*** e usando la rotellina del mouse, utile se si è in presenza di un metodo molto lungo, si può ridurre il livello di zoom temporaneamente per tenere sotto occhio una parte di codice più ampia.



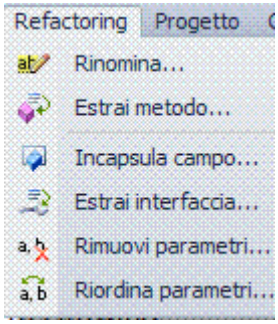
È possibile selezionare una parte di testo qualsiasi, premere il tasto ***ALT*** e il pulsante sinistro del mouse, tenere premuto il pulsante sinistro e spostare il mouse da sinistra a destra e dall'altro verso il basso, per selezionare solo la parte di codice desiderata, questa selezione può essere copiata ed incollata in altre selezioni rettangolari.



TODO
Se s'inserisce una stringa di nome ***TODO*** dopo l'indicatore di commento, nell'elenco attività è visualizzato un collegamento al commento.
Per aprire l'elenco attività, fare clic su ***Visualizza/Elenco attività (CTRL+ALT+K).***
Per passare al commento nell'editor, fare doppio clic.



Refactoring
È una modifica al codice sorgente di un'applicazione che ne migliora la leggibilità o ne semplifica la struttura senza però modificarne il funzionamento e quindi il risultato.
Prima di Visual Studio, il lavoro di refactoring del codice era abbastanza lungo e laborioso e richiedeva un pesante intervento manuale nel codice.
Inoltre, l'intervento manuale facilmente poteva portare all'introduzione di errori senza contare la consistente perdita di tempo, ora è possibile effettuare questo processo con pochi clic del mouse e modificare un intero progetto senza che questo comporti l'introduzione di errori e senza perdita di tempo.
Fare clic sul menu ***Refactoring***, oppure nell'editor fare clic con il pulsante destro.

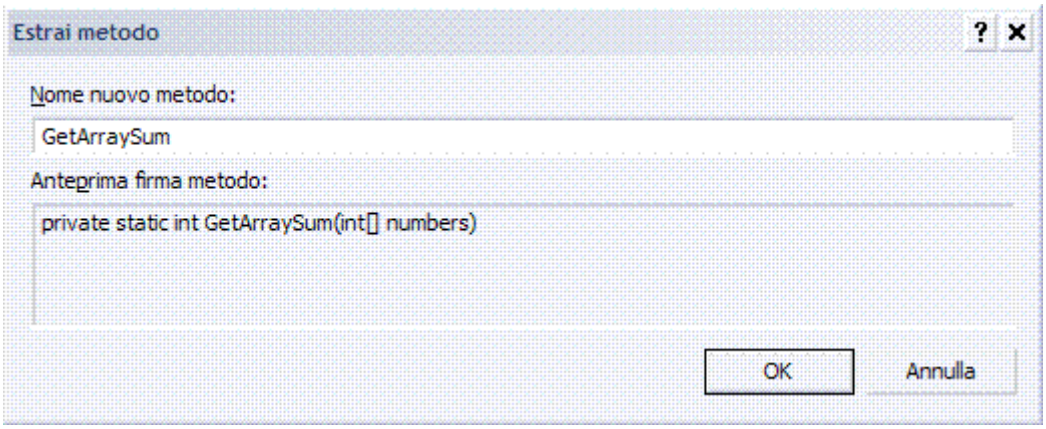


Rinomina…
Permette di rinominare un metodo, un *namespace*, un parametro o un tipo, apportando la modifica all'intero progetto in cui è contenuto l'elemento modificato.
Naturalmente questo comando di refactoring non esegue una semplice find e replace ma considera il particolare elemento selezionato e quindi apporta le dovute sostituzioni.
Richiede solo di specificare il nuovo nome da dare all'elemento selezionato e, come per altri comandi di refactoring, richiede di specificare se apportare le modifiche anche nei commenti e nelle stringhe testuali, oltre naturalmente alla possibilità di visualizzare una finestra di preview prima di procedere con l'operazione.
Nel caso specifico della ridenominazione di un metodo, si può anche scegliere se rinominare gli overload dello stesso oppure no.

Estrai metodo…
Permette di estrapolare un nuovo metodo da una porzione di codice esistente.
Accade sovente durante lo sviluppo di scrivere porzioni di codice che in qualche modo possono essere riutilizzate in altre parti del progetto.
Sarebbe quindi scomodo e ridondante ripetere diverse volte le stesse righe di codice, senza contare il fatto che in caso di modifica dovremmo necessariamente modificare tutti i punti nel progetto in cui queste righe di codice sono state utilizzate.

```
public class MyClass
      {  public MyClass()
         { }
         public decimal GetAverage(int[] numbers)
         {  int sum = 0;
            foreach (int number in numbers)
            {  sum += number;  }
            return (decimal)sum / (decimal)numbers.Length;
         }
      }
```

Questa classe contiene il metodo *GetAverage* che preso un array di valori *int* ne calcola la media aritmetica.
I momenti del calcolo sono due: il primo quando calcola la somma di tutti i valori dell'array, il secondo quando divide questa somma per il numero di elementi.
È evidente che il calcolo della somma degli elementi è un'operazione di cui si può aver bisogno in altre parti dell'applicazione, è quindi opportuno separare questo codice dal metodo *GetAverage* affinché possa vivere di vita propria ed essere riutilizzato.
Per fare questo selezionare il comando ***Estrai metodo…*** selezionando le righe di codice relative al calcolo della somma e cliccando con il tasto destro del mouse sulla selezione.



Appare una finestra di dialogo nella quale si deve specificare solo il nome del nuovo metodo da creare e si può vedere anche un'anteprima di come apparirà la firma del metodo, premuto quindi il tasto ***OK*** il codice è modificato nel modo seguente.

```
public class MyClass
{  public MyClass()
   { }
   public decimal GetAverage(int[] numbers)
   {  int sum = GetArraySum(numbers);
      return (decimal)sum / (decimal)numbers.Length;
   }
   private static int GetArraySum(int[] numbers)
   {  int sum = 0;
      foreach (int number in numbers)
      { sum += number; }
      return sum;
   }
}
```

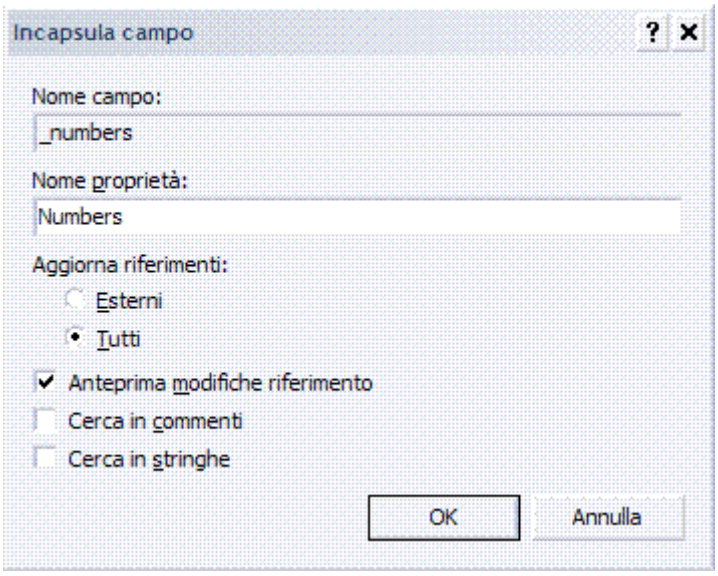
Visual Studio ha creato un nuovo metodo *GetArraySum* contenente le righe di codice da

selezionate precedentemente, poi ha creato un parametro del metodo corrispondente all'array d'input su cui calcolare la somma e ha inserito come valore di ritorno del metodo la variabile *sum*.
Fatto questo ha sostituito tutto il codice selezionato con la chiamata al nuovo metodo passandogli come parametro proprio l'array di numeri da sommare.
A questo punto si può togliere il metodo *GetArraySum* dalla classe e posizionarlo in una classe *Helper* da cui potrà essere richiamato molteplici volte da altre classi del progetto.

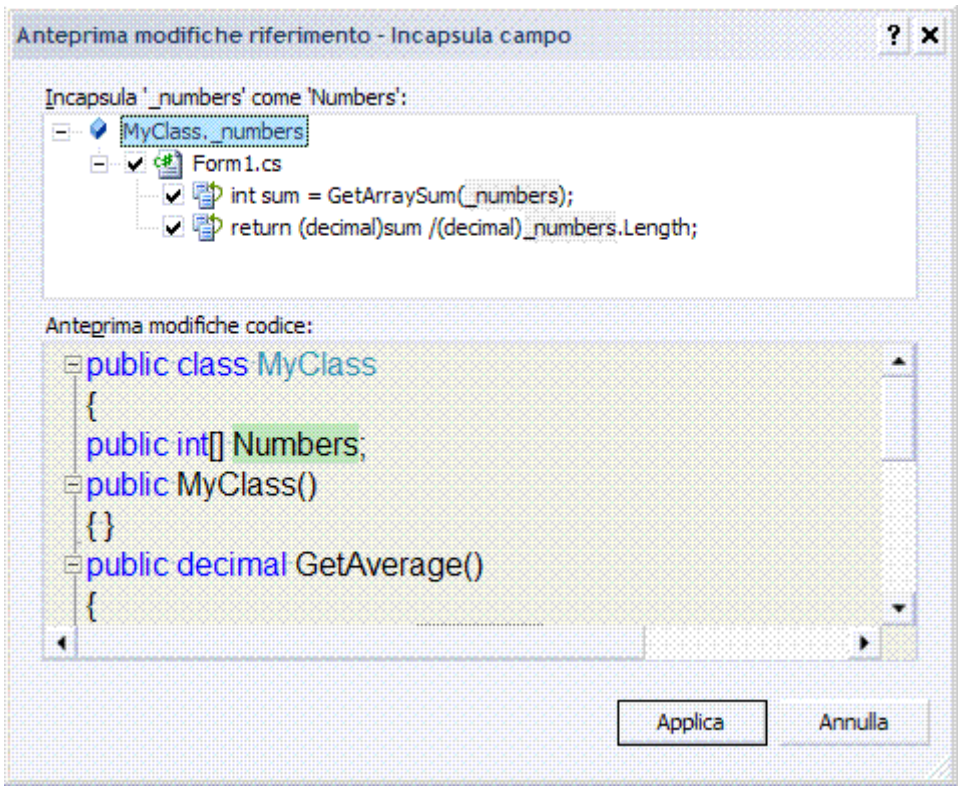
Incapsula campo…
Permette d'incapsulare un campo pubblico all'interno di una proprietà utilizzando un campo privato come variabile di appoggio.
Non è corretto definire un campo pubblico all'interno di una classe perché questo viola il principio dell'incapsulamento e non permette di avere il controllo dei valori assegnati al campo.
Nell'esempio della classe che calcola la media, si è modificata in modo tale da contenere un campo che permette di specificare l'array sul quale calcolare la media piuttosto che passarlo come parametro dal relativo metodo.

```
public class MyClass
{ public int[] _numbers;
  public MyClass()
{ }
public decimal GetAverage()
{ int sum = GetArraySum(_numbers);
  return (decimal)sum /(decimal)_numbers.Length;
}
}
```

È sufficiente cliccare con il tasto destro del mouse sul campo pubblico che si vuole incapsulare e scegliere solo tra alcune opzioni all'interno della finestra di dialogo che appare, per esempio *public int[] _numbers;.*
Si deve nell'ordine selezionare il nome da dare alla proprietà, il modo in cui deve essere effettuato l'aggiornamento dei riferimenti, se visualizzare una preview prima di apportare le modifiche, se effettuare le sostituzioni anche nei commenti e nelle stringhe di testo.
La modalità di aggiornamento ***Esterni*** permette di aggiornare tutti i riferimenti del campo pubblico nel progetto eccetto quelli della classe stessa in cui il campo è contenuto, mentre la modalità ***Tutti*** aggiorna tutti i riferimenti al campo, anche quelli presenti nella stessa classe in cui è definito.



Fare clic su ***OK***, è visualizzata l'anteprima.



Fare clic su ***Applica*** e il codice è modificato nel modo seguente.

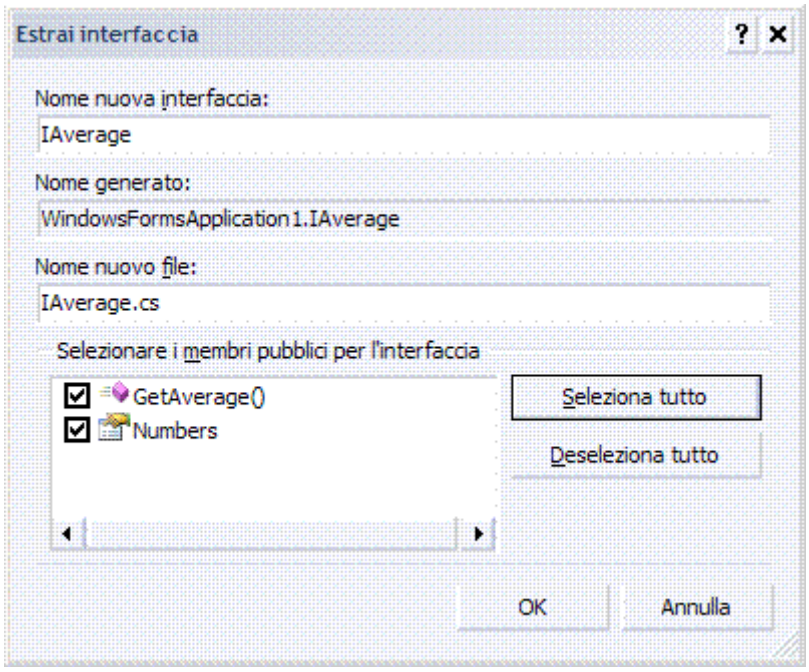
```
public class MyClass
{ private int[] _numbers;
public int[] Numbers
{ get { return _numbers; }
  set { _numbers = value; }
}
public MyClass()
{ }
public decimal GetAverage()
{int sum = GetArraySum(Numbers);
  return (decimal)sum /(decimal)Numbers.Length;
```

```
}
}
```

Le modifiche apportate sono: *private* al posto di *public* sul campo *_numbers*, la creazione della nuova proprietà *Numbers* che espone il campo privato *_numbers* e la sostituzione nel codice di tutti i riferimenti a *_numbers* in riferimenti alla nuova proprietà *Numbers*.

Estrai interfaccia…

Nel momento in cui una classe presenta metodi, proprietà o eventi che potrebbero essere fatti propri da altre classi, automaticamente si pensa di creare un'interfaccia da cui far derivare la classe: questo comando estrae un'interfaccia da una classe.
È sufficiente posizionarsi con il cursore all'interno di una classe e cliccare in un punto vuoto qualsiasi con il tasto destro del mouse e nell'ordine specificare il nome da dare all'interfaccia, il nome del file su disco che la conterrà, i metodi da includere nell'interfaccia.
Nella classe per il calcolo delle medie aritmetiche è evidente che si possono individuare dei metodi e proprietà generiche che possono essere utilizzati anche per il calcolo di medie diverse da quella aritmetica, per esempio geometrica, quadratica.
Selezionare la proprietà *Numbers* ed il metodo *GetAverage()* e generare un'interfaccia che si chiami *IAverage*.



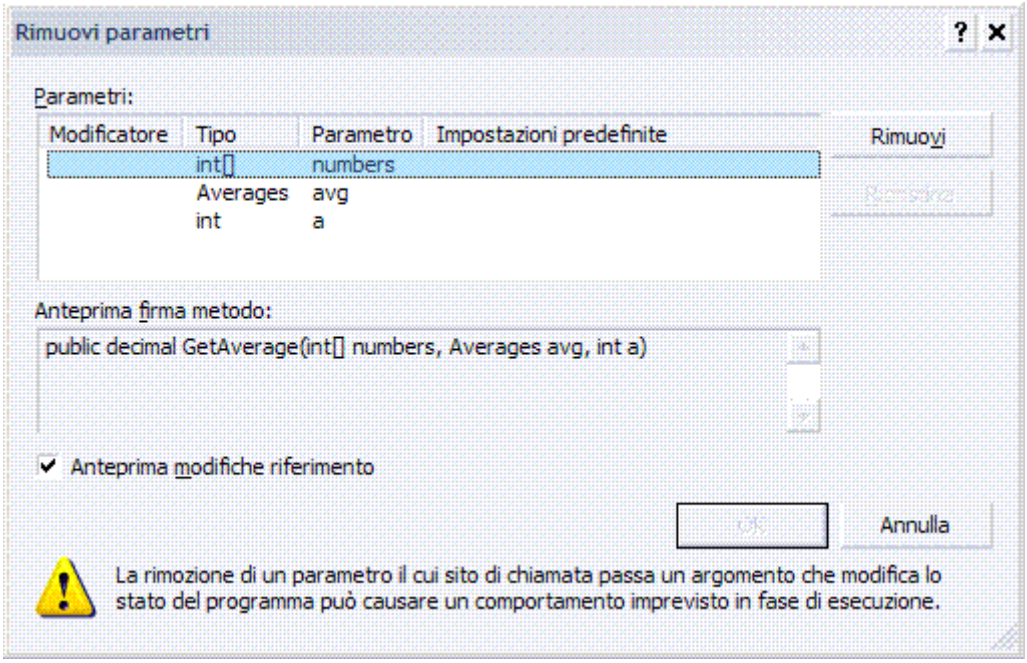
Fare clic su ***OK***, è generato il file IAVERAGE.CS contenente la seguente interfaccia.

```
using System;
namespace WindowsFormsApplication1
{  interface IAverage
   {   decimal GetAverage();
       int[] Numbers { get; set; }
   }
}
```

La classe è fatta ereditare da *IAverage*.

```
public class MyClass : WindowsFormsApplication1.IAverage
```

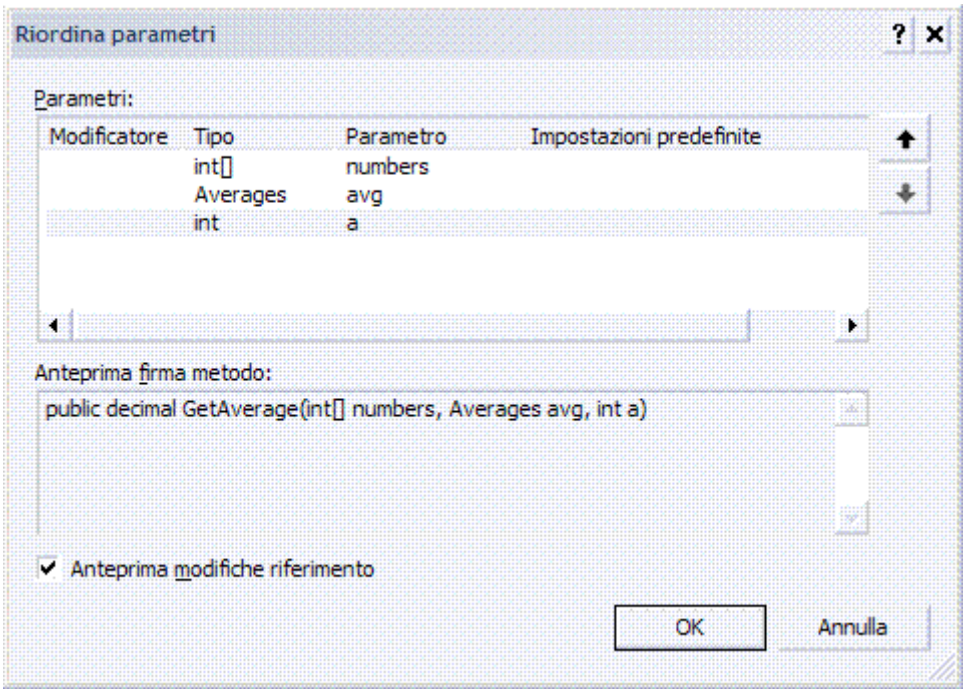
Rimuovi parametri…
Rimuove un parametro da un metodo aggiornando di conseguenza tutte le chiamate al metodo stesso.
Selezionare il metodo su cui agire dal menu contestuale, è visualizzata la finestra di dialogo che chiede di specificare i parametri da rimuovere o eventualmente ripristinare, nello stesso momento, non dopo che la modifica è stata apportata.
Come anche il messaggio di alert dice nella finestra di dialogo, bisogna fare attenzione nel rimuovere parametri che sono stati utilizzati all'interno del metodo stesso perché eliminandoli si potrebbero creare situazioni di errore dovute al fatto che il parametro è utilizzato senza essere stato dichiarato.



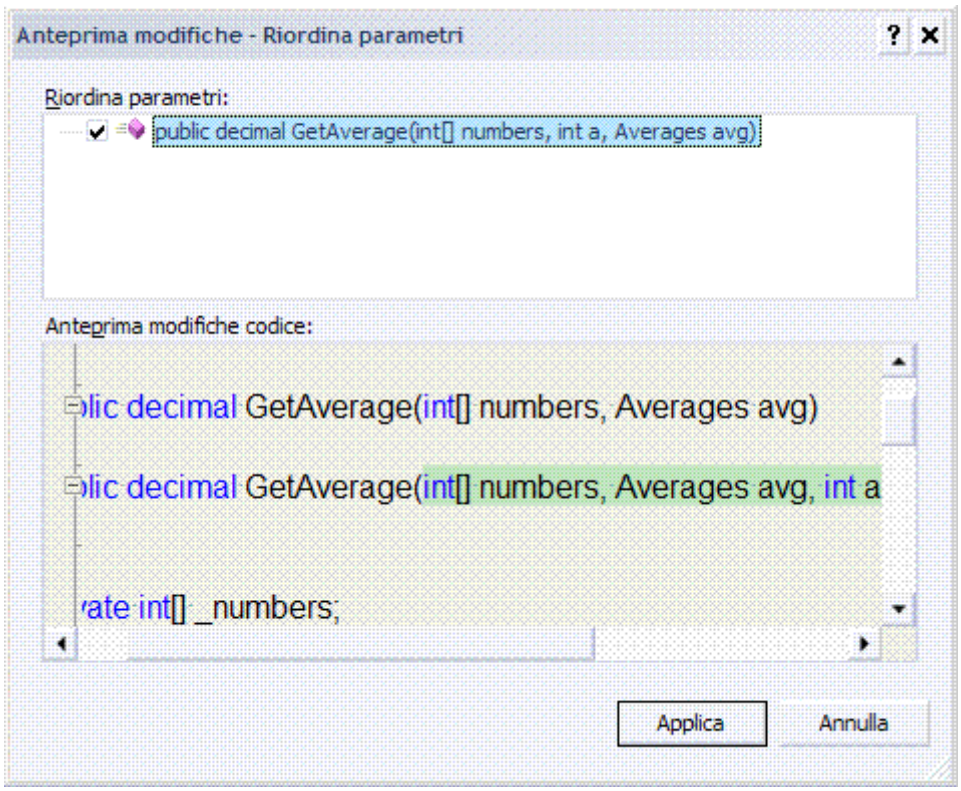
Riordina parametri…
Consente di riordinare i parametri di un metodo.
Per esempio, dati una serie di metodi in overload.

```
public decimal GetAverage( int[] numbers)
{ }
public decimal GetAverage( int[] numbers, Averages avg)
{ }
public decimal GetAverage( int[] numbers, int a,Averages avg)
{ }
```

Si vede che l'ultimo overload ha il parametro *a* prima del parametro *avg* ma sarebbe più corretto che quest'ultimo mantenesse la stessa posizione rispetto agli overload precedenti, quindi la seconda posizione e che il nuovo parametro *a* andasse in coda agli altri.
Rimuovi parametri permette di effettuare questo cambiamento con semplicità, si deve soltanto selezionare il metodo i cui parametri si desidera riordinare e dal menu contestuale.



Nella finestra scegliere l'ordine corretto dei parametri agendo su di essi con i tasti freccia sulla destra e poi premere il tasto ***OK*** dopo aver selezionato l'opzione per visualizzare una preview delle modifiche prima di confermarle.

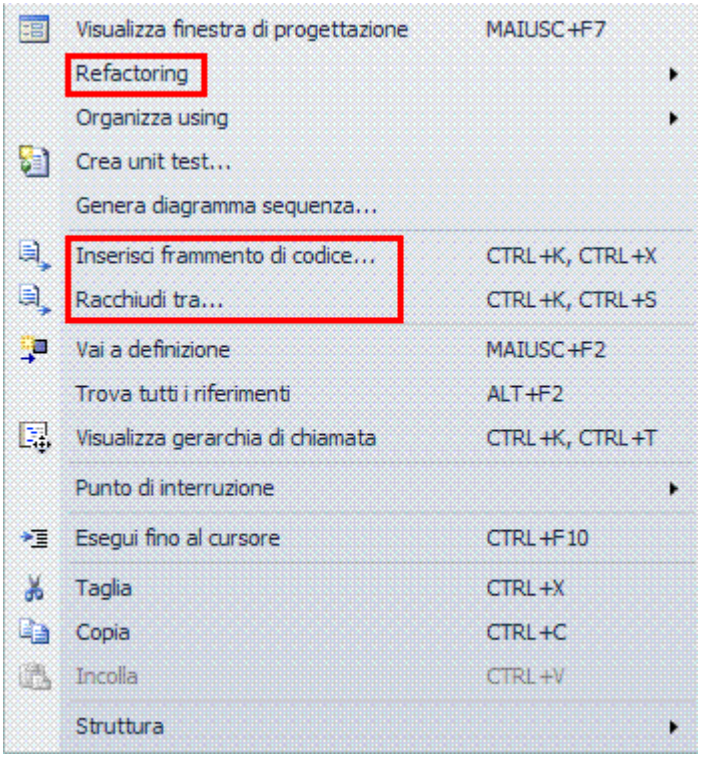


Il parametro è stato correttamente spostato dopo tutti gli altri e altresì qualsiasi riferimento a questo metodo all'interno del progetto e degli altri collegati, è stato aggiornato per rispecchiare la nuova disposizione dei parametri.
Fare clic su ***Applica*** e il codice è modificato nel modo seguente.

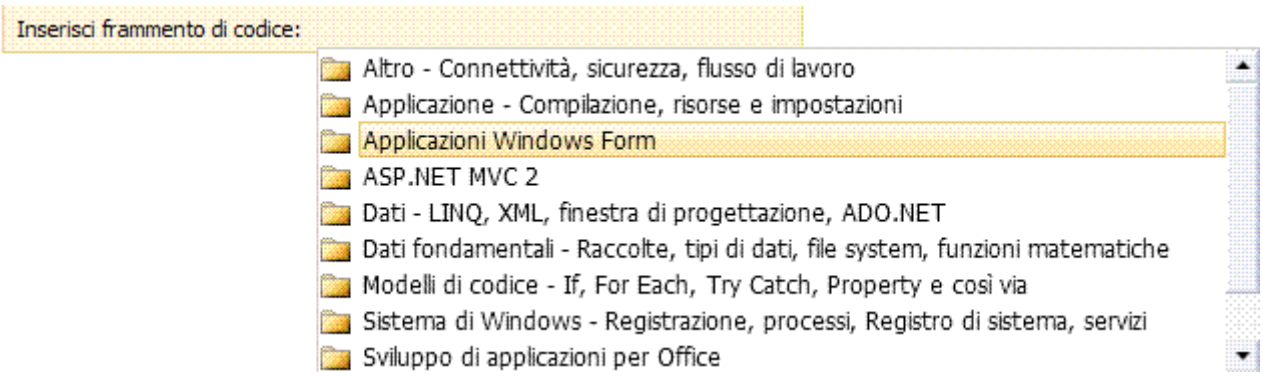
```
public decimal GetAverage(int[] numbers, Averages avg, int a)
{ }
```

Le modifiche apportate dalle funzioni di refactoring non sono applicate solo nel progetto corrente ma sono estese anche a tutti i progetti correlati.
Ad esempio se si sta sviluppando un Windows Form che utilizza un progetto Class Library e si effettua una modifica di refactoring, ad esempio un rinomina di un metodo, all'interno della Class Library, saranno aggiornati anche tutti i riferimenti del progetto Windows Form verso quel metodo o tipo modificato.

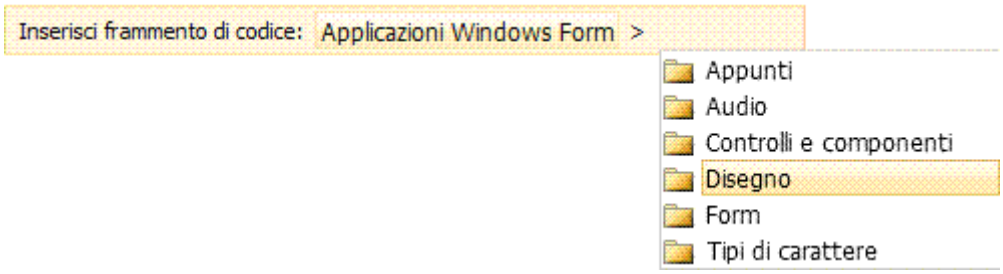
Snippet - Surround
Gli snippet sono porzioni di codice di uso comune, come il codice dei costrutti *for* e *while*, che possono essere inseriti facendo clic col tasto destro del mouse nella finestra del codice, oppure, più semplicemente, digitando il nome dello snippet e premendo due volte il tasto TAB, queste porzioni di codice possono contenere dei parametri, in tal caso, dopo l'inserimento, il cursore si posiziona automaticamente sul primo di essi: per spostarsi tra i parametri, si usa il tasto TAB.
Il codice è presente nel percorso seguente in formato XML.
C:\Programmi\Microsoft Visual Studio 10.0\VB\Snippets\1033 - 1040
Il surround è la possibilità di circondare un blocco d'istruzioni con un costrutto.



Esempio, in VB.NET ***Inserisci frammento di codice….***
Selezionare ***Applicationi Windows Form***.

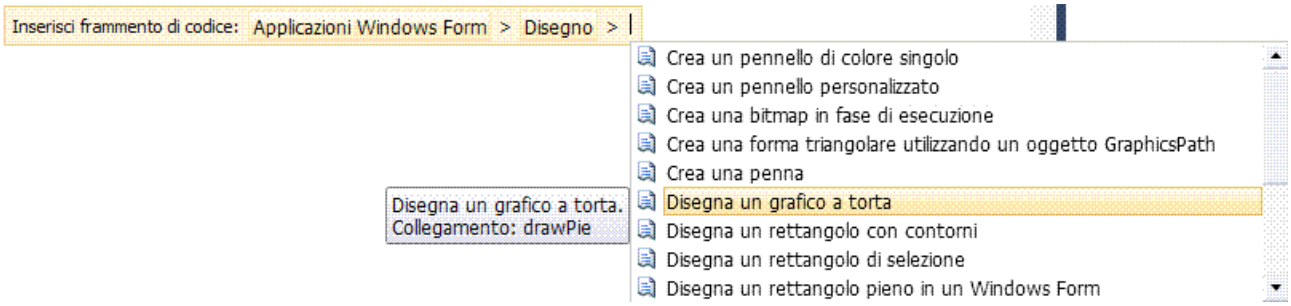


Quindi ***Disegno***.



Si hanno a disposizione 17 algoritmi già “preconfezionati” e pronti ad essere utilizzati

Disegna un grafico a torta consente di creare un grafico.

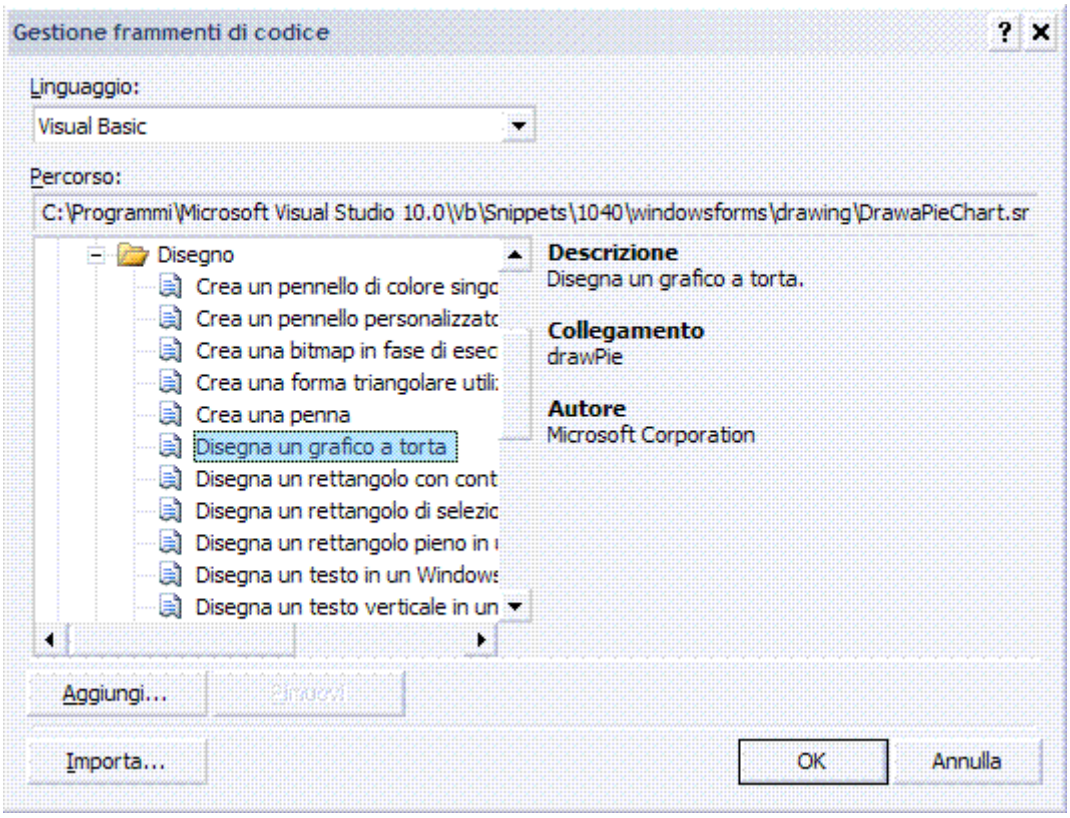


Fare clic per inserire nell’editor il codice dello snippet.

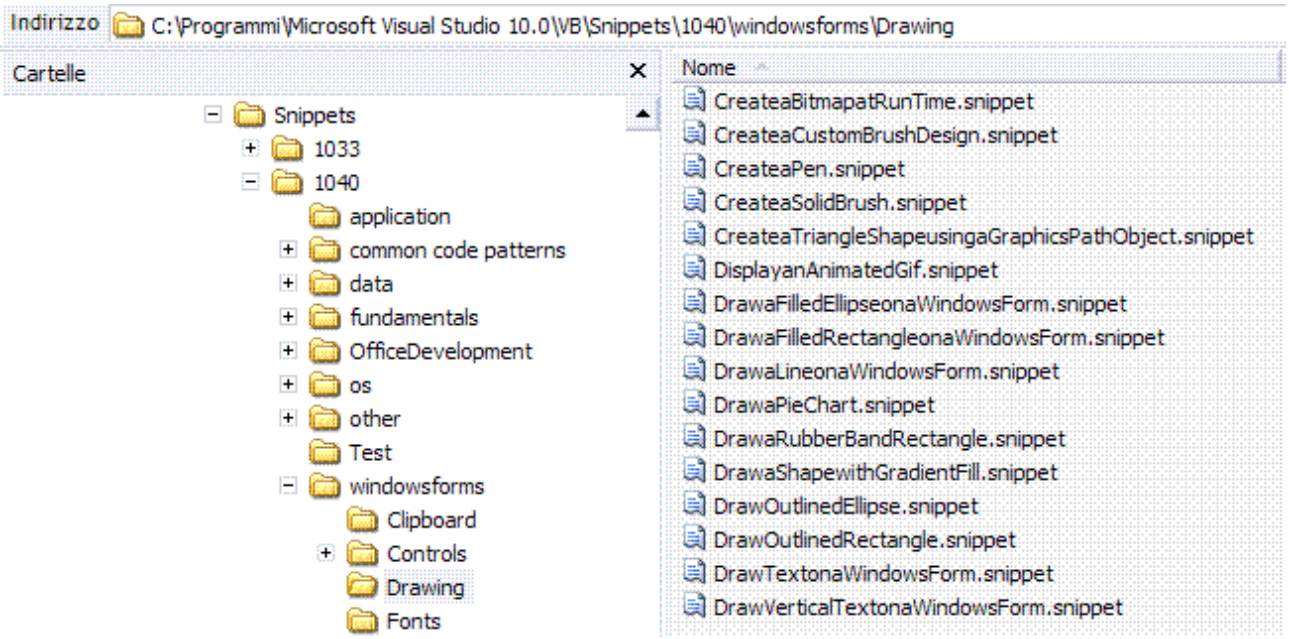
```
' Shows how to call the DrawPieChart method
Public Sub DrawPieChartHelper()
    Dim percents = {10, 20, 70}
    Dim colors = {Color.Red, Color.CadetBlue, Color.Khaki}
    Using graphics = Me.CreateGraphics()
        Dim location As New Point(10, 10)
        Dim size As New Size(150, 150)
        DrawPieChart(percents, colors, graphics, location, size)
    End Using
End Sub
' Draws a pie chart.
Public Sub DrawPieChart(ByVal percents() As Integer, ByVal colors() As Color, ByVal
surface As Graphics, ByVal location As Point, ByVal pieSize As Size)
    ' Check if sections add up to 100.
    Dim sum = 0
    For Each percent In percents
        sum += percent
    Next
    If sum <> 100 Then
        Throw New ArgumentException("Percentages do not add up to 100.")
    End If
    If percents.Length <> colors.Length Then
     Throw New ArgumentException("There must be the same number of percents and
colors.")
    End If
    Dim percentTotal = 0
    For percent = 0 To percents.Length() - 1
        Using brush As New SolidBrush(colors(percent))
```

```
            surface.FillPie(brush,
                            New Rectangle(location, pieSize),
                            CSng(percentTotal * 360 / 100),
                            CSng(percents(percent) * 360 / 100))
        End Using
        percentTotal += percents(percent)
    Next
    Return
End Sub
```

Visual Studio offre uno strumento per gestire gli snippets nel menu seguente.
Strumenti/Gestione frammenti di codice… (CTRL+K, CTRL+B).



La finestra che si apre consente d'impostare, aggiungere o rimuovere elementi o gruppi di elementi, selezionando uno snippet, appare anche il relativo percorso del file su disco con estensione .SNIPPET.
Contenuto della directory cui appartiene il file è insieme a tutti gli altri concernenti lo stesso argomento.



Se si apre un file è in formato XML.

File DRAWPIECHART.SNIPPET

```
<?xml version="1.0" encoding="utf-8"?>
<CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>Disegna un grafico a torta</Title>
      <Author>Microsoft Corporation</Author>
      <Description>Disegna un grafico a torta.</Description>
      <Shortcut>drawPie</Shortcut>
    </Header>
    <Snippet>
      <References>
        <Reference>
          <Assembly>System.Drawing.dll</Assembly>
        </Reference>
      </References>
      <Imports>
        <Import>
          <Namespace>Microsoft.VisualBasic</Namespace>
        </Import>
        <Import>
          <Namespace>System.Drawing</Namespace>
        </Import>
        <Import>
          <Namespace>System</Namespace>
        </Import>
      </Imports>
      <Declarations>
        <Literal>
          <ID>Percent1</ID>
          <Type>Integer</Type>
          <ToolTip>Sostituire con la percentuale della torta.</ToolTip>
          <Default>10</Default>
        </Literal>
```

```
<Literal>
  <ID>Percent2</ID>
  <Type>Integer</Type>
  <ToolTip>Sostituire con la percentuale della torta.</ToolTip>
  <Default>20</Default>
</Literal>
<Literal>
  <ID>Percent3</ID>
  <Type>Integer</Type>
  <ToolTip>Sostituire con la percentuale della torta.</ToolTip>
  <Default>70</Default>
</Literal>
<Literal>
  <ID>Color1</ID>
  <Type>Color</Type>
  <ToolTip>Sostituire con il colore per la sezione.</ToolTip>
  <Default>Color.Red</Default>
</Literal>
<Literal>
  <ID>Color2</ID>
  <Type>Color</Type>
  <ToolTip>Sostituire con il colore per la sezione.</ToolTip>
  <Default>Color.CadetBlue</Default>
</Literal>
<Literal>
  <ID>Color3</ID>
  <Type>Color</Type>
  <ToolTip>Sostituire con il colore per la sezione.</ToolTip>
  <Default>Color.Khaki</Default>
</Literal>
<Literal>
  <ID>XLocation</ID>
  <Type>Integer</Type>
  <ToolTip>Sostituire con la coordinata X della posizione del disegno.</ToolTip>
  <Default>10</Default>
</Literal>
<Literal>
  <ID>YLocation</ID>
  <Type>Integer</Type>
  <ToolTip>Sostituire con la coordinata Y della posizione del disegno.</ToolTip>
  <Default>10</Default>
</Literal>
<Literal>
  <ID>Width</ID>
  <Type>Integer</Type>
  <ToolTip>Sostituire con la larghezza.</ToolTip>
  <Default>150</Default>
</Literal>
<Literal>
  <ID>Height</ID>
  <Type>Integer</Type>
  <ToolTip>Sostituire con l'altezza.</ToolTip>
  <Default>150</Default>
</Literal>
```

```
    </Declarations>
    <Code Language="VB" Kind="method decl">
     <![CDATA[    ' Shows how to call the DrawPieChart method
   Public Sub DrawPieChartHelper()
      Dim percents = {$Percent1$, $Percent2$, $Percent3$}
      Dim colors = {$Color1$, $Color2$, $Color3$}
      Using graphics = Me.CreateGraphics()
         Dim location As New Point($XLocation$, $YLocation$)
         Dim size As New Size($Width$, $Height$)
         DrawPieChart(percents, colors, graphics, location, size)
      End Using
   End Sub
' Draws a pie chart.
Public Sub DrawPieChart(ByVal percents() As Integer, ByVal colors() As Color,
                  ByVal surface As Graphics, ByVal location As Point,ByVal pieSize As Size)
   ' Check if sections add up to 100.
   Dim sum = 0
   For Each percent In percents
      sum += percent
   Next
   If sum <> 100 Then
      Throw New ArgumentException("Percentages do not add up to 100.")
   End If
   If percents.Length <> colors.Length Then
Throw New ArgumentException("There must be the same number of percents and
colors.")
   End If
   Dim percentTotal = 0
   For percent = 0 To percents.Length() - 1
      Using brush As New SolidBrush(colors(percent))
         surface.FillPie(brush,New Rectangle(location, pieSize),
                  CSng(percentTotal * 360 / 100),CSng(percents(percent) * 360 / 100))
      End Using
      percentTotal += percents(percent)
   Next
   Return
End Sub]]>
    </Code>
   </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

La sintassi è semplice, l'elemento *CodeSnippets* è quello di primo livello e contiene uno o più elementi *CodeSnippet* che costituisce l'elemento base per definire il frammento di codice.
CodeSnippet si articola in due sotto elementi.

1. *Header*

Contiene le informazioni descrittive del frammento, possono essere contenuti gli elementi seguenti.

Nome elemento	Tipo	Descrizione	Min./Max
Author	Facoltativo	Nome dell'autore.	0/1

Description	Facoltativo	Descrizione del frammento che appare nel Tooltip.	0/1
HelpURL	Facoltativo	URL Contenente ulteriori informazioni.	0/1
Keywords	Facoltativo	Contiene elementi Keyword, ovvero parole chiave personalizzate a scopo di ricerca o categorizzazione.	0/1
Shortcut	Facoltativo	Testo di collegamento utilizzabile per inserire il frammento con l'operazione: <testo> + TAB.	0/1
SnippetTypes	Facoltativo	Contiene elementi SnippetType che specificano il comportamento dello snippet.	0/1
Title	Obbligatorio	Nome dello snippet come appare in Intellisense.	1/1

I valori di un elemento *SnippetType* possono essere i seguenti.
- ✓ *SurroundsWith*: permette d'inserire il frammento di codice intorno a un segmento di codice.
- ✓ *Expansion*: permette d'inserire il frammento di codice nella posizione del cursore.

2. *Snippet*
Permette di definire, oltre al codice vero e proprio, il contesto dove esso si colloca.
Si possono avere i gruppi di elementi seguenti.

References
Definisce i riferimenti a librerie di sistema o personali che saranno aggiunti automaticamente al progetto, se non già presenti, al momento in cui s'inserisce il frammento, tali riferimenti sono definiti in un numero illimitato di sotto elementi *Reference* che contengono: un elemento Assembly per definire il nome dell'assembly che sarà aggiunto al progetto e, opzionalmente, un elemento **URL** (*Uniform Resource Locator*) contenente l'indirizzo web dove è possibile trovare informazioni sull'assembly aggiunto.

Import
Definisce gli spazi dei nomi che sono automaticamente aggiunti in testa al file di codice utilizzando lo snippet, questo evita di dover far riferimento al nome completo dei tipi nel codice che si va a definire, *Imports* contiene più elementi *Import* che, sotto l'elemento *Namespace*, definiscono lo spazio dei nomi relativo.

Declarations
Definisce i valori letterali o gli oggetti del frammento di codice che l'utente può modificare, è un segnaposto dei valori effettivi, tali valori sono espressi da due tipi di elementi.
1. *Literal* per i valori letterali.
2. *Object* per gli oggetti.

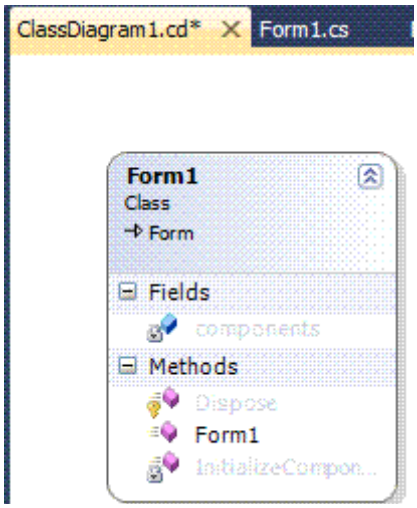
Ogni elemento *Literal* o *Object* deve definire un elemento *ID* che possa essere richiamato nel codice e un elemento *Default* che contiene il valore predefinito.
È possibile anche definire un elemento *ToolTip* che sarà mostrato quando l'utente si sposta sul segnaposto per descrivere l'oggetto o il valore.
Per gli elementi *Object* è obbligatorio anche il sotto elemento *Type* che definisce il tipo cui appartiene l'oggetto.
All'interno del codice si può inserire un segnaposto definito come *Literal* o *Object*

attraverso il suo *ID* racchiuso tra il simbolo $ (o altro simbolo che possiamo definire nell'elemento *Code*.
L'elemento *Code* è quello più importante, perché racchiude il codice, ha tre attributi.

1. *Delimiter* facoltativo, specifica il delimitatore utilizzato per descrivere i valori letterali e gli oggetti nel codice, se non altrimenti definito, il delimitatore è *$*.
2. *Kind* facoltativo, specifica il tipo di codice contenuto nel frammento e pertanto le posizioni in cui il frammento può essere inserito. I valori disponibili sono i seguenti.
   2.1. *method body* il frammento deve essere inserito all'interno di una dichiarazione di metodo.
   2.2. *method decl* il frammento deve essere inserito all'interno di una classe o un modulo.
   2.3. *type decl* il frammento deve essere inserito all'interno di una classe, un modulo o uno spazio dei nomi.
   2.4. *page* il frammento di codice deve essere utilizzato come codice all'interno di una pagina di progetto web.
   2.5. *file* il frammento è un file di codice completo.
   2.6. *any* il frammento può essere inserito in qualsiasi posizione.
3. *Language* (obbligatorio) specifica il linguaggio del frammento di codice, i valori disponibili sono i seguenti.
   3.1. *VB*.
   3.2. *CSharp*.
   3.3. *XML*.

All'interno dell'elemento Code è inserito il codice vero e proprio, preferibilmente all'interno dei tag *<![CDATA[...]]>* per evitare che alcuni simboli presenti nel codice, ad esempio < o >, possano essere interpretati come XML.

Class Designer
Progettazione classi per costruire in modo grafico lo schema delle classi, usando lo stile **UML** (*Unified Modeling Language*), il grafico è salvato in un file CLASSDIAGRAM1.CD.



Per visualizzare tutti gli elementi associati al progetto, inclusi i file nascosti o esclusi, fare clic sul pulsante ***Mostra tutti i file*** .
Le cartelle nascoste ***bin*** e ***obj*** saranno quindi visualizzate.
Per aggiornare lo stato di tutti gli elementi associati al progetto, fare clic sul pulsante ***Refresh*** .
Saranno visualizzati eventuali nuovi elementi aggiunti.
Fare clic su FORM1.CS in ***Esplora soluzioni***, saranno visualizzati altri due pulsanti.
Per visualizzare la finestra del codice, fare clic sul pulsante ***Visualizza/Codice (F7)*** .

Per visualizzare la finestra di progettazione, fare clic sul pulsante.
Visualizza/Finestra di progettazione (MAIUSC+F7)  .

HELP ONLINE/OFFLINE

Il sistema è basato interamente su pagine che sono aperte nel browser, sia che si utilizzi l'help online, sia che si utilizzi quello installato in locale, offline.
La ricerca è costituita da una gerarchia di collegamenti e da una casella di testo per inserire la stringa da ricercare.
Semplice la modalità di aggiornamento dell'help installato in locale, attraverso la voce di menu ***?/Controlla aggiornamenti***.

ADD-IN

Sono moduli software che sfruttando la particolare architettura di Visual Studio, permettono di aggiungere funzionalità non presenti in origine nel prodotto.
Strumenti/Gestione estensioni…
Permette di gestire tutti gli add-in e le estensioni di Visual Studio.
Le estensioni sono categorizzate in tre macro categorie: controlli, template e strumenti ognuna delle quali è poi suddivisa ulteriormente per categoria.
Per esempio, *NDoc* estrapola commenti e permette di portarli fuori dal codice sorgente sotto forma di file HTML o file Help di Windows, invece *GhostDoc* nel momento stesso in cui si stanno scrivendo i commenti del codice basta un clic del mouse e scrive automaticamente una bozza di documentazione estrapolandone il testo dal nome stesso della classe o del metodo e dei suoi parametri.

.NET FRAMEWORK 4.0

È stato introdotto il tipo *dynamic*, che consente di definire struttura e comportamenti che non sono risolti dal compilatore, bensì durante la fase di run-time.
Diventa possibile quindi utilizzare oggetti definiti con linguaggi dinamici come Python e Ruby direttamente nel codice C#; l'esecuzione del codice scritto con linguaggi dinamici è garantito da un nuovo run-time chiamato **DLR** (*Dynamic Language Runtime*).
La dinamicità introdotta a livello del Framework consente anche di gestire oggetti la cui struttura può variare durante la vita di un oggetto: si pensi a una pagina HTML che tramite Javascript è modificata nei suoi elementi; è possibile interagire da C#/VB con gli elementi **DOM** (*Document Object Model*) della pagina senza la necessità di tipizzare la struttura della pagina stessa.

```
dynamic d = GetDynamicObject(…);
d.Fai(7);
```

È possibile invocare il metodo *Fai* con qualunque numero e tipo di parametri senza che il compilatore esegua il controllo sulla sicurezza dei tipi, type-safety.
Ovviamente a run-time l'oggetto *d* deve esporre tale metodo, altrimenti si riceve un'eccezione.
È possibile definire *dynamic lookup*, ovvero scrivere metodi, operatori, proprietà e invocazione di oggetti bypassando il controllo sui tipi effettuato dal compilatore.

PARALLEL EXTENSIONS

La versione 2010 di Visual Studio e il .NET Framework 4.0 forniscono nativamente il supporto alla programmazione parallela.
Nuovo modello di multithreading basato sull'elaborazione parallela, incluso anche in **PLINQ** (*Parallel LINQ*), una nuova implementazione di LINQ to Object.
In pratica, si può decidere di lavorare a livello più "basso" gestendo manualmente thread,

semafori e mutex, oppure lavorare a un livello più alto e quindi più semplice; questa astrazione si raggiunge utilizzando i metodi *Begin/End* delle classi che li supportano oppure il controllo *BackgroundWorker*.
La versione 4.0 semplifica ulteriormente la gestione del parallelismo liberando il programmatore dal dover utilizzare API asincrone.

```
var source = Enumerable.Range(1,10000);
var query = from num in source.AsParallel()
            where Compute(num) > 0
            select num;
```

Il codice esegue la chiama alla funzione *Compute(num)* parallelizzandone l'esecuzione su più thread senza necessità di esplicitare queste funzionalità nel codice.
L'unica differenza rispetto al codice LINQ è l'introduzione del metodo *AsParallel*().

Windows Seven

Nel run-time della versione 4.0 sono presenti alcune funzionalità che semplificano l'integrazione con le nuove caratteristiche specifiche di Windows Seven, come ad esempio *JumpList, ProgressBar, Thumbnails Toolbar* e *Icon Overlay*.

Esempio, codice che sfrutta la *JumpList*.

```
JumpList jumpList = new JumpList();
JumpList.SetJumpList(Application.Current, jumpList);
JumpTask jumpTask = new JumpTask();
jumpTask.Title = "DevLeap";
jumpTask.Description = "DevLeap Training";
jumpTask.CustomCategory = "Customer Relation";
jumpTask.ApplicationPath = "devleap.exe";
jumpList.JumpItems.Add(jumpTask);
jumpTask = new JumpTask();
jumpTask.Title = "ThinkAhead";
jumpTask.Description = "ThinkAhead UX";
jumpTask.CustomCategory = "Customer Relation";
jumpTask.ApplicationPath = "Gemini.exe";
jumpList.JumpItems.Add(jumpTask);
jumpList.Apply();
```

Si sono aggiunti due elementi alla *JumpList*, entrambi nella sezione *Customer Relation*; il primo con label *DevLeap* punta a *devleap.exe*, mentre il secondo elemento denominato *ThinkAhead*, punta all'applicativo *Gemini.exe*.

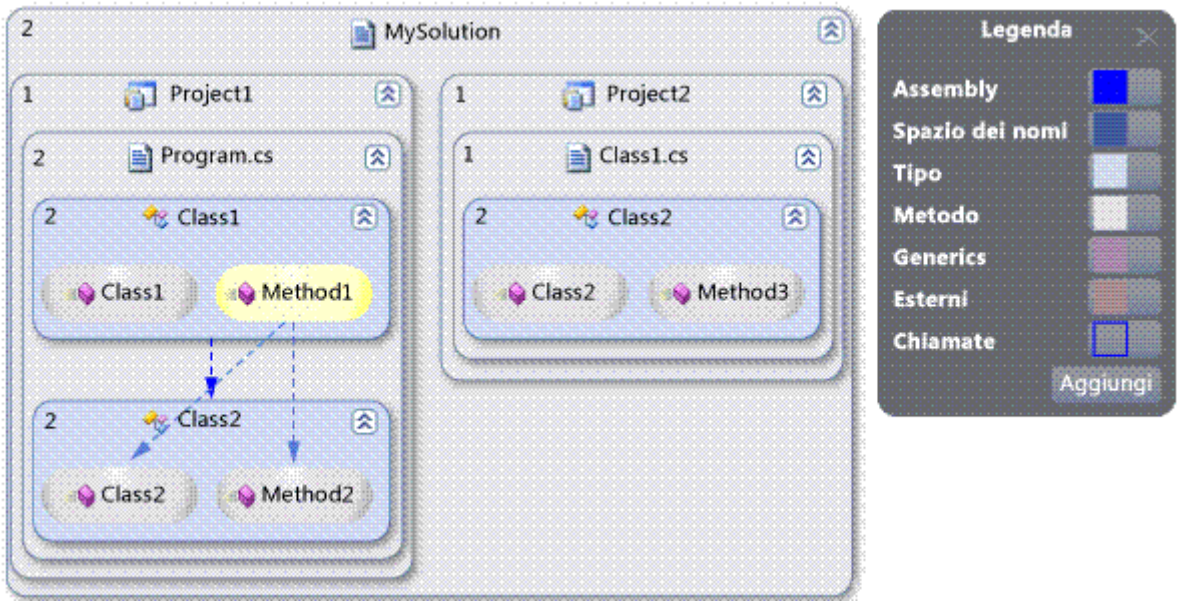
Grafici di dipendenze

Particolare attenzione a tutto il ciclo di vita di un'applicazione, dalla progettazione e modellazione fino ai test e alla distribuzione.
È possibile utilizzare grafici di dipendenze per ottenere maggiori informazioni sull'organizzazione e sulle relazioni presenti nel codice esistente.
Questi grafici rappresentano le strutture come nodi e le relazioni come collegamenti, indicati mediante frecce tra i nodi.
Le dipendenze nel grafico sono rappresentate dai seguenti tipi di collegamenti.

- ✓ Un collegamento singolo rappresenta una singola dipendenza tra due nodi.
- ✓ Un collegamento di aggregazione rappresenta tutte le dipendenze rivolte nella stessa direzione tra due gruppi.

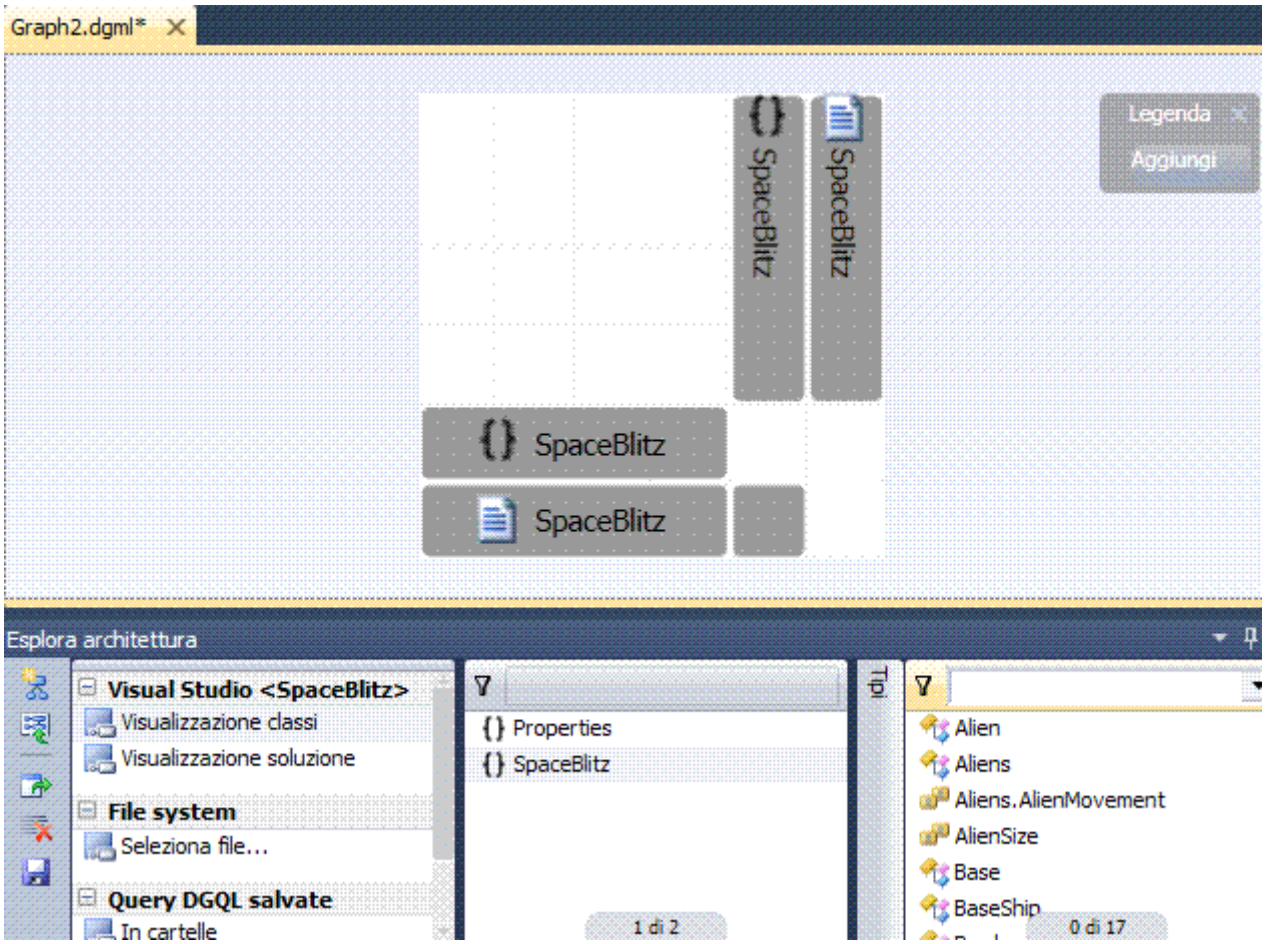
✓ Un collegamento tra gruppi è semplicemente un collegamento tra due nodi appartenenti a gruppi diversi.

Esempio, modalità di rappresentazione di strutture e relazioni di contenimento come gruppi all'interno del grafico.



ESPLORA ARCHITETTURA

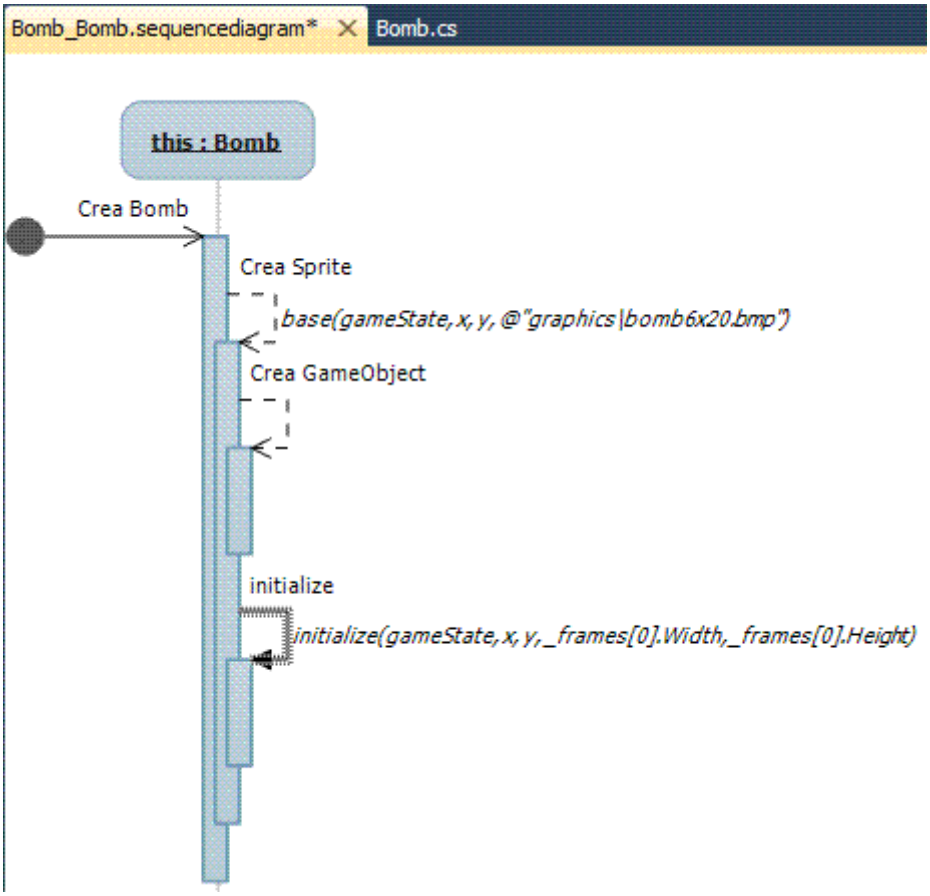
Consente di navigare in modo grafico le relazioni fra le classi e le relative chiamate.



L’applicazione consta di vari layer applicativi che possono essere navigati in modo veloce e intuitivo tramite lo strumento.
I colori indicano la tipologia di oggetto.
Una seconda modalità di utilizzo dello strumento è la matrice che indica l’intersezione fra i vari elementi applicativi dove i colori indicano la tipologia di relazione.

Diagramma di sequenza

Visualizza un'interazione, in pratica è una sequenza di messaggi tra tipiche istanze di classi, componenti, sottosistemi o attori.
Il designer consente di definire i layer di un’applicazione e le regole secondo cui i componenti dei vari layer possono “parlarsi”.



Questo strumento è molto utile per “controllare” un’applicazione esistente rispetto al modello desiderato e per imporre il layering corretto agli sviluppatori.
Se si utilizza questo strumento affiancandolo a un’altra visualizzazione l’***Esplora architettura*** si ottiene un controllo completo sull’interazione dei layer e dei componenti dell’applicazione.

Windows Azure

Strumenti specifici per sviluppare applicazioni per Windows Azure, cioè per il cloud computing, una nuova piattaforma per il web, dotata di alta disponibilità, 24/24 ore e 7/7 giorni, gestione automatica e indipendenza dall'hardware e dalla dislocazione fisica.

APPLICAZIONI

INTRODUZIONE

All'interno della cartella del progetto si trovano diversi file.

File PRIMA APPLICAZIONE.SLN
Rappresenta la soluzione del progetto ovvero, l'insieme dei file che si useranno: modelli, immagini, suoni, icone.

```
Microsoft Visual Studio Solution File, Format Version 11.00
# Visual Studio 2010
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "Prima applicazione", "Prima
applicazione\Prima applicazione.csproj", "{85FDC1BF-2E9E-4FB4-A85B-
E0E40F0FB8AC}"
EndProject
Global
	GlobalSection(SolutionConfigurationPlatforms) = preSolution
		Debug|x86 = Debug|x86
		Release|x86 = Release|x86
	EndGlobalSection
	GlobalSection(ProjectConfigurationPlatforms) = postSolution
		{85FDC1BF-2E9E-4FB4-A85B-E0E40F0FB8AC}.Debug|x86.ActiveCfg =
Debug|x86
		{85FDC1BF-2E9E-4FB4-A85B-E0E40F0FB8AC}.Debug|x86.Build.0 =
Debug|x86
		{85FDC1BF-2E9E-4FB4-A85B-E0E40F0FB8AC}.Release|x86.ActiveCfg =
Release|x86
		{85FDC1BF-2E9E-4FB4-A85B-E0E40F0FB8AC}.Release|x86.Build.0 =
Release|x86
	EndGlobalSection
	GlobalSection(SolutionProperties) = preSolution
		HideSolutionNode = FALSE
	EndGlobalSection
EndGlobal
```

File ASSEMBLYINFO.CS

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Prima applicazione")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany(" ")]
[assembly: AssemblyProduct("Prima applicazione")]
[assembly: AssemblyCopyright("Copyright © 2011")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
// Setting ComVisible to false makes the types in this assembly not visible
```

```
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]
// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("2137eb7e-f98a-4396-a776-2f0f573e2676")]
// Version information for an assembly consists of the following four values:
//      Major Version
//      Minor Version
//      Build Number
//      Revision
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

SVILUPPO CROSS LANGUAGE

L'interoperabilità tra linguaggi aderenti al CLS è immediata, da una classe base scritta in C++ ne sarà derivata una in VB.NET, dalla quale ne sarà derivata una in MSIL.

File PLUTO.CPP

```
#using <mscorlib.dll>
using namespace System;
public class VCBase
{ protected:
   VCBase(){
      Console::WriteLine(" Eseguo VCBase::VCBase() costruttore.");
   }
public:
   virtual void Method() = 0;
   void MethodThatThrows(){
      throw(gcnew OutOfMemoryException("Non ci sono più risorse!"));
   }
};
```

File PLUTO.VB

```
Option Explicit On
Option Strict On
Imports System
Public Class VBDerived
    Inherits VCBase
    Public Sub New()
        Console.WriteLine("  Eseguo VBDerived.New() costruttore.")
    End Sub
    Public Overrides Sub Method()
        Console.WriteLine(" Eseguo VBDerived.Method()  metodo virtuale")
    End Sub
End Class
```

File PLUTO.IL

```
.module extern VBDerived.netmodule
.class public auto ansi ILDerived
       extends [.module VBDerived.netmodule]VBDerived
```

```
{
  .method public specialname rtspecialname
          instance void .ctor() il managed
  {
    .maxstack  1
    .locals init (class System.Object[] V_0)
    IL_0000:  ldarg.0
    IL_0001:  call       instance void [.module VBDerived.netmodule]VBDerived::.ctor()
    IL_0006:  ldstr      "   Executing the ILDerived::ctor() constructor"
    IL_000b:  call       void [mscorlib]System.Console::WriteLine(class System.String)
    IL_0010:  ret
  }
  .method public virtual instance void Method() il managed
  {
    .maxstack  1
    .locals init (class System.Object[] V_0)
    IL_0000:  ldstr      " Eseguo ILDerived::Method()  metodo virtuale"
    IL_0005:  call       void [mscorlib]System.Console::WriteLine(class System.String)
    IL_000a:  ret
  }
}
```

File PLUTO.CS

```
using System;
public class CSDerived:ILDerived{
   public CSDerived(){
      Console.WriteLine("Eseguo CSDerived.CSDerived() costruttore.");
   }
    override public void Method(){
       Console.WriteLine(" Eseguo CSDerived.Method()  metodo virtuale.");
   }
}
```

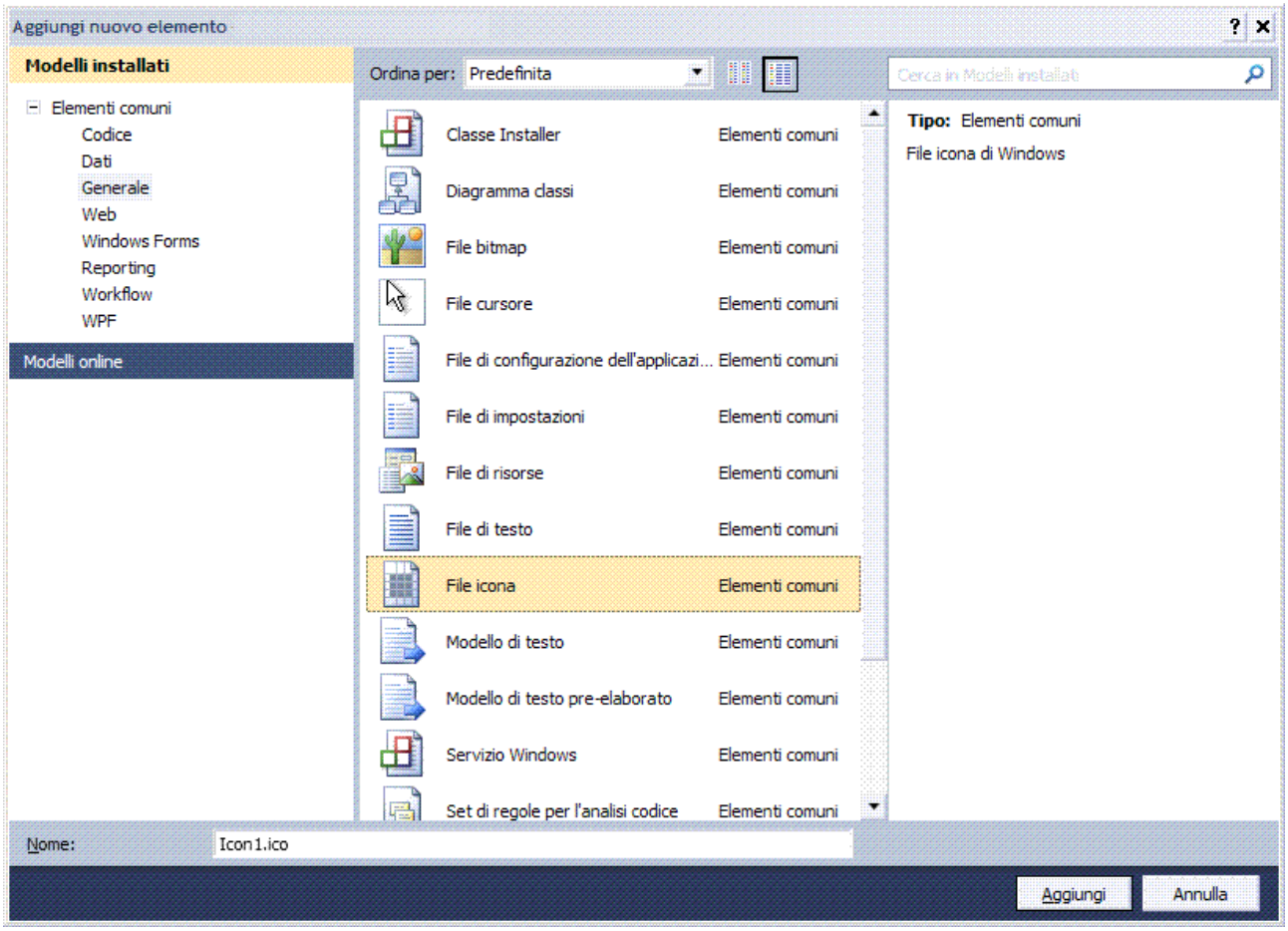
File CROSSLANG.CS

```
class App
{  static void Main(string[] args)
   {  CrossObj(); }
      static void CrossObj()
      {
      // Crea un array di oggetti per immagazzinare quelli definiti negli altri linguaggi
      VCBase[] objects = new VCBase[3];
      // Carica nella posizione 0 dell'array l'oggetto creato usando IL
      Console.WriteLine("\nCreo l'oggetto:  ILDerived");
      objects[0] = new ILDerived();
      // Carica nella posizione 1 dell'array l'oggetto creato usando C#
      Console.WriteLine("\nCreating object:  CSDerived");
      objects[1] = new CSDerived();
      // Carica nella posizione 2 dell'array l'oggetto
      //creato usando VB
      Console.WriteLine("\nCreo l'oggetto:  VBDerived");
      objects[2] = new VBDerived();
      // Chiama il metodo virtuale di ogni oggetto
      Console.WriteLine("\nChiamata dei metodi");
      foreach(VCBase obj in objects) { obj.Method(); }
```

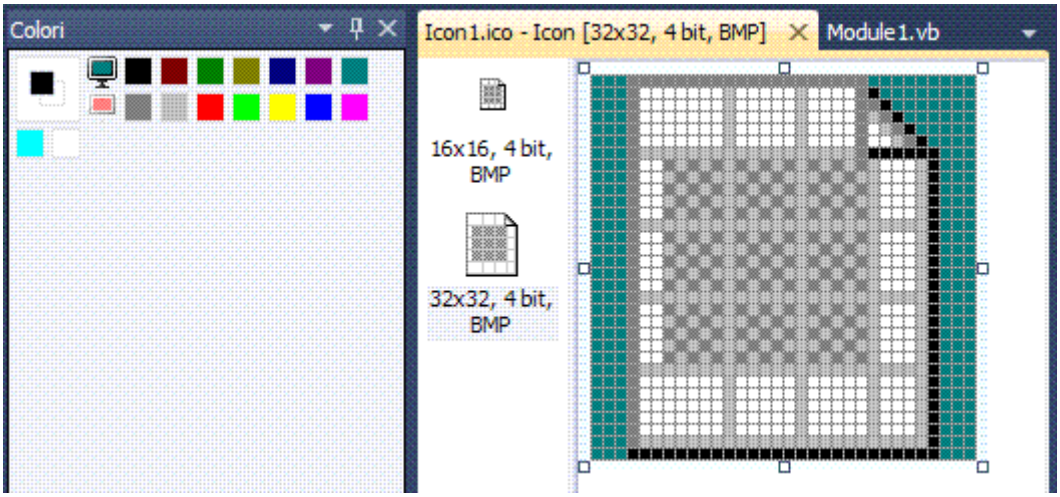
```
        Console.ReadKey();
        }
}
```

ICONA

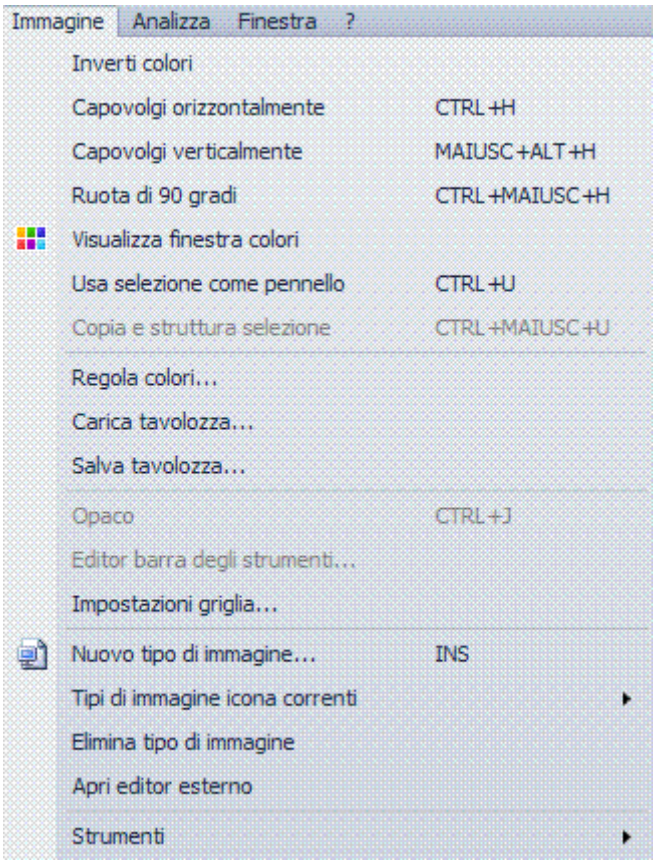
Fare clic destro in ***Esplora soluzioni*** sul nome del progetto, scegliere il menu seguente ***Aggiungi/Nuovo elemento… (CTRL+MAIUSC+A)***
Alla voce ***Elementi comuni*** selezionare ***Generale*** e in ***Ordina per: File icona***.



Nel campo ***Nome:*** al posto di ***Icon1.ico***, digitare il nome da dare all'icona e fare clic sul pulsante ***Aggiungi***.
Appare l'ambiente di lavoro che consente di creare l'icona del progetto e che è composto dai seguenti elementi.

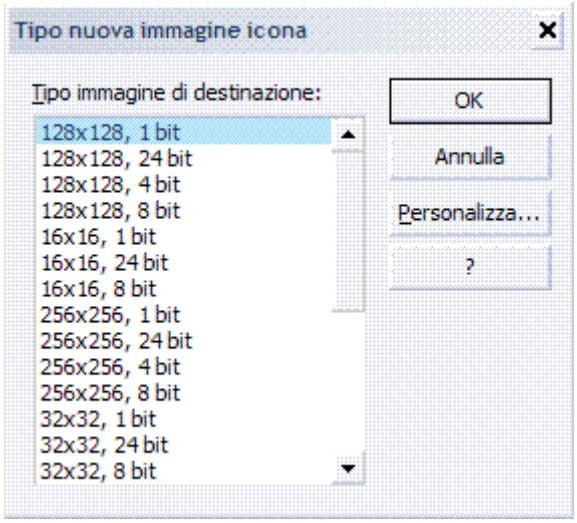


La barra con la tavolozza dei colori.
Due riquadri che contengono rispettivamente l'icona in dimensioni reali e l'icona ingrandita si può disegnare su entrambi.
Il menu ***Immagine*** con i suoi elementi e la barra delle icone.



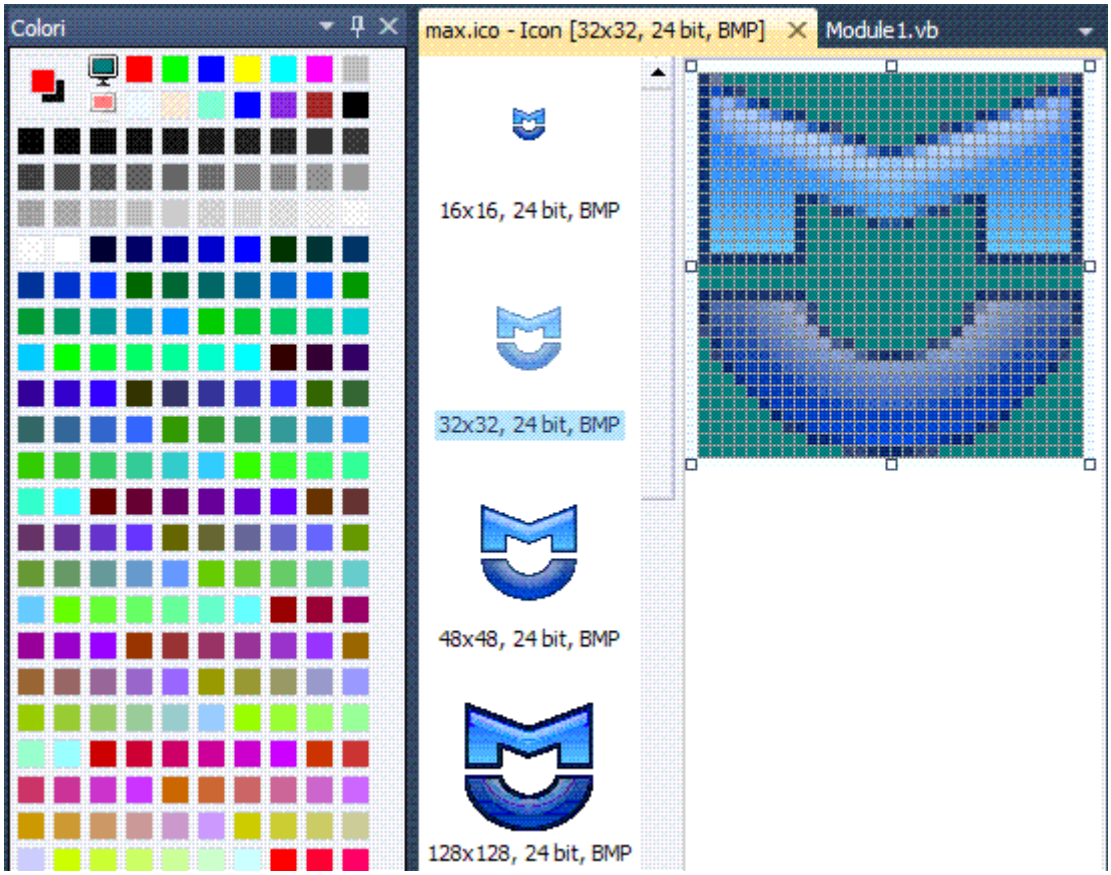
Visualizza/Barre degli strumenti/Editor immagini

Le impostazioni di default dell'icona sono due 16X16 e 32X32 4 bit, 16 colori, ma sono modificabili selezionando il menu ***Immagine/Nuovo tipo di immagine… (INS)***.



Una volta definito il formato dell'icona, si può creare sia disegnandola a mano libera sia

copiando un disegno o parte di esso nell'apposito riquadro.
Gli strumenti disponibili sono simili a quelli che sono messi a disposizione da altre applicazioni di editing d'immagini: matita, pennello, aerografo, selezioni, figure geometriche.
Tramite il menu ***Immagine***, inoltre, è possibile invertire i colori, capovolgere l'immagine orizzontalmente o verticalmente oppure ruotarla di 90 gradi.
Una volta ultimato il disegno ricordarsi di definire "trasparenti" le aree esterne ad esso, in questo modo lo sfondo dell'immagine è esattamente quello del desktop e l'icona è molto più professionale.
Selezionare lo strumento ***Aerografo*** e, poi, sulla tavolozza fare clic sul piccolo monitor di colore verde, a questo punto fare clic sulle aree del disegno che si vuole rendere trasparenti.



Salvare l'icona e chiudere l'editor.
L'icona, una volta creata, genera automaticamente un file ICO nella directory principale del progetto.
Per incorporare l'icona nell'applicazione, selezionare il menu ***Progetto/Proprietà di …*** fare clic sulla scheda ***Applicazione*** e includere l'icona nella casella combinata ***Icona:***.

DEBUGGER

INTRODUZIONE

È uno strumento utile per individuare e risolvere alcuni problemi del codice.
Quando un algoritmo si comporta diversamente dal previsto, restituendo un risultato erroneo o sollevando un'eccezione inattesa, il debugger è un alleato prezioso.

PROFILER

INTRODUZIONE

Serve per misurare le prestazioni del codice e prevenire e curare i problemi di abuso della memoria e delle risorse di calcolo.
Affinché la qualità di un software risulti elevata, non è sufficiente che questo funzioni soltanto, ma bisogna anche che funzioni al meglio, significa che il codice deve essere corretto, ma anche efficiente e che non debba sprecare spazio né tempo oltre lo stretto indispensabile.
Dato un software potrebbe accadere che, sotto alcune condizioni non verificate in precedenza, emerga un problema di spazio, troppa memoria occupata o di tempo, elaborazione troppo lunga, non osservato durante i test funzionali.
Bisogna allora andare alla ricerca di quella porzione del codice che è causa del problema, il profiler è lo strumento che segnala quali sono le parti del codice che consumano più risorse.
Il profiler mentre esegue il codice colleziona ed elabora una serie d'informazioni, tra le quali: numero e dimensione degli oggetti istanziati, tempo di esecuzione di ciascun metodo, interazioni tra oggetti ed interazioni tra thread.
Le informazioni raccolte possono essere elaborate e mostrate in tempo reale, ma è anche possibile accumularle e salvarle sul disco per esaminarle con calma in un secondo momento.
Alcuni filtri possono essere applicati sui dati raccolti, in modo da scremarli da tutte le informazioni non necessarie.
Ad esempio, se si sta ricercando la causa di un rallentamento, conviene farsi segnalare dal profiler soltanto i metodi la cui esecuzione ha una durata superiore alla media.
Se si cerca un punto in cui è sprecata la memoria, invece, conviene chiedere i nomi dei metodi che hanno istanziato più oggetti.

GUIDA IN LINEA

INTRODUZIONE

Tutte le applicazioni contengono al loro interno un manuale per l'utente, l'attivazione avviene facendo clic sul menu ***? (F1)***.
La guida in linea per un'applicazione Visual Studio può essere di due tipi.

1. WinHelp, costruita interamente con tecnologie Windows.
2. HTML Help, costruita utilizzando la tecnologia HTML.

Le guide HTML sono più moderne ed efficienti ed ormai hanno soppiantato le WinHelp.
Sul mercato sono disponibili vari pacchetti per l'implementazione e la distribuzione di guide in linea, per esempio RoboHelp e Flash MX.
Gli strumenti fornito da Microsoft sono i seguenti.

1. WinHelp.
2. HTML Help Workshop.

Essi sono dei compilatori che ricevono in input dei file opportunamente strutturati e producono in output i file della guida, da agganciare ad un progetto.
WinHelp è un compilatore che lavora soltanto con la CLI mentre il Workshop è un vero ambiente di sviluppo.
Un progetto di una guida in linea è composto dai seguenti componenti.

- ✓ Un insieme di pagine denominate ***topics***, argomenti.
- ✓ Un file di contesto, ***context*** file, che rappresenta gli argomenti principali in una rappresentazione ad albero.
- ✓ Un file d'indice, ***index*** file, che contiene l'elenco delle chiavi di ricerca.
- ✓ Un insieme di file, comandi, che contengono informazioni sui link tra i vari topics.
- ✓ Un insieme di file di supporto immagini, video e sonori.

I topics nel caso di WinHelp sono definiti attraverso un file **RTF** (*Rich Text Format*) mentre nel caso HTML Help sono definiti come file HTML.
Le informazioni sui vari file della guida, sono organizzate in un file di progetto che nel caso di WinHelp è un file in formato ASCII con estensione HPJ, mentre nel caso dell'HMTL Help è un file del Workshop con estensione HHP.

WINHELP

1. Creazione della Guida: RTF

La creazione del file degli argomenti comporta la stesura del testo che sarà visualizzato quando l'utente attiva la ***Guida*** e l'inserimento di codici di controllo che consentono i collegamenti ipertestuali, è un file creato in Word e salvato come RTF.
All'interno della ***Guida***, è possibile consultare argomenti in due modi: selezionando uno degli argomenti sottolineati si passa ad un'altra finestra; oppure selezionando un argomento sottolineato da una linea tratteggiata si visualizza una finestra a comparsa.
Sottolineare una parola.

1. Evidenziarla.
2. ***Formato/Carattere/Sottolineatura Doppia***.
3. Posizionare il cursore subito dopo la parola, non ci devono essere spazi.
4. ***Formato/Carattere/Nascosto***.
5. Inserire il contrassegno, indica a quale argomento bisogna saltare, se è seguito da ***@pathname*** salta ad un altro file, invece ***>windowname*** salta ad un'altra finestra.
6. ***Inserisci/Interruzione Di pagina***.
7. Bisogna indicare che si tratta della destinazione indicata al punto 5, per fare questo posizionare il cursore sulla nuova pagina e si selezioni ***Inserisci/Note personalizzata***,

si digiti # identifica in modo univoco un argomento ed il contrassegno. A questo punto Word divide la finestra in due parti, visualizzando nella parte inferiore le note e nella parte superiore il cursore dopo # che appare nel testo che si sta impostando.

8. ***File/Salva con nome/*.RTF***.

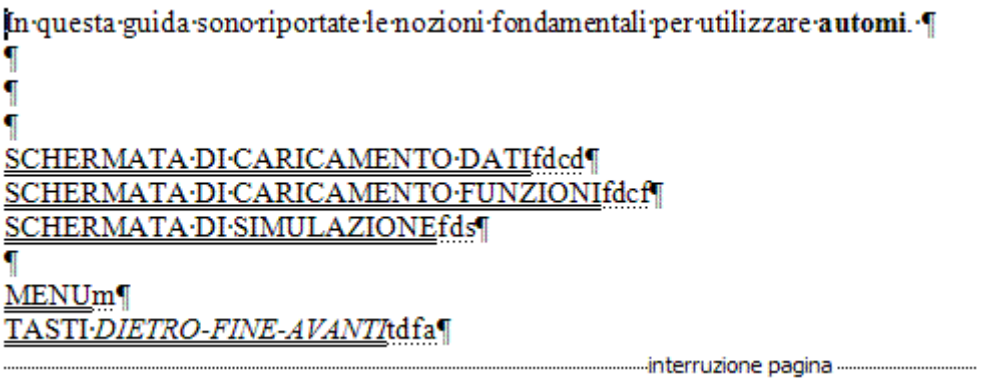
Sottolineare una parola con una linea tratteggiata.

1. Evidenziarla.
2. ***Formato/Carattere/Sottolineatura Singola***.
3. Posizionare il cursore subito dopo la parola non ci devono essere spazi.
4. ***Formato/Carattere/Nascosto***.
5. Inserire il contrassegno, indica a quale argomento bisogna saltare.
6. ***Inserisci/Interruzione Di pagina***.
7. Posizionare il cursore sulla nuova pagina e si selezioni ***Inserisci/Note personalizzata***, si digiti # con il commento della finestra a comparsa.
8. ***File/Salva con nome/*.RTF***.

Per ciascun argomento si deve assegnare oltre al contrassegno, anche un titolo ed un insieme di parole chiave, il simbolo ***$*** assegna alla pagina corrente un titolo all'argomento che apparirà nella casella di riepilogo quando l'utente utilizza l'opzione ***Cerca*** della ***Guida***.
Il simbolo ***K*** definisce una parola chiave che l'utente può utilizzare nella ricerca degli argomenti, più parole chiave sono separate dal punto e virgola.
Il simbolo **+** definisce la successione degli argomenti, è facoltativo.
L'inserimento d'immagini avviene in due modi.

1. Diretto nel file RTF.
2. In fase di compilazione mediante ***{bmc pathname}***.

In Word non usare l'apostrofo, ma ***Inserisci/Simbolo/Testo normale***.
Nel titolo inserire ***$***, in questo modo nella cronologia sparisce: argomento senza titolo.



2. Creazione del file di progetto: HPJ

È in formato ASCII, con struttura simile ai file INI, in pratica una serie di sezioni che determinano l'aspetto del file di ***Guida*** ed alcuni parametri necessari alla compilazione.

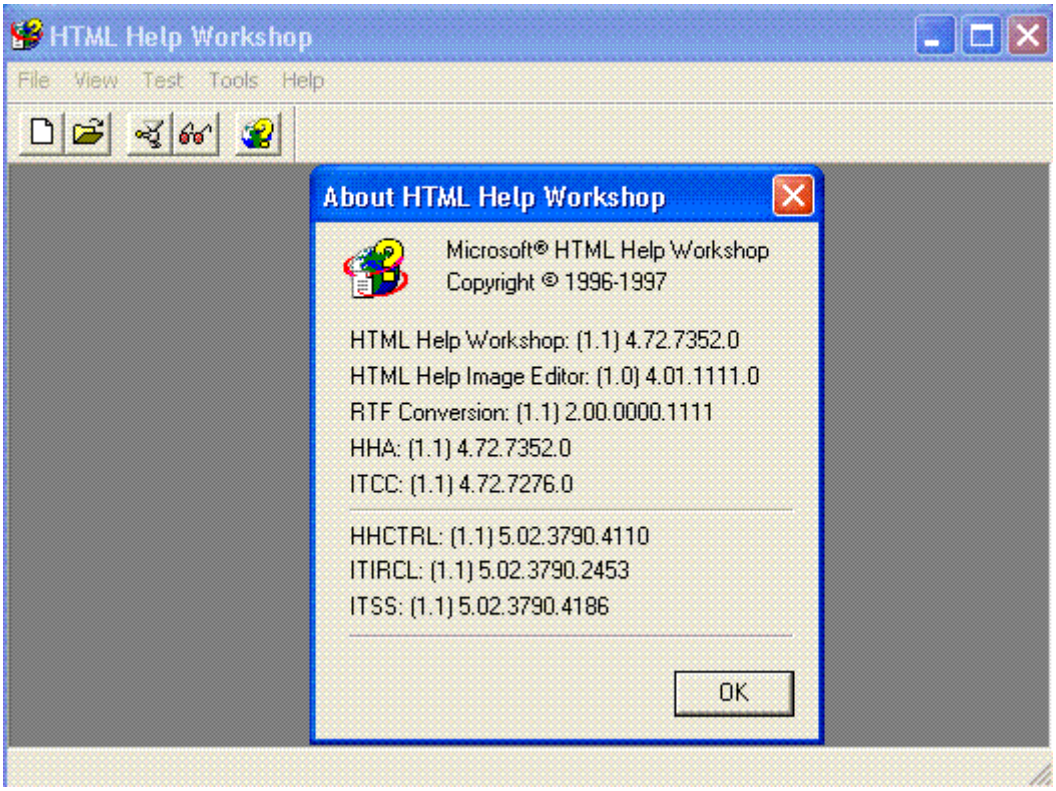
```
[OPTIONS]
errorlog = filename.err      ;salva gli errori in fase di compilazione
title = ... GUIDA            ;titolo della Guida
compress = false             ;con true il file è compattato
warning = 3                  ;visualizza tutti i messaggi di avvertimento
contents = contents          ;nome della prima finestra che contiene il sommario argomenti
[FILES]
filename.rtf                 ;il file che contiene il testo della Guida
[WINDOWS]
main = "titolo",(0,0,1023,1023),,,(192,192,192)
;si possono specificare titolo, posizione e dimensione della finestra
[CONFIG]
```

[MAP]

3. Creazione del file di Guida: HLP
Una guida sviluppata in questo modo è un file compilato con in input un file con estensione HLP e in output si avrà il file HLP.
Il compilatore può essere uno dei seguenti: HC.EXE, HC31.EXE, HCP.EXE e HCRTF.EXE che produce un help file a 32 bit.

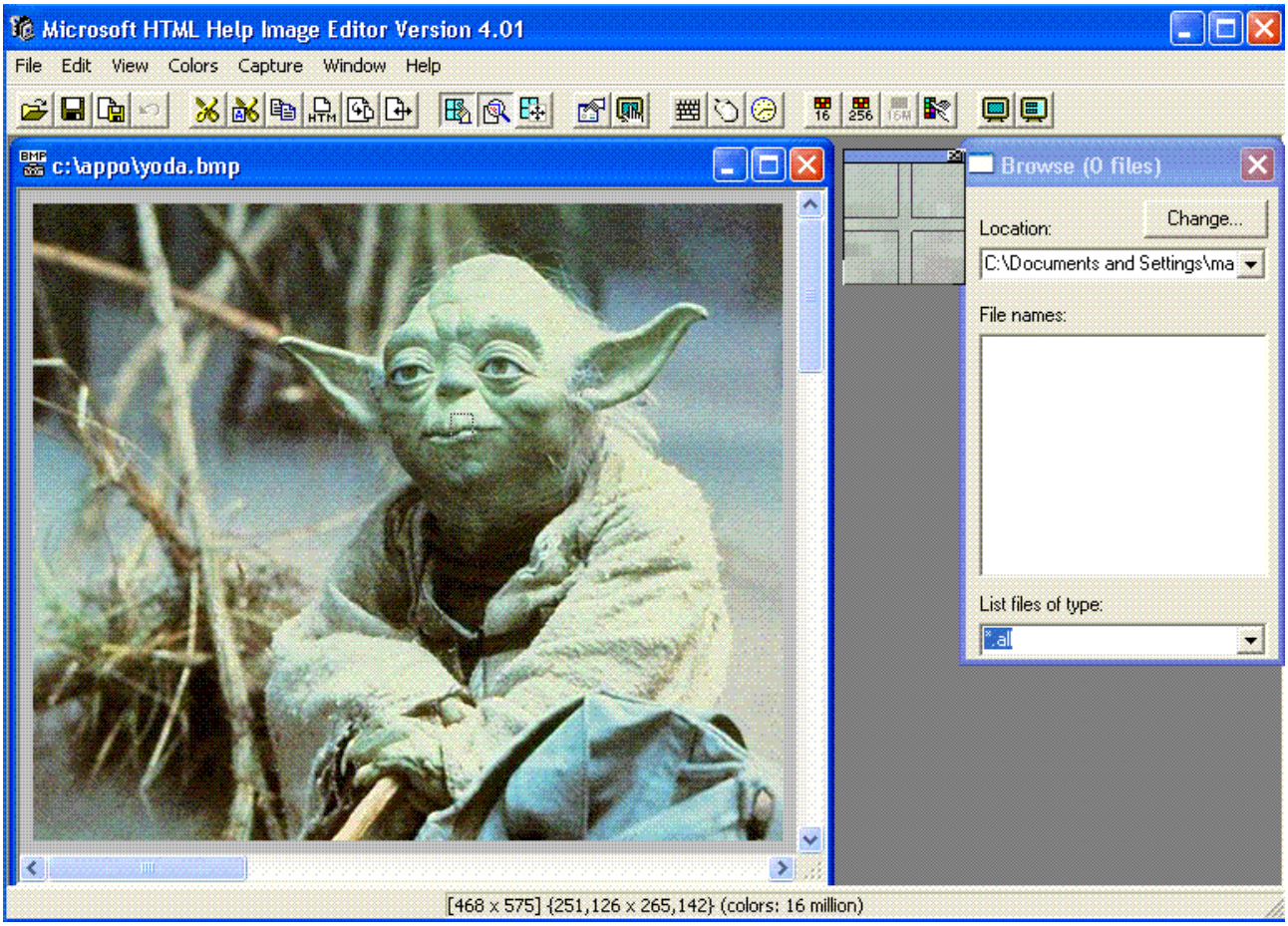
HTML HELP WORKSHOP

Una guida sviluppata in formato HTML Help è un file compilato con estensione CHM, è un ambiente che permette d'implementare guide in linea utilizzando le caratteristiche dell'HTML che possono essere agganciate ad un progetto, sviluppato con Visual Studio, oppure pubblicate su Internet.



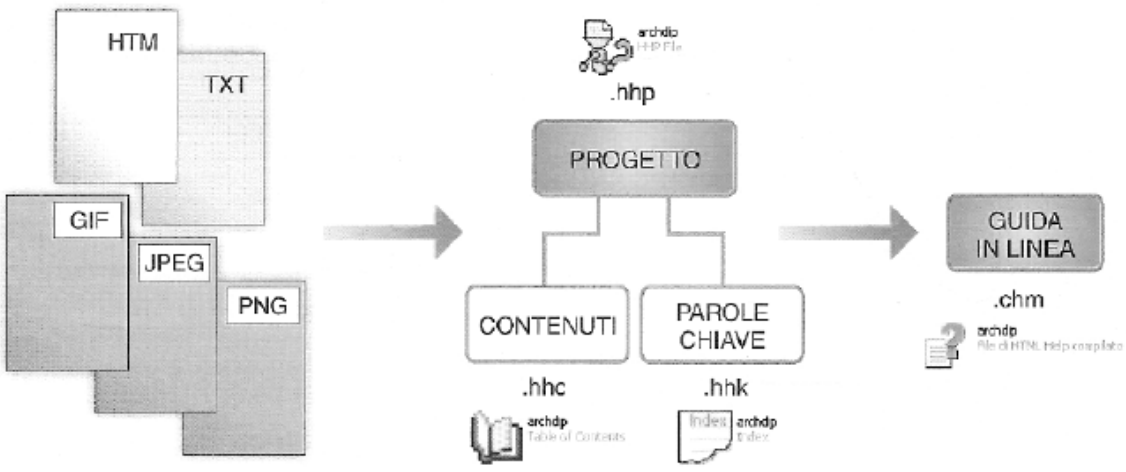
Posizionate sulla barra degli strumenti ci sono cinque icone.
New: crea un nuovo file.
Open: apre un file esistente.
Compile HTML file: compila un progetto.
View compiled file: visualizza un file già compilato.
Display on line information: apre la guida.
L'ambiente è composto dai seguenti componenti.

- ✓ Help Viewer, basa il suo funzionamento sui componenti d'Internet Explorer in particolare su SHDOCVW.DLL per questo è compatibile con tutti i sistemi operativi Microsoft, supporta: HTM, HTML, TXT, ActiveX, Java, JScript, VBScript ed i formati immagini: JPEG, GIF, PNG.
- ✓ Help components.
- ✓ Help Image Editor, è un tool per gestire le immagini, è attivato con il menu ***Tools/HTML Help Image Editor*** e permette di fare varie operazioni sulle immagini, per esempio: conversione di tipi, modifica, organizzazione in album, capture; naturalmente le immagini possono essere inserite nelle pagine, topics, della guida.



1. Creazione della guida

HTML Workshop organizza tutte le risorse della guida in un ***Project***, file HHP.
Le informazioni sugli argomenti della guida sono conservate nel ***Content***, file HHC.
Le parole chiave sono conservate nell'***Index***, file HHK.
Quando il progetto è completato, si esegue la compilazione che produce il file **CHM** (*Compressed HTML Help*), la guida pronta per essere utilizzata dall'applicazione.
Consiste in un insieme di pagine scritte in un subset di HTML e con un indice di hyperlink, è ottimizzato per la lettura, dato che i file sono indicizzati, tutti i file sono compressi con algoritmo LZX e i browser CHM possono visualizzare l'indice accanto al testo della pagina.
Nei file CHM è possibile associare ad ogni controllo dell'interfaccia una parola chiave e quindi aprire la guida selezionando proprio la pagina che descrive la funzionalità desiderata.



Importante è il lavoro di progettazione per definire i contenuti, la sequenza, i collegamenti

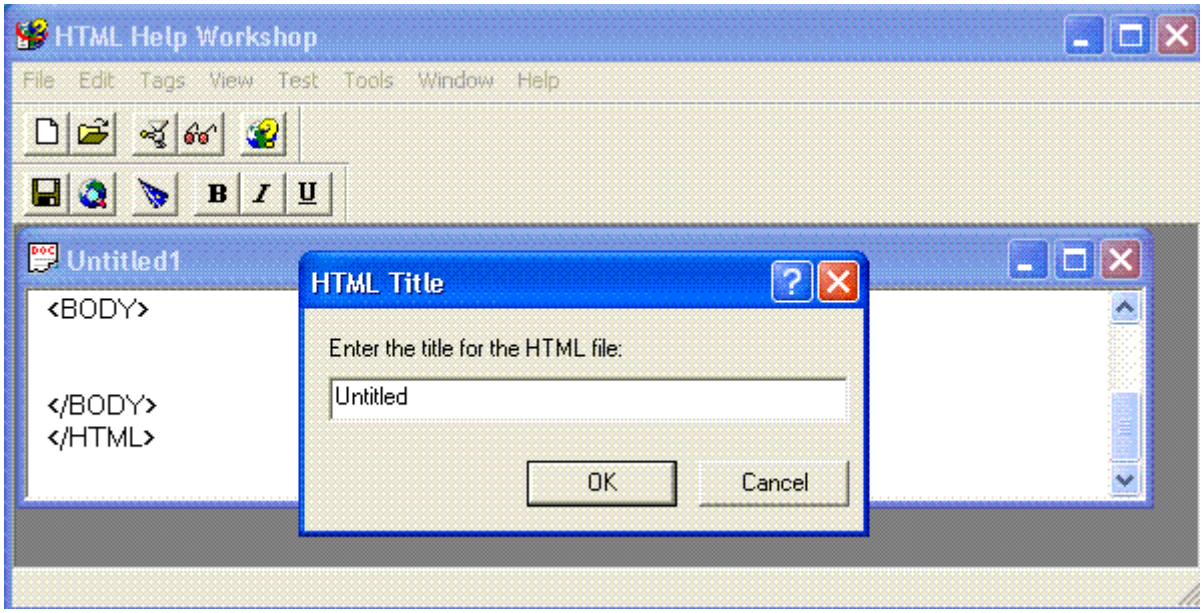
e le parole chiave.
Per esempio, la guida in linea deve contenere gli argomenti e i sotto argomenti.

1. Presentazione dell'applicazione.
   1.1. Le informazioni.
   1.2. L'identificazione.
2. Il menu dell'applicazione.
   2.1. Inserisci.
   2.2. Modifica.
   2.3. Visualizza.

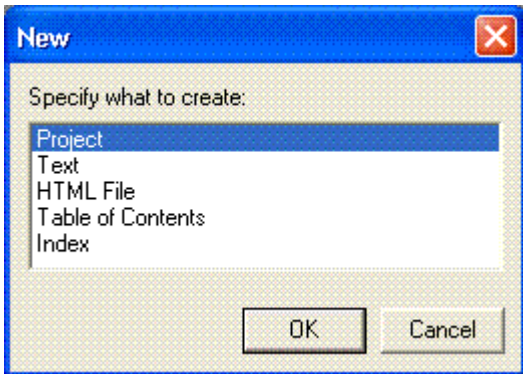
In corrispondenza di ciascun contenuto si deve progettare la pagina HTM relativa, ottenendo i seguenti file.
DEFAULT.HTM, è la pagina di apertura della guida, la presentazione.
INFORMAZIONI.HTM.
IDENTIFICAZIONE.HTM.
MENU.HTM.
INSERISCI.HTM.
MODIFICA.HTM.
VISUALIZZA.HTM.
All'interno di ciascun file HTM s'inseriscono i link che consentono all'utente di effettuare collegamenti tra i diversi file.
Durante la progettazione bisogna anche costruire la tabella delle parole chiave della guida e le corrispondenze con i file ai quali esse si riferiscono.

Parole chiave	File
Archivio	default, informazioni
Dati	informazioni, identificazione

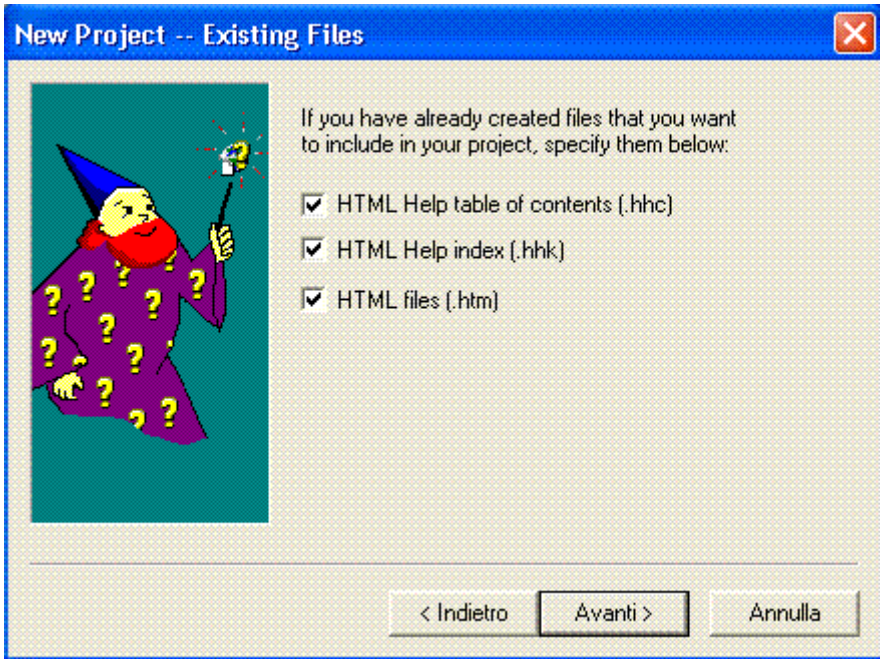
Fare clic sul menu ***File/New/HTML File***.



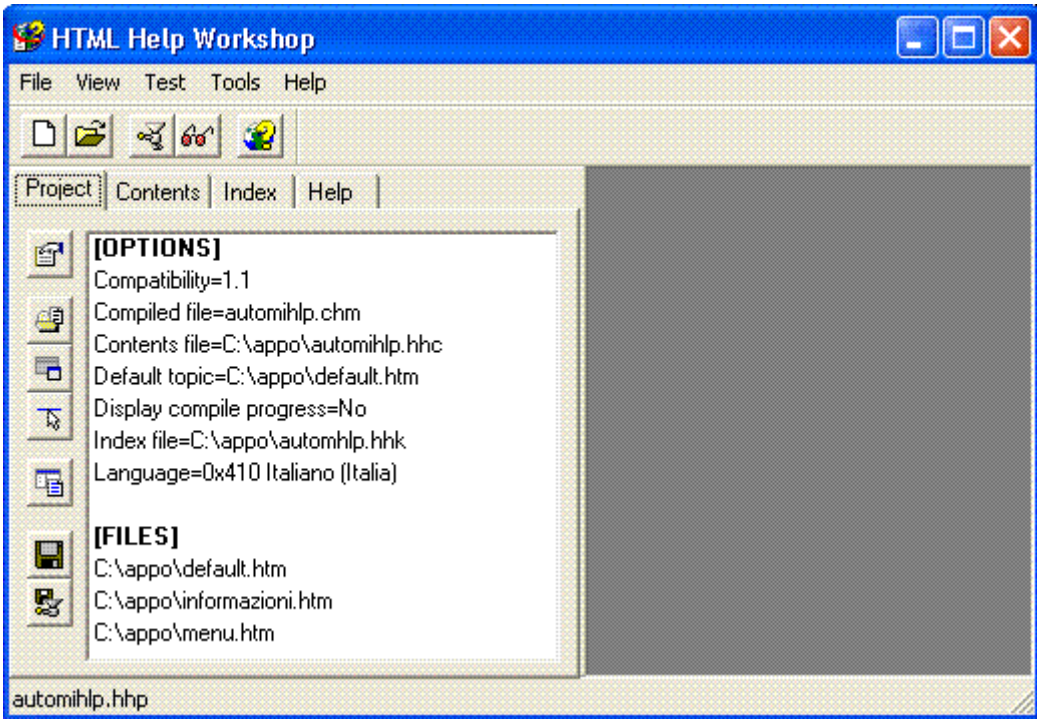
Per creare un nuovo progetto, fare clic sul menu ***File/New/Project***, con una serie di schermate guidate da un wizard si costruisce il progetto passo per passo.



Occorre assegnare il nome del progetto e selezionare i file che si vogliono inserire.



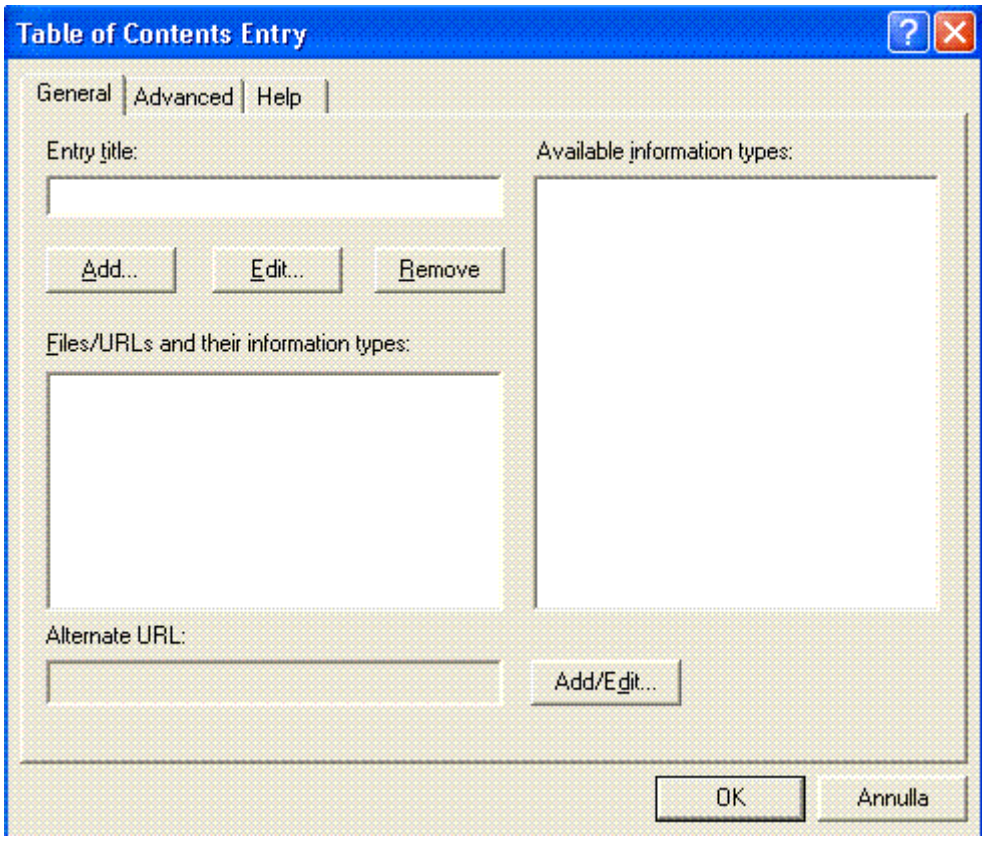
Al termine il progetto è creato su disco con il nome assegnato e l'estensione HHP.



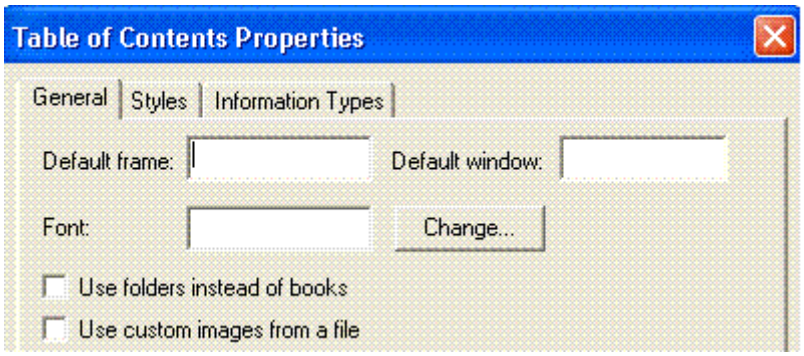
Le icone a sinistra della scheda ***Project*** consentono di effettuare le seguenti operazioni.
Impostare le opzioni del progetto.
Inserire o rimuovere file dalla sezione ***[FILES]***.
Visualizzare il codice HTM dei file.
Salvare il progetto e lanciare la compilazione.

2. Creazione della tabella dei contenuti
I ***Contents*** sono gli argomenti che sono visualizzati nella scheda ***Sommario*** della guida.
Fare clic sulla scheda ***Contents*** e selezionare il file con estensione HHC da inserire nel progetto, se esiste, altrimenti si visualizza la seguente finestra.

Gli argomenti sono specificati nella finestra che si apre facendo clic sull'icona ***Insert a page***.
Nella casella ***Entry title:*** s'inserisce il titolo da assegnare a un argomento della guida che sarà inserito nel ***Sommario***, il pulsante ***Add…*** consente di selezionare il file da associare al titolo.
Le frecce a sinistra verso l'alto e verso il basso permettono di modificare l'ordine degli argomenti nell'elenco, invece le frecce verso destra e verso sinistra stabiliscono o rimuovono i rientri creando così livelli gerarchici che diventeranno argomenti e sotto argomenti nel ***Sommario*** della guida.

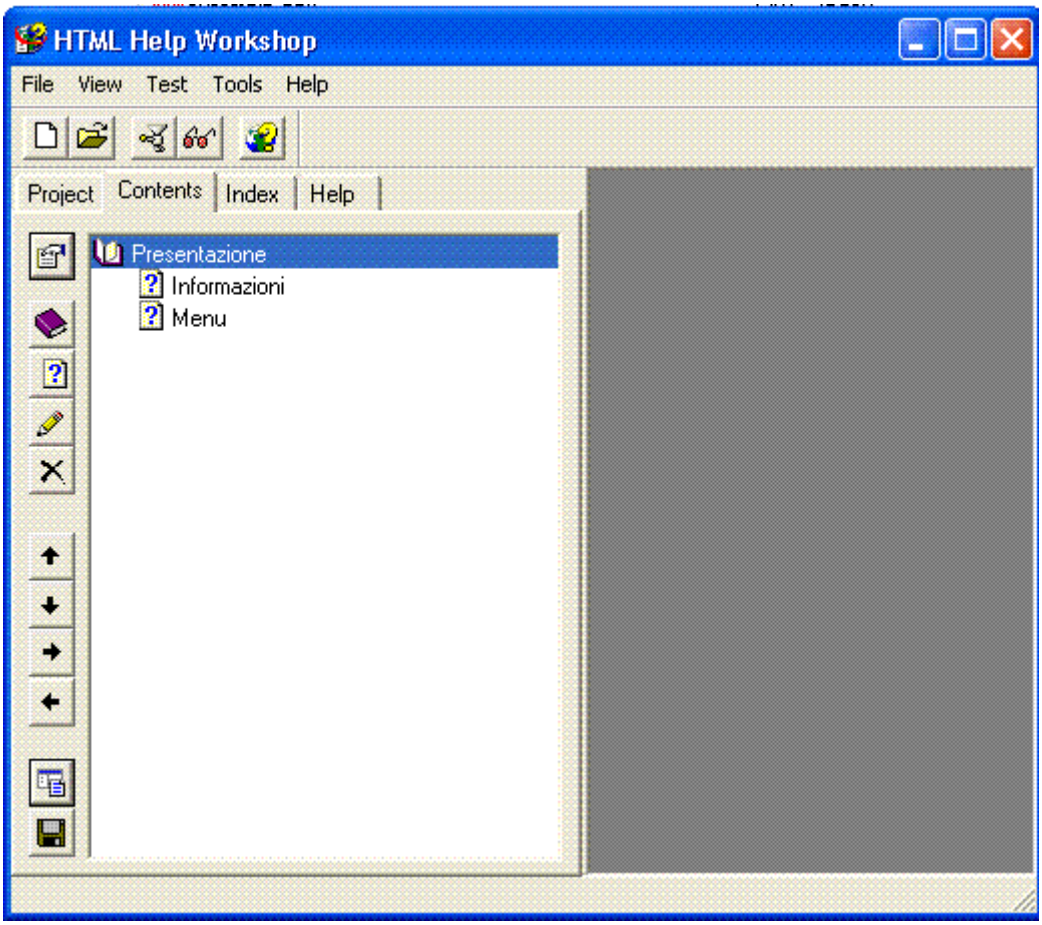


Fare clic sull'icona ***Content properties*** della finestra principale.



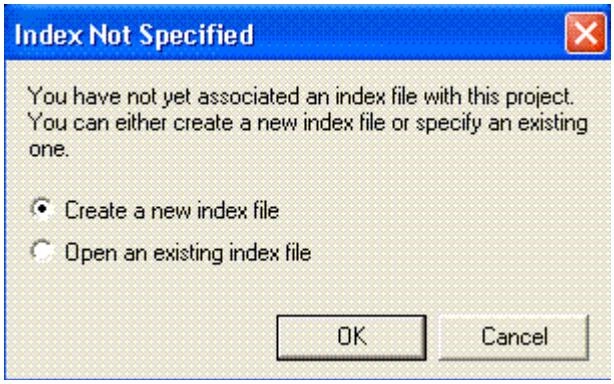
In questa finestra si decidono le due forme delle icone visualizzate accanto agli argomenti.

1. L'immagine di un libro per gli argomenti e di un foglio con il punto interrogativo per i sotto argomenti, default.
2. L'immagine di una cartella per gli argomenti e di un foglio per i sotto argomenti, segno di spunta nella casella ***Use folders instead of books***.

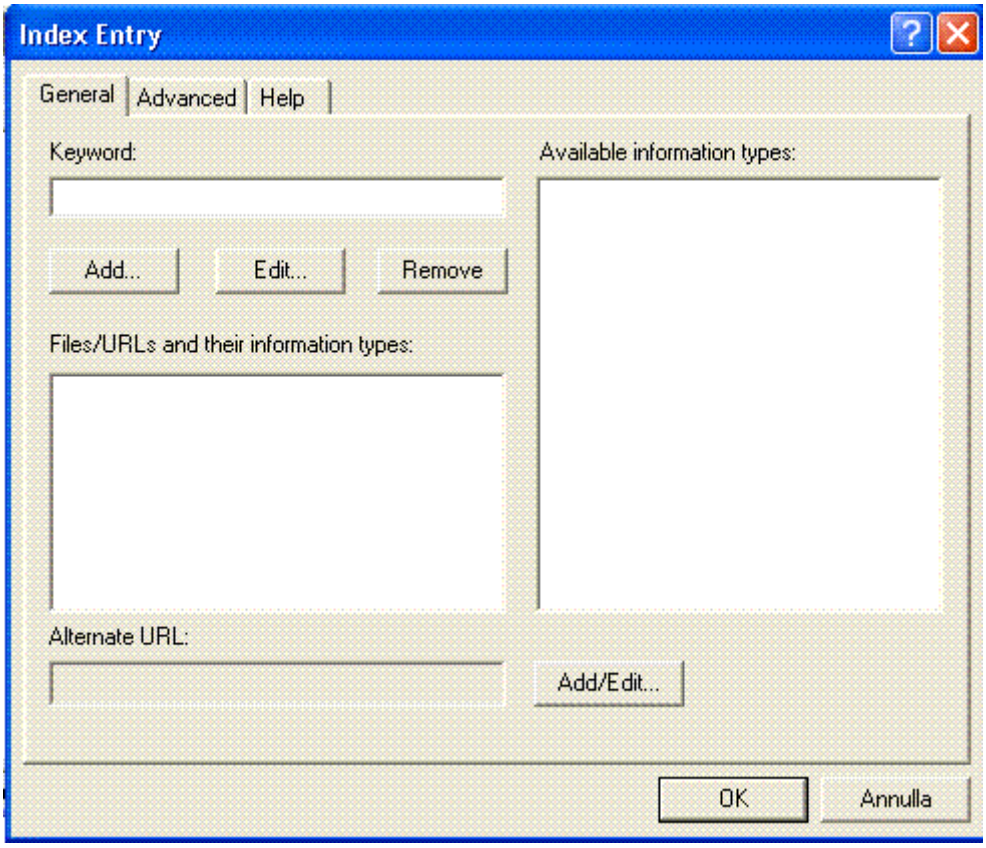


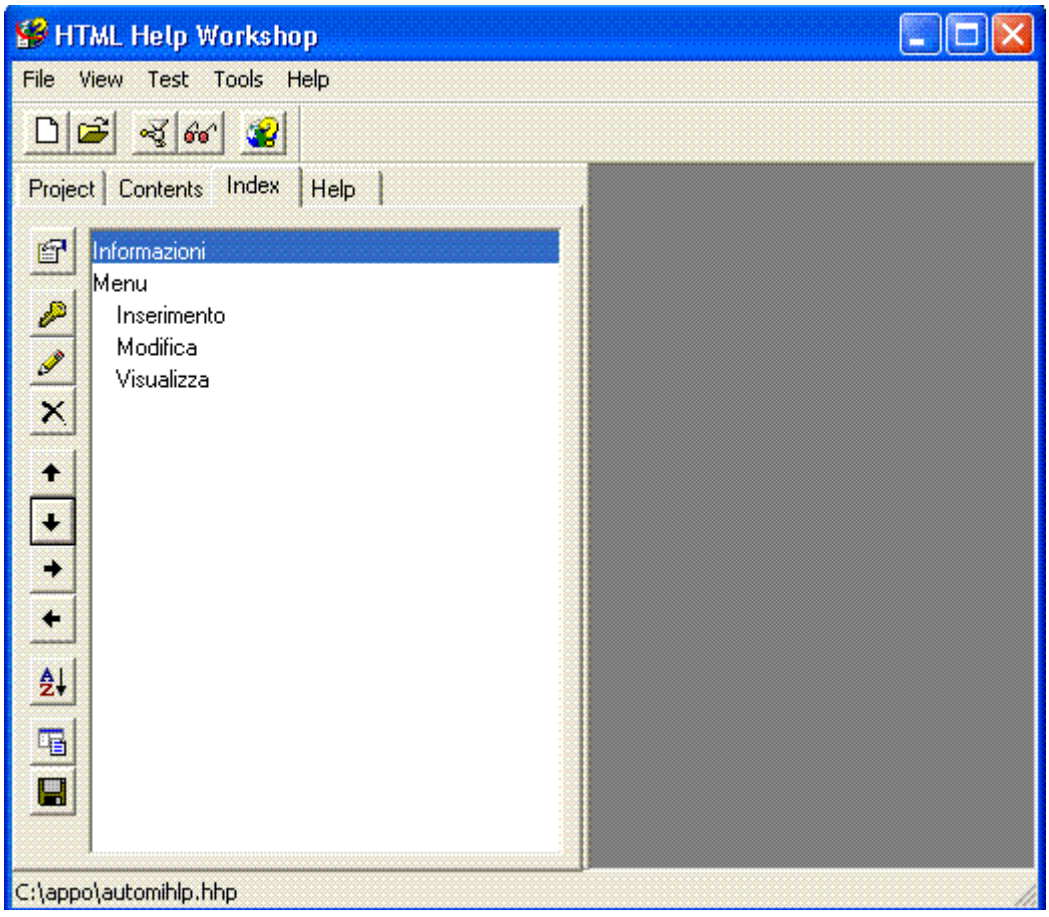
3. Creazione dell'indice delle parole chiave

Fare clic sulla scheda ***Index*** e selezionare il file con estensione HHK da inserire nel progetto, se esiste, altrimenti si visualizza la seguente finestra.



Fare clic sull'icona ***Insert a keyword*** della finestra principale, sono le parole chiave che compariranno nella scheda ***Indice*** della guida.
Nella casella ***Keyword:*** s'inserisce la parola chiave, il pulsante ***Add…*** consente di selezionare il file da associare alla parola chiave.
Le frecce a sinistra verso l'alto e verso il basso permettono di modificare l'ordine delle parole chiave nell'elenco, invece le frecce verso destra e verso sinistra stabiliscono o rimuovono i rientri creando così livelli gerarchici.

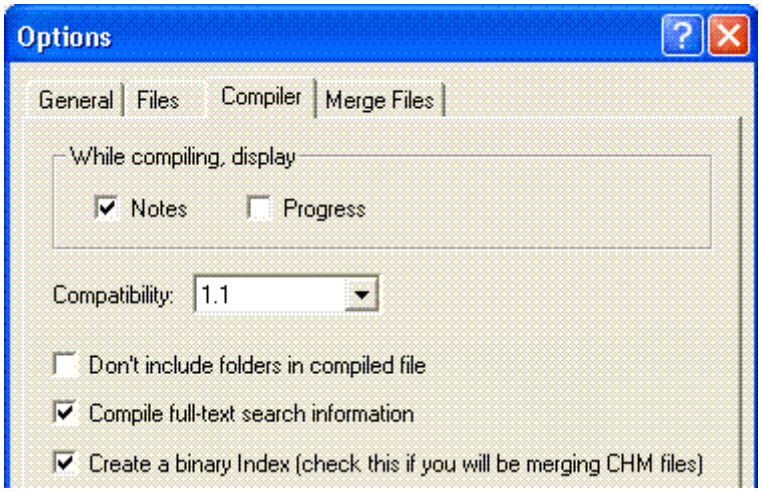




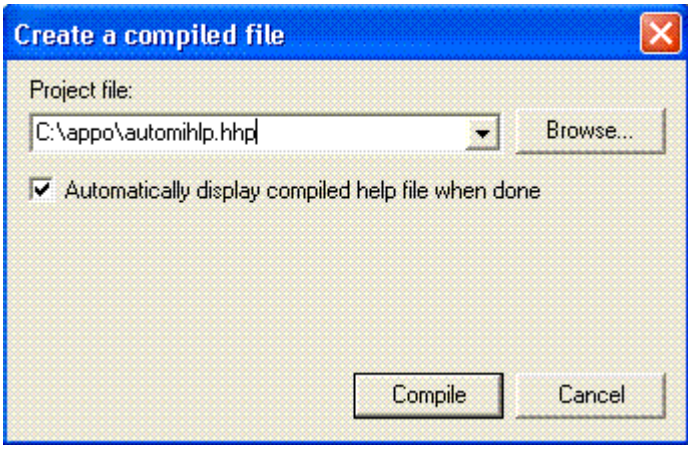
4. Compilazione del progetto

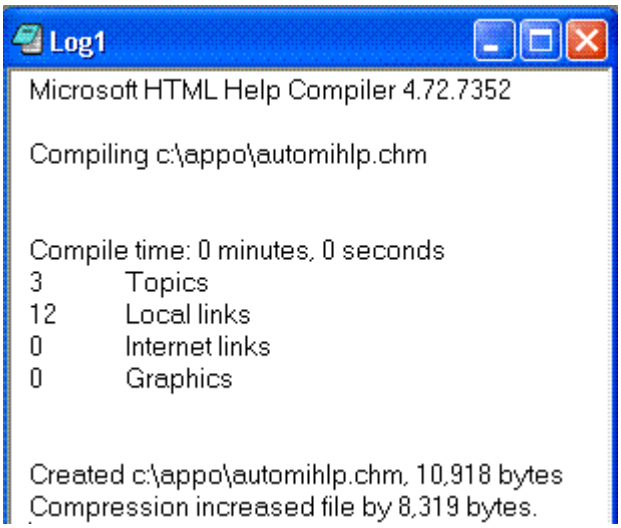
La compilazione, se non genera errori, produce come risultato il file CHM, che è il file della guida dell'applicazione usata dall'utente.

Prima di lanciare la compilazione, è buona regola di programmazione definire le opzioni del progetto, fare clic sulla scheda ***Project*** icona ***Change project options***.
Nella finestra che si apre, selezionare la scheda ***Compiler*** si può mettere il segno di spunta nella casella ***Compile full-text search information:*** in questo modo si chiede al compilatore di creare una guida che consenta la ricerca libera su tutto il testo.
Attivando questa opzione, nell'help della guida compare anche la scheda ***Cerca*** e dopo aver scelto una parola, la guida visualizza tutte le pagine che la contengono, evidenziando in ciascuna la parola trovata.

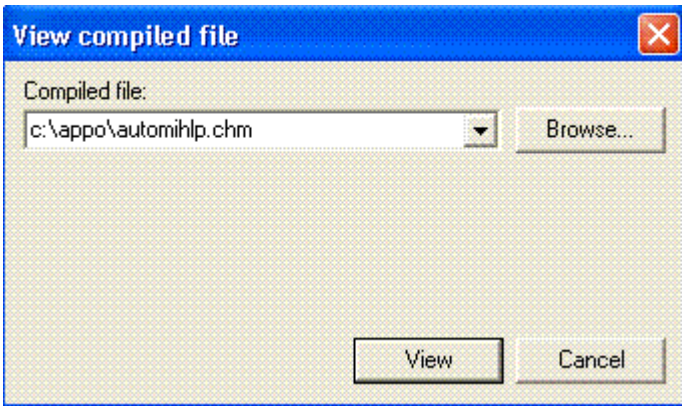


Per compilare il progetto, selezionare il menu ***File/Compiled…***.
Se la compilazione non genera errori, si crea il file AUTOMIHLP.CHM e presenta automaticamente l'anteprima della guida.

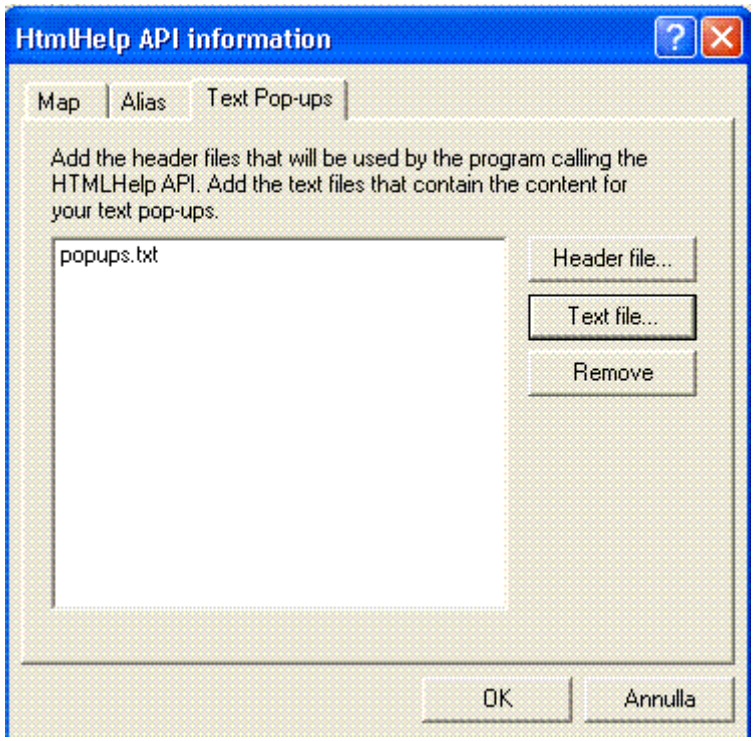




La visualizzazione della guida si attiva anche con il menu ***View/Compiled File…*** .



Popups
Sono predisposti attraverso un file di testo rispettando la seguente sintassi.
✓ *.topic "numero"*
✓ *"testo del messaggio"*
Per esempio si possono definire i seguenti popups.
• *.topic 20000 – Informazioni.*
• *.topic 20010 – Inserimento.*
• *.topic 20020 – Modifica.*
Il testo è inserito nel file POPUPS.TXT e salvato nella stessa cartella del progetto.
Quindi bisogna associare il file al progetto, selezionare l'icona a sinistra della scheda ***Project*** di nome ***HtlmHelp API information***, nella finestra che si apre fare clic sulla scheda ***Text Pop-ups***.



Dopo queste operazioni, nella finestra è visualizzato un nuovo file, compilare il progetto.

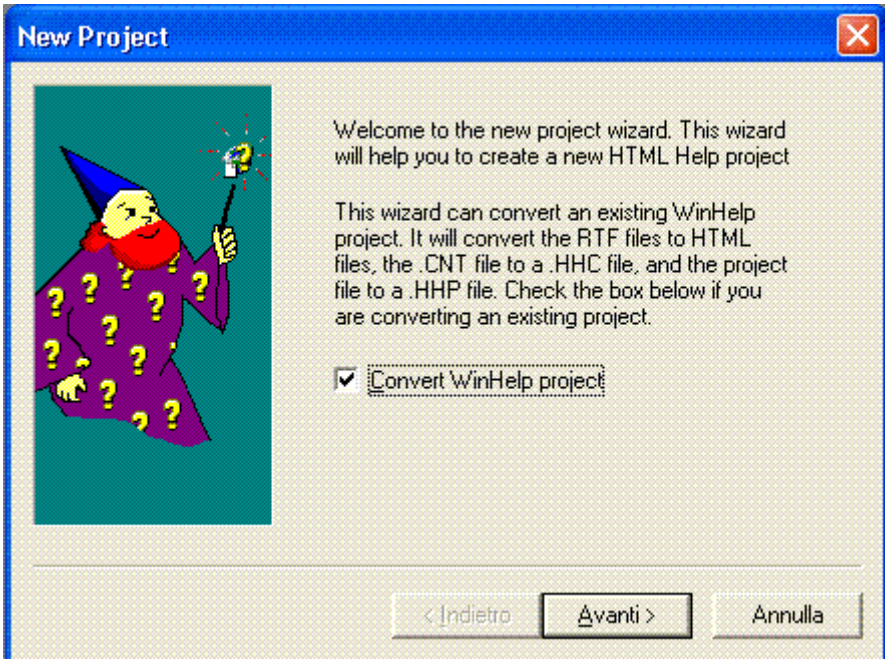
```
[OPTIONS]
Compatibility=1.1
Compiled file=automihlp.chm
Contents file=automihlp.hhc
Default topic=default.htm
Display compile progress=No
Full-text search=Yes
Index file=automihlp.hhc
Language=0x410 Italiano (Italia)

[FILES]
default.htm
informazioni.htm
menu.htm

[TEXT POPUPS]
popups.txt
```

Il nome è utilizzato quando si aggancia il file CHM al progetto, la sintassi è la seguente.
automihlp.chm::/popups.txt

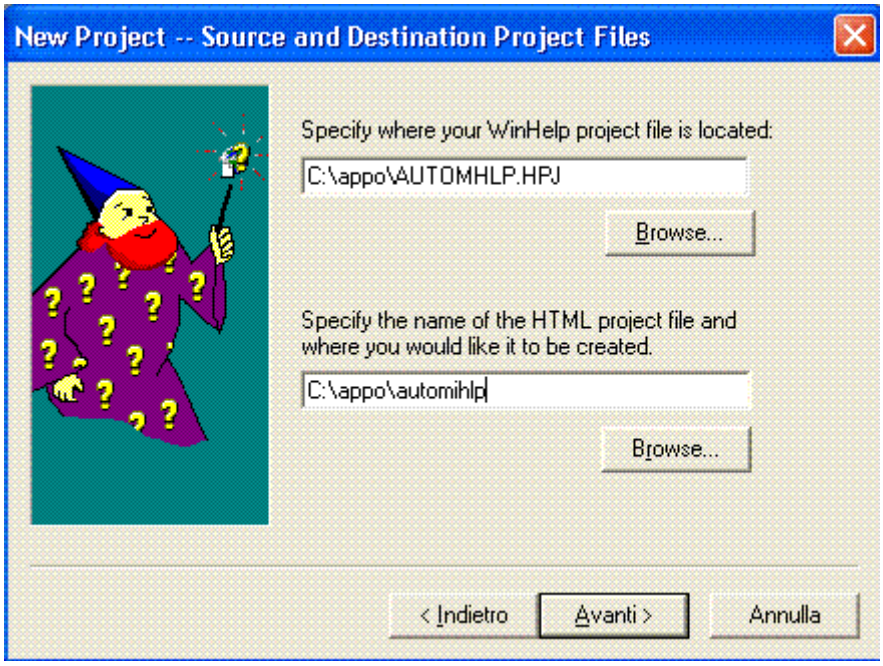
Esempio, alla variabile *VarHelp* che contiene il path del file CHM dovrà essere assegnato il seguente valore.
VarHelp = App.Path & "\htmlhelp\esempio.chm::/popups.txt"

Questa assegnazione, però, non deve trarre in inganno infatti, il file POPUPS.TXT non deve essere distribuito insieme alla guida dato che è incluso nel file compilato.

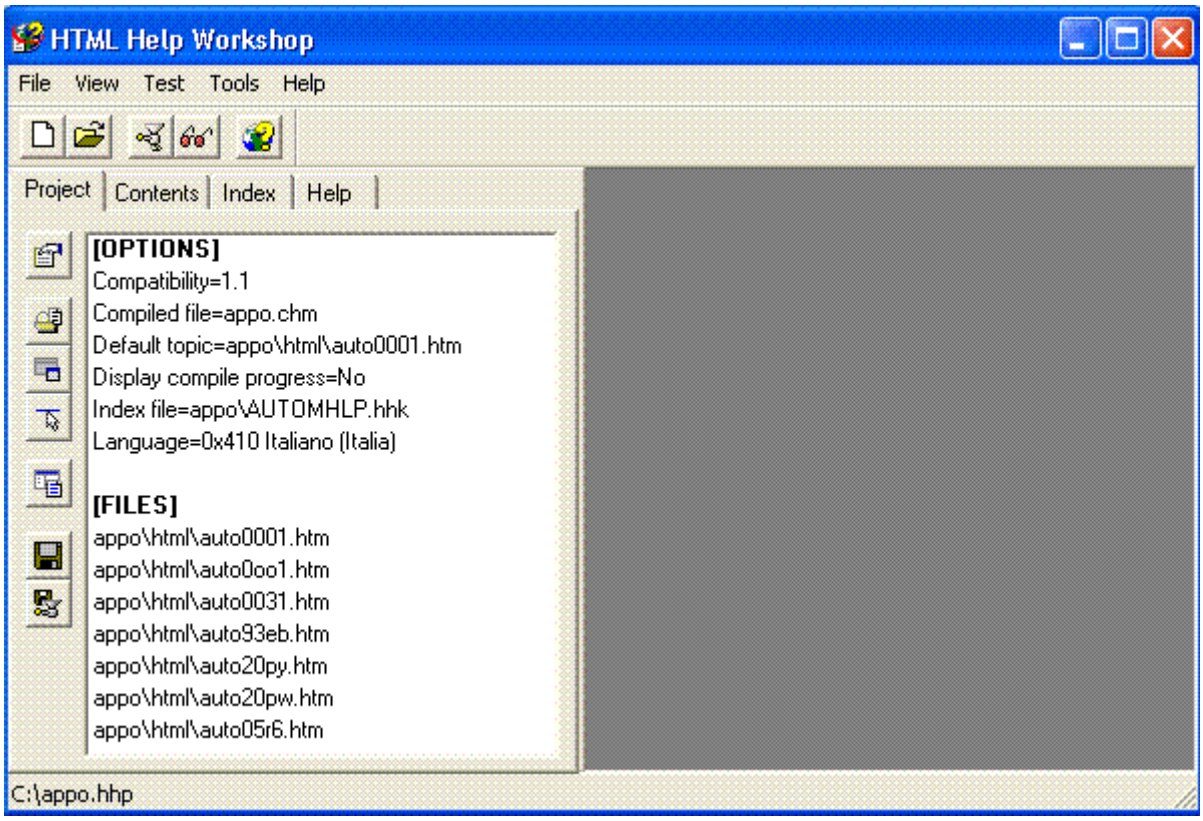
Conversione di un progetto WinHelp in HTML Help
Creare un nuovo progetto con il menu ***File/New/Project***, il wizard si attiva e sulla prima maschera si deve selezionare ***Convert WinHelp project***.



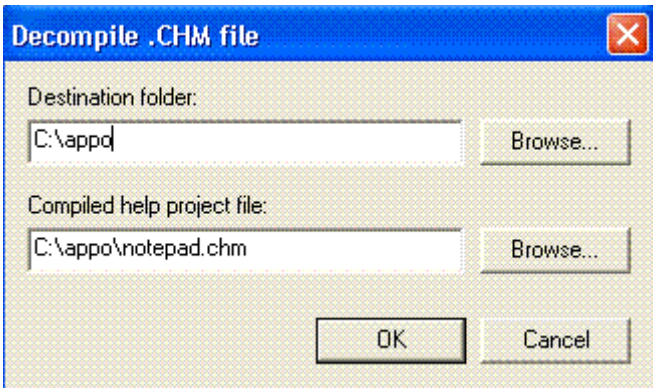
Nella schermata successiva scegliere la cartella dove si trova i file del progetto HPJ.



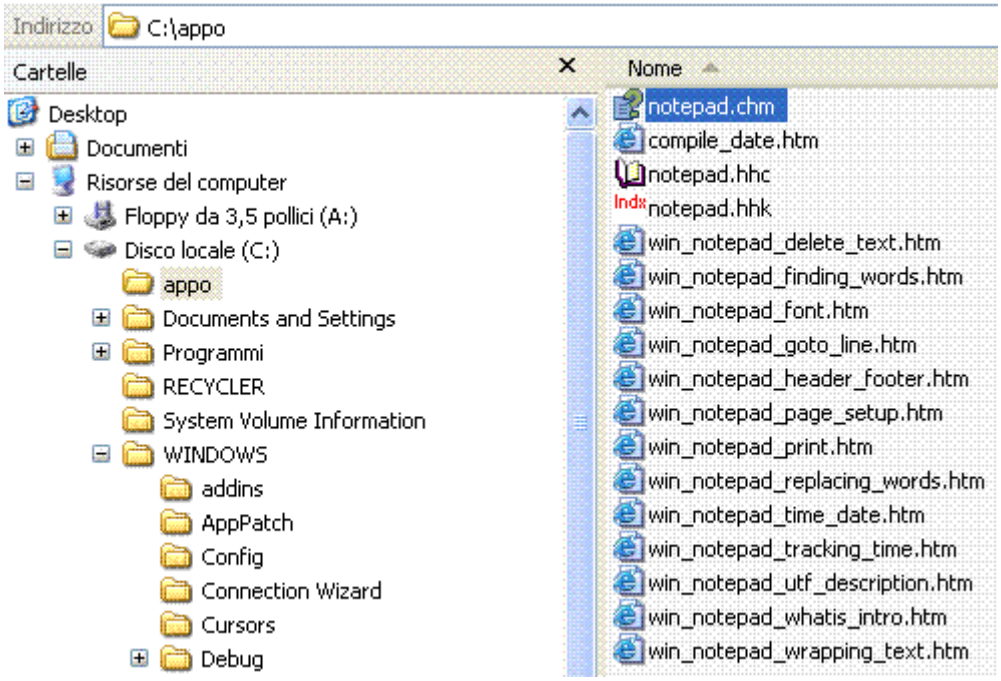
Il wizard crea: il file di progetto, il file di contesto e i topics HTM che sono inseriti in una cartella chiamata HTML.



Decompilazione di un progetto
Per decompilare un file CHM, fare clic sul menu ***File/Decompiled…***.

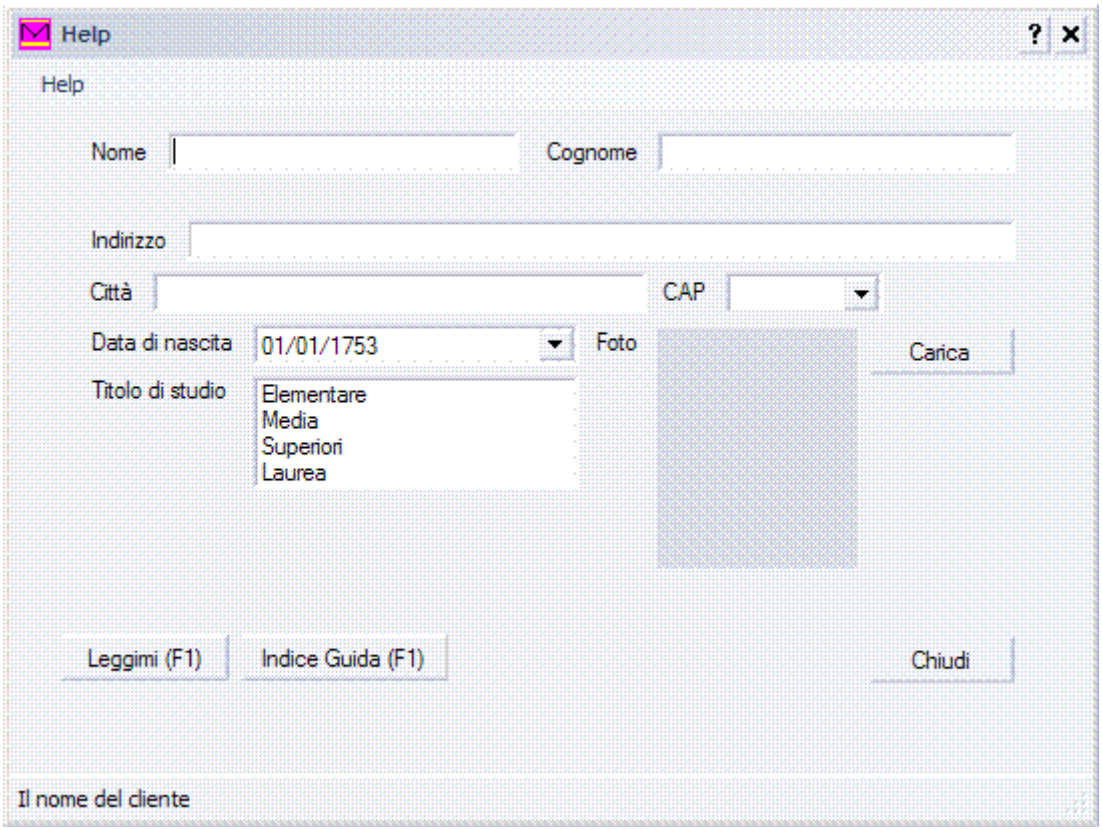


Dopo aver fatto clic su ***OK***, nella cartella sono creati i seguenti file.



APPLICAZIONE

Il .NET Framework fornisce le classi necessarie ad implementare sistemi di guida per le applicazioni Windows Forms.
Scrivere la documentazione e i file della guida di un'applicazione è spesso un compito sottovalutato o addirittura tralasciato.
Esistono diverse tecniche per aggiungere una funzionalità di guida ad un'applicazione, la maggior parte sono diventate uno standard de facto sia nel mondo Windows sia Linux.
Ad esempio, ogni applicazione ha un menu help ***?***, il tasto ***F1*** attiva un help contestuale sulla funzione che si sta utilizzando, fumetti o tooltip che appaiono posizionando il mouse su un controllo e spiegando la sua funzione.
Progettare un'applicazione Windows costituita da una singola finestra e da qualche controllo, per esempio pulsanti e caselle di testo, per inserire i dati personali.



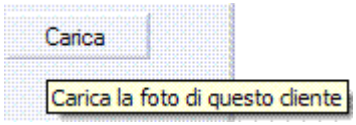
Tooltip

L'utente si chiede, ad esempio, cosa serve il pulsante ***Carica***, utilizzando un tooltip quando si posiziona il puntatore del mouse su di esso si può far apparire una spiegazione. Fare clic sulla ***Casella degli strumenti*** e trascinare il componente *ToolTip* sul form per fare in modo che ogni controllo sia dotato di una nuova proprietà, *Tooltip on Nometooltip*.

Impostando una stringa essa sarà utilizzata come *ToolTip* per il controllo.
La stessa cosa si può ottenere via codice, per creare un'istanza del *ToolTip* è necessario invocare il costruttore di default e poi utilizzare il metodo *SetTooltip* specificando il controllo e la stringa da mostrare.

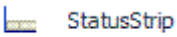
```
ToolTip mioTooltip = new ToolTip();
mioTooltip.SetToolTip(btCarica, "Carica la foto di questo cliente");
mioTooltip.SetToolTip(btClose, "Chiudi la finestra");
```

Una sola istanza di *ToolTip* può essere utilizzata con tutti i controlli presenti sul form.
Posizionando il puntatore del mouse sul pulsante si ottiene la seguente spiegazione.



Barra di stato

Un altro sistema per mostrare informazioni su ciò che sta accadendo su una finestra dell'applicazione, è quello di mostrare una stringa sulla barra di stato, che varia ad esempio spostando il focus da un controllo all'altro, mostrando in tempo reale una descrizione dell'informazione data dal controllo.
Fare clic sulla ***Casella degli strumenti*** e trascinare il componente *StatusStrip* sul form.



StatusStrip crea una *ToolStripStatusLabel* al suo interno, sarà necessario gestire l'evento *MouseMove* dei controlli interessati e dell'intero form, impostando la proprietà *Text* della *ToolStripStatusLabel*.

```
private void txtNome_MouseMove(object sender, MouseEventArgs e)
      { this.toolStripStatusLabel.Text = "Il nome del cliente"; }
private void txtCognome_MouseMove(object sender, MouseEventArgs e)
      { this.toolStripStatusLabel.Text = "Il cognome del cliente"; }
private void dateTimePicker1_MouseMove(object sender, MouseEventArgs e)
      { this.toolStripStatusLabel.Text = "La data di nascita del cliente"; }
private void listBox1_MouseMove(object sender, MouseEventArgs e)
      { this.toolStripStatusLabel.Text = "Il titolo di studio del cliente"; }
private void txtIndirizzo_MouseMove(object sender, MouseEventArgs e)
      { this.toolStripStatusLabel.Text = "L'indirizzo del cliente"; }
private void txtCittà_MouseMove(object sender, MouseEventArgs e)
      { this.toolStripStatusLabel.Text = "La città di resdienza del cliente"; }
private void cboCAP_MouseMove(object sender, MouseEventArgs e)
      { this.toolStripStatusLabel.Text = "Il CAP di residenza del cliente"; }
private void pictureBoxFoto_MouseMove(object sender, MouseEventArgs e)
      { this.toolStripStatusLabel.Text = "La foto del del cliente"; }
private void Form1_MouseMove(object sender, MouseEventArgs e)
      { this.toolStripStatusLabel.Text = ""; }
```

Nel caso dell'evento *MouseMove* relativo al form, si resetta il contenuto con una stringa vuota.

Classe *ErrorProvider*

Permette di dare all'utente dell'applicazione un feedback su determinate situazioni di errore, è utilizzata quando dei controlli da valorizzare sono riempiti con valori non validi. Per esempio, una *TextBox* che deve contenere solo numeri, oppure che deve essere validata secondo certi criteri, come numeri solo positivi o stringhe in un certo formato. Mediante i metodi della classe *ErrorProvider*, è possibile mostrare all'utente un'icona al lato di un determinato controllo che non è stato valorizzato correttamente.
Fare clic sulla ***Casella degli strumenti*** e trascinare il componente *ErrorProvider* sul form. Per esempio, il nome del cliente è un campo obbligatorio, quando si valida il form, chiudendolo, è possibile verificare la presenza del nome del cliente e se il campo è vuoto mostrare un'icona di errore.

```
private void btClose_Click(object sender, EventArgs e)
      {  if (txtNome.Text == String.Empty)
         {  errorProvider = new ErrorProvider();
            errorProvider.SetError(txtNome, "Inserire il nome del cliente");
         }
}
```

L'icona mostra un punto esclamativo, per default sulla destra del controllo interessato, che lampeggia un determinato numero di volte e che mostra un *ToolTip* posizionandovi sopra il puntatore.

È possibile configurare diversi aspetti dell'*ErrorProvider*, per mezzo di proprietà e metodi.
Impostando la proprietà *BlinkRate* è possibile variare la velocità di lampeggio dell'icona di errore, data in millisecondi e che per default è di 250 millisecondi.
Se invece, ad esempio, si vuole che l'icona lampeggi ad intervalli di 1 secondo, sarà sufficiente impostare il valore di *BlinkRate* a1000.

```
errorProvider.BlinkRate = 1000;
```

La proprietà *BlinkStyle* permette di definire quando l'icona deve lampeggiare, usando uno dei valori dell'enumerazione *ErrorBlinkStyle*.
Per default il blink è sempre attivo, quindi impostata a *AlwaysBlink* e quindi esso si verifica ad ogni occorrenza dell'errore.
Se invece si desidera che l'icona rimanga fissa, si può usare il valore *NeverBlink*, o ancora se il lampeggio deve verificarsi solo su errori diversi da eventuali errori precedenti sullo stesso controllo, bisogna utilizzare il valore *BlinkIfDifferentError*.

```
errorProvider.BlinkStyle = ErrorBlinkStyle.AlwaysBlink;
```

È possibile configurare il posizionamento dell'icona stessa oppure un'icona personalizzata al posto del punto esclamativo.
Il metodo *SetIconAlignment* imposta l'allineamento dell'icona rispetto al controllo associato, utilizzando uno dei valori dell'enumerazione *ErrorIconAlignment*.
Per esempio se si vuol posizionare l'icona alla destra della *TextBox* e allineata al centro.

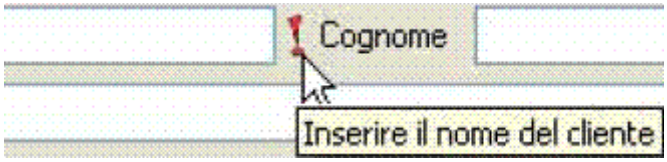
```
errorProvider.SetIconAlignment(txtNome, ErrorIconAlignment.MiddleRight);
```

La distanza dal controllo è controllata dal metodo *SetIconPadding*, con cui è possibile specificare il numero di pixel da utilizzare per il distanziamento.

```
errorProvider.SetIconPadding(txtNome, 1);   // un solo pixel di padding
```

Per utilizzare un'icona personalizzata basta impostare la proprietà *Icon*.

```
errorProvider.Icon = new Icon("erroricon.ico");
```

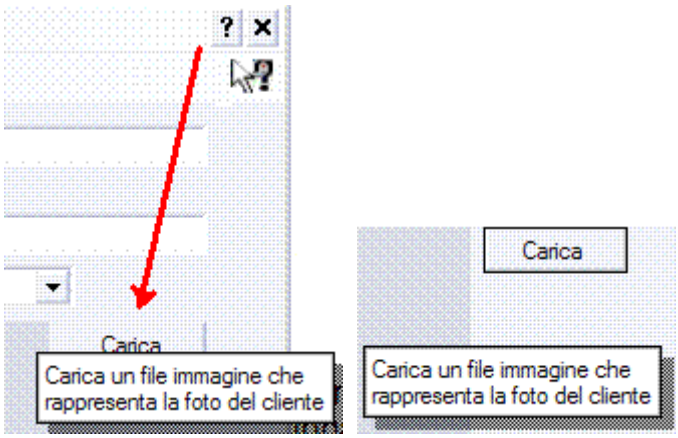


Classe *HelpProvider*
L'help contestuale consente di fornire all'utente aiuto ed informazioni su un particolare controllo, semplicemente selezionandolo e utilizzando i tasti di scorciatoia opportuni.
Per far ciò, è possibile in .NET utilizzare la classe *HelpProvider*.
Fare clic sulla ***Casella degli strumenti*** e trascinare il componente *HelpProvider* sul form, è possibile impostare la stringa di help per ogni controllo che si vuole, ad esempio, per la TextBox *txtNome* si può scrivere il codice seguente.

```
HelpProvider helpProvider = new HelpProvider();
```

```
helpProvider.SetHelpString(txtNome, "Il nome del cliente");
helpProvider.SetHelpString(txtCognome, "Il cognome del cliente");
helpProvider.SetHelpString(txtIndirizzo, "L'indirizzo del cliente");
helpProvider.SetHelpString(cboCAP, "Il CAP di residenza del cliente");
helpProvider.SetHelpString(txtCittà, "La città di residenza del cliente");
helpProvider.SetHelpString(lstTitoloDiStudio, "Il titolo di studio del cliente");
helpProvider.SetHelpString(pictureBoxFoto, "La foto del cliente");
helpProvider.SetHelpString(btCarica, "Carica un file immagine che rappresenta la foto del
cliente");
helpProvider.SetHelpString(btClose, "Chiude la finestra");
```

Cliccando sul ***?*** del form, il puntatore visualizzerà un punto interrogativo e ogni controllo per cui è stata impostata una stringa di help visualizzerà al clic su di esso tale stringa in una sorta di tooltip.
La stessa cosa avverrà, senza alcuna aggiunta di codice, cliccando il tasto F1 dopo aver impostato il focus su un controllo.



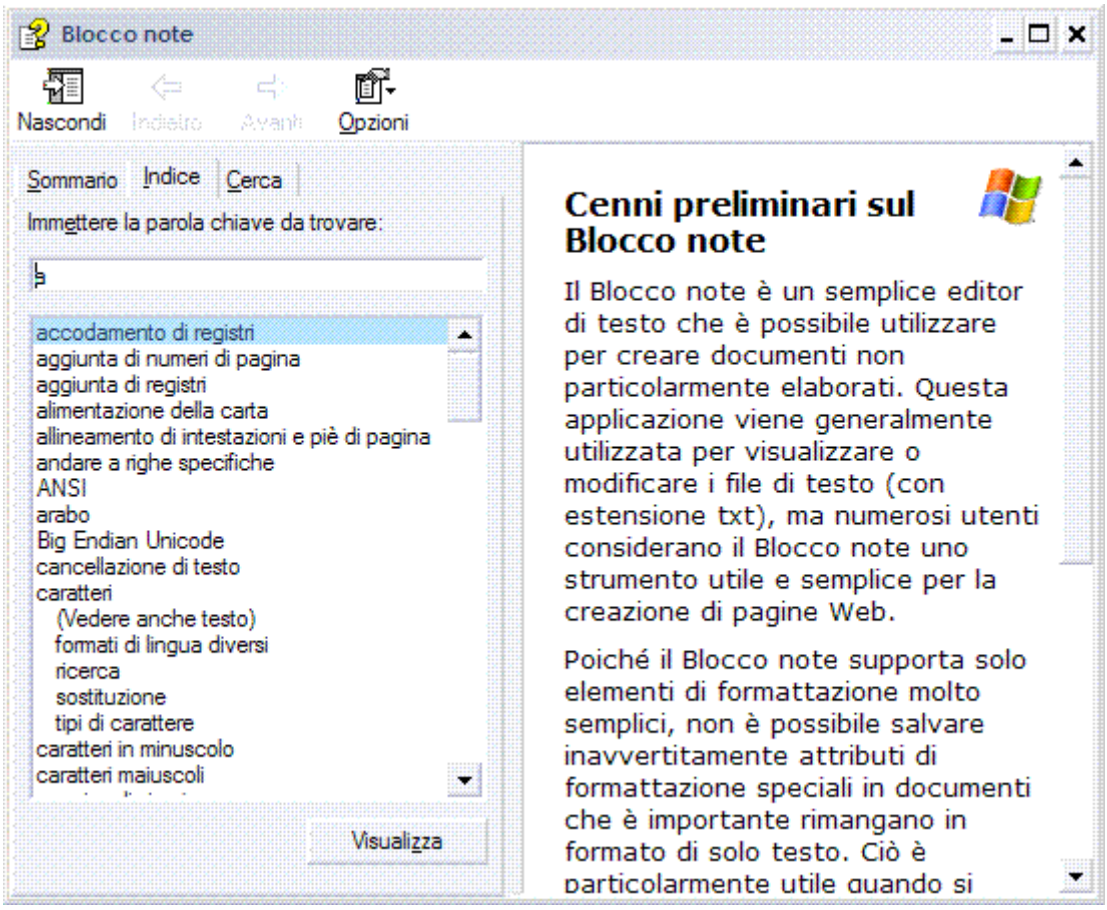
Per visualizzare un pulsante di help sulla finestra, è necessario impostare a *True* la proprietà *HelpButton* del form e quindi impostare a *False* le proprietà *MaximizeBox* e *MinimizeBox*.
Ciò visualizzerà sulla barra del titolo della finestra un pulsante con un punto interrogativo, che una volta cliccato attiverà l'help contestuale, situazione verificabile dal fatto che anche il puntatore del mouse cambierà.
Dopo aver aggiunto il componente *HelpProvider*, ogni controllo sarà dotato di una nuova proprietà *HelpString on NomeHelpProvider* nella finestra delle proprietà, da impostare con la stringa di *Help* per il controllo stesso.

Componenti provider
Esistono dei componenti forniti dal .NET Framework, che pur non essendo visibili all'utente permettono di dotare i controlli grafici di nuove proprietà, sono gli extender provider e comprendono: *Tooltip*, *ErrorProvider* ed *HelpProvider* e gli sviluppatori possono creare il proprio provider implementando l'interfaccia *IExtenderProvider*.
La classe *HelpProvider* permette di aprire un file di guida esterno, sia HTML sia CHM.
Per l'applicazione si usa un file CHM qualunque, per esempio il file di guida di Blocco note, NOTEPAD.CHM.
Se la proprietà *HelpNamespace* è stata impostata ad un percorso di un file di guida, quest'ultimo sarà visualizzato usando i parametri che devono essere impostati mediante il metodo *SetHelpNavigator* ad uno dei valori dell'enumerazione *HelpNavigator*.
- ✓ *AssociateIndex*: apre la guida alla prima lettera dell'argomento specificato.
- ✓ *Find*: apre la guida alla pagina di ricerca.
- ✓ *Index*: apre la guida alla pagina dell'indice.

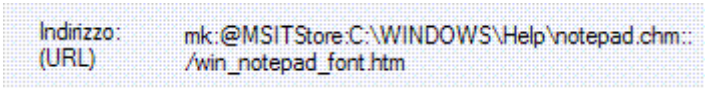
- ✓ *KeywordIndex*: apre la guida all'argomento dell'indice specificato se esiste, oppure a quello più vicino.
- ✓ *TableOfContents*: apre il sommario la guida.
- ✓ *Topic*: apre lo specifico argomento se esiste.
- ✓ *TopicId*: apre l'argomento identificato dal numero specificato.

Per esempio, alla pressione del tasto F1 con il focus su un dato controllo, aprire il file NOTEPAD.CHM, posizionandosi alla prima parola che inizia con una determinata lettera. È necessario in questo caso impostare il valore *HelpNavigator.Index* e quindi la lettera che interessa.

```
HelpProvider chmHelp = new HelpProvider();
chmHelp.HelpNamespace = "notepad.chm";
chmHelp.SetHelpNavigator(btHelpIndex, HelpNavigator.Index);
chmHelp.SetHelpKeyword(btHelpIndex, "a");
```

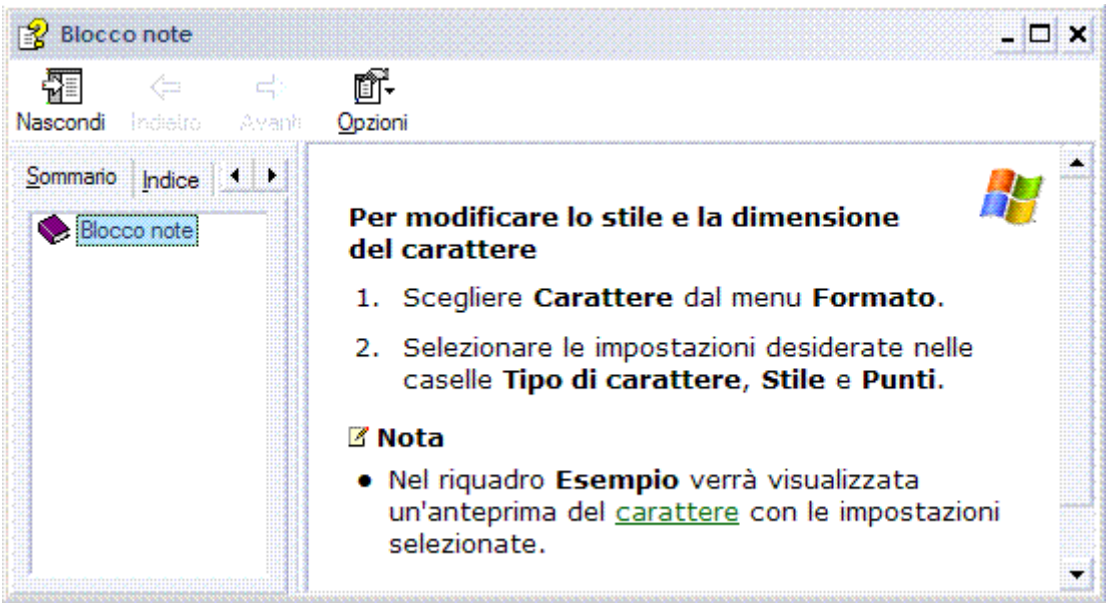


Se si vuol aprire la guida posizionandosi ad un determinato sotto argomento, bisogna fornire il percorso dell'argomento nel file, si può ottenere cliccando con il tasto destro, nella pagina dell'argomento stesso e poi cliccando su proprietà.



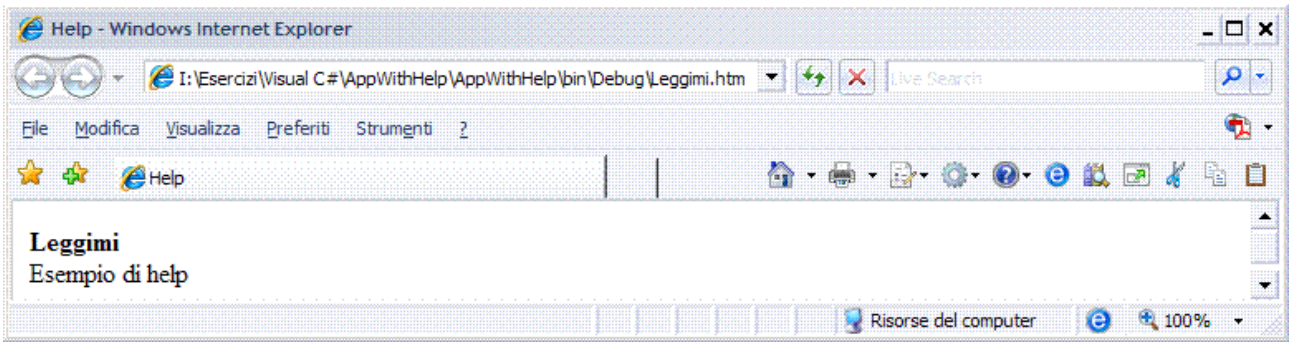
Quindi bisogna utilizzare come *HelpNavigator* il parametro *Topic*.

```
chmHelp.SetHelpNavigator(btHelpIndex, HelpNavigator.Topic);
chmHelp.SetHelpKeyword(btHelpIndex, "notepad.chm::/win_notepad_font.htm");
```



Per impostare il file HTM da aprire, basta impostare la proprietà *HelpNamespace* utilizzando il percorso del file e invocare il metodo *SetShowHelp* per indicare che l'help è attivo su un determinato controllo.

```
HelpProvider htmlHelpProvider = new HelpProvider();
htmlHelpProvider.SetShowHelp(btReadMe, true);
htmlHelpProvider.HelpNamespace = "Leggimi.htm";
```

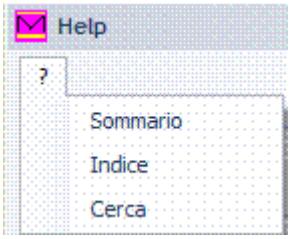


Se è impostata la proprietà *HelpNamespace* allora la *HelpString* sarà ignorata e non sarà più visualizzata come tooltip premendo il tasto F1 o utilizzando l'HelpButton del form, ma al suo posto sarà aperto il file di guida stesso.
La *HelpString* sarà comunque accessibile per ogni evenienza invocando il metodo *GetHelpString*.

Menu
La classe *HelpProvider* utilizza internamente i metodi della classe *Help*, che fornisce allo sviluppatore solo metodi statici pubblici, quindi è utilizzabile direttamente nel codice.
Utilizzarla, per esempio, per creare il classico menu di help, *?*, con le voci *Sommario*, Indice e *Cerca* e aprire il file guida posizionandolo proprio ad una delle tre voci.
Invocare il metodo *ShowHelp*, con l'adeguato *HelpNavigator* ed eventualmente con una chiave di ricerca.

```
private void sommarioToolStripMenuItem_Click(object sender, EventArgs e)
    { Help.ShowHelp(this, "notepad.chm", HelpNavigator.TableOfContents); }
private void indiceToolStripMenuItem_Click(object sender, EventArgs e)
    { Help.ShowHelp(this, "notepad.chm", HelpNavigator.Index); }
```

```
private void cercaToolStripMenuItem_Click(object sender, EventArgs e)
      { Help.ShowHelp(this, "notepad.chm", HelpNavigator.Find,"keyword"); }
```



OFFUSCATORE

INTRODUZIONE

Il compilatore è lo strumento che permette di passare da un linguaggio di alto livello, per esempio C, C++, C#, VB.NET a un codice binario, per esempio codice macchina, bytecode di Java, IL di .NET.
La decompilazione è il processo inverso alla compilazione, si passa dal codice binario a un linguaggio di più alto livello.
Decompilare il codice macchina in linguaggi come C o C++ è difficile.

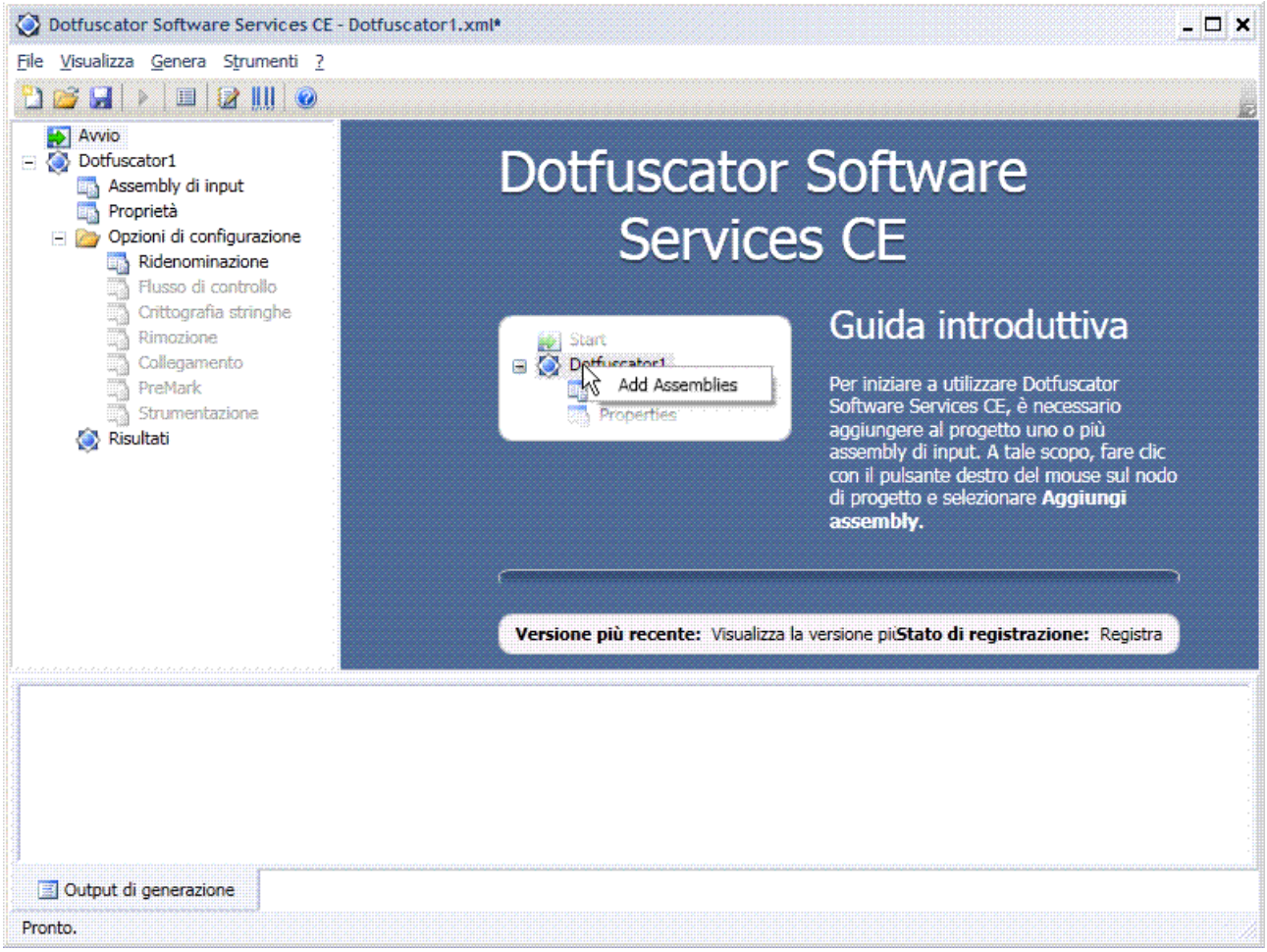
Disassemblatori

Se i decompilatori per codice macchina sono poco funzionali, si fa ricorso a un disassemblatore, che è uno strumento più semplice di un decompilatore, permette di tradurre il codice macchina in un sorgente assembly.
L'assembly, però, cambia da assemblatore a assemblatore e da versione a versione dello stesso assemblatore: è difficile decompilare anche partendo dall'assembly

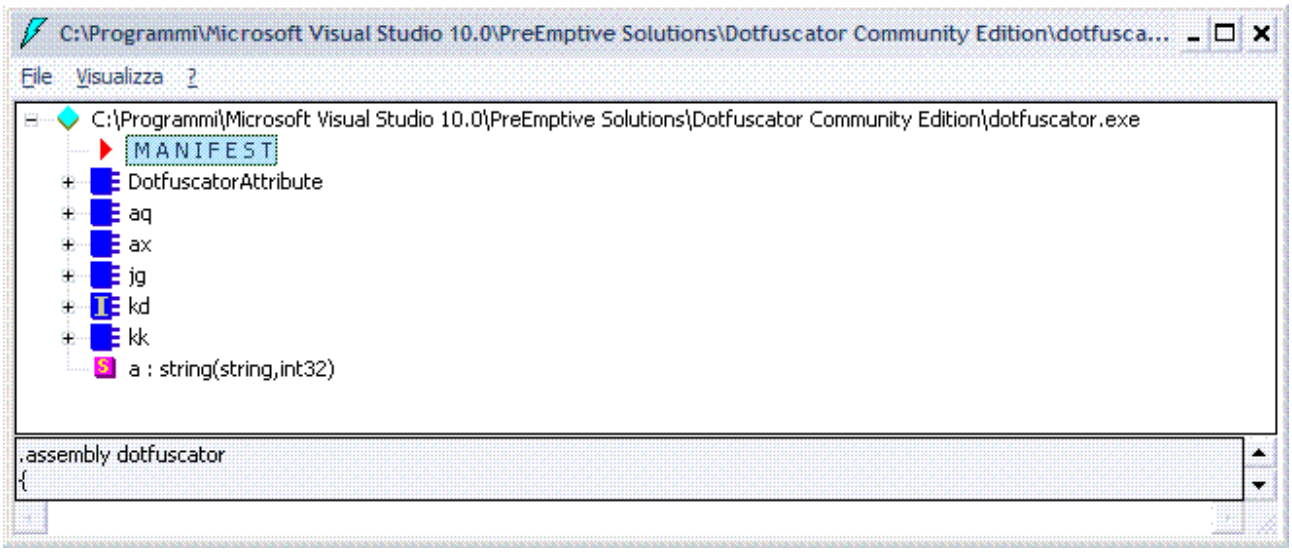
La decompilazione dei codici binari intermedi, per esempio il bytecode di Java o IL di .NET, invece, è semplice e permette quasi sempre di ottenere buoni risultati.
Solo i commenti scompaiono, poiché il compilatore non li include nel codice binario intermedio, ma per tutto il resto si può dire che la leggibilità del codice è rimasta inalterata.
Allora se le applicazioni sono vulnerabili alla decompilazione, si deve proteggere il codice servendosi di un offuscatore.
Il mercato propone diverse tipologie di offuscatori di codice, per esempio per il bytecode di Java, per codice PHP, per .NET.
Gli offuscatori agiscono sul codice binario intermedio e ne alterano il contenuto in modo da renderne vana la decompilazione, per esempio cambiano i nomi delle variabili e dei metodi non pubblici, riscrivono parte del codice in maniera criptica e compiono altre operazioni di mascheramento degli algoritmi.
Gli offuscatori, inoltre, garantiscono anche un vantaggio collaterale: rendono il codice binario intermedio più snello ed efficiente, perché lo alterano servendosi anche di riduzioni e ottimizzazioni.

DOTFUSCATOR SOFTWARE SERVICES

Gli eseguibili .NET forniscono tutte le informazioni sulle classi, i *namespace*, inoltre l'IL è compilato in maniera molto regolare, rispetto all'assembly.
Permette l'offuscamento e la compattazione .NET e consente di proteggere l’applicazione dalla decompilazione nonché di renderla più snella e efficiente.
Inoltre, offre la possibilità d’inserire funzionalità aggiuntive pregenerate che consentono la tracciatura dell'utilizzo, il rilevamento delle alterazioni e la scadenza delle applicazioni .NET.
In Visual Studio fare clic su ***Strumenti/Dotfuscator Software Services***.
Si apre la seguente schermata di avvio, l’applicazione è composta da tre pannelli: la struttura ad albero di spostamento, l'area di lavoro e l'output di generazione.



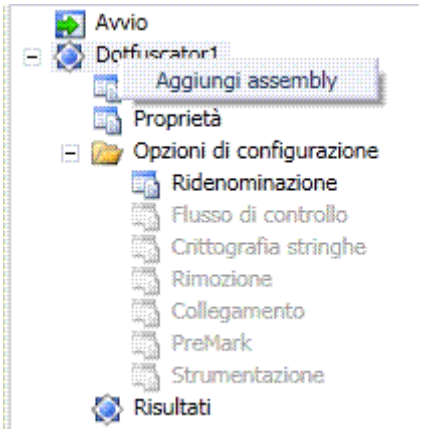
Lo stesso DOTFUSCATOR è offuscato, visto che è scritto in .NET, quindi per avere un'idea della protezione offerta è sufficiente analizzare il suo stesso eseguibile.
Ad ogni modo se si apre con ILDASM o con un decompiler il DOTFUSCATOR si vede che tutti i *namespace*, tutte le classi e tutte le funzioni sono state rinominate.
Si trovano nomi tipo *a, b, c, d*; oppure *aq, ax.*



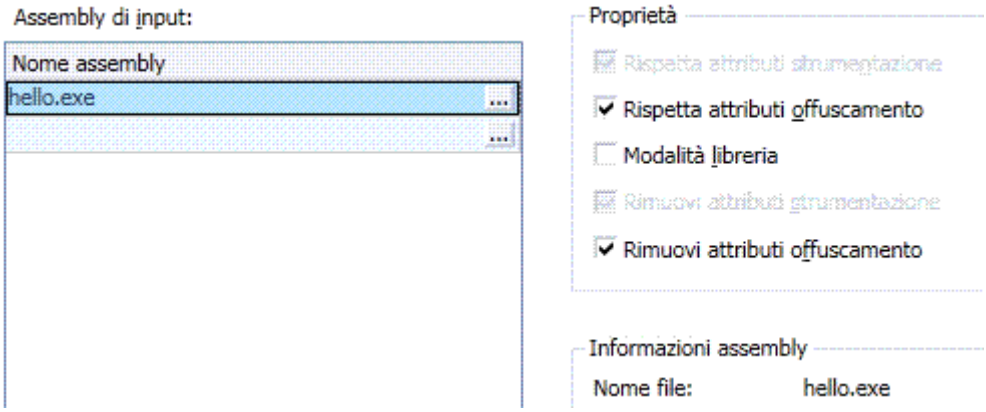
Rinominare metodi e classi non basta a fornire una buona protezione, certo anche quello va fatto, ma è il minimo, allora il problema fondamentale è che se si vuole proteggere un assembly in qualsiasi maniera possibile le opzioni sono tante e si può anche fare un bel lavoro, soprattutto con una conoscenza approfondita della struttura degli assembly, ma le opzioni diminuiscono esponenzialmente se si vuole restare nella logica del .NET.

In altre parole, si può fare un packer e crittare il codice IL ed eseguire assembler in una DLL o nell'assembly stesso che decritta il codice.
Il problema è che così facendo il .NET perde gran parte della sua ragione d'essere: la portabilità e la flessibilità.

Esempio.
Per iniziare aggiungere uno o più assembly d'input al progetto, è necessario che nell'area di lavoro sia aperta la schermata ***Assembly di input***.
È possibile attivare questa schermata facendo clic con il pulsante destro del mouse sul nodo del progetto nella struttura ad albero di spostamento e selezionando ***Aggiungi assembly*** oppure selezionando l'opzione ***Assembly di input*** nella struttura ad albero di spostamento.

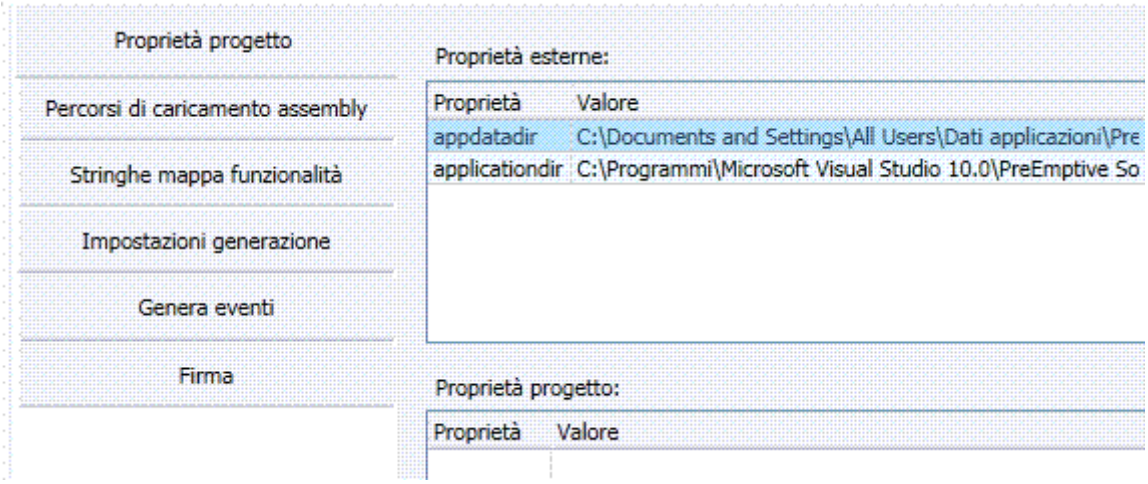


Aggiungere HELLO.EXE

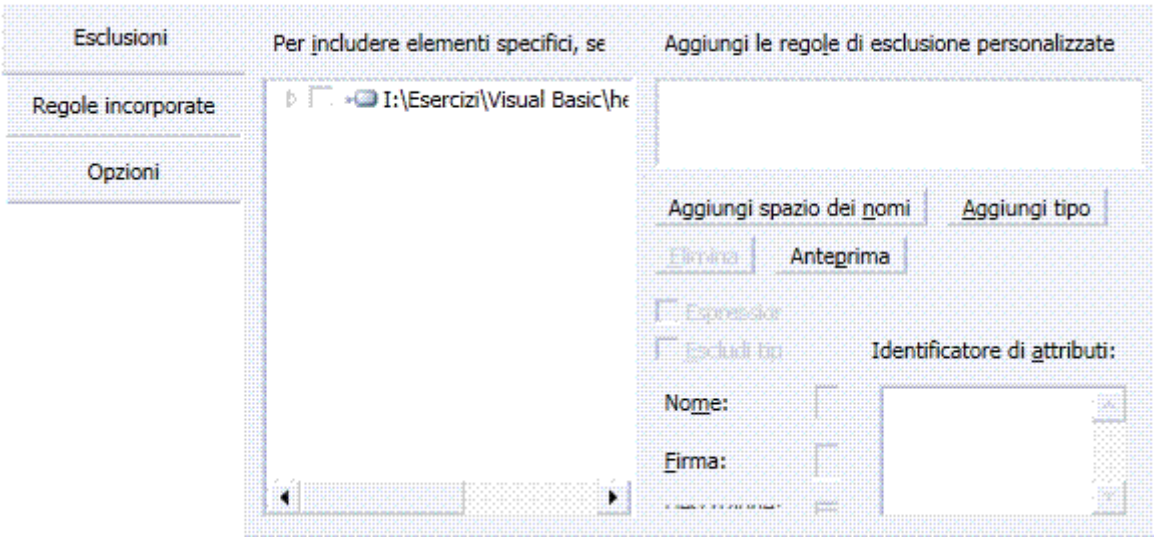


Rispetta attributi di strumentazione: sono attributi personalizzati che possono essere applicati nel sorgente per tenere traccia della stabilità, delle funzionalità e dell'utilizzo dell'applicazione nonché per aggiungere funzionalità di periodo di validità.
Rispetta attributi di offuscamento: sono attributi personalizzati che possono essere applicati nel sorgente per dichiarare esplicitamente l'inclusione o l'esclusione di tipi, metodi, enumerazioni, interfacce o membri dai vari tipi di offuscamento, utilizzare l'attributo *using System.Reflection.ObfuscationAttribute*
Modalità libreria: l'input selezionato costituisce una libreria, ai fini di offuscamento, una libreria è definita come un assembly cui fanno riferimento altri componenti non specificati come uno degli input in questa esecuzione, quando è offuscato un assembly i gli elementi visibili pubblicamente non sono rinominati, pertanto l'API pubblica resta accessibile ai chiamanti esterni.
Rimuovi attributi di offuscamento: consente la rimozione di tutti gli attributi di offuscamento una volta completata l'elaborazione, affinché gli assembly di output non

contengano indizi su come sono stati offuscati.
La voce ***Proprietà*** della struttura ad albero di navigazione consente di visualizzare la configurazione nell'area di lavoro, contiene sei schede.

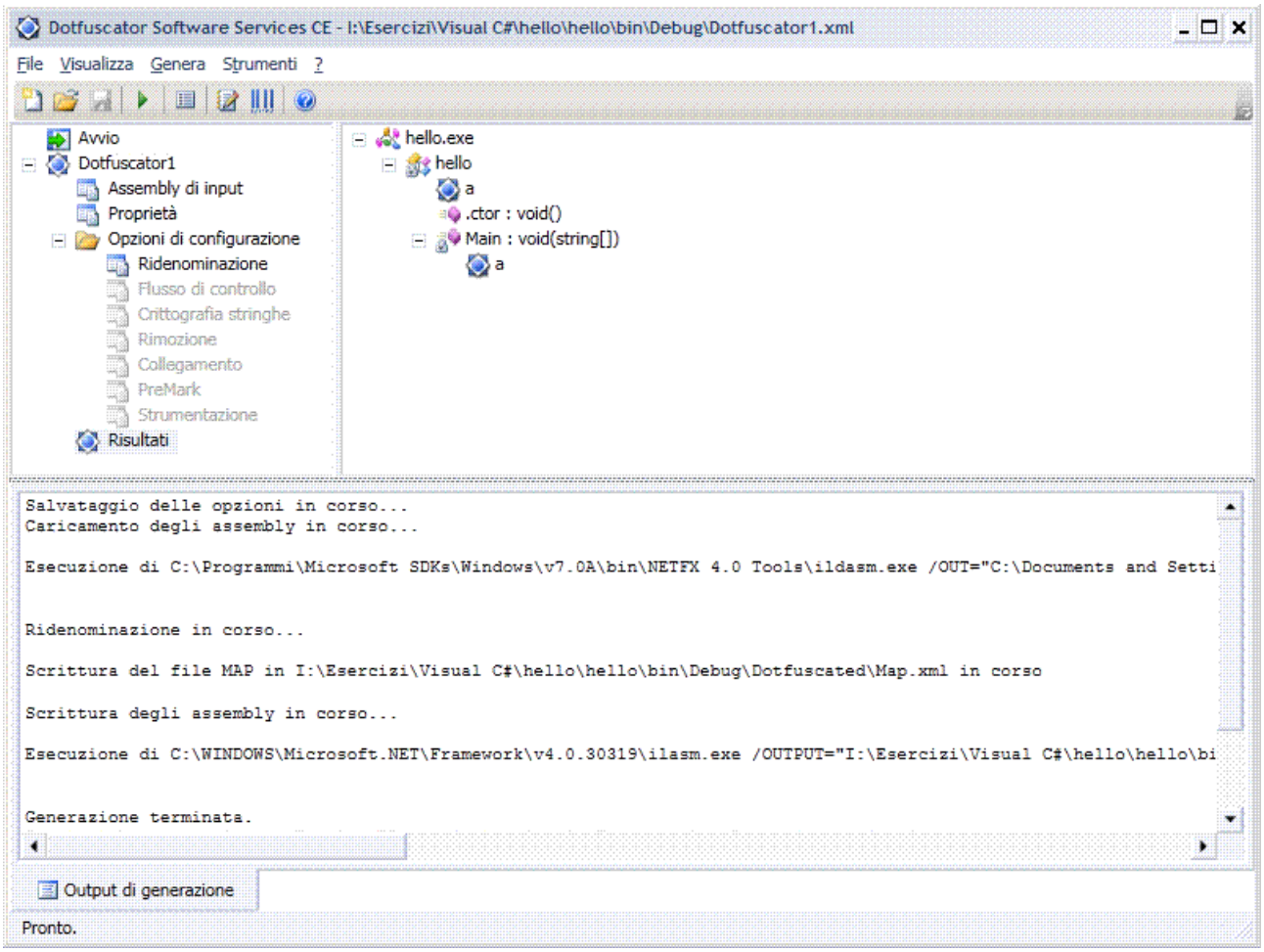


Nella sezione ***Opzioni di configurazione*** è possibile impostare le opzioni di ridenominazione e le esclusioni, scegliere le regole di ridenominazione incorporate e configurare le impostazioni di strumentazione.
Nell'editor di ***Ridenominazione*** sono visualizzate tre schede.
La scheda ***Esclusioni***, utilizzata per impostare graficamente le regole di esclusione.
La scheda ***Regole incorporate*** in cui sono visualizzate regole di esclusione della ridenominazione preconfigurate che si applicano a tecnologie o tipi di applicazione specifici.
La scheda ***Opzioni***, utilizzata per configurare altre opzioni relative alla ridenominazione.



La ***Strumentazione*** consente di aggiungere alle applicazioni funzionalità pregenerate di tracciatura dell'utilizzo dell'applicazione, rilevamento di alterazione binaria e scadenza dell'applicazione senza richiedere codice aggiuntivo, è possibile definire i punti d'inserimento tramite attributi personalizzati nel sorgente o tramite attributi estesi specificati nell'interfaccia utente.

Fare clic su ***Genera/Genera il progetto (CTRL+B).***
Dopo aver generato il progetto, è possibile visualizzare i risultati nella scheda ***Risultati***.

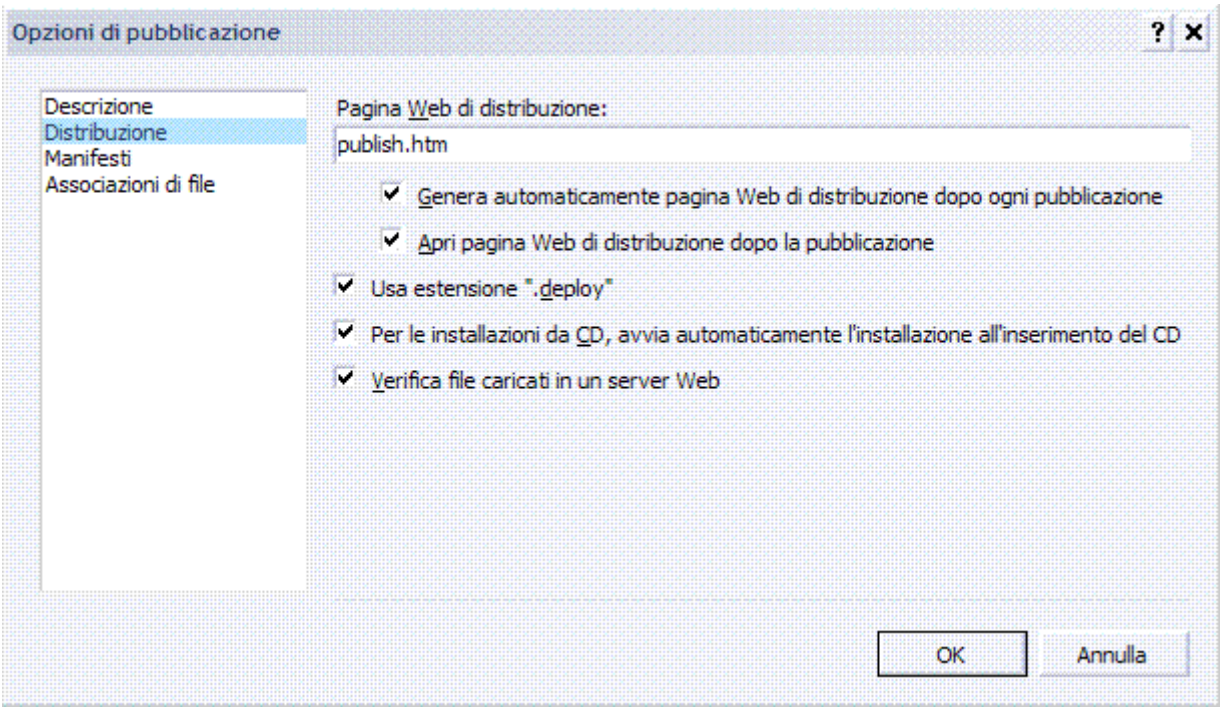


Il sorgente risulta assai meno leggibile dell'originario e quindi, assai più difficile da comprendere.

DISTRIBUZIONE APPLICAZIONI

MODO TRADIZIONALE

Fare clic sul menu ***Progetto/Proprietà di mioprogetto…***
Fare clic sulla scheda ***Pubblica*** per aprire la pagina, quindi scegliere il pulsante ***Opzioni...***



Selezionare la casella di controllo ***Per le installazioni da CD, avvia automaticamente installazione all'inserimento del CD***.
Al momento della pubblicazione dell'applicazione sarà copiato un file AUTORUN.INF nel percorso di pubblicazione.
Scegliere il pulsante ***Pubblicazione guidata...*** e seguire il wizard come prima.

File AUTORUN.INF
Contiene le istruzioni che il sistema deve eseguire all'inserimento o al collegamento della periferica.
È un file di testo con estensione posto nella cartella principale dell'unità di memoria.

[autorun]
open=nomefile.exe
icon=icona.ico
label=software

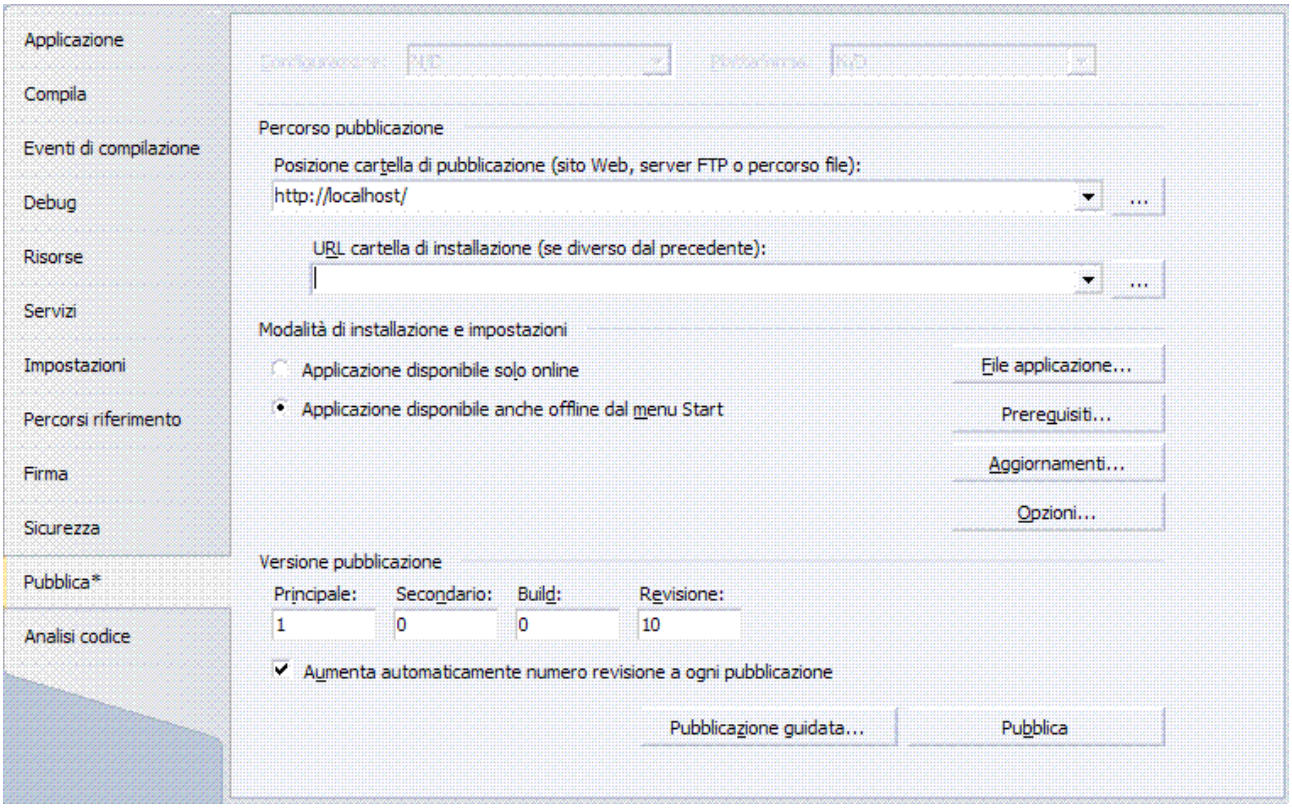
Nell'esempio NOMEFILE.EXE è il nome del file eseguibile e ICONA.ICO è il file dell'icona che apparirà al posto dell'icona di default in Esplora risorse, se non è specificata l'icona sarà utilizzata quella di sistema dell'unità.
Per aprire una pagina web è possibile utilizzare la funzione *start*; nel caso però non sia installato un browser sul PC dell'utente sarà visualizzato un errore.

[autorun]
open=start index.html

CLICKONCE

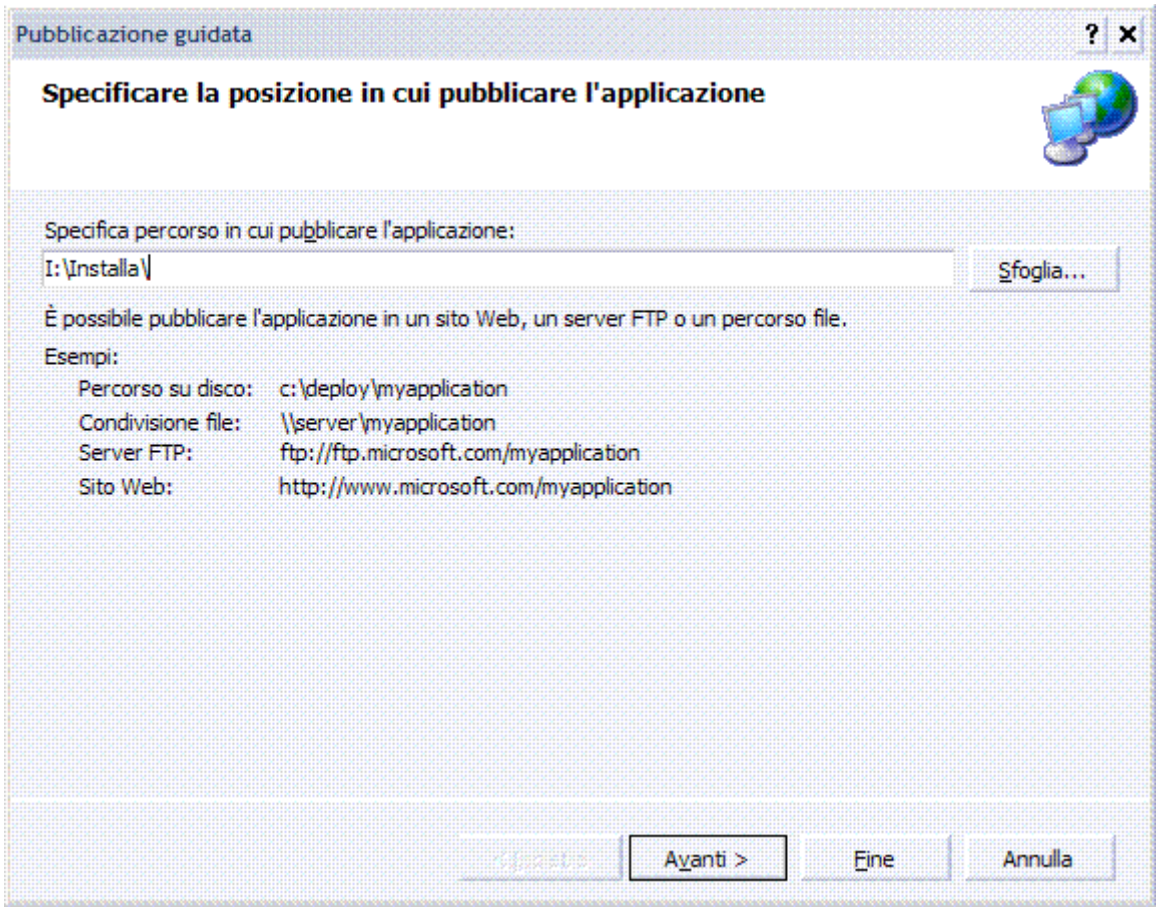
Nonostante la distribuzione di un'applicazione sia il passo finale del ciclo di produzione, non deve essere considerata ultimo anche in ordine d'importanza.
Stabilire a priori la modalità di distribuzione e installazione del software è fondamentale per una buona riuscita del prodotto.
Spesso le applicazioni devono essere immediatamente fruibili da parte dell'utente finale, facilità d'installazione e manutenzione costituiscono requisiti importantissimi.
Questo tipo di problematica ha favorito lo sviluppo di applicazioni web che per loro natura non necessitano d'installazione e sono aggiornate semplicemente lato server.
Da qui la necessità di un modello capace di rendere semplice e sicura la distribuzione delle applicazioni client.
ClickOnce consente di ottenere esattamente questo: una distribuzione semplice, efficace, sicura delle applicazioni Windows, non invasiva sul client di destinazione.

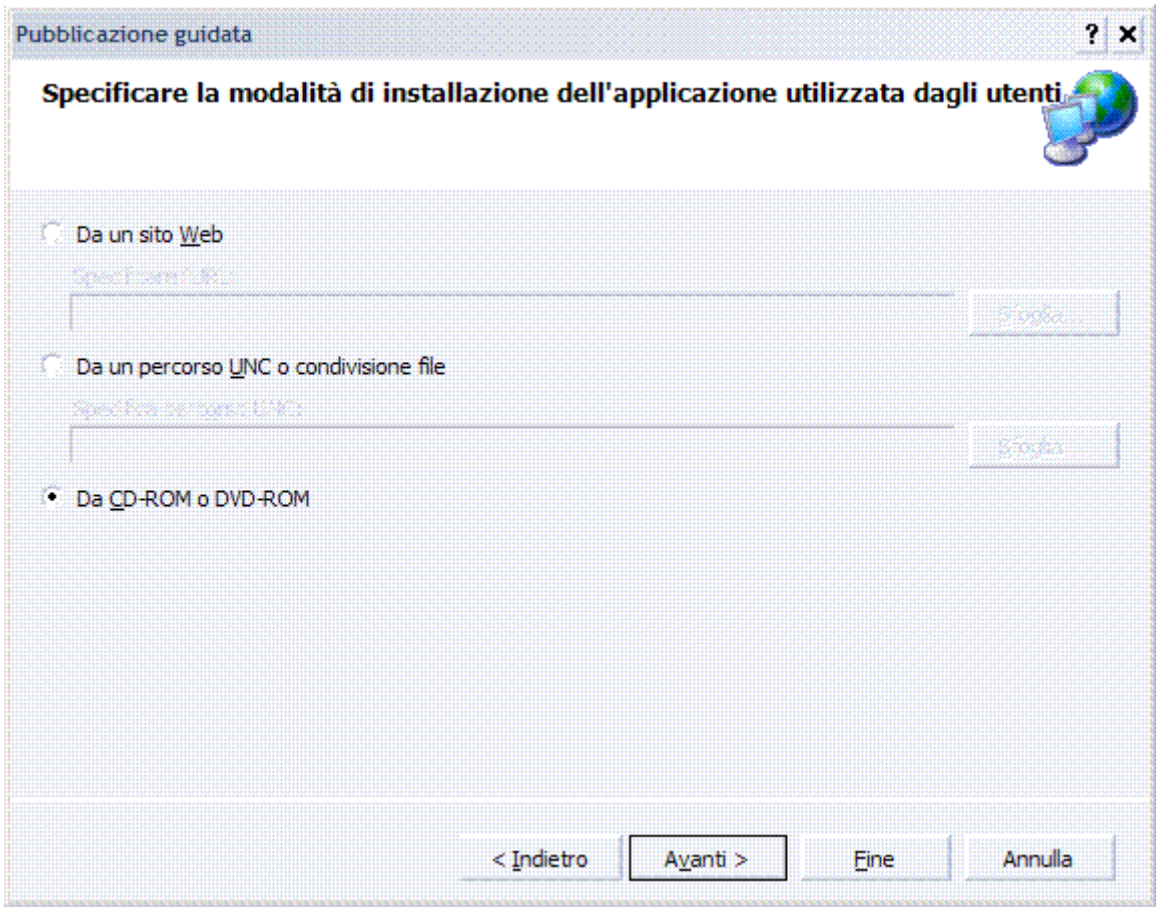
Esempio, sviluppare un'applicazione per un ipotetico cliente, la cui necessità sia quella d'implementare un servizio di registrazione anagrafiche distribuito, rilasciato a tutte le sue agenzie geograficamente distanti tra loro.
Ogni agenzia utilizza il software per registrare quotidianamente i dati dei nuovi clienti acquisiti, tutte le agenzie devono possedere in ogni determinato istante la stessa identica versione dell'applicazione.
Terminata la fase di progettazione, sviluppo e test dell'applicazione, si è pronti a procedere con la distribuzione su tutti i client.
Fare clic sul menu ***Progetto/Proprietà di …***
Fare clic sulla scheda ***Pubblica*** per aprire la pagina, quindi scegliere il pulsante ***Pubblicazione guidata...***

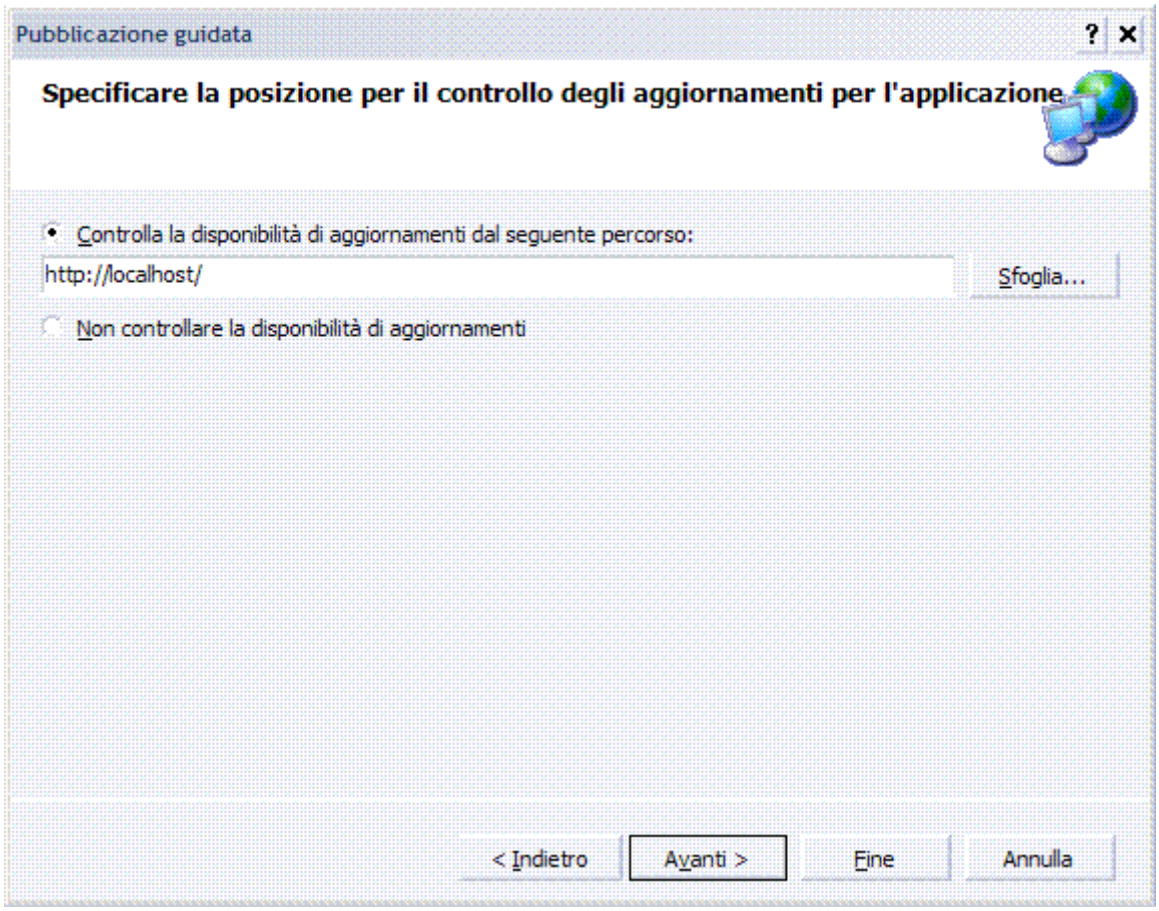


Ad ogni pubblicazione, ClickOnce incrementa il numero di build per distinguere le versioni e consentire al run-time di verificare la presenza di aggiornamenti.
È possibile rimuovere questa caratteristica e scegliere d'impostare manualmente il numero di build deselezionando il controllo check box.

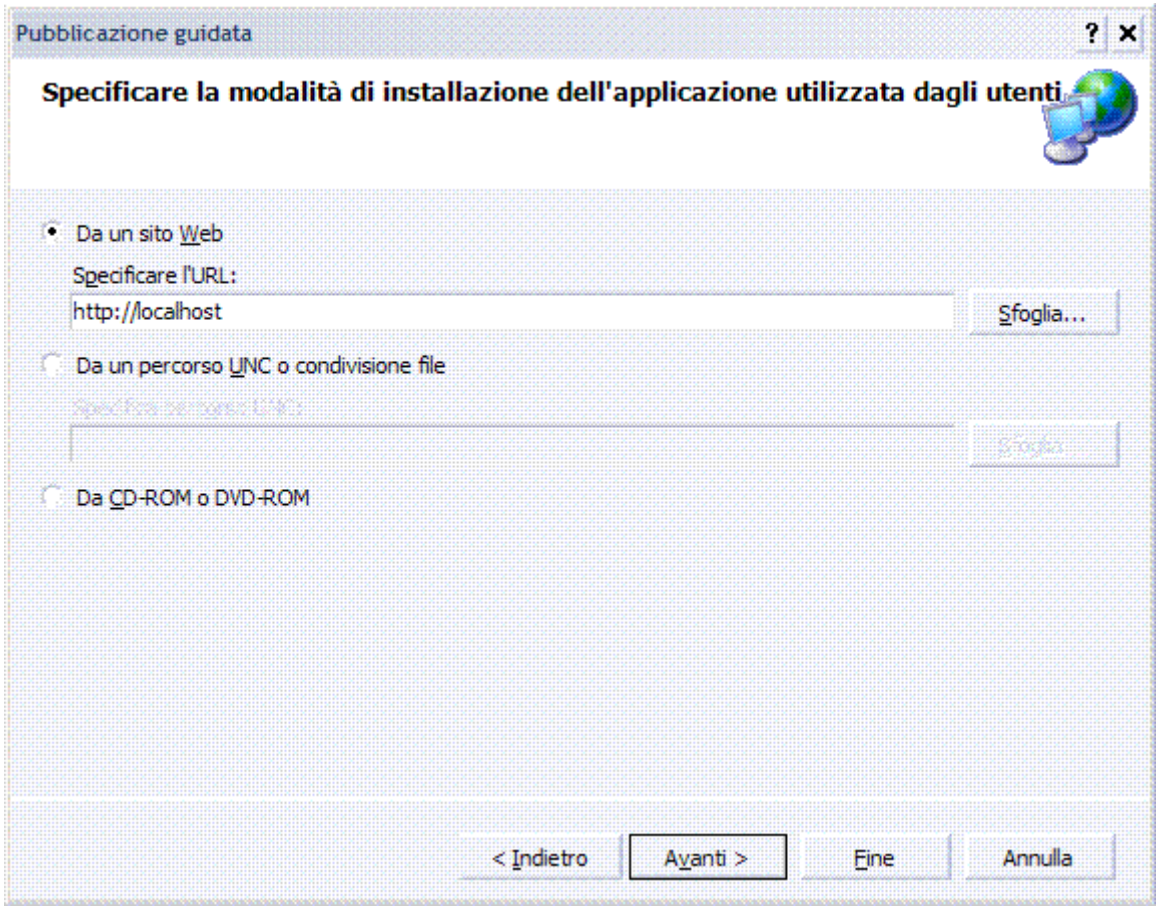
Visual Studio presenta un wizard che permette di selezionare la modalità di distribuzione.



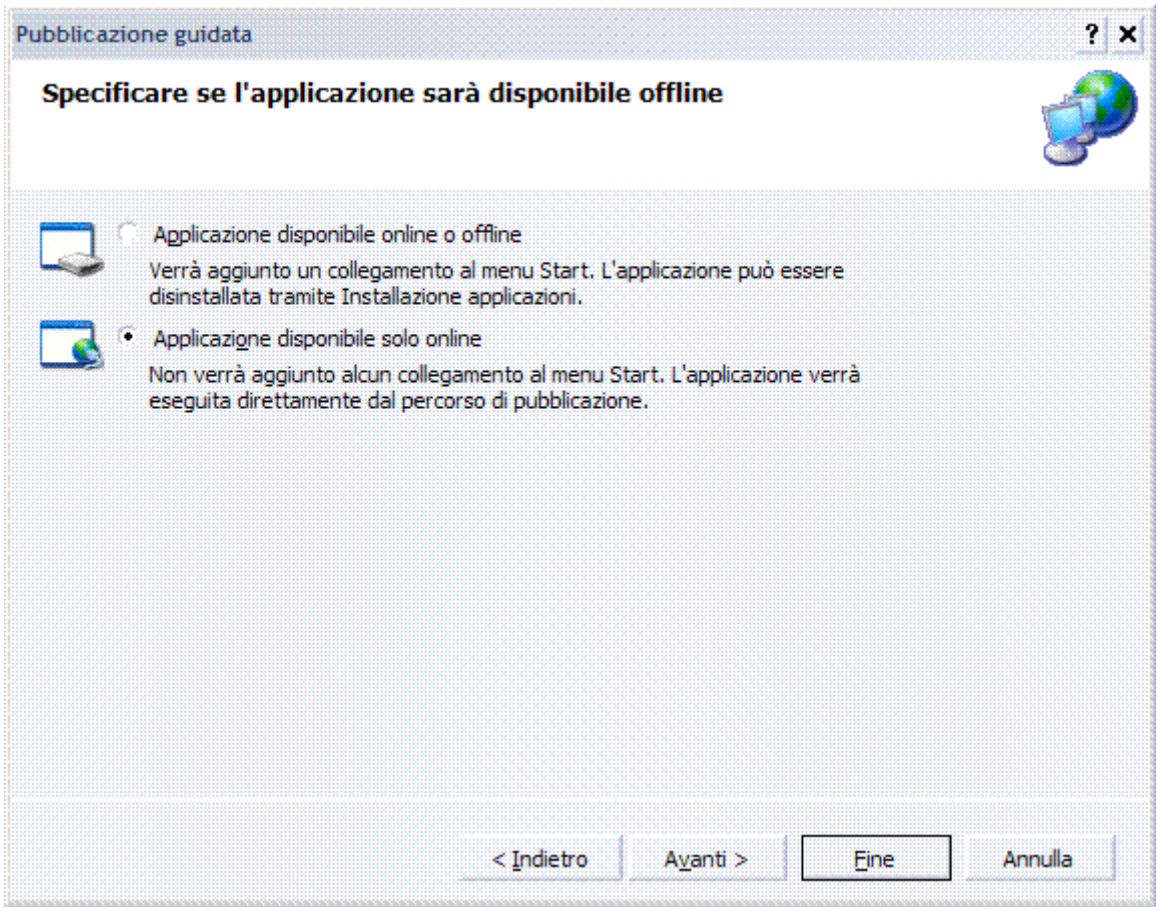


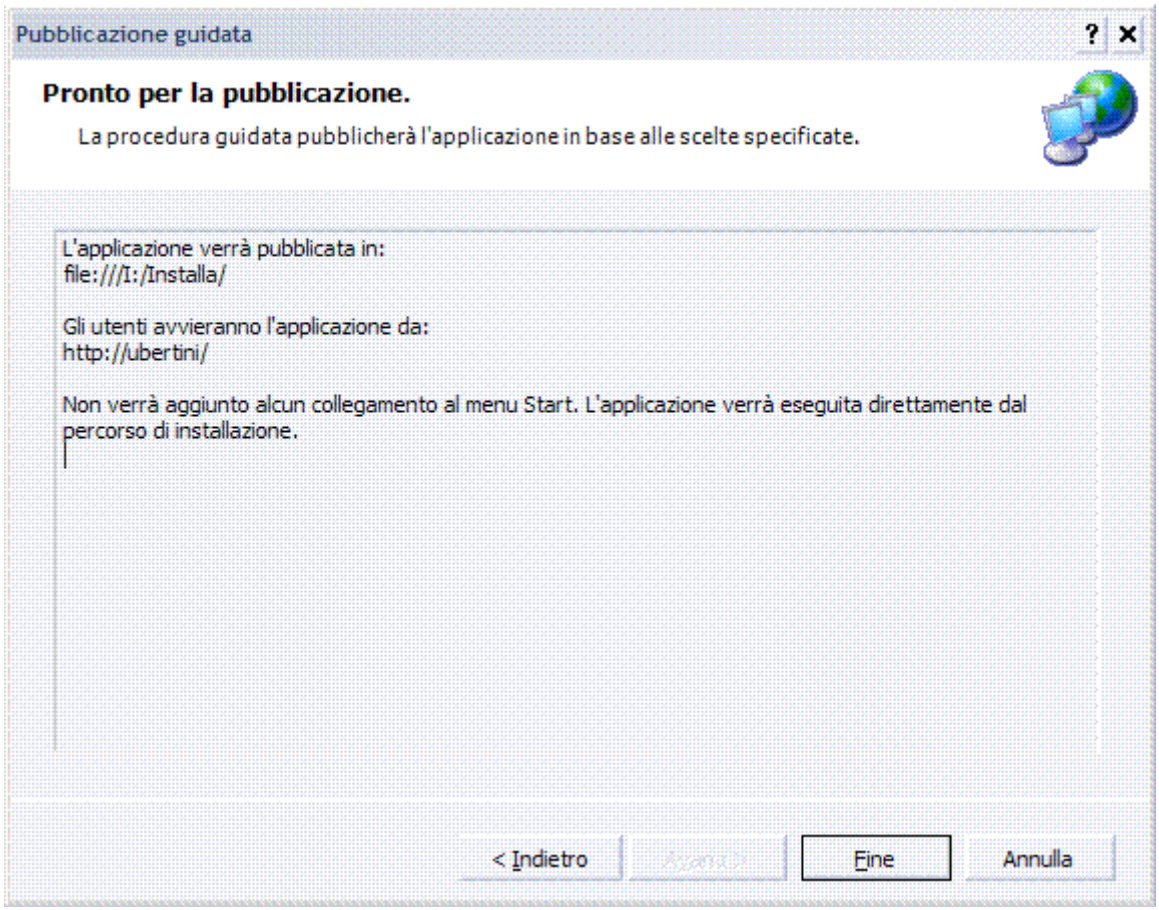


Data la differente posizione geografica dei client, si sceglie di distribuire l'applicazione attraverso un server web.



L’applicazione è disponibile solo online.





Visual Studio compila il progetto, crea i file di pubblicazione ed infine copia i file in una

cartella all'interno della directory specificata.
È creata una cartella, il cui nome è composto dal nome dell'applicazione più il numero di versione pubblicato, contenente tutti i file associati al deploy corrente.
Al termine della procedura di pubblicazione, Visual Studio mostra la pagina web generata dove sono riportati il nome dell'applicazione, l'ultima versione pubblicata e, se disponibile, l'eventuale publisher.
Utilizzando l'URL di pubblicazione i client possono eseguire l'applicazione cliccando sul tasto Run.
A questo punto il run-time ClickOnce esegue i controlli di sicurezza per verificare se l'applicazione è firmata e se la firma è valida e poi, tramite CAS, se i permessi richiesti sono compatibili con quelli impostati per l'esecuzione o l'installazione dall'intranet, che è la zona di provenienza/pubblicazione che si sta utilizzando.
Qualora almeno uno di questi controlli fallisse, sarà mostrato un Security Warning, una finestra che chiede all'utente una conferma sull'azione da eseguire, in altre parole se l'applicazione deve essere eseguita o meno.
La Security Warning può essere evitata firmando l'applicazione ClickOnce e impostando i permessi per una corretta esecuzione.

La distribuzione e l'esecuzione delle applicazioni ClickOnce sono basate su due fondamentali file manifest.

.DEPLOY
Descrive le informazioni di distribuzione su cui spiccano la descrizione della versione attualmente disponibile, il riferimento all'application manifest, le modalità d'installazione e di update dell'applicazione e la firma del manifest di deploy.

.APPLICATION
Descrive nel maggior dettaglio l'applicazione distribuita con ClickOnce, riportando le informazioni d'identificazione sull'assembly principale distribuito e su tutte le sue relative dipendenze.
L'esecuzione dell'applicazione è associata al file .APPLICATION il cui percorso è passato come parametro alla libreria DFSHIM.DLL eseguita utilizzando il comando RUNDLL.EXE.

Esecuzione dell'applicazione in modalità offline
Se il cliente desidera rendere l'applicazione utilizzabile anche da quei client che non sono sempre connessi alla rete basta riaprire la soluzione in Visual Studio e rieseguire la distribuzione, nel wizard si lascia invariato il percorso di pubblicazione e si modifica, invece, l'opzione di disponibilità dell'applicazione impostando la modalità online e offline e si termina il wizard.
Visual Studio riesegue la compilazione del progetto, incrementando il numero di build e riesegue la pubblicazione del progetto nella cartella indicata.
La pagina di publish presenta ancora una volta il nome dell'applicazione, il numero di versione incrementato di build e non più il tasto Run, ma il tasto Install.
Cliccando su install ClickOnce riesegue i controlli di sicurezza, visualizzando la Security Warning qualora non validi, provvede ad eseguire un'installazione dell'applicazione sul client e infine esegue l'applicazione appena installata.
La differenza con la precedente modalità di distribuzione è data dalla presenza di un collegamento nel menu ***Start***.
L'installazione sul client non è reale o comunque non simile a quanto avviene con il modello di setup classico.
L'applicazione è in pratica installata localmente nella cache di ClickOnce ed eseguita da quella posizione dopo aver verificato che nella cartella di pubblicazione, se disponibile, non è presente una nuova versione, nel qual caso è chiesto se effettuare l'aggiornamento.

Update dell'applicazione
Se il cliente chiede di sostituire nel form d'inserimento anagrafiche, il campo di testo per l'inserimento delle regioni con una combo box che ne consente la selezione, dopo aver aggiornato l'applicazione ripetere il processo di pubblicazione.
Al termine del processo di generazione della nuova build e della successiva pubblicazione da parte del wizard, Visual Studio mostra la pagina di pubblicazione.
Per verificare che l'update dell'applicazione funziona correttamente, provare ad eseguire l'applicazione dal menu ***Start***, proprio come farebbe un ipotetico utente.
ClickOnce accerta la presenza di una nuova versione e chiede all'utente se desidera effettuare l'aggiornamento.
Internamente è generato un hash per ogni build dell'applicazione.
Se la cartella di pubblicazione è disponibile, ClickOnce verifica l'hash installato con l'hash della versione più recente disponibile.
Se quest'ultimo è maggiore a quello installato, ClickOnce avvisa l'utente della disponibilità di una nuova versione e chiede se eseguire l'aggiornamento, altrimenti procede con la normale esecuzione dell'applicazione.
Al termine dell'aggiornamento, ClickOnce esegue la nuova versione dell'applicazione installata.
Ideale per aggiornamenti poco frequenti.

Aggiornamento dell'applicazione programmaticamente
ClickOnce usa un set di API disponibili attraverso il *namespace System.Deployment*, che consente di verificare la disponibilità di nuove versioni ed eventualmente di procedere al download.
Preparare un box dove un alert avvisa della presenza di aggiornamenti e un *Button* che consente di aggiornare l'applicazione.
Aggiungere al form un controllo *GroupBox* e inserire al suo interno un controllo *Label* e un *Button*.
Aggiungere poi al form un controllo *Timer* che ad intervalli regolari verifica la presenza di aggiornamenti.
Impostare il *Timer* su un intervallo di 30 secondi modificando la proprietà *Interval* su 30000, in millisecondi.
Selezionare il metodo da richiamare allo scadere dell'intervallo impostando il valore dell'evento *Tick* su *CheckUpdate* e premere invio sulla propertybox per generare il relativo codice.
Nel codice del form importare il seguente codice.

```
using System.Deployment.Application;
```

Nel metodo *CheckUpdate* inserire il seguente codice.

```
ApplicationDeployment deploy =ApplicationDeployment.CurrentDeployment;
if (deploy.CheckForUpdate())
{this.lblUpdate.Text = "È disponibile una nuova versione.";
 this.btnUpdate.Enabled = true;
}
else
{this.lblUpdate.Text = "Nessun aggiornamento disponibile.";
 this.btnUpdate.Enabled = false;
}
```

ApplicationDeployment è la classe più importante del *namespace* in quanto, oltre a gestire gli aggiornamenti, consente di operare sull'istanza corrente, recuperare le informazioni di deploy, verificare se l'applicazione è eseguita per la prima volta, fino ad operazioni più

complesse come il download di assembly o gruppi di assembly non ancora installati.
La proprietà statica *CurrentDeployment* consente di recuperare l'istanza corrente e di verificare la presenza di nuove versioni utilizzando il metodo *CheckForUpdate().*
Se sono presenti nuove versioni nella cartella di pubblicazione, è abilitato il pulsante *btnUpdate* che permette di eseguire il codice per l'aggiornamento.
Nel gestore dell'evento clic del pulsante *btnUpdate* inserire il seguente codice.

```
ApplicationDeployment deploy =ApplicationDeployment.CurrentDeployment;
deploy.UpdateCompleted += new
AsyncCompletedEventHandler(deploy_UpdateCompleted);
deploy.UpdateAsync();
```

Al clic sul pulsante si recupera l'istanza corrente, impostare il gestore per l'evento *UpdateCompleted* e avviare l'aggiornamento in modalità asincrona richiamando il metodo *UpdateAsync*.
È anche possibile effettuare l'aggiornamento in modalità sincrona utilizzando il metodo *Update* che ritorna *true* se il download e l'installazione è andata a buon fine e *false* se l'operazione non è riuscita.
Al termine dell'aggiornamento, si può continuare a lavorare con la versione corrente e ottenere la nuova versione solo alla successiva esecuzione, oppure, sempre attraverso il codice, chiedere all'utente di riavviare l'applicazione per completare l'aggiornamento.
Per ottenere questo risultato inserire il seguente codice nel gestore dell'evento *UpdateCompleted*.

```
void deploy_UpdateCompleted(object sender,AsyncCompletedEventArgs e)
{DialogResult result =MessageBox.Show("Aggiornamento completato! Riavviare
l'applicazione?", "",MessageBoxButtons.YesNo,MessageBoxIcon.Question);
if (result == DialogResult.Yes)
{Application.Restart();}
}
```

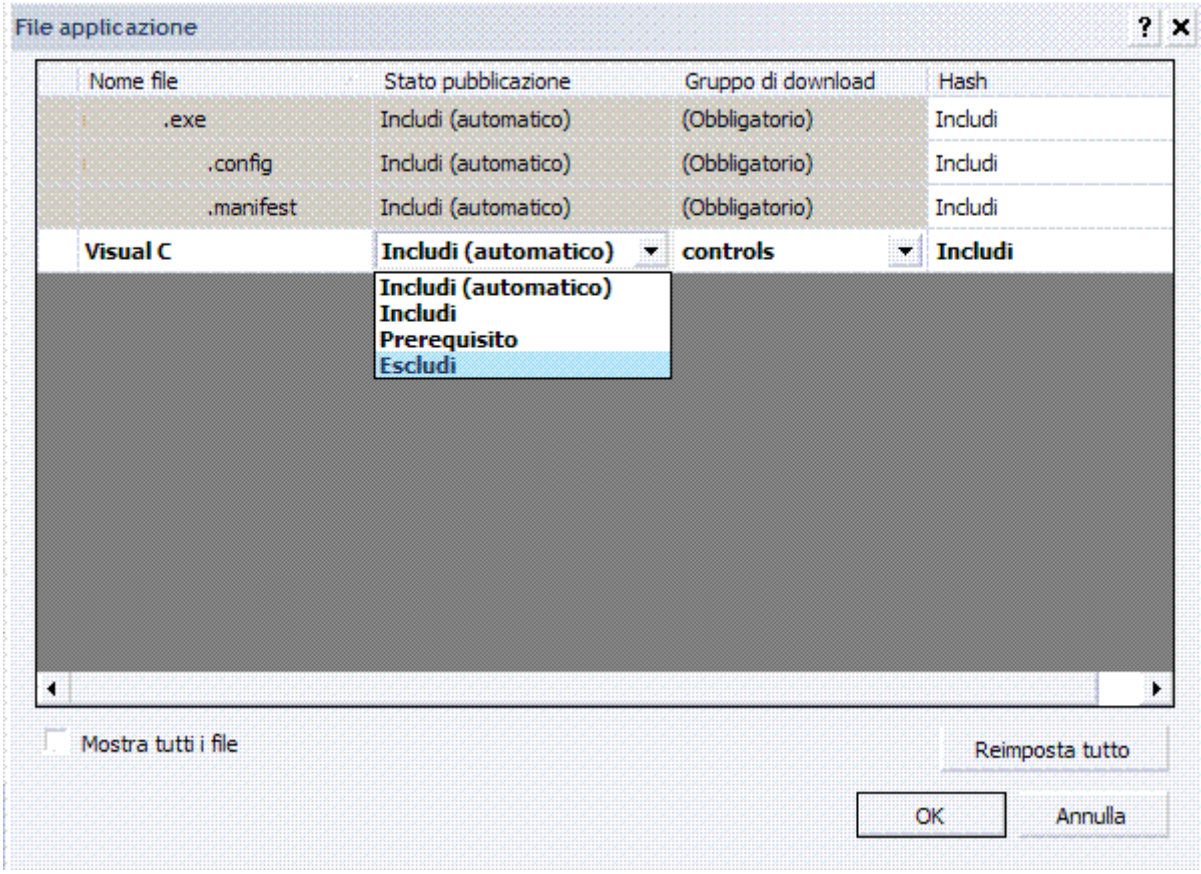
È importante sottolineare, però, che l'applicazione così impostata necessita di richiedere i permessi massimi di esecuzione, *Full Trust*.
È anche possibile recuperare le informazioni sulla nuova versione disponibile utilizzando il metodo seguente *CheckForDetailedUpdate(),* che restituisce un'istanza dell'oggetto *UpdateCheckInfo*.
L'oggetto contiene le seguenti proprietà.

Proprietà	**Descrizione**
AvailableVersion	Proprietà di tipo *Version* contenente la nuova versione disponibile.
IsUpdateRequired	Indica se l'aggiornamento è richiesto per il corretto funzionamento dell'applicazione.
MinimumRequiredVersion	Di tipo *Version* indica la versione minima richiesta per poter effettuare l'aggiornamento
UpdateAvailable	Valore booleano che indica se un aggiornamento è disponibile
UpdateSizeBytes	Restituisce la dimensione dei file da aggiornare.

Download on demand
A volte è necessario minimizzare i file da includere nell'installazione, riducendo quindi il tempo di download e di startup dell'applicazione.
ClickOnce consente d'installare gli assembly o i file anche successivamente, utilizzando le

API incluse in *System.Deployment* e introducendo il concetto di late-download o download a gruppi.
Per esempio, si deve fornire il controllo aggiuntivo *Calendar* utilizzato dai client solo in alcuni casi e contenuto in un assembly esterno all'applicazione.
Basta aggiungere un riferimento alla libreria contenente il controllo e inserire il codice necessario per poterlo richiamare.
Senza pesare sullo startup dell'applicazione è possibile creare gruppi di file e avviare il download dei file non installati solo quando necessario.
Fare clic sul menu ***Progetto/Proprietà di mioprogetto…***
Fare clic sulla scheda ***Pubblica*** per aprire la pagina, quindi scegliere il pulsante ***File applicazione...***
È visualizzata la dialog box contenente l'elenco dei file che saranno distribuiti via ClickOnce, ognuno con lo stato di pubblicazione e il gruppo di download di appartenenza.
Lo stato di pubblicazione può assumere tre valori.

1. ***Includi***, indica che il gruppo è incluso dall'installazione di base.
2. ***Prerequisito***, che imposta il gruppo come un prerequisito fondamentale per il funzionamento dell'applicazione.
3. ***Escludi***, che esclude il gruppo dall'installazione di base.



Utilizzando questa sequenza si crea il gruppo *controls* e lo si assegna alla libreria di controlli precedentemente aggiunta alla soluzione.
Prima di richiamare ed eseguire il codice contenuto in un controllo appartenente al gruppo *controls*, si deve verificare se gli assembly e i file relativi sono stati già scaricati ed installati sul client, in caso contrario si procede con il download.
Il codice da implementare è il seguente che verifica se il gruppo *controls* è stato già scaricato dalla cartella di pubblicazione ed eventualmente procede al download della libreria di controlli.

ApplicationDeployment deploy =ApplicationDeployment.CurrentDeployment;

```
if (!deploy.IsFileGroupDownloaded("controls"))
{deploy.DownloadFileGroup("controls");}
// qui il codice da eseguire
```

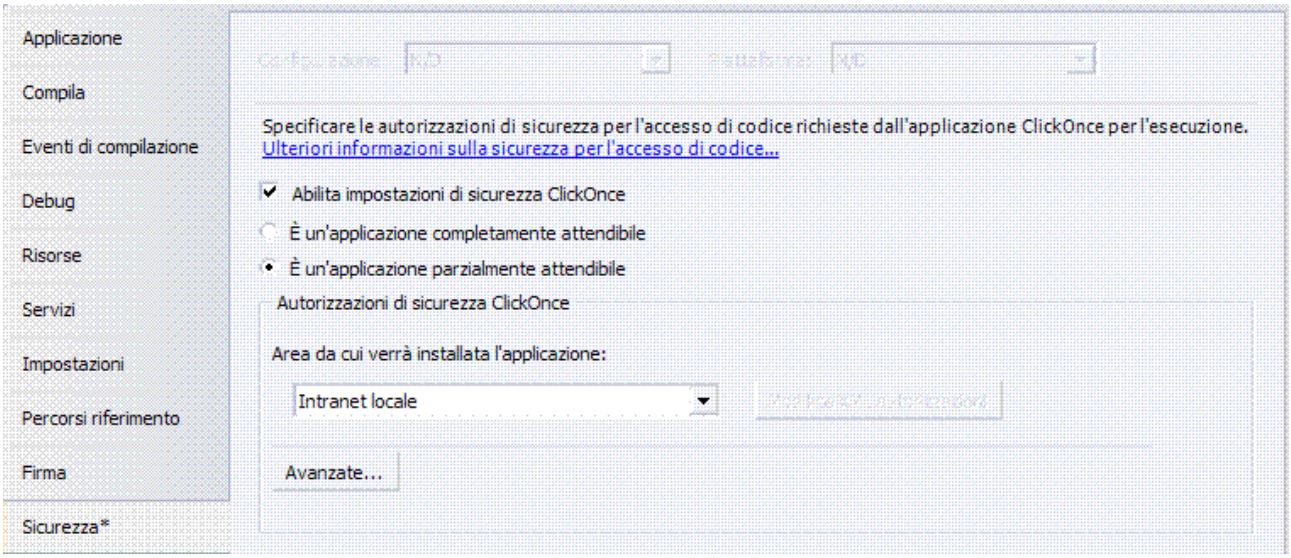
Al pari della procedura di upload è possibile utilizzare anche il metodo asincrono *DownloadFileGroupAsync* e gestire l'evento *DownloadFileGroupCompleted* per catturare il termine del download.

CAS (*Code Access Security*)
Consente di prevenire l'esecuzione di codice non attendibile o che richiede elevati permessi per essere eseguito.
Il run-time verifica, al caricamento di un assembly, la sua attendibilità sulla base dell'evidence, calcolata utilizzando parametri come la firma dell'assembly via strong-name o certificato digitale, la sua zona di esecuzione, utilizzandola per identificare il gruppo di codice di appartenenza e i relativi permessi di esecuzione.
I gruppi di codice, code group, sono determinati dagli amministratori di sistema.
Se i permessi richiesti non coincidono con il gruppo di codice di riferimento, il run-time genera un'eccezione di sicurezza.

Le applicazioni ClickOnce seguono il seguente processo di esecuzione.
Come si nota dai link riportati dalla pagina PUBLISH.HTM, è eseguito un file *.APPLICATION che altro non è che il deployment manifest, un file XML dove sono descritte informazioni strettamente legate alla pubblicazione attraverso ClickOnce come, ad esempio, l'identità dell'applicazione e come essa deve essere distribuita.
Il file .APPLICATION è eseguito perché associato alla libreria di classi *COM DFSHIM.DLL*, eseguita attraverso la stringa seguente.

rundll32.exe dfshim.dll,ShOpenVerbApplication %1,

Dove.
%1 rappresenta il percorso completo del file .APPLICATION.
Dal manifest è recuperato il file da eseguire e che sarà sottoposto alle regole di CAS che è il processo di verifica e validazione dei permessi di un assembly dipendentemente dalla zona in cui esso è eseguito.
Anche le applicazioni ClickOnce sono sottoposte al controllo della CAS e per questo motivo è importante dichiarare quelli che sono i permessi di esecuzione che si richiedono e che andranno a formare il permission set dell'assembly.
ClickOnce consente d'impostare sia manualmente sia utilizzando procedure guidate il livello di sicurezza da applicare.
Attraverso la scheda ***Sicurezza*** è possibile, infatti, dichiarare l'applicazione come parzialmente attendibile ed impostare la zona da cui sarà eseguita, ClickOnce calcolerà i permessi relativi alle impostazioni date.
Se, ad esempio, s'imposta l'intranet, saranno attribuiti i permessi minimi di esecuzione *FileDialogPermission, IsolatedStorageFilePermission, SecurityPermission, UIPermission e PrintingPermission*.
Ma se l'applicazione richiede in realtà permessi maggiori, come ad esempio, la scrittura sul file system identificata dal *FileIOPermission*, sarà generata un'eccezione.
Nell'esempio l'applicazione è eseguita o installata attraverso l'intranet, motivo per cui gli sono garantiti i diritti minimi di esecuzione.



Se, viceversa, l'applicazione necessita e richiede diritti di esecuzione superiori a quelli garantiti per la zona, ClickOnce chiederà all'utente di confermarne l'esecuzione.
Ma se si deve distribuire via Internet e l'applicazione richiede diritti superiori a quelli possibili con l'intranet, si hanno due alternative.

1. Distribuire l'applicazione utilizzando supporti removibili che ottengono i permessi di una zona completamente attendibile, pur mantenendo la modalità di aggiornamento via Internet.
2. Ottenere un certificato digitale per garantire l'autenticità del codice.

Con ClickOnce, infatti, è possibile firmare i file .APPLICATION utilizzando certificati digitali rilasciati da **CA** (*Certification Authority*) riconosciute.
Questo implica, però, un passaggio aggiuntivo che consenta la registrazione del publisher sul PC client, al fine di riconoscerlo come attendibile.
Dopo la registrazione nello store locale, si è in grado di eseguire correttamente e senza errori l'applicazione ClickOnce.

Creazione e utilizzo degli assembly con nome sicuro

Un nome sicuro è costituito dall'identità dell'assembly, corrispondente al nome semplice in formato testo, al numero di versione e alle informazioni sulle impostazioni eventualmente disponibili, nonché da una chiave pubblica e da una firma digitale.
Tale nome è generato da un file assembly tramite la chiave privata corrispondente.
Nel file assembly è contenuto il manifesto dell'assembly, in cui si trovano i nomi e gli hash di tutti i file che costituiscono l'assembly.
Un assembly con nome sicuro può utilizzare solo tipi di altri assembly con nome sicuro.
In caso contrario, la protezione dell'assembly con nome sicuro risulterebbe compromessa.
Processo di firma di un assembly con un nome sicuro e la successiva creazione di riferimenti basati su tale nome.
Un assembly A è creato con un nome sicuro mediante uno dei metodi seguenti.

1. L'hash del file contenente il manifesto dell'assembly è firmato dall'ambiente o dallo strumento di sviluppo con la chiave privata dello sviluppatore, tale firma digitale è memorizzata nel file eseguibile portabile **PE** (*Portable Executable*) contenente il manifesto dell'assembly A.
2. L'assembly B è un consumer dell'assembly A, nella sezione dei riferimenti del manifesto dell'assembly B è incluso un token che rappresenta la chiave pubblica dell'assembly A, un token è una parte della chiave pubblica completa, è utilizzato al posto della chiave stessa in modo da occupare meno spazio.
3. Quando l'assembly è inserito nella **GAC** (*Global Assembly Cache*), la firma con nome sicuro è verificata da CLR, quando si effettua l'associazione tramite nome sicuro in

fase di esecuzione, CLR confronta la chiave archiviata nel manifesto dell'assembly B con la chiave utilizzata per generare il nome sicuro per l'assembly A, se il controllo di protezione del .NET Framework ha esito positivo e l'associazione riesce, nell'utilizzare l'assembly B si può dare per certo che le informazioni contenute nell'assembly A non sono state alterate e provengono effettivamente dal relativo sviluppatore.

Per firmare un assembly con un nome sicuro, è necessario disporre di una coppia di chiavi, pubblica e privata.

Tale coppia di chiavi crittografiche, pubblica e privata, è utilizzata durante la compilazione per creare un assembly con nome sicuro.

È possibile creare una coppia di chiavi utilizzando lo strumento SN.EXE.

Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>sn /?
Utilità Nome sicuro di Microsoft (R) .NET Framework Versione 4.0.30319.1
Copyright (c) Microsoft Corporation. Tutti i diritti riservati.
Sintassi: sn [-q|-quiet] <opzione> [<parametri>]
Opzioni:
-c [<csp>]
Imposta o reimposta il nome del CSP da utilizzare per le operazioni MSCORSN.
-d <contenitore>
Elimina il contenitore di chiavi <contenitore>.
-D <assembly1> <assembly2>
Verifica che <assembly1> e <assembly2> differiscano solo nelle firme.
-e <assembly> <file di output>
Estrae la chiave pubblica da <assembly> e la inserisce in <file di output>.
-i <file di input> <contenitore>
Installa la coppia di chiavi da <file di input> nel contenitore di chiavi <contenitore>.
-k [<dimensione chiave>] <file di output>
Genera una nuova coppia di chiavi della dimensione specificata e la scrive nel <file di output>.
-m [y|n]
Attiva (y), disattiva (n) o verifica (nessun parametro) se i contenitori
di chiavi sono specifici del computer o dell'utente.
-o <file di input> [<file di output>]
Converte la chiave pubblica in <file di input> nel file di testo <file di output> con elenco di valori in byte decimali separato da virgole. Se <file di output> è omesso, il testo viene copiato negli Appunti.
-p <file di input> <file di output>
Estrae la chiave pubblica dalla coppia di chiavi in <file di input> e la esporta in <file di output>.
-pc <contenitore> <file di output>
Estrae la chiave pubblica dalla coppia di chiavi in <contenitore> e la esporta in <file di output>.
-Pb [y|n]
Abilita (y), disabilita (n) o verifica (nessun parametro) i criteri CLR per
ignorare la firma con nome sicuro delle applicazioni attendibili sui relativi assembly.
-q
Modalità non interattiva. Questa opzione deve essere specificata per prima sulla riga di comando ed evita la visualizzazione dei messaggi, ad eccezione di quelli di errore.
-R[a] <assembly> <file di input> [-ecma]
Firma di nuovo l'assembly firmato o parzialmente firmato con la coppia di chiavi in <file di input>.
Se si utilizza -Ra, vengono ricalcolati gli hash per tutti i file nell'assembly.
Se si utilizza -ecma, il file di input viene trattato come chiave reale per la firma ECMA.

-Rc[a] <assembly> <contenitore> [-ecma]

Firma di nuovo l'assembly firmato o parzialmente firmato con la coppia di chiavi nel contenitore <contenitore>. Se si utilizza -Rca, sono ricalcolati gli hash per tutti i file nell'assembly -ecma, il contenitore è trattato come chiave reale per la firma ECMA.

-Rh <assembly>

Ricalcola hash per tutti i file nell'assembly.

-t[p] <file di input>

Visualizza il token per la chiave pubblica in <file di input> (con la chiave pubblica se si utilizza l'opzione -tp).

-T[p] <assembly>

Visualizza il token per la chiave pubblica di <assembly> (con la chiave pubblica se si utilizza l'opzione -Tp).

-TS <assembly> <file di input>

Applica firma di test all'assembly firmato o parzialmente firmato con la coppia di chiavi in <file di input>.

-TSc <assembly> <contenitore>

Applica firma di test all'assembly firmato o parzialmente firmato con la coppia di chiavi nel contenitore di chiavi <contenitore>.

-v[f] <assembly> [{-ecmakey <file di chiave> | -ecmacontainer <contenitore>}]

Verifica la coerenza interna della firma con nome sicuro di <assembly>. Specificando -vf si impone la verifica anche se disattivata nel Registro. Se si specifica -ecmakey, il file di chiave viene trattato come chiave ECMA reale.Se si specifica -ecmacontainer, il contenitore viene trattato come chiave ECMA reale.

-Vl

Elenca le impostazioni correnti per la verifica del nome sicuro nel computer in uso.

-Vr <assembly> [<elenco utenti>] [<file di input>]

Registra <assembly> per l'omissione della verifica (con un elenco facoltativo di nomi utente separati da virgole e una chiave pubblica di test facoltativa in <file di input>).

*<assembly> può essere specificato con * per indicare tutti gli assembly*

*oppure con *,<token della chiave pubblica> per indicare tutti gli assembly con il token della chiave pubblica specificato. I token delle chiavi pubbliche dovrebbero essere specificati come una stringa di cifre esadecimal*

i.

-Vu <assembly>

Annulla la registrazione di <assembly> per l'omissione della verifica. Per la denominazione di <assembly> si usano le stesse regole valide per -Vr.

-Vx

Rimuove tutte le voci per le quali è omessa la verifica.

-?

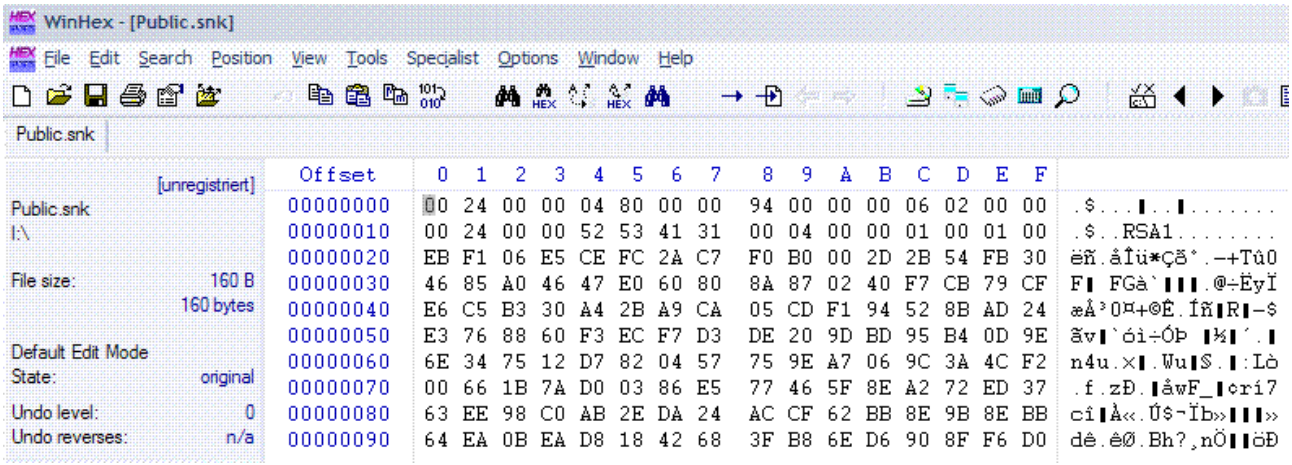
-h

Visualizza questo argomento della Guida.

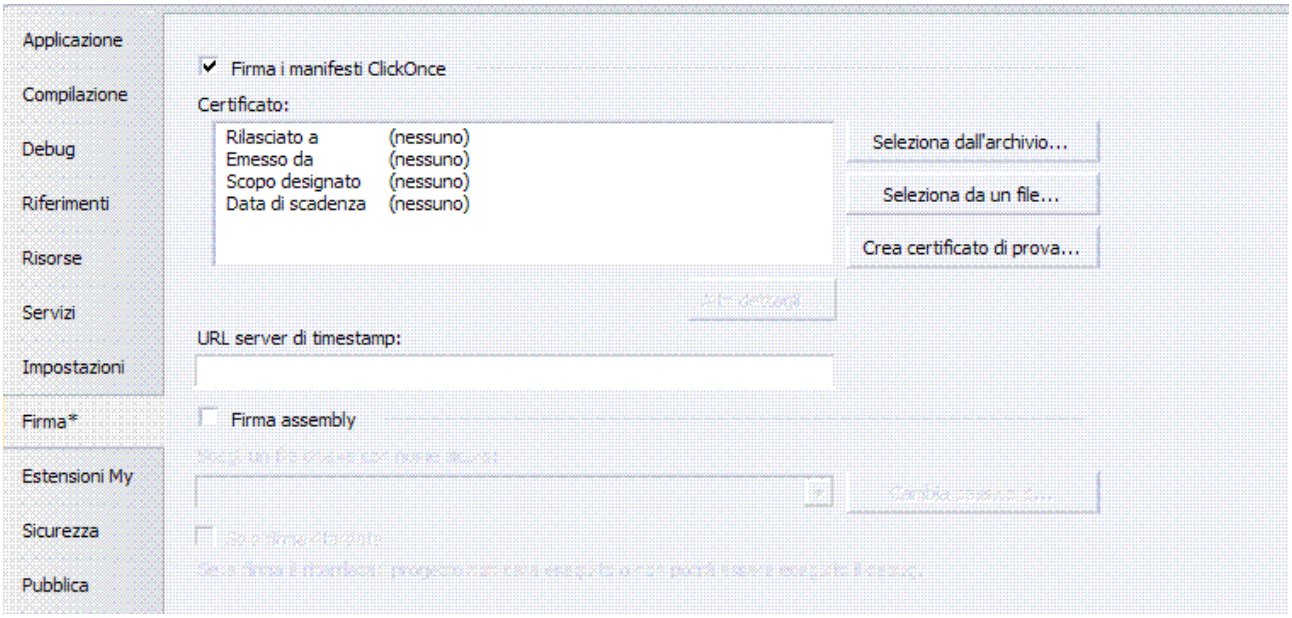
C:\Programmi\Microsoft Visual Studio 10.0\VC>sn -k DigitalSignTools.snk
Utilità Nome sicuro di Microsoft (R) .NET Framework Versione 4.0.30319.1
Copyright (c) Microsoft Corporation. Tutti i diritti riservati.
Coppia di chiavi scritta in DigitalSignTools.snk

Estrarre quindi la chiave pubblica dalla coppia di chiavi e copiarla in un file distinto.

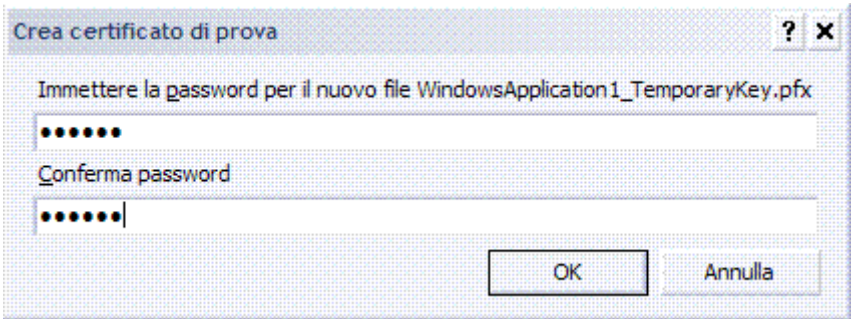
C:\Programmi\Microsoft Visual Studio 10.0\VC>sn -p DigitalSignTools.snk Public.snk
Utilità Nome sicuro di Microsoft (R) .NET Framework Versione 4.0.30319.1
Copyright (c) Microsoft Corporation. Tutti i diritti riservati.
La chiave pubblica è stata scritta in Public.snk



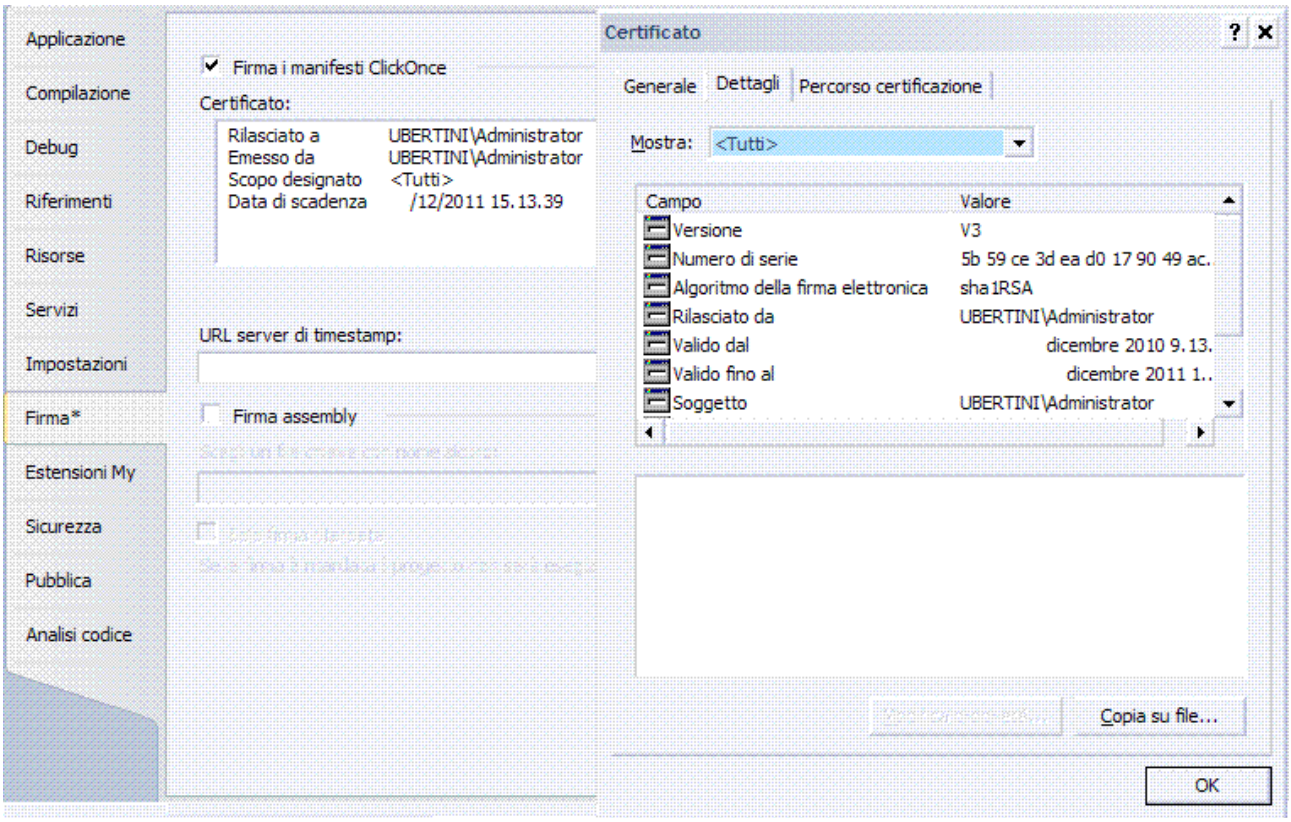
È possibile creare una coppia di chiavi utilizzando l'ambiente MDE.
Fare clic sul menu ***Progetto/Proprietà di …***, fare clic sulla scheda ***Firma***.
Segno di spunta su ***Firma i manifesti ClickOnce***.



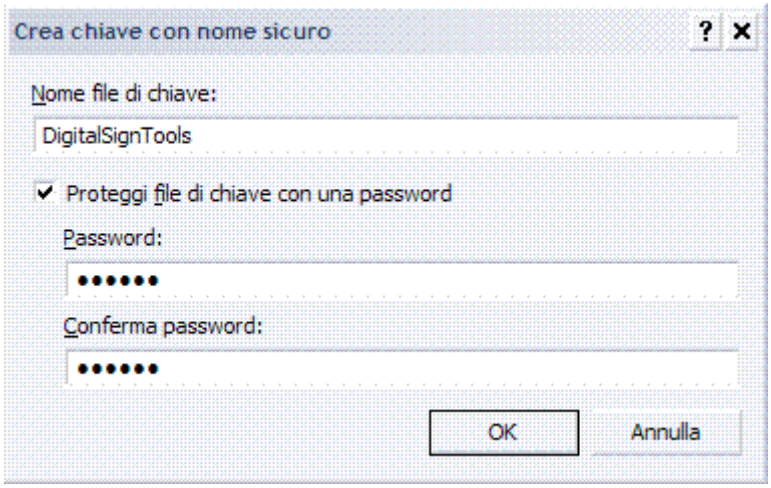
Fare clic sul pulsante ***Crea certificato di prova…***



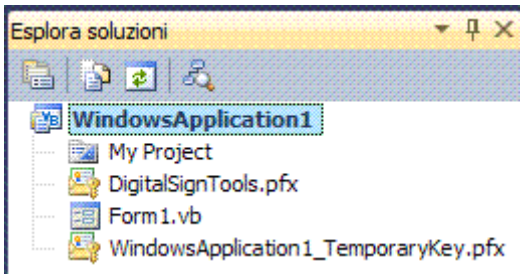
Fare clic su ***OK*** e fare clic sul pulsante ***Altri dettagli…***

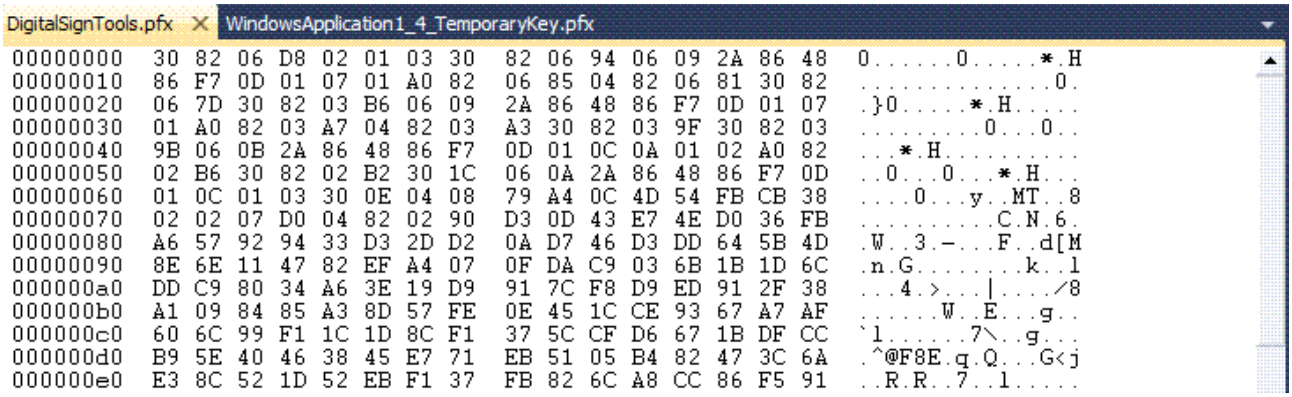


Segno di spunta su ***Firma assembly***.
Nella casella di testo ***Scegli un file chiave con nome sicuro...*** selezionare ***<Nuova...>.***



In Esplora soluzioni sono creati i seguenti file.





Sono disponibili diversi modi per firmare un assembly con un nome sicuro.

1. Utilizzo degli attributi dell'assembly per inserire le informazioni relative al nome sicuro nel codice: *AssemblyKeyFileAttribute* o *AssemblyKeyNameAttribute*, a seconda della posizione del file di chiave da utilizzare.
2. Utilizzo di opzioni del compilatore, ad esempio */keyfile* o */delaysign* in C# e Visual Basic oppure l'opzione del linker */KEYFILE* o */DELAYSIGN* in C++.
3. Utilizzo dello strumento Assembly Linker, AL.EXE.

Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>al
Microsoft (R) Assembly Linker versione 10.0.30319.1
Copyright (C) Microsoft Corporation. All rights reserved.
Uso: al [opzioni] [file di origine]
Opzioni: (occorre specificare '/out')
/? o /help Visualizza questo messaggio relativo all'uso
@<nomefile> Legge il file di risposta per ulteriori opzioni
/algid:<id> Algoritmo utilizzato per apporre un numero hash ai file (esadecimale)
/base[address]:<indirizzo> Indirizzo di base della libreria
/bugreport:<nomefile> Crea un file di report sui bug
/comp[any]:<testo> Nome della società
/config[uration]:<testo> Stringa di configurazione
/copy[right]:<testo> Informazioni sul copyright
/c[ulture]:<testo> Impostazioni cultura supportate
/delay[sign][+|-] Ritarda la firma dell'assembly
/descr[iption]:<testo> Descrizione
/e[vidence]:<nomefile> File di prove della sicurezza da incorporare
/fileversion:<versione> Versione Win32 facoltativa (esegue l'override della versione dell'assembly)
/flags:<flag> Flag dell'assembly (esadecimale)
/fullpaths Visualizza i file che utilizzano nomi di file completi
/keyf[ile]:<nomefile> File contenente la chiave per firmare l'assembly
/keyn[ame]:<testo> Nome del contenitore di chiavi per firmare l'assembly
/main:<metodo> Specifica il nome del metodo del punto di ingresso
/nologo Evita la visualizzazione del messaggio di avvio e delle informazioni sul copyright
/out:<nomefile> Nome del file di output per il manifesto dell'assembly
/platform:<testo> Limita le piattaforme su cui il codice può essere eseguito; i valori devono essere x86, Itanium,

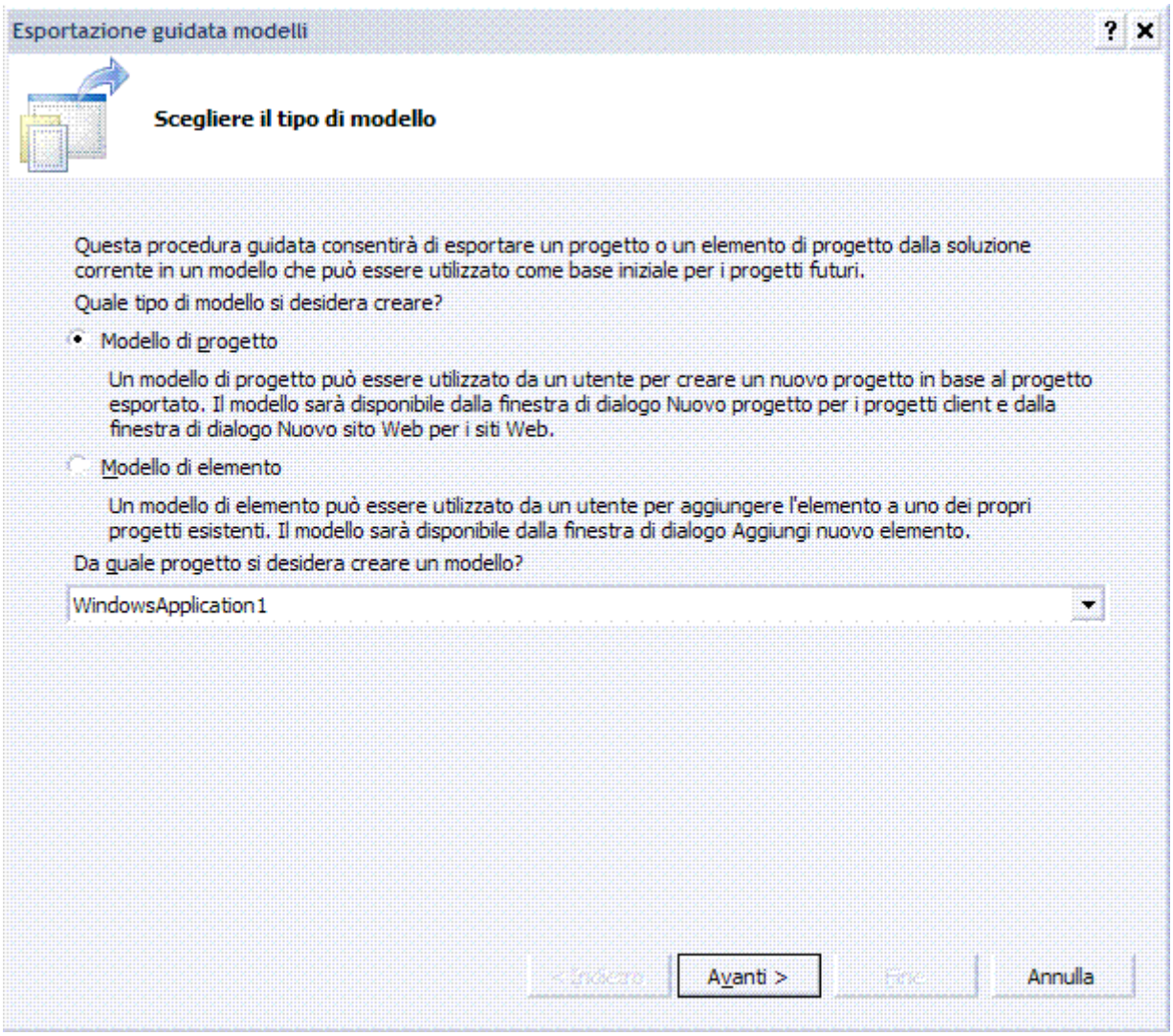
x64 o anycpu (predefinito)
/prod[uct]:<testo> Nome del prodotto
/productv[ersion]:<testo> Versione del prodotto
/t[arget]:lib[rary] Crea una libreria
/t[arget]:exe Genera un file eseguibile da console
/t[arget]:win[exe] Crea un eseguibile Windows
/template:<nomefile> Specifica un assembly da cui ottenere le opzioni predefinite
/title:<testo> Titolo
/trade[mark]:<testo> Informazioni sul marchio
*/v[ersion]:<versione> Versione (utilizzare * per generare automaticamente i numeri non specificati)*
/win32icon:<nomefile> Utilizza questa icona per l'output
/win32res:<nomefile> Specifica il file di risorse Win32

Origini: (è necessario specificare almeno un'origine)
<nomefile>[,<file di destinazione>] aggiunge il file all'assembly
/embed[resource]:<nomefile>[,<nome>[,Private]] incorpora il file come risorsa nell'assembly
/link[resource]:<nomefile>[,<nome>[,<file di destinazione>[,Private]]] collega il file come risorsa all'assembly

PROGETTO DI MODELLO

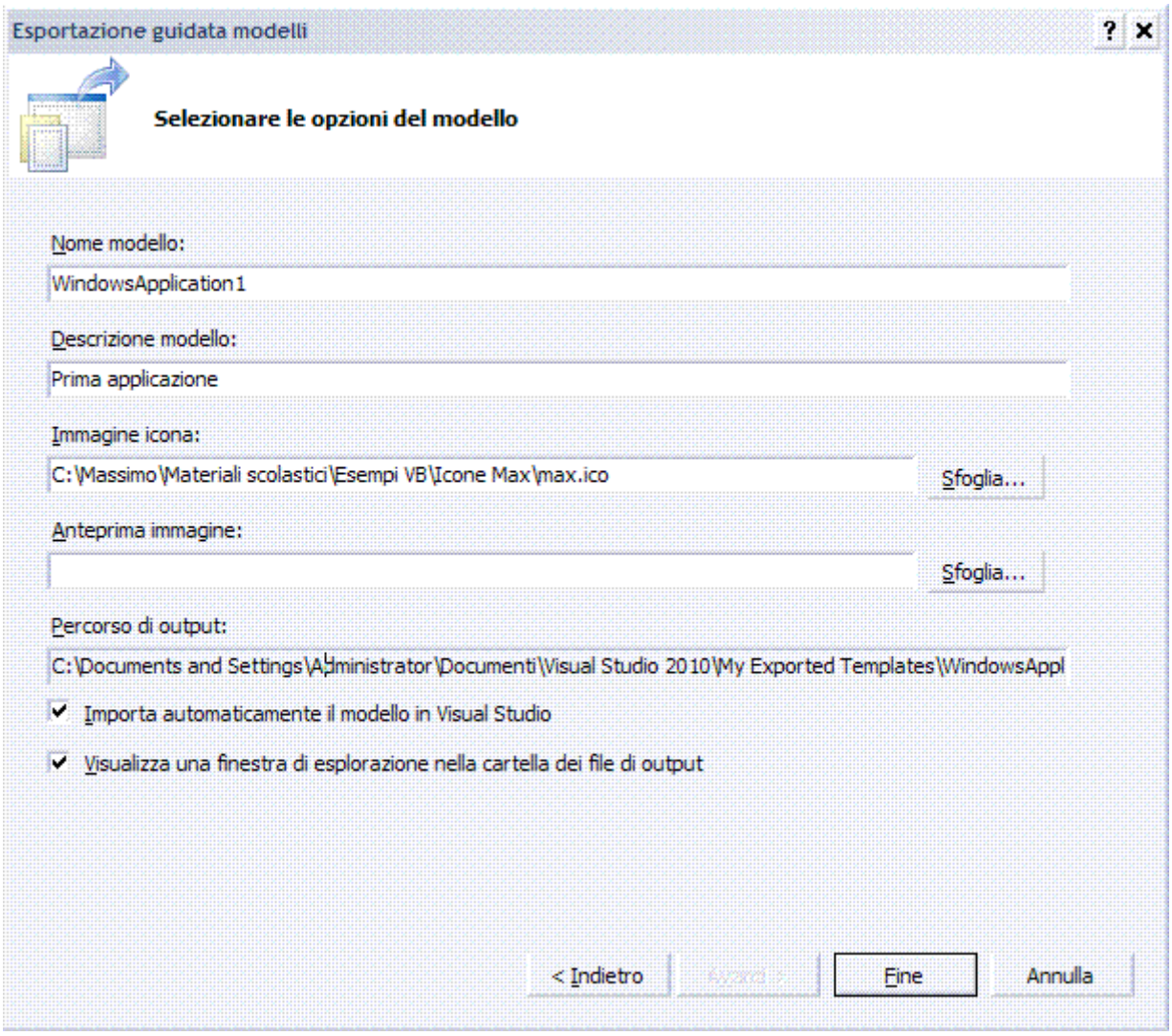
INTRODUZIONE

È utilizzato come punto di partenza per la creazione di un progetto e contiene le impostazioni, i riferimenti e i file per iniziare un determinato tipo di progetto.
Il primo passo per la creazione di un modello è quello di creare un progetto che sia compilato senza generare errori.
Il secondo passo è quello di produrre la documentazione.
Dopo avere completato il progetto e la documentazione, si può creare il modello di progetto facendo clic sul menu ***File/Esporta modello…*** si avvia il wizard ***Esportazione guidata modelli***.

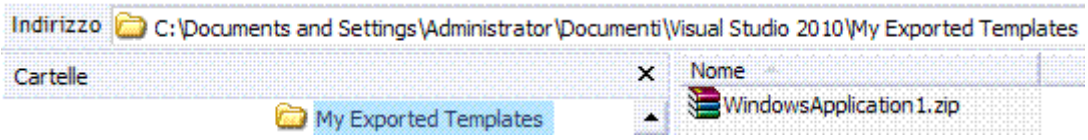


Nella prima finestra di dialogo, si deve scegliere il tipo di modello.
Selezionare la prima voce e fare clic su ***Avanti***.

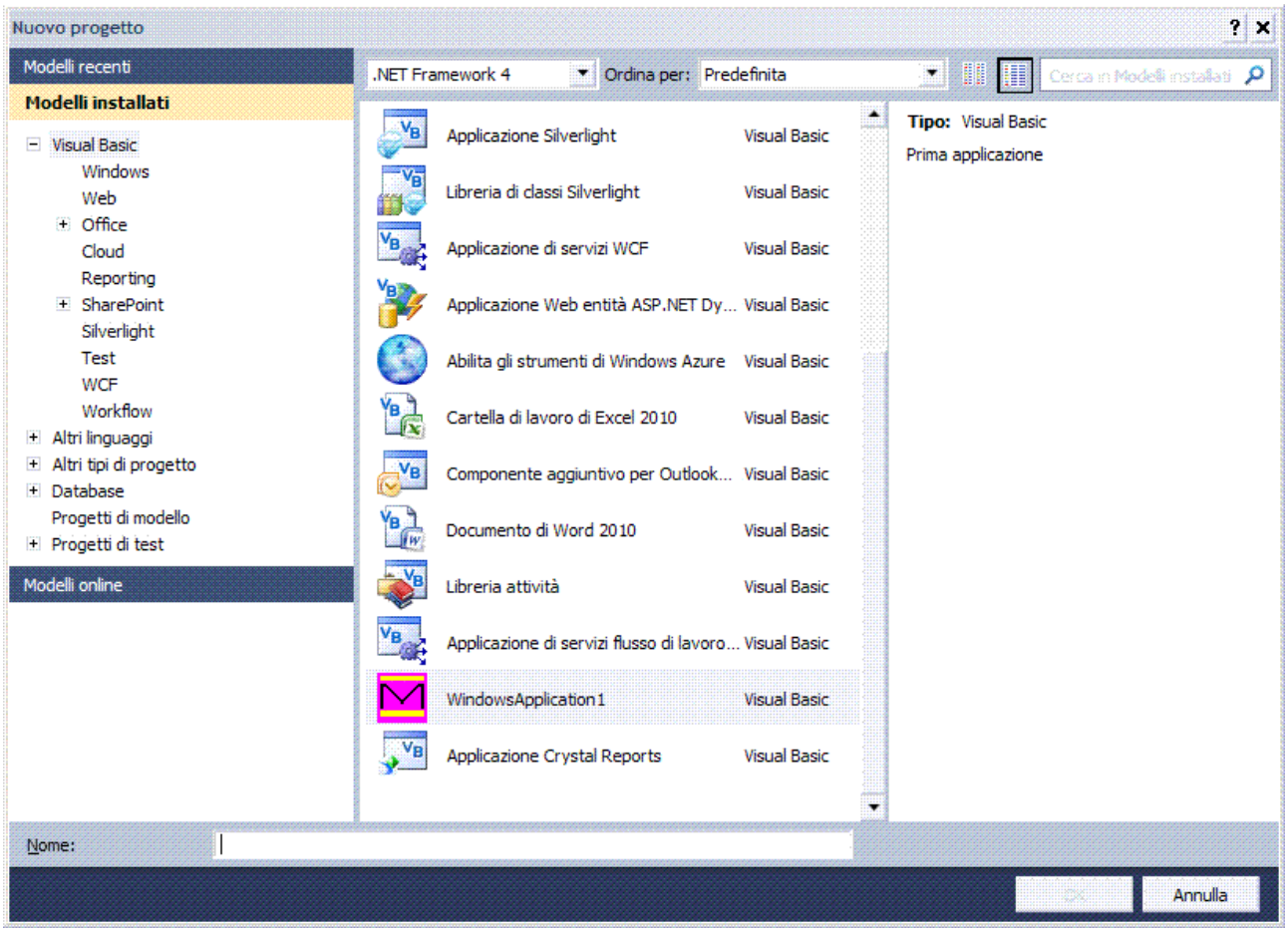
Nella seconda finestra si può selezionare il nome, la descrizione e l'icona del modello.



Dopo le opportune selezioni fare clic su ***Fine***.
Il progetto sarà compresso ed esportato in un file con estensione ZIP e inserito nella cartella seguente.



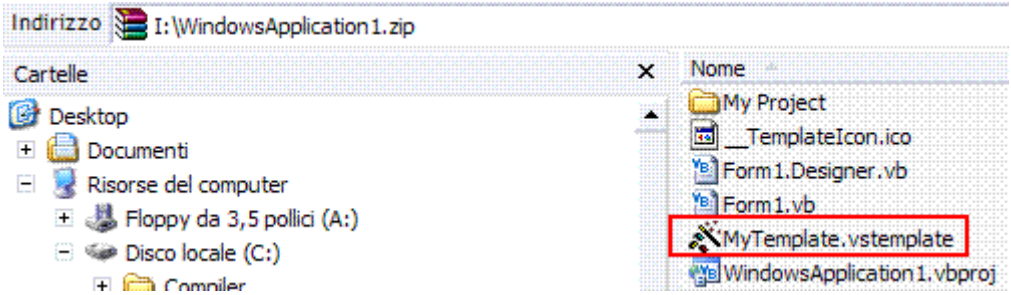
Infine, si trova nella finestra ***Nuovo progetto***, ***Modelli installati*** il modello appena creato.



Un modello contiene la documentazione dettagliata che spiega i vari componenti del progetto, oppure una guida dettagliata per eventuali personalizzazioni a completamento del progetto stesso.

METADATI

Se si apre il file ZIP generato dal wizard, si nota come sia stato creato un file con estensione VSTEMPLATE.



Ogni modello include questo file che racchiude tutti i metadati in formato XML che forniscono a VB.NET le informazioni indispensabili per visualizzare il modello nella finestra di dialogo ***Nuovo progetto*** e le informazioni necessarie per creare un nuovo progetto dal modello.
I tre componenti fondamentali sono i seguenti.

1. *VSTemplate*

Permette d'identificare il modello come modello di progetto o di elemento in base al valore

dell'attributo *Type* e indica il numero di versione del modello per mezzo dell'attributo *Version*.

2. *TemplateData*
Permette d'indicare i dati che classificano il modello e di definire come deve essere visualizzato il modello, nella finestra di dialogo ***Nuovo progetto*** o ***Aggiungi nuovo elemento***.

3. *TemplateContent*
Permette d'indicare tutti i file inclusi nel modello.

File MYTEMPLATE.VSTEMPLATE

```
<VSTemplate Version="3.0.0"
xmlns="http://schemas.microsoft.com/developer/vstemplate/2005" Type="Project">
  <TemplateData>
    <Name>WindowsApplication1</Name>
    <Description>Prima applicazione</Description>
    <ProjectType>VisualBasic</ProjectType>
    <ProjectSubType>
    </ProjectSubType>
    <SortOrder>1000</SortOrder>
    <CreateNewFolder>true</CreateNewFolder>
    <DefaultName>WindowsApplication1</DefaultName>
    <ProvideDefaultName>true</ProvideDefaultName>
    <LocationField>Enabled</LocationField>
    <EnableLocationBrowseButton>true</EnableLocationBrowseButton>
    <Icon>__TemplateIcon.ico</Icon>
  </TemplateData>
  <TemplateContent>
    <Project TargetFileName="WindowsApplication1.vbproj"
File="WindowsApplication1.vbproj" ReplaceParameters="true">
      <ProjectItem ReplaceParameters="true"
TargetFileName="Form1.vb">Form1.vb</ProjectItem>
      <ProjectItem ReplaceParameters="true"
TargetFileName="Form1.Designer.vb">Form1.Designer.vb</ProjectItem>
      <Folder Name="My Project" TargetFolderName="My Project">
        <ProjectItem ReplaceParameters="true"
TargetFileName="Application.myapp">Application.myapp</ProjectItem>
        <ProjectItem ReplaceParameters="true"
TargetFileName="Application.Designer.vb">Application.Designer.vb</ProjectItem>
        <ProjectItem ReplaceParameters="true"
TargetFileName="AssemblyInfo.vb">AssemblyInfo.vb</ProjectItem>
        <ProjectItem ReplaceParameters="true"
TargetFileName="Resources.resx">Resources.resx</ProjectItem>
        <ProjectItem ReplaceParameters="true"
TargetFileName="Resources.Designer.vb">Resources.Designer.vb</ProjectItem>
        <ProjectItem ReplaceParameters="true"
TargetFileName="Settings.settings">Settings.settings</ProjectItem>
        <ProjectItem ReplaceParameters="true"
TargetFileName="Settings.Designer.vb">Settings.Designer.vb</ProjectItem>
      </Folder>
    </Project>
  </TemplateContent>
</VSTemplate>
```

Nella sezione *TemplateContent* importante è l'attributo *Project* che indica, in dettaglio, tutti i file o le directory da aggiungere al progetto, all'interno si trova il parametro figlio *ProjectItem*, che ammette i seguenti attributi.

TargetFileName
Permette d'indicare il nome e il percorso di ogni elemento quando è creato un progetto dal modello, è possibile utilizzare l'attributo *TargetFileName* per creare una struttura di directory diversa dalla struttura di directory presente nel file ZIP del modello.
ReplaceParameters
È un valore booleano che permette d'indicare se nell'elemento specificato, i valori dei parametri dovranno essere sostituiti nel momento in cui è creato un progetto dal modello, il valore predefinito è *false*.
OpenInEditor
È un valore booleano che permette d'indicare se l'elemento specificato, dovrà essere aperto nell'editor di VB.NET quando è creato un progetto dal modello, il valore predefinito è *false*.
OpenInWebBrowser
È un valore booleano che permette d'indicare se l'elemento specificato, dovrà essere aperto nel browser quando è creato un progetto dal modello, nel browser è possibile aprire solo i file XHTML e i file di testo locali del progetto, non è possibile aprire URL esterni, il valore predefinito è *false*.
OpenInHelpBrowser
È un valore booleano che permette d'indicare se l'elemento specificato, dovrà essere aperto nel visualizzatore della Guida quando è creato un progetto dal modello, nel browser della Guida è possibile aprire solo i file XHTML e i file di testo locali del progetto, non è possibile aprire URL esterni, il valore predefinito è *false*.
OpenOrder
Permette d'indicare un valore numerico che rappresenta l'ordine in cui gli elementi saranno aperti nei rispettivi editor, tutti i valori devono essere multipli di 10, ad essere aperti per primi saranno gli elementi con valori *OpenOrder* inferiori.

In un modello la prima pagina visualizzata, all'apertura, deve essere quella relativa alla documentazione, aprire il file con estensione VSTEMPLATE con un editor qualsiasi e individuare l'elemento *ProjectItem* che contiene il file di documentazione.
Nell'editor di codice porre *OpenInEditor="true"*.
Nel browser porre *OpenInWebBrowser="true"*.
Nel visualizzatore della Guida porre *OpenInHelpBrowser="true".*
È possibile, inoltre, specificare più file e anche l'ordine in cui dovranno essere aperti, per esempio porre l'attributo *OpenOrder="10"* per fare in modo che la documentazione sia visualizzata per prima.

```
<ProjectItem OpenOrder="10"
   OpenInWebBrowser="true">Doc\Webcam.htm
   </ProjectItem>
```

È possibile organizzare i modelli installati in base alle proprie esigenze creando delle sotto categorie personalizzate all'interno della cartella del linguaggio di programmazione.
Per creare le sottocategorie è sufficiente creare le sotto directory corrispondenti, in questo modo nelle finestre di dialogo ***Nuovo progetto*** e ***Aggiungi nuovo elemento*** le sottocategorie saranno rappresentate come cartelle virtuali all'interno di ciascun linguaggio.
Affinché i modelli possano essere mostrati nelle finestre di dialogo ***Nuovo progetto*** e ***Aggiungi nuovo elemento***, i file che lo compongono devono essere inseriti in uno dei

due percorsi predefiniti.
Per impostazione predefinita, i modelli installati con VB.NET risiedono nei seguenti percorsi.

C:\Programmi\Microsoft Visual Studio 10.0\Common7\IDE\ProjectTemplates\VisualBasic
C:\Programmi\Microsoft Visual Studio 10.0\Common7\IDE\ItemTemplates\VisualBasic

Se in questi percorsi esiste un file compresso che contiene un file con estensione VSTEMPLATE, nelle finestre di dialogo sarà visualizzato il modello corrispondente.

MONO

INTRODUZIONE

È la soluzione per realizzare soluzioni .NET multipiattaforma che siano perfettamente compatibili con piattaforme diverse da Windows.

È un progetto open source coordinato da Novell, precedentemente da Ximian, per creare un insieme di strumenti compatibili con il .NET Framework, secondo gli standard **ECMA** (*European Computer Manufacturers Ass*ociation) (ECMA-334 e ECMA-335).
Nel 2000 la società Ximian, **GNOME** (*GNU Network Object Model Environment*), fondata e diretta da **Miguel de Icaza** (Città del Messico, 1972), s'interessò al .NET Framework e il 19 luglio 2001 annunciò il progetto open source Mono alla conferenza O'Reilly.
Il 30 giugno 2004 fu rilasciata la versione 1.0, attualmente è disponibile la versione 2.0.
LinuxWorld 2006: Boston, Mono vince il premio come miglior piattaforma di sviluppo.
Attualmente de Icaza è il vice presidente allo sviluppo della Novell.

Piattaforme hardware

- ✓ Architettura x86 e x64.
- ✓ PowerPC.
- ✓ **ARM** [*Advanced **RISC** (Reduced Instruction Set Computer) Machine].*
- ✓ Alpha.

Sistemi operativi

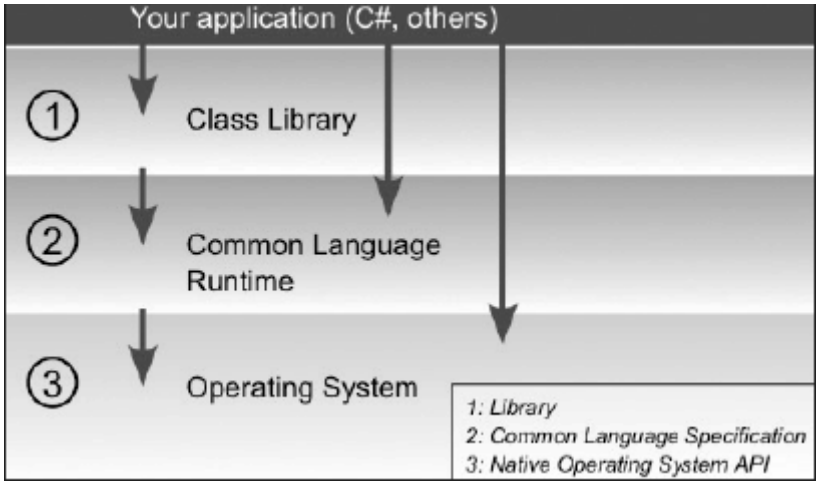
- ✓ Linux.
- ✓ Windows.
- ✓ Mac OSX.
- ✓ **BSD** (*Berkeley Software Distribution*).
- ✓ Sun Solaris.
- ✓ Nintendo Wii.
- ✓ Sony Playstation 3.
- ✓ Apple iPhone.

Linguaggi di programmazione

- ✓ C#.
- ✓ C.
- ✓ Java.
- ✓ BOO, linguaggio progettato da Rodrigo Barreto de Oliveira, simile al Python.
- ✓ Nemerle.
- ✓ Visual Basic.NET.
- ✓ Python.
- ✓ Javascript.

- ✓ Oberon.
- ✓ PHP.
- ✓ Object Pascal.
- ✓ Lua.
- ✓ Cobra.
- ✓ Component Pascal.
- ✓ Delta Forth.
- ✓ DotLisp.
- ✓ #Smalltalk.
- ✓ Ruby.
- ✓ Ada.
- ✓ Logo.
- ✓ Kylix Delphi per Linux.

ARCHITETTURA



Librerie.

- ✓ **GTK**# (*GNOME ToolKit*), binding per il kit di sviluppo GTK per Unix e Windows.
- ✓ #ZipLib: libreria per la gestione di formati di compressione.
- ✓ Tao Framework: bindings per sviluppare utilizzando OpenGL.
- ✓ Mono.Directory.LDAP/Novell.Directory.LDAP: accesso **LDAP** (*Lightweight Directory Access Protocol*) per applicativi .NET.
- ✓ Mono.Data: supporto per PostgreSQL, MySQL, Firebird, Sybase **ASE** (*Adaptive Server Enterprise*), **IBM** (*International Business Machines*) DB2, SQLite, Microsoft SQL Server, Oracle e **ODBC** (*Open DataBase Connectivity*).
- ✓ Mono.Cairo: binding per il motore 2D di rendering Cairo.
- ✓ Mono.Posix / Mono.UNIX: binding per realizzare applicativi POSIX in linguaggio C#.
- ✓ Mono.Remoting.Channels.Unix: supporto per utilizzo di socket Unix.
- ✓ Mono.Security: enhanced security e crypto framework.
- ✓ Mono.Math: generazione di BigInteger e numeri primi.
- ✓ Mono.Http: supporto per la realizzazione di servers **HTTP** (*Hyper Text Transfer Protocol*) e dei servizi associati.
- ✓ Mono.XML: supporto esteso per **XML** (*eXtensible Markup Language*).
- ✓ Managed.Windows.Forms (System. Windows.Forms): supporto per la realizzazione d'interfacce che utilizzano Windows.Forms in maniera cross platform.
- ✓ Remoting.CORBA: supporto per **CORBA** (*Common Object Request Broker Architecture*).
- ✓ Ginzu: supporto per il middleware **ICE** (*Internet Communication Engine*) che consente la programmazione di tali servizi in C++, .NET, Java, Python, Objective-C, Ruby e

PHP.

Componenti

- ✓ Componenti Core, forniscono grammatica e semantica in linguaggio C# e lo standard CLI.
- ✓ Stack Mono/Linux/GNOME, fornisce tutti quegli strumenti open source che sono utilizzati per lo sviluppo di applicativi.
- ✓ Stack di compatibilità con le tecnologie Microsoft, fornisce le tecnologie necessarie per consentire compilazione ed esecuzione in ambienti non Windows del codice .NET.

Mono supporta anche Silverlight, questa tecnologia è nota come *Moonlight*.
Quando si realizza un'applicazione che si appoggia a una libreria grafica, generalmente si può scegliere tra un limitato sotto insieme di soluzioni, in ambito Windows si parla di **WPF** (*Windows Presentation Foundation*); con Mono si ha un'ulteriore libertà di scelta: le applicazioni potranno utilizzare GTK, Qyoto, Qt4, Cocoa, wxNet.

Esempio, applicazione console per MC OSX.

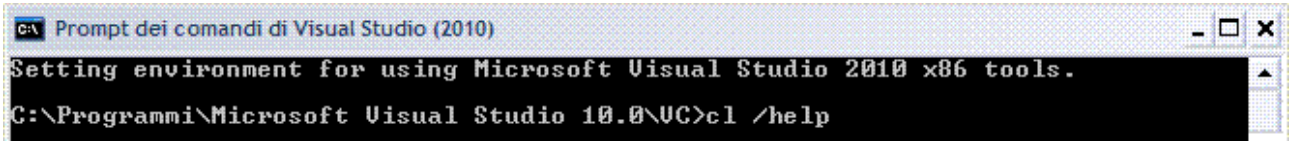
```
using System;
namespace Mono
{
   class MainClass
   {
      public static void Main(string[] args)
      {
         Console.WriteLine("Hello World!");
      }
   }
}
```

Non c'è alcuna differenza, quindi è semplice effettuare un porting di un progetto.

Esempio, applicazione grafica per MC OSX.

```
using System;
using System.Windows.Forms;
public class HelloWorld : Form
{
   static public void Main()
   {
      Application.Run(new HelloWorld());
   }
   public HelloWorld()
   {
      Text = "Hello Mono World";
   }
}
```

COMPILAZIONE CLI (*COMMAND LINE INTERFACE*)



Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>cl /help
Microsoft (R) 32-bit C/C++ Optimizing Compiler versione 16.00.30319.01 per 80x86
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.
OPZIONI DEL COMPILATORE C/C++
-OTTIMIZZAZIONE-
/O1 Riduce al minimo lo spazio /O2 Ottimizza la velocità
/Ob<n> Espansione inline (n predefinito=0)
/Od Disabilita le operazioni di ottimizzazione (predefinita)
/Og Abilita l'ottimizzazione globale /Oi[-] Abilita le funzioni intrinseche
/Os Ottimizza lo spazio del codice /Ot Ottimizza la velocità del codice
/Ox Combina le opzioni di ottimizzazione
/Oy[-] Abilita l'omissione dei puntatori ai frame
-GENERAZIONE DEL CODICE-
/GF Abilita la condivisione delle stringhe in sola lettura
/Gm[-] Abilita la ricompilazione minima /Gy[-] Separa le funzioni per il linker
/GS[-] Abilita i controlli di sicurezza /GR[-] Abilita RTTI di C++
/GX[-] Abilita EH (equivalente a /EHsc) di C++
/EHs Abilita EH (senza eccezioni SEH) di C++
/EHa Abilita EH con eccezioni SEH) di C++
/EHc Usa nothrow come impostazione predefinita di extern "C"
/fp:<except[-]|fast|precise|strict> Scegliere il modello a virgola mobile:
except[-] Considera le eccezioni di virgola mobile durante la generazione del codice
fast Modello a virgola mobile "fast"; i risultati sono meno prevedibili
precise Modello a virgola mobile "precise"; i risultati sono prevedibili
strict Modello a virgola mobile "strict" (implica /fp:except)
/Qfast_transcendentals Genera intrinseci FP inline anche con /fp:except
/GL[-] Abilita la generazione di codice in fase di collegamento
/GA Ottimizza per l'applicazione Windows
/Ge Impone il controllo dello stack per tutte le funzioni
/Gs[num] Controlla le chiamate di verifica dello stack
/Gh Abilita la chiamata di funzione _penter
/GH Abilita la chiamata di funzione _pexit
/GT Genera accessi TLS indipendenti da fiber
/RTC1 Abilita i controlli rapidi (/RTCsu)
/RTCc Converte in controlli di tipo più limitato
/RTCs Verifica runtime dello stack frame
/RTCu Controlli di utilizzo locale non inizializzati
/clr[:opzione] Compila per Common Language Runtime, dove opzione è:
pure Produce un file di output solo IL (senza codice eseguibile nativo)
safe Produce un file di output verificabile solo IL
oldSyntax Accetta la sintassi delle estensioni gestite di Visual C++ 2002/2003

initialAppDomain Abilita il comportamento AppDomain iniziale di Visual C++ 2002
noAssembly Non produce un assembly /Gd Convenzione di chiamata __cdecl
/Gr Convenzione di chiamata __fastcall /Gz Convenzione di chiamata __stdcall
/GZ Abilita i controlli dello stack (/RTCs)
/QIfist[-] Usa FIST anziché ftol()
/hotpatch assicura la spaziatura interna delle funzioni per le immagini su cui è applicabile una patch a caldo
/arch:<SSE|SSE2|AVX> Requisiti minimi dell'architettura della CPU. Scegliere una delle seguenti opzioni:
SSE Consente di utilizzare le istruzioni disponibili con le CPU abilitate per SSE
SSE2 Consente di utilizzare le istruzioni disponibili con le CPU abilitate per SSE2
AVX Consente di utilizzare le istruzioni
/Qimprecise_fwaits Genera elementi FWAIT solo esternamente all'istruzione "Try" e non al suo interno
/Qsafe_fp_loads Genera caricamenti FP sicuri
-FILE DI OUTPUT-
/Fa[file] Specifica un file di listato dell'assembly
/FA[scu] Configura il listato dell'assembly
/Fd[file] Specifica un file PDB /Fe<file> Specifica un file eseguibile
/Fm[file] Specifica un file map /Fo<file> Specifica un file oggetto
/Fp<file> Specifica un file di intestazione precompilata
/Fr[file] Specifica il file di origine del browser
/FR[file] Specifica un file SBR esteso
/Fi[file] Specifica un file pre-elaborato
/doc[file] Elabora i commenti relativi alla documentazione XML ed eventualmente specifica il file XDC
-PREPROCESSORE-
/AI<dir> Specifica il percorso di ricerca dell'assembly
/FU<file> Impone l'utilizzo di assembly/modulo
/C Non rimuove i commenti /D<nome>{=|#}<testo> Definisce una macro
/E Pre-elabora in stdout
/EP Pre-elabora in, senza istruzione #line
/P Pre-elabora nel file /Fx Unisce il codice inserito al file
/FI<file> Specifica il file di inclusione da utilizzare
/U<nome> Rimuove la macro definita in precedenza
/u Rimuove tutte le macro definite in precedenza
/I<dir> Specifica il percorso di ricerca/X Ignora "posizioni standard"
-LINGUAGGIO-
/Zi Abilita le informazioni di debug
/Z7 Abilita informazioni di debug obsolete
/Zp[n] Comprime le strutture allineandole su un limite di n byte
/Za Disabilita le estensioni /Ze Abilita le estensioni (predefinita)
/Zl Omette il nome della libreria predefinita in OBJ
/Zg Genera i prototipi delle funzioni /Zs Solo controllo della sintassi
/vd{0|1|2} Abilita/disabilita vtordisp /vm<x> Tipo di puntatori ai membri
/Zc:arg1[,arg2] Conformità al linguaggio C++, i cui argomenti possono essere:
forScope[-] Impone C++ standard per le regole di ambito
wchar_t[-] wchar_t è il tipo nativo, non un typedef
auto[-] Impone il nuovo significato di C++ standard per auto
trigraphs[-] Abilita i trigrammi (disattivato per impostazione predefinita)
/ZI Abilita le informazioni di debug di Modifica e continuazione
/openmp Abilita le estensioni del linguaggio OpenMP 2.0
-VARIE-
@<file> Opzioni dei file di risposta

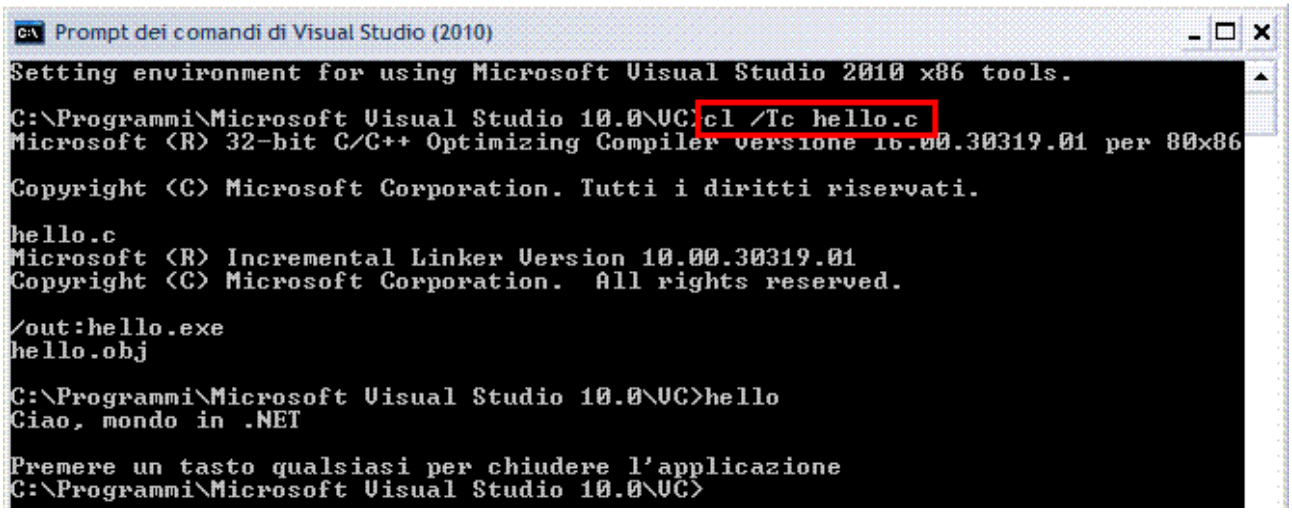
/?, /help Visualizza questo argomento della Guida
/bigobj Genera il formato di oggetto esteso
/c Solo compilazione, nessun collegamento
/errorReport:opzione Segnala a Microsoft gli errori interni del compilatore
none Non invia la segnalazione
prompt Richiede l'invio immediato della segnalazione
queue Al successivo accesso di amministratore, richiede l'invio della segnalazione (predefinita)
send Invia la segnalazione automaticamente
/FC Utilizza i percorsi completi nella diagnostica
/H<num> Lunghezza massima dei nomi esterni
Il tipo char predefinito /J è unsigned
/MP[n] Utilizza fino a 'n' processi per la compilazione
/nologo Non visualizza le informazioni sul copyright
/showIncludes Visualizza i nomi dei file di inclusione
/Tc<file di origine> Compila il file come file .c
/Tp<file di origine> Compila il file come file .cpp
/TC Compila tutti i file come file .c /TP Compila tutti i file come file .cpp
/V<stringa> Imposta la stringa di versione
/w Disabilita tutti gli avvisi /wd<n> Disabilita l'avviso n
/we<n> Considera l'avviso n come un errore
/wo<n> Riporta l'avviso n una sola volta
/w<l><n> Imposta il livello di avviso 1-4 per n
/W<n> Imposta il livello di avviso (valore predefinito n=1)
/Wall Abilita tutti gli avvisi
/WL Anilita le informazioni di diagnostica su una sola riga
/WX Considera gli avvisi come errori /Yc[file] Crea un file PCH
/Yd Inserisce informazioni di debug in ogni OBJ
/Yl[sym] Inserisce riferimenti PCH per la libreria di debug
/Yu[file] Utilizza il file PCH /Y- Disabilita tutte le opzioni PCH
/Zm<n> Massima allocazione di memoria (% del valore predefinito)
/Wp64 Abilita gli avvisi relativi alla portabilità a 64 bit
- COLLEGAMENTO -
/LD Crea un file .DLL /LDd Crea un libreria di debug .DLL
/LN Crea un .netmodule
/F<num> Imposta la dimensione dello stack
/link [librerie e opzioni del linker]
/MD Effettua il collegamento con MSVCRT.LIB
/MT Effettua il collegamento con LIBCMT.LIB
/MDd Effettua il collegamento con la libreria di debug MSVCRTD.LIB
/MTd Effettua il collegamento con la libreria di debug LIBCMTD.LIB
-ANALISI CODICE-
/analyze[:WX-] Abilita l'analisi codice WX- Gli avvisi dell'analisi codice non devono essere considerati errori anche se viene richiamato /WX

APPLICAZIONE CONSOLE C

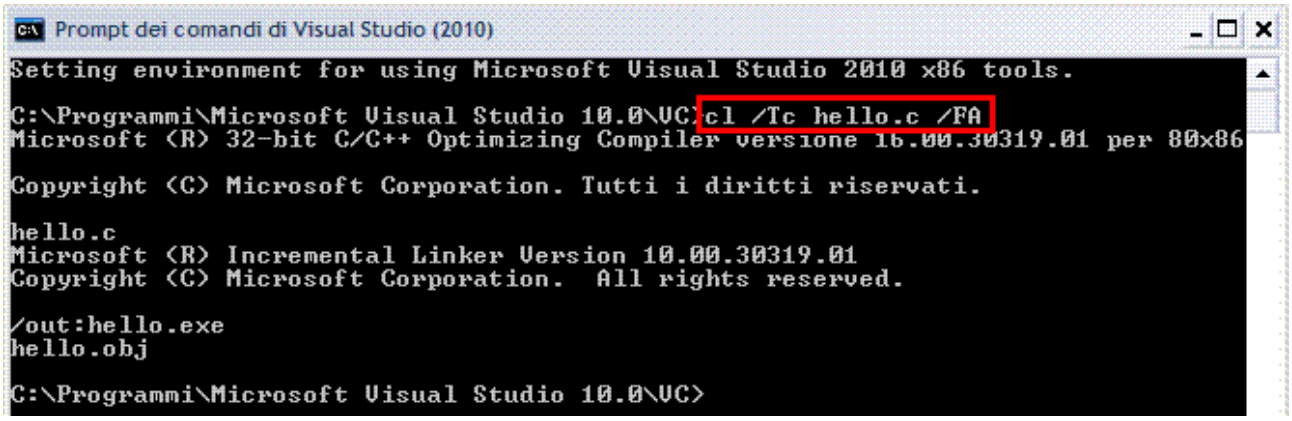
Sono semplici da costruire, non hanno grafica e hanno interfaccia **CUI** (*Character User Interface*) che permette all'utente d'interagire con la tastiera e una finestra.

```
/* Nome dell'applicazione: hello.c
 Programmatore:
 Descrizione: */
#include <stdio.h>
```

```
#include <conio.h>
int main (void)
{       printf ("Ciao, mondo in .NET");
        printf("\n\n");
         printf ("Premere un tasto qualsiasi per chiudere l'applicazione");
        getch();return(0);
}
```



APPLICAZIONE ASSEMBLY



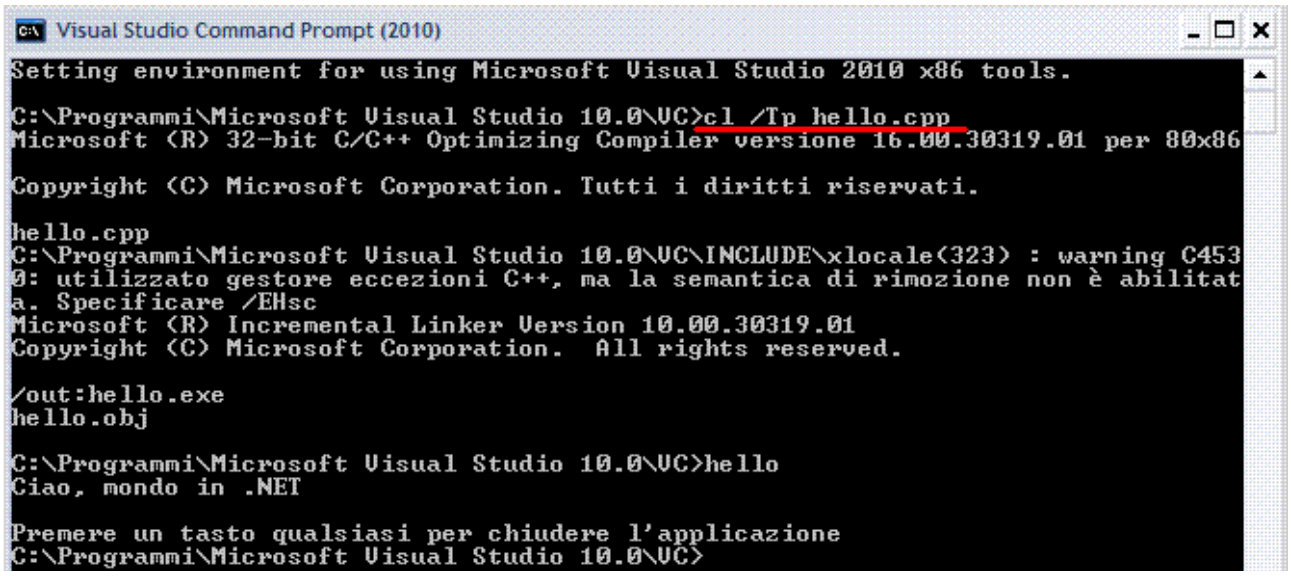
Il compilatore genera il file HELLO.ASM.

```
; Listing generated by Microsoft (R) Optimizing Compiler Version 16.00.30319.01
        TITLE C:\Programmi\Microsoft Visual Studio 10.0\VC\hello.c
        .686P
        .XMM
        include listing.inc
        .model flat
INCLUDELIB LIBCMT
INCLUDELIB OLDNAMES
_DATA   SEGMENT
$SG2898 DB      'Ciao, mondo in .NET', 00H
$SG2899 DB      0aH, 0aH, 00H
        ORG $+1
$SG2900 DB      'Premere un tasto qualsiasi per chiudere l''applicazione', 00H
_DATA   ENDS
```

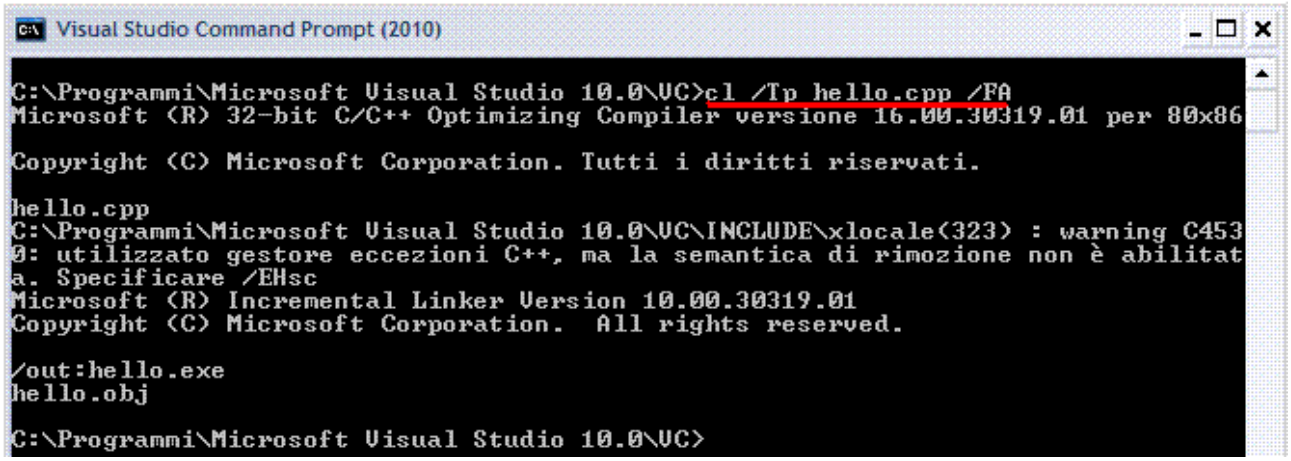
```
PUBLIC	_main
EXTRN	_getch:PROC
EXTRN	_printf:PROC
; Function compile flags: /Odtp
_TEXT	SEGMENT
_main PROC
; File c:\programmi\microsoft visual studio 10.0\vc\hello.c
; Line 7
	push	ebp
	mov	ebp, esp
	push	OFFSET $SG2898
	call	_printf
	add	esp, 4
; Line 8
	push	OFFSET $SG2899
	call	_printf
	add	esp, 4
; Line 9
	push	OFFSET $SG2900
	call	_printf
	add	esp, 4
; Line 10
	call	_getch
	xor	eax, eax
; Line 11
	pop	ebp
	ret	0
_main ENDP
_TEXT	ENDS
END
```

APPLICAZIONE CONSOLE C++

```
// Nome dell'applicazione: hello.cpp
// Programmatore:
// Descrizione:
#include <iostream>
#include <conio.h>
using namespace std;
int main ( void)
{  cout<<"Ciao, mondo in .NET";
   cout<<"\n\n";
   cout<<"Premere un tasto qualsiasi per chiudere l'applicazione";
    getch();return(0);
}
```



APPLICAZIONE ASSEMBLY



Il compilatore genera il file HELLO.ASM.

C++ UNMANAGED E C++ MANAGED

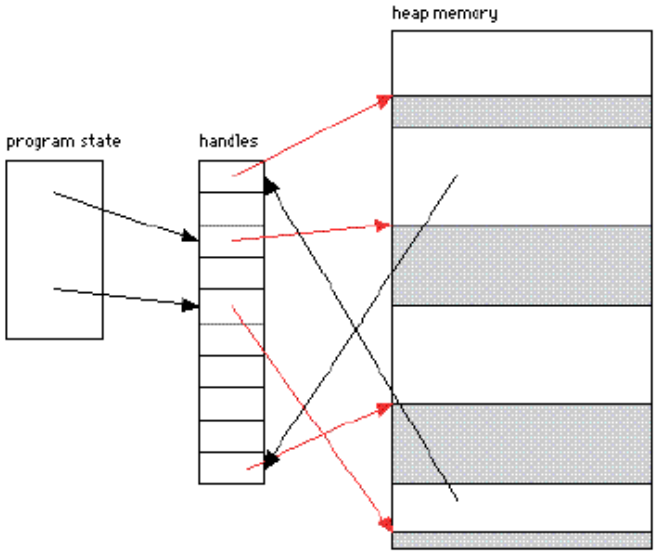
Come riutilizzare codice scritto in C++ nella piattaforma .NET.
Problema: la differenza che sussiste tra l'allocazione esplicita di un oggetto nell'heap rispetto alla presenza di una GC.
La parola chiave *new* assume in C++ un significato diverso rispetto al C# o VB.NET.
In C# ad esempio la creazione di un oggetto è fatta in questo modo.

```
public void method(...)
{ ObjectX obj = new ObjectX();
  obj.m();
}
```

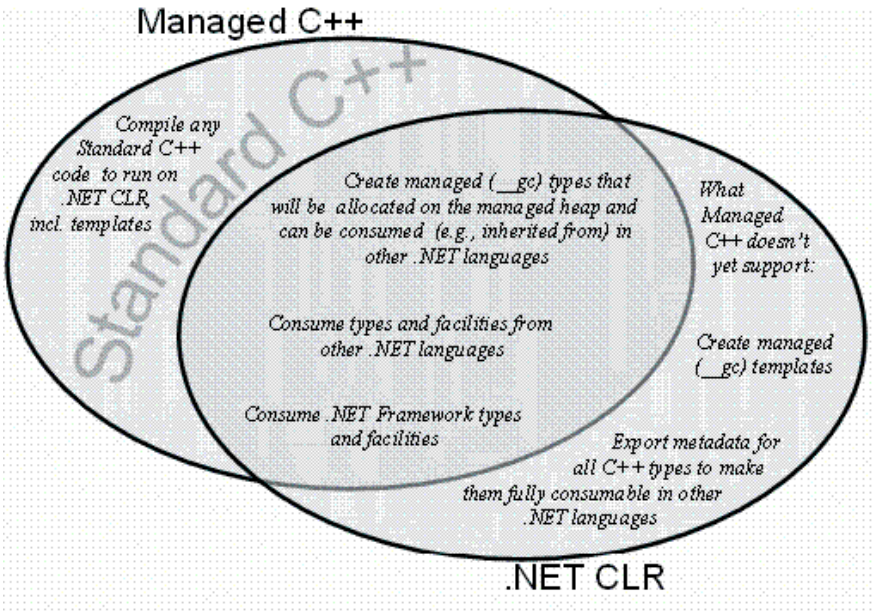
Quando *method* restituirà il controllo al chiamante, di *obj* non ci sarà più traccia.
In realtà quello che succede dietro le quinte è che periodicamente il GC controllerà quali degli oggetti allocati non sono più referenziati e provvederà così a liberare lo spazio di memoria relativo.
In C++ la cosa è più complicata.

```
public void method(...)
{ ObjectX * obj = new ObjectX();
   obj->m();
   delete obj;
}
```

L'operatore *new*, che si occupa di riservare memoria per l'oggetto da allocare, in questo caso restituisce non l'oggetto bensì il puntatore ad esso, o meglio ancora l'indirizzo della memoria heap dove esso è allocato.



Non essendo presente il GC, al momento della restituzione del controllo al chiamante nessuno si occuperà di liberare o meglio deallocare la memoria; per questo motivo esiste un operatore simmetrico a *new* che si chiama *delete* che si occupa di deallocare la memoria precedentemente riservata.
Visual Studio permette di programmare in tutti i linguaggi previsti da .NET e tra questi c'è anche il C++, la doppia esigenza di non perdere i “vecchi programmatori di C/C++” e quella di permettere una “transizione indolore” alla nuova piattaforma .NET ha portato a progettare la seguente architettura.

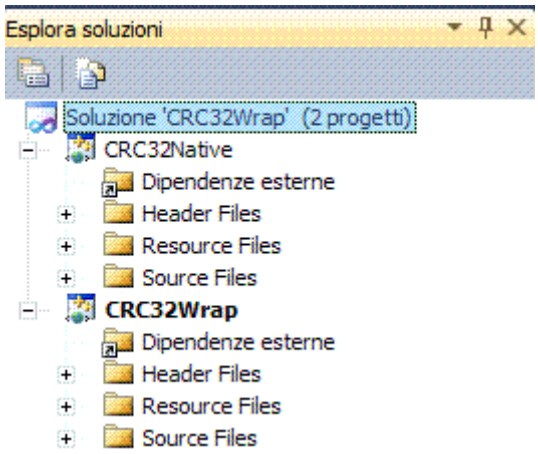


Esempio, calcolo del **CRC**32 (*Cyclic Redundancy Check*) di qualsiasi file.
È un algoritmo di hashing utilizzato per esempio da winzip e dal peer to peer eMule, elabora qualunque quantità di bit e restituisce una stringa di bit di lunghezza fissa e predefinita.
Si tratta di una famiglia di algoritmi che soddisfa i requisiti seguenti.

1. L'algoritmo ritorna una stringa di numeri e lettere a partire dal documento di qualsiasi dimensione che è entrato, tale stringa è detta digest.
2. La stringa è univoca per ogni sequenza di byte e ne è un identificatore, è per questo che l'algoritmo è utilizzabile per la firma digitale.
3. L'algoritmo non è invertibile, ossia non è possibile ricostruire il documento originale a partire dalla stringa che è ritornata in output.

C'è quindi la possibilità di avere digest diversi anche cambiando un solo bit dell'ingresso.

Per prima cosa creare una nuova Soluzione di tipo Visual C++ Libreria di classi che si chiama CRC32Wrap e aggiungere il progetto dove è contenuto il codice nativo, che si chiama, CRC32Native.



File CRC.H

```
#pragma once
#include "stdafx.h"
class CCrc32
{
  public:
```

```
    enum EPolynomials
    {
      polyStandard = 0x04C11DB7,
      polyStandardReversed = 0xEDB88320
    };
  public:
    CCrc32(DWORD dwPoly);
    void PutByte(BYTE byByte)
    {
            unsigned uTop = m_dwRegister >> 24;
            uTop ^= byByte;
            m_dwRegister = (m_dwRegister << 8) ^ m_adwTable[uTop];
    }
    void PutBuffer(BYTE* pbyBuf, DWORD dwBufSize);
    DWORD Done()
    {
      DWORD dwTmp = m_dwRegister;
      m_dwRegister = 0;
      return dwTmp;
    }
  protected:
    DWORD m_dwPoly; // really 33-bits key, counting implicit 1 top-bit
    DWORD m_dwRegister;
    DWORD m_adwTable[256];
};
// This class currently only works with the polynomial used by the
// Dallas Semicondictor 1-wire network protocol
class CCrc8
{
  public:
    CCrc8();
    void PutByte(BYTE byByte);
    BYTE Done()
    {
      BYTE byTmp = m_byRegister;
      m_byRegister = 0;
      return byTmp;
    }
  protected:
    static const BYTE m_byPoly; // really 9-bits key, counting implicit 1 top-bit
    static const BYTE m_abyTable[256];
    BYTE m_byRegister;
};
```

File CRC32NATIVE.H

```
#pragma once
using namespace System;
namespace CRC32Native {
public ref class Class1
{
        // TODO: Add your methods for this class here.
};
}
```

File STDFAX.H

```
#pragma once
typedef unsigned long DWORD;
typedef unsigned char BYTE;
```

File CRC.CPP

```
#pragma once
#include "CRC.h"
CCrc32::CCrc32(DWORD dwPoly)
{
  m_dwPoly = dwPoly;
  m_dwRegister = 0;
  unsigned uScan;
  unsigned uScan2;
        for (uScan = 0; uScan < 256; ++uScan)
        {
                DWORD dwReg = uScan << 24;
                for (uScan2 = 0; uScan2 < 8; ++uScan2)
                {
                        bool bTopBit = ((dwReg & 0x80000000) != 0);
                        dwReg <<= 1;
                        if (bTopBit)
                                dwReg ^= m_dwPoly;
                }
                m_adwTable[uScan] = dwReg;
        }
}
void CCrc32::PutBuffer(BYTE* pbyBuf, DWORD dwBufSize)
{ register BYTE* pbyCurrent = pbyBuf;
  register DWORD dwLeft = dwBufSize;
  register unsigned uTop;
  register DWORD dwR = m_dwRegister;
  while (dwLeft--)
  {
    uTop = dwR >> 24;
          uTop ^= *(pbyCurrent++);
          dwR = (dwR << 8) ^ m_adwTable[uTop];
  }
  m_dwRegister = dwR;
}
const BYTE CCrc8::m_byPoly = 0x31; // actually, 0x131, counting implicit 1 bit 8
const BYTE CCrc8::m_abyTable[256] =
{ 0, 94, 188, 226, 97, 63, 221, 131, 194, 156, 126, 32, 163, 253, 31, 65,
  157, 195, 33, 127, 252, 162, 64, 30, 95, 1, 227, 189, 62, 96, 130, 220,
  35, 125, 159, 193, 66, 28, 254, 160, 225, 191, 93, 3, 128, 222, 60, 98,
  190, 224, 2, 92, 223, 129, 99, 61, 124, 34, 192, 158, 29, 67, 161, 255,
  70, 24, 250, 164, 39, 121, 155, 197, 132, 218, 56, 102, 229, 187, 89, 7,
  219, 133, 103, 57, 186, 228, 6, 88, 25, 71, 165, 251, 120, 38, 196, 154,
  101, 59, 217, 135, 4, 90, 184, 230, 167, 249, 27, 69, 198, 152, 122, 36,
  248, 166, 68, 26, 153, 199, 37, 123, 58, 100, 134, 216, 91, 5, 231, 185,
  140, 210, 48, 110, 237, 179, 81, 15, 78, 16, 242, 172, 47, 113, 147, 205,
  17, 79, 173, 243, 112, 46, 204, 146, 211, 141, 111, 49, 178, 236, 14, 80,
  175, 241, 19, 77, 206, 144, 114, 44, 109, 51, 209, 143, 12, 82, 176, 238,
  50, 108, 142, 208, 83, 13, 239, 177, 240, 174, 76, 18, 145, 207, 45, 115,
```

```
  202, 148, 118, 40, 171, 245, 23, 73, 8, 86, 180, 234, 105, 55, 213, 139,
  87, 9, 235, 181, 54, 104, 138, 212, 149, 203, 41, 119, 244, 170, 72, 22,
  233, 183, 85, 11, 136, 214, 52, 106, 43, 117, 151, 201, 74, 20, 246, 168,
  116, 42, 200, 150, 21, 75, 169, 247, 182, 232, 10, 84, 215, 137, 107, 53
};
CCrc8::CCrc8()
{  m_byRegister = 0;}
void CCrc8::PutByte(BYTE byByte)
{  m_byRegister = m_abyTable[m_byRegister ^ byByte];}
```

Implementare un wrapper, ossia un involucro, che fa da collegamento tra il .NET e il C++ Unmanaged.
Creare la classe wrapper che contenga al proprio interno un puntatore all'oggetto di cui tale classe è wrapper.
Definire i costruttori ed i distruttori in modo tale che il GC sappia trattare correttamente la classe in questione.
Il distruttore *protected*, che inizia con il simbolo *!*, è quello che sarà invocato dal GC quando deciderà di liberare l'heap memory dagli oggetti non più referenziati.

File CRC32WRAP.H

```
#pragma once
#include "..\..\CRC32Native\CRC32Native\CRC.h"
namespace CRC {
public ref class CRC32
	{
	public:
		CRC32(System::UInt32 poly) : m_poCRC32(new CCrc32(poly)){}
		~CRC32()    { delete m_poCRC32; }
		void put(BYTE byByte){list.Add(byByte);};
		System::UInt32 Done();
		static const System::UInt32 directPoly=0x04C11DB7;
		static const System::UInt32 reversePoly=0xEDB88320;
	protected:
		!CRC32()    { delete m_poCRC32; }
	private:
		CCrc32 * m_poCRC32;
		System::Collections::Generic::List<BYTE> list;
	};
}
```

Fornire la classe wrapper di tutti i metodi necessari affinché possa essere correttamente usata da qualsiasi altro linguaggio .NET, ciò implica ad esempio che tale classe dovrà esporre solo i tipi previsti dal Framework stesso, per esempio al posto dei *typedef* usati nel codice nativo *DWORD* diventa *Uint32*, ossia un intero senza segno.

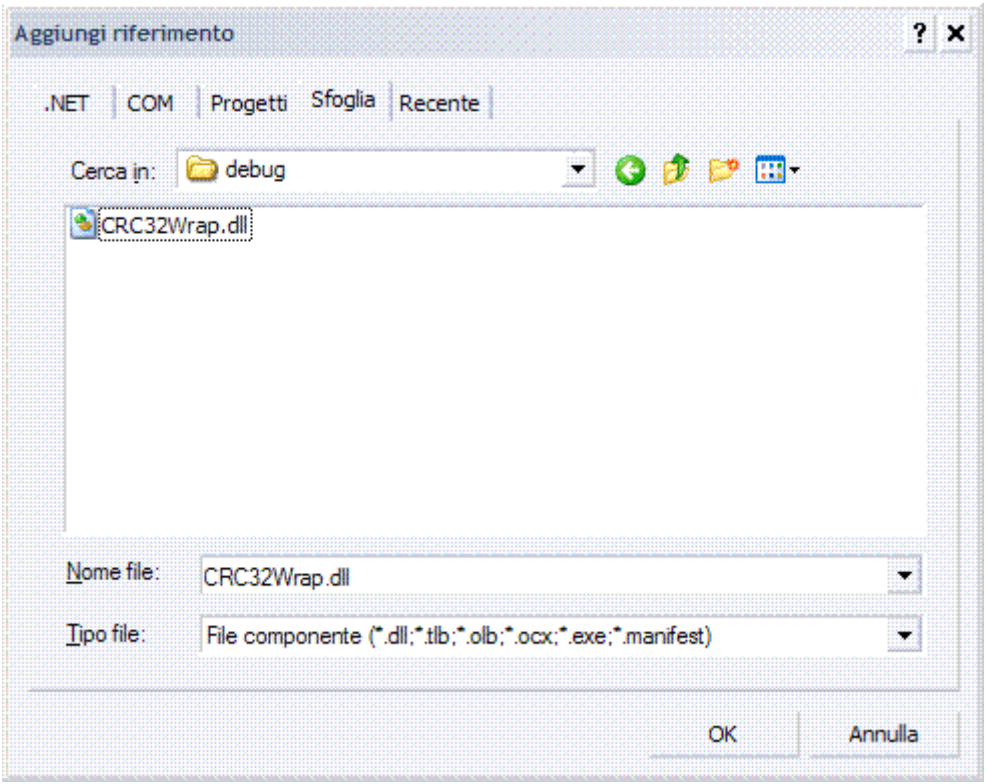
File CRC32WRAP.CPP

```
#include "stdafx.h"
#include "CRC32Wrap.h"
#include "StdAfx.h"
System::UInt32 CRC::CRC32::Done()
{	BYTE * pbyBuffer = new BYTE[list.Count];
	for(int i=0; i<this->list.Count; i++)
	{
		pbyBuffer[i] = this->list[i];
```

```
        }
        this->m_poCRC32->PutBuffer(pbyBuffer,list.Count);
        System::Int32 result = this->m_poCRC32->Done();
        delete[] pbyBuffer;
        return result;
}
```

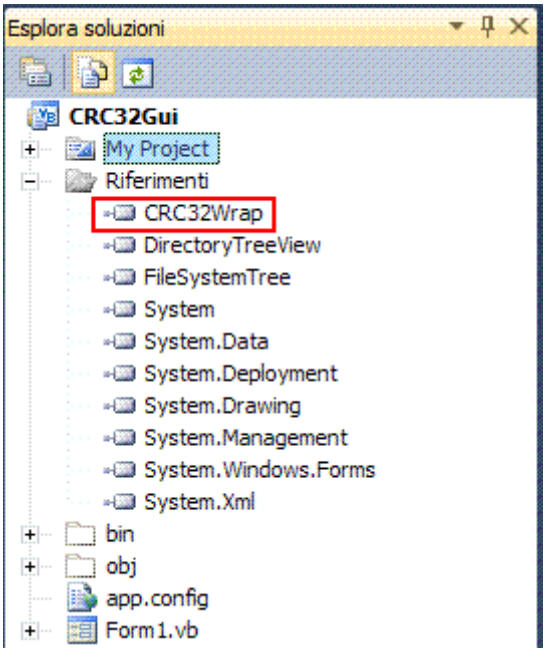
La classe wrap prevede due soli metodi *public*: *put* e *done*.

È possibile usare la libreria in qualsiasi applicazione .NET facendo clic con il pulsante destro su ***Riferimenti/Aggiungi riferimento…***.

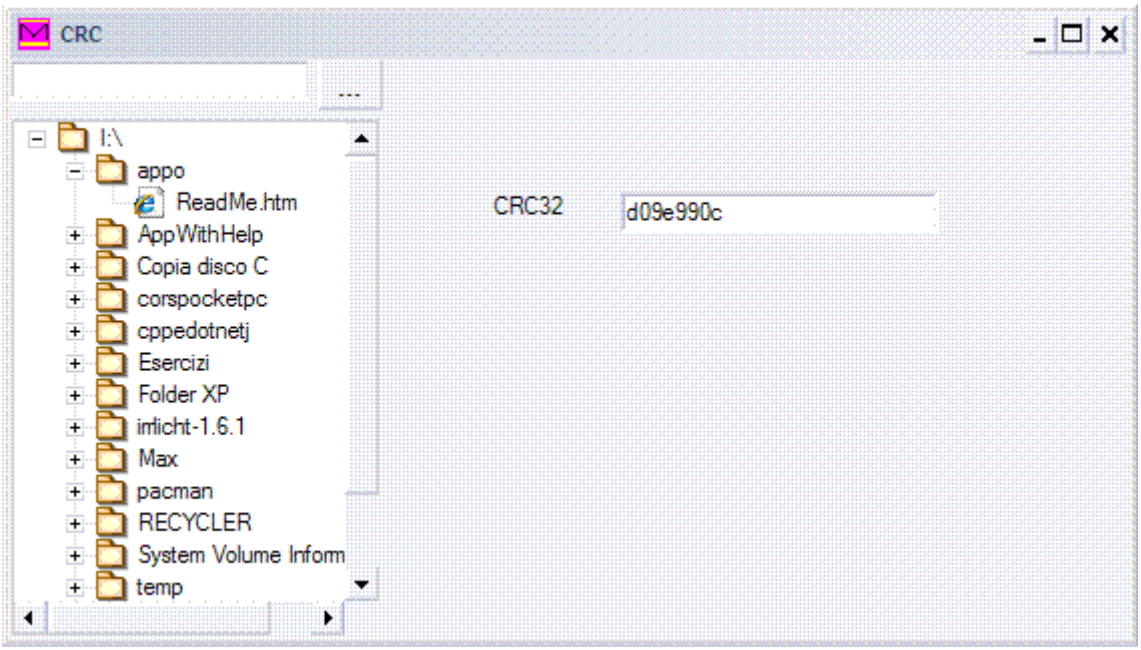


Per esempio, VB.NET cattura un evento di selezione su un componente per il browsing del file system e visualizza l'estratto hashing CRC32 in formato esadecimale, l'applicazione svolge i seguenti compiti.

1. Verificare se il file esiste.
2. Leggere tutti i byte del file.
3. Istanziare la classe CRC32.
4. Riempire il buffer con i byte del file.
5. Chiamare il metodo *done* per ottenere l'estratto hash.

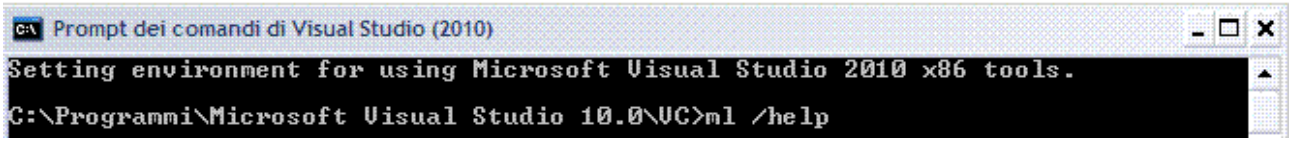



```
Public Class Form1
    Public Sub New()
        InitializeComponent()
        treeView.Load("I:\")
    End Sub
    Private Sub treeView_AfterSelect(ByVal sender As Object, ByVal e As
System.Windows.Forms.TreeViewEventArgs) Handles treeView.AfterSelect
        If IO.File.Exists(treeView.SelectedNode.FullPath) Then
            Dim crc32 As New CRC.CRC32(CRC.CRC32.directPoly)
            Dim contents As Byte() = IO.File.ReadAllBytes(treeView.SelectedNode.FullPath)
            Dim myByte As Byte
            For Each myByte In contents
                crc32.put(myByte)
            Next
            TextBox1.Text = crc32.Done.ToString("x")
        End If
    End Sub
End Class
```


CRC
...
I:\
appo
ReadMe.htm
AppWithHelp
Copia disco C
corspocketpc
cppedotnetj
Esercizi
Folder XP
irrlicht-1.6.1
Max
pacman
RECYCLER
System Volume Inform
temp
CRC32
d09e990c

MASM (*MICROSOFT MACRO ASSEMBLER*)

COMPILAZIONE CLI (*COMMAND LINE INTERFACE*)



Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>ml /help
Microsoft (R) Macro Assembler Version 10.00.30319.01
Copyright (C) Microsoft Corporation. All rights reserved.
ML [/options] filelist [/link linkoptions]
/Bl<linker> Use alternate linker /safeseh Assert all exception
/c Assemble without linking handlers are declared
/Cp Preserve case of user identifiers /Sf Generate first pass listing
/Cu Map all identifiers to upper case /Sl<width> Set line width
/Cx Preserve case in publics, externs /Sn Suppress symbol-table listing
/coff generate COFF format object file /Sp<length> Set page length
/D<name>[=text] Define text macro /Ss<string> Set subtitle
/EP Output preprocessed listing to stdout /St<string> Set title
/F <hex> Set stack size (bytes) /Sx List false conditionals
/Fe<file> Name executable /Ta<file> Assemble non-.ASM file
***/Fl[file] Generate listing** /w Same as /W0 /WX*
***/Fm[file] Generate map** /WX Treat warnings as errors*
/Fo<file> Name object file /W<number> Set warning level
/Fr[file] Generate limited browser info /X Ignore INCLUDE environment path
/FR[file] Generate full browser info /Zd Add line number debug info
***/G<c|d|z> Use Pascal, C, or Stdcall calls** /Zf Make all symbols public*
/I<name> Add include path /Zi Add symbolic debug info
/link <linker options and libraries> /Zm Enable MASM 5.10 compatibility
/nologo Suppress copyright message /Zp[n] Set structure alignment
/omf generate OMF format object file /Zs Perform syntax check only
/Sa Maximize source listing
/errorReport:<option> Report internal assembler errors to Microsoft
none - do not send report
prompt - prompt to immediately send report
queue - at next admin logon, prompt to send report
send - send report automatically

MSIL (*MICROSOFT INTERMEDIATE LANGUAGE*)

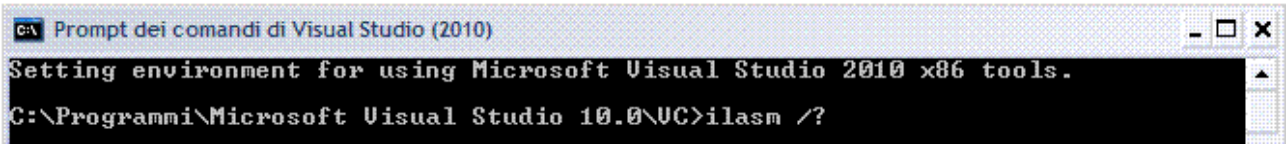
INTRODUZIONE

Il linguaggio MSIL è il formato in cui i compilatori .NET emettono la rappresentazione eseguibile del codice sorgente, il supporto JIT provvede a convertire il codice MSIL in codice eseguibile nativo per la CPU appropriata.
È un linguaggio molto simile all'assembler, rimanendo comunque molto più chiaro e semanticamente più complicato.
Ci sono diverse versioni del compilatore JIT per convertire il codice MSIL in codice nativo.
Vantaggi.

1. Il sistema operativo può fare controlli di consistenza e verifiche di sicurezza sul codice prima di eseguirlo.
2. Il codice intermedio può essere compilato nel modo più opportuno nel momento più indicato per stabilire tutti i parametri di compilazione: ottimizzazioni specifiche per piattaforma.

Svantaggi.
Penalizzazione delle prestazioni.
Tutto questo assomiglia a Java, però ci sono delle differenze.
Un compilatore Java crea bytecode, che in fase di esecuzione è interpretato tramite JVM.
Il .NET Framework crea un codice nativo.
JVM è solo Java, il .NET Framework è multi linguaggio, quelli di Microsoft e altri che possono aggiungersi grazie al *namespace* *System.Reflection.Emit*.

COMPILAZIONE CLI (*COMMAND LINE INTERFACE*)



Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>ilasm
Usage: ilasm [Options] <sourcefile> [Options]
Options:
/NOLOGO Don't type the logo
/QUIET Don't report assembly progress
/NOAUTOINHERIT Disable inheriting from System.Object by default
/DLL Compile to .dll
/EXE Compile to .exe (default)
/PDB Create the PDB file without enabling debug info tracking
/DEBUG Disable JIT optimization, create PDB file, use sequence points from PDB
/DEBUG=IMPL Disable JIT optimization, create PDB file, use implicit sequence points
/DEBUG=OPT Enable JIT optimization, create PDB file, use implicit sequence points
/OPTIMIZE Optimize long instructions to short
/FOLD Fold the identical method bodies into one
/CLOCK Measure and report compilation times
/RESOURCE=<res_file> Link the specified resource file (.res) into resulting .exe or .dll*
/OUTPUT=<targetfile> Compile to file with specified name
(user must provide extension, if any)
/KEY=<keyfile> Compile with strong signature (<keyfile> contains private key)

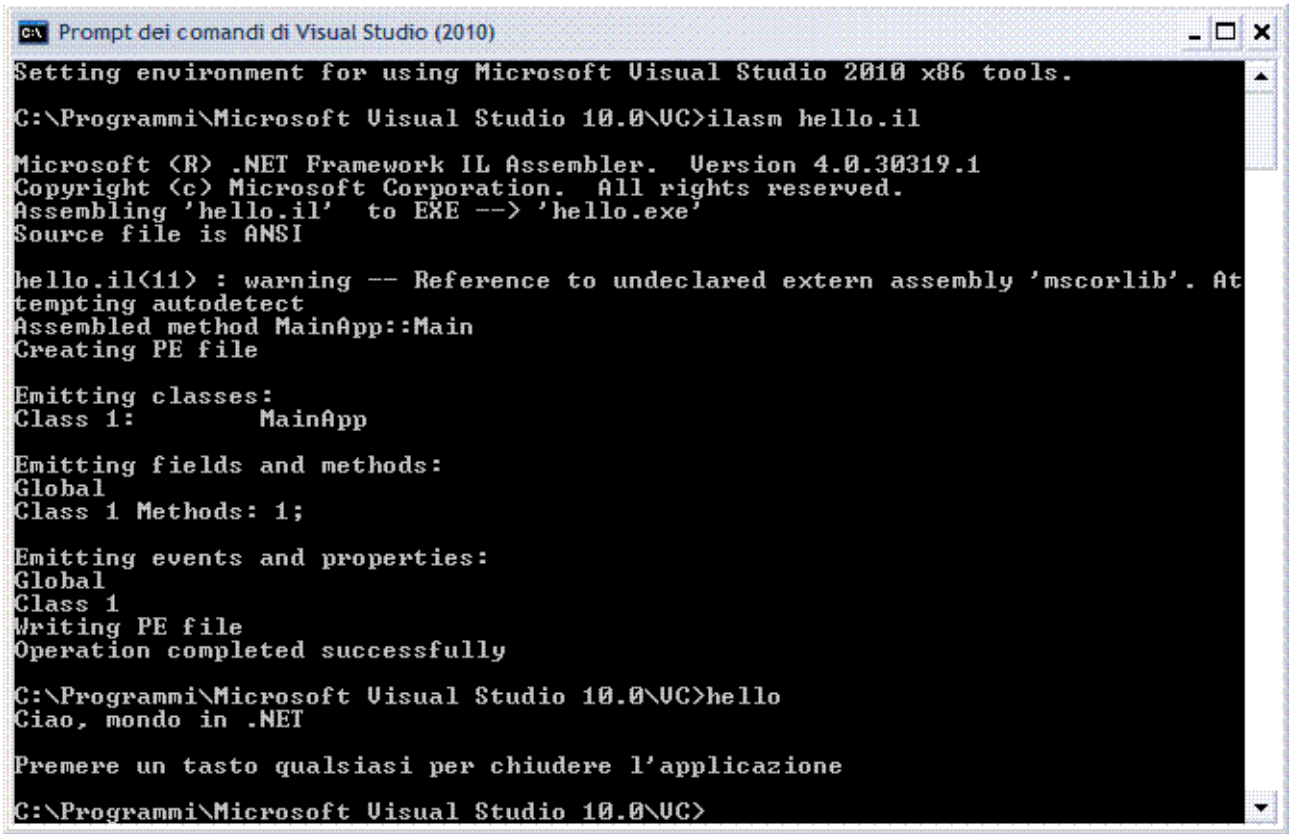
```
/KEY=@<keysource>   Compile with strong signature
                    (<keysource> is the private key source name)
/INCLUDE=<path>     Set path to search for #include'd files
/SUBSYSTEM=<int>    Set Subsystem value in the NT Optional header
/FLAGS=<int>        Set CLR ImageFlags value in the CLR header
/ALIGNMENT=<int>    Set FileAlignment value in the NT Optional header
/BASE=<int>     Set ImageBase value in the NT Optional header
                (max 2GB for 32-bit images)
/STACK=<int>    Set SizeOfStackReserve value in the NT Optional header
/MDV=<version_string>   Set Metadata version string
/MSV=<int>.<int>   Set Metadata stream version (<major>.<minor>)
/PE64           Create a 64bit image (PE32+)
/NOCORSTUB      Suppress generation of CORExeMain stub
/STRIPRELOC     Indicate that no base relocations are needed
/ITANIUM        Target processor: Intel Itanium
/X64            Target processor: 64bit AMD processor
/ENC=<file>     Create Edit-and-Continue deltas from specified source file
Key may be '-' or '/'
Options are recognized by first 3 characters
Default source file extension is .il
Target defaults:
/PE64       => /PE64 /ITANIUM
/ITANIUM    => /PE64 /ITANIUM
/X64        => /PE64 /X64
```

APPLICAZIONE CONSOLE

Sono semplici da costruire, non hanno grafica e hanno interfaccia **CUI** (*Character User Interface*) che permette all'utente d'interagire con la tastiera e una finestra.

```
// Nome dell'applicazione: hello.il
// Programmatore:
// Descrizione:
.assembly hello
{
  .hash algorithm 0x00008004
  .ver 0:0:0:0
}
.module hello.exe
.class private auto ansi beforefieldinit MainApp extends [mscorlib]System.Object
{
.method private hidebysig static void  Main(string[] args) cil managed
 {
  .entrypoint
  // Code size       36 (0x24)
  .maxstack  8
  ldstr      "Ciao, mondo in .NET"
  call       void [mscorlib]System.Console::WriteLine(string)
  call       void [mscorlib]System.Console::WriteLine()
  ldstr      "Premere un tasto qualsiasi per chiudere l'applicazione"
  call       void [mscorlib]System.Console::WriteLine(string)
  call                                        valuetype       [mscorlib]System.ConsoleKeyInfo
[mscorlib]System.Console::ReadKey()
  pop
```

```
  ret
 }
}
```


Prompt dei comandi di Visual Studio (2010)
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>ilasm hello.il
Microsoft (R) .NET Framework IL Assembler. Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.
Assembling 'hello.il' to EXE --> 'hello.exe'
Source file is ANSI
hello.il(11) : warning -- Reference to undeclared extern assembly 'mscorlib'. At
tempting autodetect
Assembled method MainApp::Main
Creating PE file
Emitting classes:
Class 1:   MainApp
Emitting fields and methods:
Global
Class 1 Methods: 1;
Emitting events and properties:
Global
Class 1
Writing PE file
Operation completed successfully
C:\Programmi\Microsoft Visual Studio 10.0\VC>hello
Ciao, mondo in .NET
Premere un tasto qualsiasi per chiudere l'applicazione
C:\Programmi\Microsoft Visual Studio 10.0\VC>

MODULO 2

C#

Linguaggio
Compilazione CLI
Struttura di un’applicazione
Espressioni
Strutture di controllo
Array
Struct
Function/Metodi
Multithread
Remoting
Accesso alle API di Windows
Programmazione OO
Gestione delle eccezioni
Gestione file
Windows Forms

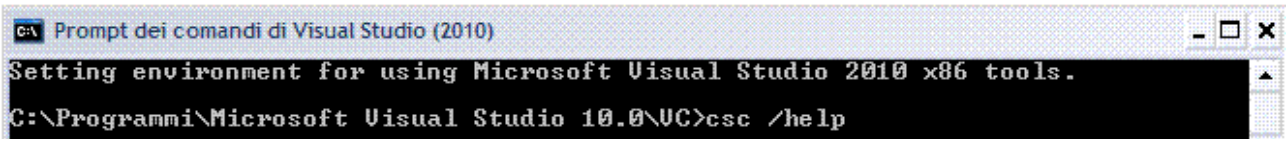
LINGUAGGIO

INTRODUZIONE

Non è il successore di C++.
Non è più potente di VB.NET.
Non produce codice “più veloce” di VB.NET.
Non è il linguaggio “ufficiale” di .NET, ma tutto in .NET è una classe.
Non soffre di difetti imputabili a compatibilità all’indietro.
È case-sensitive e standard ECMA/ISO.
Linguaggio OOP: ha forti analogie con C++ e Java.
Supporta l’uso di puntatori in modalità *unsafe*.
Può essere integrato nelle pagine web.

COMPILAZIONE CLI (*COMMAND LINE INTERFACE*)

INTRODUZIONE



Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>csc /help
Compilatore Microsoft (R) Visual C# 2010 versione 4.0.30319.1
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.

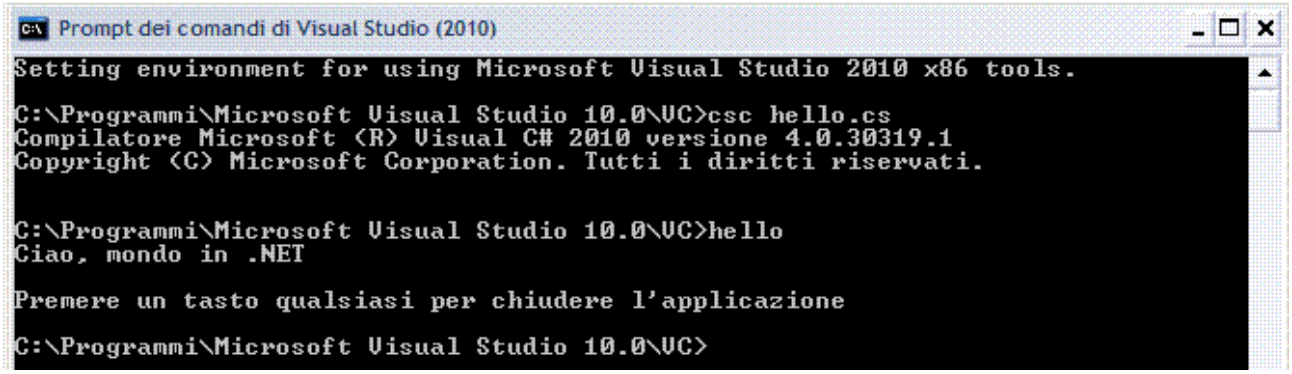
Opzioni del compilatore Visual C# 2010
- FILE DI OUTPUT -
/out:<file> Specificare il nome del file di output (impostazione predefinita: nome base del file con la classe principale o il primo file)
/target:exe Compila un file eseguibile da console (default) (Forma breve: /t:exe)
/target:winexe Compila un file eseguibile Windows (Forma breve: /t:winexe)
/target:library Compila una libreria (Forma breve: /t:library)
/target:module Compila un modulo che può essere aggiunto ad altro assembly. (Forma breve: /t:module)
/delaysign[+|-] Ritarda la firma dell'assembly utilizzando solo la parte pubblica della chiave con nome sicuro
/doc:<file> File XML della documentazione da generare
/keyfile:<file> Specificare un file di chiave con nome sicuro
/keycontainer:<stringa> Specificare un contenitore di chiavi con nome sicuro
/platform:<stringa> Limitare le piattaforme su cui è possibile eseguire questo codice: x86, Itanium, x64 o anycpu. Il valore predefinito è anycpu.
- FILE DI INPUT -
/recurse:<carjolly> Include tutti i file presenti nella directory corrente e nelle relative sottodirectory in base alle specifiche dei caratteri jolly
/reference:<alias>=<file> Crea riferimento a metadati dai file assembly specificati utilizzando l'alias indicato (Forma breve: /r)
/reference:<elenco file> Crea riferimento a metadati dai file assembly specificati (Forma breve: /r)
/addmodule:<elenco file> Collega i moduli specificati nell'assembly
/link:<elenco file> Incorpora metadati dai file di assembly di interoperabilità (Forma breve: /l)
- RISORSE -
/win32res:<file> Specificare un file di risorse Win32 (.res)
/win32icon:<file> Utilizza questa icona per l'output
/win32manifest:<file> Specificare un file manifesto Win32 (XML)
/nowin32manifest Non includere il manifesto Win32 predefinito
/resource:<inf_ris> Incorporare la risorsa specificata (Forma breve: /res)
/linkresource:<inf_ris> Collegare la risorsa specificata all'assembly. (Forma breve: /linkres)
Dove il formato di resinfo è <file>[,<nome stringa>[,public|private]]
- GENERAZIONE CODICE -
/debug[+|-] Crea informazioni di debug.
/debug:{full|pdbonly} Specificare il tipo di debug ('full' è

l'impostazione predefinita e consente di
associare un debugger a un programma in esecuzione)
/optimize[+|-] Abilita ottimizzazioni (Forma breve: /o)
- ERRORI E AVVISI -
/warnaserror[+|-] Segnala tutti gli avvisi come errori
/warnaserror[+|-]:<elenco avvisi>
Segnala avvisi specifici come errori
/warn:<n> Imposta livello avvisi (0-4) (Forma breve: /w)
/nowarn:<elenco avvisi> Disabilita messaggi di avviso specifici
- LINGUAGGIO -
/checked[+|-] Genera controlli dell'overflow
/unsafe[+|-] Ammetti codice 'unsafe'
/define:<elenco simboli> Definisci simboli di compilazione condizionale
(Forma breve: /d)
/langversion:<stringa> Specificare la modalità della versione di lingua:
ISO-1, ISO-2, 3 o Default
- VARIE -
@<file> Legge il file di risposta per ulteriori opzioni
/help Visualizza questo messaggio relativo all'uso (Forma breve: /?)
/nologo Non visualizza il messaggio di copyright del compilatore
/noconfig Non includere automaticamente il file CSC.RSP
- AVANZATE -
/baseaddress:<indirizzo> Indirizzo di base della libreria da compilare
/bugreport:<file> Creare un file di report sui bug
/codepage:<n> Specificare la tabella codici da utilizzare per
l'apertura dei file di origine.
/utf8output Restituisci messaggi del compilatore usando la
codifica UTF-8
/main:<tipo> Specificare il tipo che contiene il punto di
ingresso, ignorando tutti gli altri punti di
ingresso possibili. (Forma breve: /m)
/fullpaths Il compilatore genera percorsi completi
/filealign:<n> Specificare l'allineamento utilizzato per le
sezioni del file di output
/pdb:<file> Specificare il nome file delle informazioni di
debug (impostazione predefinita: nome file di
output con estensione .pdb)
/nostdlib[+|-] Ometti riferimenti a libreria standard (mscorlib.dll)
/lib:<elenco file> Specificare directory aggiuntive in cui cercare i riferimenti
/errorreport:<stringa> Specificare come gestire gli errori interni del
compilatore: prompt, send, queue o none.
L'impostazione predefinita è queue.
/appconfig:<file> Specificare un file di configurazione
dell'applicazione contenente le impostazioni di
associazione dell'assembly
/moduleassemblyname:<stringa> Nome dell'assembly di cui farà parte questo modulo

APPLICAZIONE CONSOLE

Sono semplici da costruire, non hanno grafica e hanno interfaccia **CUI** (*Character User Interface*) che permette all'utente d'interagire con la tastiera e una finestra.

```
// Nome dell'applicazione: hello.cs
// Programmatore:
// Descrizione:
using System;
   class hello
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Ciao, mondo in .NET");
         Console.WriteLine();
        Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione");
       Console.ReadKey();
      }
   }
```



STRUTTURA DI UN'APPLICAZIONE

INTRODUZIONE

```
using SpazioDiNomi;
namespace UnNamespace
{       class MiaClasse
        {...}
        struct MiaStruct
        {...}
        interface ImiaInterfaccia
        {...}
        delegate tipo MioDelegate();
        enum MiaEnum
        {...}
        namespace UnAltroNamespace
        { ... }
        class ClassePrincipale
        {       public static void Main(string[] args)
        {...}
        }
}
```

Tutto deve essere contenuto in una classe, *hello*, dichiarata con la parola chiave *class* che contiene il metodo *Main*, è il metodo di avvio dell'applicazione, deve essere *static* in quanto deve poter essere usato prima che sia creata un'istanza della classe che lo contiene.
Tutti le applicazioni in C#, comprese quelle per Windows, devono avere una e solo una classe con all'interno un metodo *Main*, altrimenti in fase di run-time si otterrà un messaggio di errore.
Tutti i tipi reference, sono derivati dal tipo base *System.Object*.
Ogni oggetto eredita i metodi base da *System.Object*, mediante il meccanismo di boxing.
La classe *System.Object* offre i seguenti metodi.

- ✓ *Equals (public, virtual):* permette la comparazione di due oggetti per stabilire se il loro valore è uguale.
- ✓ *GetHashCode (public, virtual):* permette ai tipi di restituire un integer a 32 bit con segno come codice hash per i suoi oggetti,.di solito è utilizzato per memorizzare oggetti in una tabella hash.
- ✓ *ToString (public, virtual):* permette ad un tipo di fornire in ritorno una stringa che rappresenta il valore dell'oggetto.
- ✓ *GetType (public, non-virtual):* restituisce un oggetto che rappresenta il tipo dell'oggetto.
- ✓ *MemberwiseClone (protected, non-virtual):* permette al tipo di costruire una nuova istanza che è copia di se stesso.
- ✓ *Finalize (protected, virtual):* permette di ripulire dalla memoria gli oggetti del tipo e rilasciare le risorse associate.

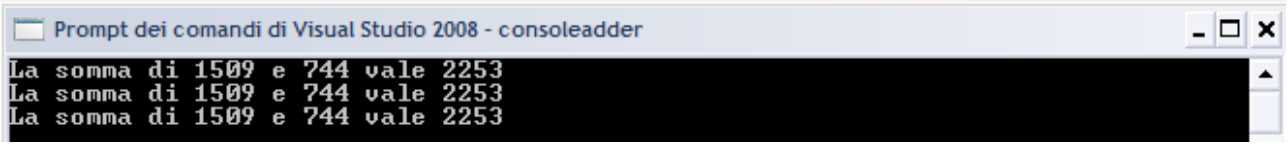
INPUT/OUTPUT

La libreria di classi del .NET Framework contiene la classe *System.Console* e per utilizzarla basta scrivere il suo nome, altrimenti il compilatore non la troverebbe.
Il metodo statico *WriteLine* stampa una riga sulla console e va a capo; il metodo statico

ReadLine legge dalla console.
La classe *Console* si trova nel *namespace System* contenuto nell'assembly *mscorlib.dll,* quindi la prima linea del programma, *using System*, permette di utilizzare la classe *Console* all'interno di *Main*.

Esempio: stampa di numeri.

```
using System;
class ConsoleAdder
{       public static void Main()
        {int a = 1509;                  // commento su una linea
                                        /* commento
                                        su due linee */
        int b = 744;
        int c = a + b;
        Console.Clear();
        Console.Write("La somma di ");
        Console.Write(a);
        Console.Write(" e ");
        Console.Write(b);
        Console.Write(" vale ");
        Console.WriteLine(c);
        Console.WriteLine("La somma di " + a + " e " + b + " vale " + c);
        // al posto dei segnaposto {0}, {1}, {2} contenuti nella stringa, si sostituiscono
        // i valori specificati nella lista separati dalla virgola
        Console.WriteLine("La somma di {0} e {1} vale {2}", a, b, c);
        Console.ReadKey();
        }
}
```



Esempio: tipo di piattaforma hardware.

```
using System;
class cpu
{public static void Main()
{ Console.Clear();
Console.WriteLine("Questa applicazione è compilata per essere eseguita da tutte le
CPU.");
Console.WriteLine("Controlla IntPtr.Size per vedere se l'applicazione gira a 32 bit o 64
bit.");
Console.WriteLine("IntPtr.Size = " + IntPtr.Size);
Console.WriteLine();
if (IntPtr.Size == 4)
        Console.WriteLine("Questa applicazione gira a a 32 bit.");
        if (IntPtr.Size == 8) Console.WriteLine("Questa applicazione gira a a 64 bit.!!!");
Console.ReadKey();
}
}
```

Formattazione
Nei segnaposti del tipo *{n}* possono essere aggiunte informazioni sull'allineamento ed il formato da utilizzare nel modo seguente.
{indice[,allineamento][:stringaDiFormato]}

Stringa di formato	Descrizione
C (Currency)	Il numero è convertito in una stringa che rappresenta una valuta nel formato locale.
D (Decimal)	Formato decimale, converte in base 10 e aggiunge degli zeri all'inizio se è specificata la precisione.
E (Exponential)	Formato scientifico o esponenziale. La precisione indica il numero di cifre decimali, 6 per default, mentre il simbolo esponenziale può essere maiuscolo o minuscolo.
F (Fixed-point)	Formato fixed-point, con la precisione che indica il numero di cifre decimali.
G (General)	Formato generale, il numero viene convertito in formato F oppure E, a seconda di quale dei due è il più compatto.
N (Number)	Il numero è convertito in una stringa con separatori delle migliaia e separatore decimale, ad es. 32,768.00.
P (Percent)	Formato percentuale.
X (Hexadecimal)	Il valore numerico è convertito in esadecimale, aggiungendo degli zeri se viene specificata la precisione.

```
using System;
class ConsoleAdder
{       public static void Main()
        {double d=123456.789;
        Console.Clear();
        Console.WriteLine("{0:C2}",d);                  // € 123.456,79
        Console.WriteLine("{0:F1}",d);                  // 123456,8
        Console.WriteLine("{0:E}",d);                   // 1,234568E+005
        Console.WriteLine("{0:X}",7123);                // 1BD3
        int i=10000;
        int j=234;
        Console.WriteLine("{0,10:D}+\n{1,10:D}=",i,j);  // 10000 + 234=
        Console.WriteLine("{0,10:D}",i+j);              // 10234
        Console.ReadKey();
        }
}
```

ORGANIZZARE LE CLASSI

namespace
Permette l'organizzazione gerarchica del codice, in pratica specifica in modo completo il nome di una classe in esso contenuta, separando i nomi dei namespace con un "." e terminando con il nome della classe stessa.

È buona regola di programmazione raggruppare classi che si riferiscono o che forniscono funzionalità correlate.
Nel .NET Framework ogni tipo, come classi, interfacce, strutture, enumerazioni e delegati, fa parte di uno spazio di nomi: *namespace*.
La parola chiave *namespace*, chiede al compilatore di includere nel namespace tutte le classi che saranno dichiarate in seguito.

Esempio di tre *namespace* annidati.

```
using System;
namespace pippo
{
    namespace pluto
    {
        namespace capitolo1
        {
          class hello
                { static void Main(string[] args)
                    { Console.WriteLine("Ciao, mondo in .NET");
                      Console.ReadKey();
                    }
                }
        }
    }
}
```

Esempio di tre *namespace* annidati in modo compatto.

```
using System;
namespace pippo.pluto.capitolo1
{
          class hello
                { static void Main(string[] args)
                    { Console.WriteLine("Ciao, mondo in .NET");
                      Console.ReadKey();
                    }
                }
}
```

I *namespace* definiscono uno spazio di dichiarazione, ma di tipo **open ended** perché un *namespace* può estendere il suo spazio su più file, questo si chiama *global namespace*.

Esempio.

```
// file classe1.cs
namespace pippo
{
Classe1
{
…
}
}
```

```
// file classe2.cs
namespace pippo
{
Classe2
{
…
}
}
```

using

Usata per evitare il nome completo della classe all'interno del codice, per esempio la classe *Console* è contenuta nel *namespace System* e senza usare l'istruzione:
using System;
si deve scrivere:

System.Console.WriteLine(“Ciao, mondo in .NET”);
Questo modo di programmare è comodo quando si hanno *namespace* annidati e quindi con nomi delle classi molto lunghi.
Può capitare che i nomi di due classi coincidano in un progetto.
```
// una libreria per gestire una anagrafica clienti in Italia
public class IndirizzoPostale
{...}
// una libreria per gestire una anagrafica clienti negli USA
public class IndirizzoPostale
{...}
```
Ci sono due classi che hanno lo stesso nome, ma risiedono in due librerie diverse.
Le due classi rappresentano indirizzi diversi, ma è impossibile includerle nella stessa applicazione perché hanno lo stesso nome.
Prima soluzione: mettere davanti al nome un prefisso: questo, però, porta a dei nomi di classi molto lunghi e rende l’applicazione meno leggibile.
```
//Italia
public class ItaliaIndirizzoPostale
 ...}
//USA
public class USAIndirizzoPostale
{...}
```
Seconda soluzione: i *namespace* risolvono il problema della collisione dei nomi di classe con una tecnica simile, ma più strutturata.
Permettono di definire uno spazio di nomi, in pratica un altro livello di visibilità (scope) in cui definire dei tipi, da dichiarare con la parola chiave *namespace*.
All'interno di uno spazio di nomi sono valide solo le dichiarazioni di tipi e di altri *namespace*, mentre i nomi di questi tipi sono unici solo all'interno dei rispettivi spazi.
I *namespace* sono quindi utili per evitare collisioni tra nomi di classe.
Hanno due caratteristiche importanti.
1. Possono essere nidificati.
2. Diversamente dalle classi, sono spazi di visibilità aperti.

```
public void miaFunzione
{
   //utilizzo la libreria A
   Italia.IndirizzoPostale.Controlla("Via del Giglio,9 00182 Roma");
   //utilizzo la B
   USA.IndirizzoPostale.Controlla("9, Rose Av. 00-143 Sacramento, CA");
}
namespace Italia
{
   public class IndirizzoPostale
   {...}
}
```
Nell’esempio si può associare due *namespace* rispettivamente alla libreria A e alla libreria B, utilizzando la classe *IndirizzoPostale* senza problemi.
Un nome di classe insieme al nome del suo *namespace* è detto nome completamente qualificato.
Il codice precedente potrebbe così assumere un nuovo aspetto, sia che si ricorra alla forma completa:
```
namespace Anagrafica
{
   namespace Italia
   {
      public class IndirizzoPostale {...}
```

```
    }
}
```
sia che si ricorra alla forma abbreviata:
```
namespace Anagrafica.Italia
{
    public class IndirizzoPostale {...}
}
```
I *namespace* sono un nome condiviso all'interno di un'applicazione.
Librerie scritte da diversi autori possono definire un *namespace* uguale; se questo avviene le classi definite all'interno di queste librerie apparterranno allo stesso *namespace*.
Per utilizzare le classi si deve qualificarle con il loro *namespace*; come nel caso dei prefissi, questa tecnica porta alla scrittura di nomi troppo lunghi.
Si può utilizzare la parola chiave *using* per includere un *namespace* nello spazio di visibilità corrente.
```
// questo codice mostra una situazione in cui non è possibile risolvere in maniera certa il
// nome della classe IndirizzoPostale*/
using Anagrafica.Italia;
using Anagrafica.USA;
...
public void controllaIndirizzo()
{
   //errore!!! Quale classe voglio indicare????
   IndirizzoPostale.Controlla("Via del Giglio,9 00182 Roma");
   ...
}
```
Grazie a *using*, il compilatore cercherà i nomi di classe all'interno dei *namespace* indicati, ovviamente non ci devono essere ambiguità.
Gli spazi di nomi permettono di creare una struttura logica per le classi.
I *namespace* sono un meccanismo che è sfruttato dal compilatore e non durante l'esecuzione del programma.
Le classi .NET sono distribuite attraverso gli assembly, durante l'esecuzione le classi appartenenti ad un certo *namespace* sono cercate dal CLR all'interno di uno o più assembly; generalmente c'è corrispondenza tra la struttura logica (il namespace) e quella fisica (il nome di un assembly).

ESPRESSIONI

IDENTIFICATORI VERBATIM

Costituiti dal carattere @ seguito da un identificatore oppure da una parola chiave del C#, ad esempio: *@if*; è buona regola di programmazione non usarli.
Sono utili solo in un caso, il .NET Framework consente un'interoperabilità fra linguaggi diversi per cui potrebbe capitare che un identificatore usato in un'applicazione, potrebbe essere una parola chiave in un altro linguaggio.

VARIABILE DI CLASSE

È una variabile dichiarata all'interno del corpo di una classe, è chiamata anche campo o variabile membro.
Un campo di una classe estende il suo scope su tutta la classe e non importa il punto in cui è dichiarata.

Esempio.

```
using System;
public class Uomo
{
// il metodo usa il campo fTemperatura dichiarato sotto
  public void StampaTemperatura()
    {
      Console.WriteLine("Temperatura="+fTemperatura);
    }
static void Main()
{       Console.Clear();
        Uomo ex=new Uomo();
        ex.StampaTemperatura();
        Console.ReadKey();
}
float fTemperatura=37.7f;
}
```

ACCESSIBILITÀ

I tipi definiti dall'utente, *classi, strutture, enum*, i membri di classi e strutture possono avere accessibilità diversa.

- ✓ *public* Accessibili a tutti.
- ✓ *protected* Solo alle classi derivate.
- ✓ *private* Nessuno (solo se stessa).
- ✓ *internal* Tutte le classi dello stesso assembly.
- ✓ *protected internal* Combinazione dei due.

Differenziare l'accessibilità di un membro è fondamentale per realizzare "encapsulation".

TIPI VALORE

Discendono dalla classe VT che a sua volta deriva dalla classe *Object*.
In C# è possibile creare nuovi tipi valore di due categorie.

1. Tipo *struct*.
2. Tipo enumerazione.

C# fornisce un insieme di tipi *struct* predefiniti che sono i tipi semplici, che sono identificati da parole chiave che sono alias per i tipi *struct* predefiniti contenuti del *namespace System*

e definiti nel CTS che è il sistema di tipi comune a tutti i linguaggi del .NET Framework.
Tipi semplici predefiniti.

Tipo semplice	Tipo CTS	CLS Compliant
sbyte	System.SByte	No
byte	System.Byte	Sì
short	System.Int16	Sì
ushort	System.UInt16	No
int	System.Int32	Sì
uint	System.UInt32	No
long	System.Int64	Sì
ulong	System.UInt64	No
char	System.Char	Sì
float	System.Single	Sì
double	System.Double	Sì
bool	System.Boolean	Sì
decimal	System.Decimal	Sì

Dichiarati nel namespace *System:*
- ✓ boolean, Byte, **Sbyte**, Char
- ✓ Int16, Int32, Int64, **UInt16**, **UInt32**, **UInt64**
- ✓ Decimal, Single, Double
- ✓ DateTime, TimeSpan
- ✓ String.

C# dichiara alcuni alias per questi tipi:
- ✓ bool (Boolean), byte (Byte), sbyte (**Sbyte**), char (Char)
- ✓ short (Int16), int (Int32), long (Int64)
- ✓ ushort (**UInt16**), uint (**UInt32**), ulong (**UInt64**)
- ✓ decimal (Decimal), float (Single), double (Double)
- ✓ Nessun alias per DateTime e TimeSpan
- ✓ string (String).

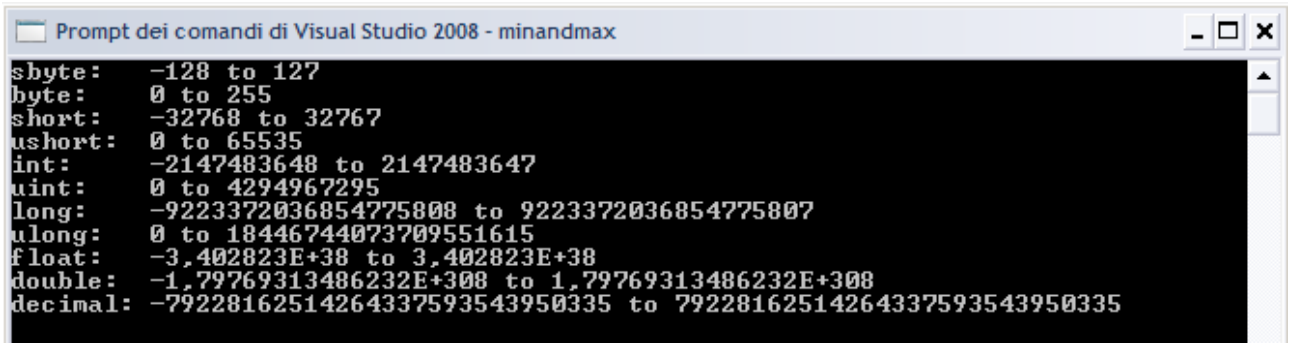
In grassetto i tipi che **non** sono conformi alle specifiche CLS.
Ad esempio, il tipo *int* possiede i membri del corrispettivo *SystemInt32* oltre a quelli della super classe *System.Object*; per ricavare il massimo e il minimo di una variabile numerica si usa la proprietà seguente.

```
int maxIntero=int.MaxValue;                 // si usa l'alias int
int minIntero=System.Int32.MinValue;        // si usa il nome completo
```

Esempio: stampare il valore minimo e massimo dei semplici.

```
using System;
class MinAndMax
{       public static void Main()
        {Console.Clear();
        Console.WriteLine("sbyte:   {0} to {1}", sbyte.MinValue,sbyte.MaxValue);
        Console.WriteLine("byte:    {0} to {1}", byte.MinValue,byte.MaxValue);
        Console.WriteLine("short:   {0} to {1}", short.MinValue,short.MaxValue);
        Console.WriteLine("ushort:  {0} to {1}", ushort.MinValue,ushort.MaxValue);
        Console.WriteLine("int:     {0} to {1}", int.MinValue, int.MaxValue);
        Console.WriteLine("uint:    {0} to {1}", uint.MinValue,uint.MaxValue);
        Console.WriteLine("long:    {0} to {1}", long.MinValue,long.MaxValue);
        Console.WriteLine("ulong:   {0} to {1}", ulong.MinValue,ulong.MaxValue);
        Console.WriteLine("float:   {0} to {1}", float.MinValue,float.MaxValue);
```

```
        Console.WriteLine("double:  {0} to {1}", double.MinValue, double.MaxValue);
        Console.WriteLine("decimal: {0} to {1}", decimal.MinValue,decimal.MaxValue);
        Console.ReadKey();
    }
}
```



Tipo enumerativo

Un'enumerazione è un insieme di valori tra di loro in relazione, ad esempio i giorni della settimana.

```
public enum Giorni
{ Lunedì=1,
  Martedì=2,
  Mercoledì=3,
  Giovedì=4,
  Venerdì=5,
  Sabato=6,
  Domenica=7
}
Giorni g=Giorni.Lunedì;          // restituisce il membro di valore 1
string s=g.ToString();           //s vale "Lunedì"
```

La classe *System.Enum* fornisce, per esempio, il metodo *Enum.GetNames* ritorna tutti i membri dell'enumerazione; il metodo *Enum.GetValues* permette di conoscere i valori.

```
foreach(string s in Enum.GetNames(typeof(Giorni)))
Console.WriteLine(s);
foreach(int i in Enum.GetValues(typeof(Giorni)))
Console.WriteLine(i);
```

LITERAL

È un suffisso e un prefisso che indica al compilatore il tipo del valore numerico.

```
int i=0xFFFF;            // valore esadecimale prefisso 0x
```

Un carattere che segue un valore numerico, indica il tipo da usare per gestire il valore stesso.

"L" o "l" indica un *long*.
"F" o "f" indica un *float*.
"D" o "d" indica un *double*.
"U" o "u" indica un *unsigned*.

La "U" o "u" può essere usata da sola , in questo caso indica un valore intero senza segno *uint*; oppure in combinazione con "L" o "l", ad esempio "UL" o "LU" ed indica un valore *ulong*.

```
long l=10L               // il suffisso L o l indica un long
float f=10f;             // f o F indica float
```

Per i tipi booleani sono definiti i due literal "true" e "false".

TIPI DI RIFERIMENTO

Sono tipi le cui istanze sono create nello heap, C# fornisce due tipi di riferimento primitivi.

1. Tipo *object*.
2. Tipo *string*.

Tipo *object*

È il padre di tutti i tipi, primitivi e non: tutti i tipi del .NET Framework derivano dalla classe *System.Object*, è la classe da cui implicitamente ogni altra deriva, se non è specificata nessuna altra classe; la parola chiave *object* è un alias per *System.Object*.
Questa classe fornisce dei metodi che ogni altra classe eredita e che può eventualmente ridefinire con un override.

Metodo	Descrizione
`public virtual string ToString()`	Restituisce una stringa che rappresenta l'oggetto.
`public virtual int GetHashCode()`	Restituisce un intero, utilizzabile come valore di hash, ad per la ricerca dell'oggetto in un elenco di oggetti.
`public virtual bool Equals(object o)`	Effettua un test di uguaglianza con un'altra istanza della classe.
`public static bool Equals(object a,object b)`	Effettua un test di uguaglianza fra due istanze della classe.
`public static bool ReferenceEquals(object a,object b)`	Effettua un test di uguaglianza per verificare se due riferimenti si riferiscono alla stessa istanza della classe.
`public Type GetType()`	Restituisce un oggetto derivato da System.Type che rappresenta il tipo dell'istanza.
`protected object MemberwiseClone()`	Effettua una copia dei dati contenuti nell'oggetto, creando un'altra istanza.
`protected virtual void Finalize()`	Distruttore dell'istanza.

Tipo *string*

Per trattare sequenze di caratteri racchiuse tra doppi apici ("…").
string s="ciao";
Da questa assegnazione è creato un oggetto *System.String*, allocato sullo heap.
string s1="s1";
string s2=s1;
Console.WriteLine(s1);
Console.WriteLine(s2);
Sono stampate due stringhe uguali perché *s1* e *s2* puntano allo stesso oggetto nello heap.
string s2="s2";
Console.WriteLine(s1);
Console.WriteLine(s2);
Essendo string un tipo di riferimento, puntando *s1* e *s2* allo stesso oggetto il cambiamento di *s2* deve riflettersi anche su *s1*.
Invece, sono stampati due valori diversi "s1" e "s2", perché quando si cambia il valore di una stringa è creato un oggetto nuovo sullo heap, quindi *s1* punta ancora al valore "s1", mentre s2 punta al nuovo oggetto "s2" tutto questo perché ogni stringa è una sequenza di caratteri immutabili e quindi non modificabili.

CONVERSIONI DI TIPO

Si converte una variabile da un tipo ad un altro.

Conversioni implicite

È trasparente e quindi è garantito che il valore da convertire non subirà perdita di dati.

Da	A
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double

Conversioni implicite

Operazione di **cast**.

È l'operazione di conversione di un tipo ad un altro, in caso d'incompatibilità è lanciata un'eccezione di tipo *InvalidcastException*.

```
{ Dog b = (Dog)a;                    // cast
  // … }
catch (InvalidcastException err)
{  // …  }
```

boxing e unboxing

Consentono di convertire un qualsiasi tipo valore nel tipo *object*, quindi in un tipo riferimento (e viceversa): è un'operazione costosa perché la memoria del VT deve essere copiata nello heap e viceversa.

```
int i=123;              // tipo valore i sullo stack
object box=i;           // tipo object box sullo heap: boxing
int n=(int)box;         // unboxing
```

Ad esempio, è possibile ottenere la stringa di *i* con il metodo seguente.

```
int i=123;
string str=i.ToString();
```

Esempio.

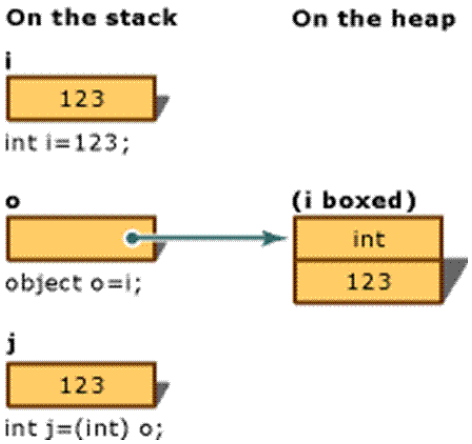
```
using System;
class MyClass
{
   static void Main(string[] args)
    {    object a;
         a = 1;   // un esempio di boxing
         Console.WriteLine(a);
         Console.WriteLine(a.GetType());
         Console.WriteLine(a.ToString());
         Console.WriteLine();
         Console.ReadKey();
    }
}
```

```
1
System.Int32
1
```

Con il boxing e l'unboxing si può realizzare un collegamento tra i VT e i RT permettendo che il valore di un VT sia convertito da e verso un oggetto.
Convertire un VT in un RT è detto boxing, il processo inverso unboxing.
Il boxing avviene automaticamente quando un VT è utilizzato in una locazione che richiederebbe l'uso di un oggetto.
Il boxing di un VT consiste nell'allocare l'istanza di un oggetto e copiare il valore nell'istanza.

Esempio.

```
using System;
class MyClass
{
   static void Main(string[] args)
    {    int i = 1;
         object o = i;        // boxing
         // verifica che o sia davvero un int
         if (o is int)  { Console.Write("O è un integer"); }
         int j = (int) o; // unboxing
         Console.ReadKey();
    }
}
```



Esempio di boxing scorretto, errore di run-time.

```
using System;
class MyClass
{   static void Main(string[] args)
    {   int intI = 123;
        object o = intI;
        // Riferimenti a oggetti incompatibili producono  InvalidCastException
        try
        {   int intJ = (short)o;
            Console.WriteLine("Unboxing OK.");
        }
        catch (InvalidCastException e)
        {  Console.WriteLine("{0} Errore: unboxing non corretto", e);
            Console.ReadKey();
        }
```

```
    }
}
System.InvalidCastException: Cast specificato non valido.
   in MyClass.Main(String[] args) in I:\Esercizi\Visual C#\sequenza_1\sequenza_1
\Program.cs:riga 11 Errore: unboxing non corretto
```

La classe *System.Convert*

Permette di convertire un tipo di base in un altro tipo di base.
Ad esempio, è possibile convertire una stringa lunga un carattere, “m”, nel corrispondente carattere “m”.

```
char a=System.Convert.ToChar(“m”);
```

Se la stringa passata come argomento è più lunga di un carattere o è vuota, sarà lanciata l’eccezione di formato non valido.
Conversione di un intero in un booleano.

```
int t=123;
int f=0;
Console.WriteLine("Convert.ToBoolean({0})={1}",t,Convert.ToBoolean(t));
Console.WriteLine("Convert.ToBoolean({0})={1}",f,Convert.ToBoolean(f));
```

Conversione di una stringa in un valore numerico.

```
string str="12";
Console.WriteLine("str={0}",str);
short s=Convert.ToInt16(str);
int i=Convert.ToInt32(str);
long l=Convert.ToInt64(str);
double d=Convert.ToDouble(str);
byte b=Convert.ToByte(str);
float f=Convert.ToSingle(str);
Console.WriteLine("short {0}",s);
Console.WriteLine("int {0}",i);
Console.WriteLine("long {0}",l);
Console.WriteLine("double {0}",d);
Console.WriteLine("byte {0}",b);
Console.WriteLine("float {0}",f);
```

Una versione del metodo Convert.ToInt32, accetta due parametri, il primo di tipo string e il secondo un intero che specificala base dalla quale si vuole convertire un valore.

```
string strHex="AB1";                    // valore esadecimale
int dec=Convert.ToInt32(strHex,16);     // conversione in decimale
Console.WriteLine(strHex+"="+dec);
```

OPERATORI

Precedenza degli operatori.

Categoria	Operatori
primari	( ) . [ ] x++ x-- new typeof checked unchecked
unari	+ - ! ~ ++x --x (tipo)x
moltiplicativi	* / %
additivi	+ -
shift	>> <<
confronto e type-testing	> < >= <= is as
uguaglianza	== !=
AND logico	&
XOR logico	^
OR logico	|
AND condizionale	&&
OR condizionale	||
ternario	?:
assegnazione	= += -= *= /= %= <<= >>= &= ^= |=

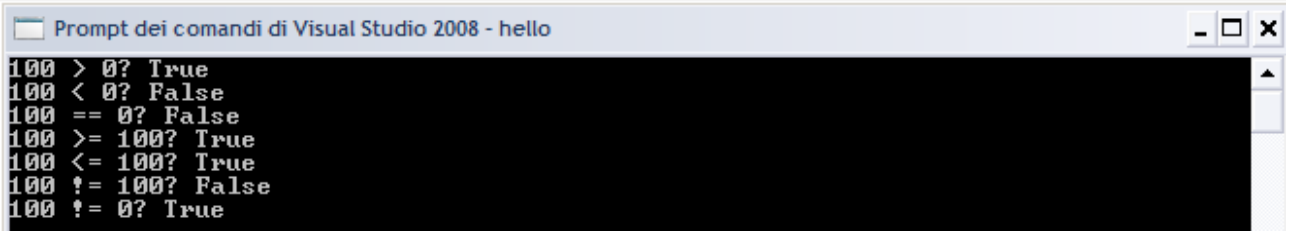
```
int x=10;
int y=3;
int resto=x%y ;                 // restituisce 1 resto della divisione tra interi
int quoz= x/y;                  //restituisce 3 quoziente troncato
```

Esempio

```
using System;
class hello
{    static void Main(string[] args)
       {
          Console.Clear();
          bool T=true;
         bool F=false;
         Console.WriteLine("Operatore AND &");
         Console.WriteLine("F & F = {0}", F&F);
         Console.WriteLine("F & T = {0}", F&T);
         Console.WriteLine("T & F = {0}", T&F);
         Console.WriteLine("T & T = {0}", T&T);
         Console.WriteLine();
         Console.WriteLine("Operatore OR |");
         Console.WriteLine("F | F = {0}", F|F);
         Console.WriteLine("F | T = {0}", F|T);
         Console.WriteLine("T | F = {0}", T|F);
         Console.WriteLine("T | T = {0}", T|T);
         Console.WriteLine();
         Console.WriteLine("Operatore XOR ^");
         Console.WriteLine("F ^ F = {0}", F^F);
         Console.WriteLine("F ^ T = {0}", F^T);
         Console.WriteLine("T ^ F = {0}", T^F);
         Console.WriteLine("T ^ T = {0}", T^T);
         Console.ReadKey();
```

```
    }
}
```


```
using System;
class hello
{
  static void Main(string[] args)
      {   bool b;int cento=100;int zero=0;
         Console.Clear();
        b=(cento>zero);
        Console.WriteLine("100 > 0? {0} ",b);
        b=(cento<zero);
        Console.WriteLine("100 < 0? {0} ",b);
        b=(cento==zero);
        Console.WriteLine(cento +" == " + zero +"? " + b);
        b=(cento>=100);
        Console.WriteLine(cento +" >= 100? " + b);
        b=(cento<=100);
        Console.WriteLine(cento +" <= 100? " + b);
        b=(cento!=100);
        Console.WriteLine(cento +" != 100? " + b);
        b=(cento!=0);
        Console.WriteLine(cento +" != 0? " + b);
         Console.ReadKey();
      }
}
```



Utilizzando gli operatori di shift e l'operatore & è possibile implementare i metodi di conversione da decimale a binario e viceversa.

```
using System;
class hello
{  static void Main(string[] args)
      {
```

```
        Console.Clear();
        Console.Write("Inserisci il valore da convertire < 32: ");
        int n = Convert.ToInt32(Console.ReadLine());
        string strBin="";
        for(int i=0;i<32;i++)
                { if( ((1<<i) & n) == 0)  strBin = "0" + strBin;
                 else strBin = "1" + strBin;
                Console.WriteLine("Conversione: " +strBin);
                }
        Console.ReadKey();
    }
}
```

Operatore dot (“.”)
Si usa per richiamare un metodo o per specificare il nome di una classe.

Operatore *new*
È usato per creare nuove istanze:
- ✓ di classi o di tipi valore;
- ✓ di array;
- ✓ di tipi delegate.

Operatore *typeof*
Permette di ottenere un oggetto *System.Type* per un dato tipo.

```
using System;
public class TestTypeOf
{
public TestTypeOf()
{
Type t1=typeof(string);
Type t2=typeof(System.String);
Console.WriteLine(t1.FullName);
Console.WriteLine(t2.FullName);
}
static void Main()
{
  Console.Clear();
  new TestTypeOf();
  Console.ReadKey();
}
}
System.String
System.String
```

Si nota che il tipo *string* ed il tipo *System.String* sono lo stesso tipo.

Operatore *is*
Determina se il valore di un’espressione è compatibile in fase di run-time con un dato tipo.

```
int i=123;
object box=i;
if(box is int)
        { int n=(int)box; }
```

Operatore *as*
È usato per convertire esplicitamente un valore in un dato tipo riferimento, ma a differenza

di un’operazione di cast, non è sollevata un’eccezione se la conversione non è possibile, è invece restituito il valore *null*.
È utilizzabile solo sui RT.
System.String str=box as System.String;
if(str==null)
Console.WriteLine("str è null");
Se si convertisse l’oggetto *box*, che contiene un intero, in un oggetto *string* si ottiene *null*. In conclusione, *as* fornisce un metodo per abbreviare codice, che fa uso di *is* per valutare se un oggetto è compatibile con un dato tipo, in caso positivo effettua la conversione, altrimenti ritorna null.
if(box is string)
str=(string)box;
else str=null;

TIPI NULL

Permettono la nullabilità dei VT, allora sono incapsulati dentro una *struct* di nome System.Nullable<T>.
Sinonimo di “Nullable<T>” è “T?”
Int? X1 = 10
X1 = null;
System.Nullable<int>x2= 100
X2 = null;
DateTime? Dt = DateTime.Now;
Dt = null;

STRUTTURE DI CONTROLLO

SEQUENZA

```
using System;
class hello
{  static void Main(string[] args)
        {
                // Istruzione vuota
        }
}
```

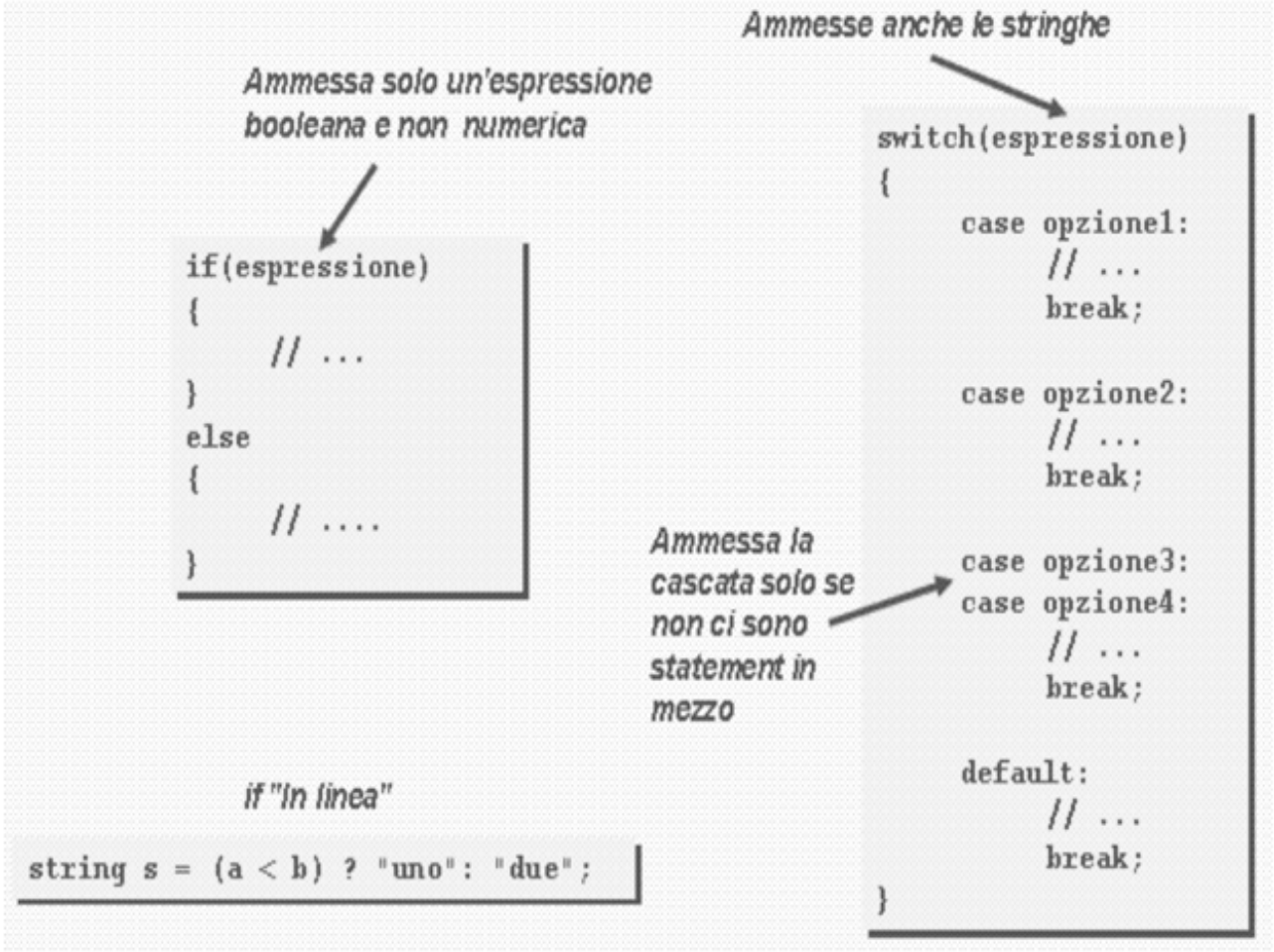
SELEZIONE

Selezione unaria

```
if cond istruzione;
```

Selezione binaria

```
if cond istruzione;
else istruzione;
```



Selezione nidificata

Il C assegna ogni *else* all'*if* più vicino che non ne possieda già uno.

```
if cond1
        if cond2 istruzione;
```

```
        else istruzione;
```
Il C# inizia la valutazione delle espressioni condizionali dall'alto verso il basso, esegue la prima istruzione corrispondente al verificarsi di *cond1*, quindi abbandona la catena di *if*.
Se non si verifica nessuna delle *condi*, è eseguito l'*else* finale (**condizione di default**).
```
if cond1
        istruzione;
else if cond2
        istruzione;
else if cond3
        istruzione;
…
else istruzione;
```

Esempio: progettare un metodo per testare se una variabile è positiva.
Primo caso: usando l'operatore *&* il metodo *IsPositive* è chiamata tre volte.
```
using System;
class hello
{
        static void Main(string[] args)
        {
                Console.Clear();
                int x=-1;int y=1;int z=2;
                if(IsPositive(x) & IsPositive(y) & IsPositive(z))
                        {Console.WriteLine("Tutti positivi");}
                Console.ReadKey();
        }
        // testa se una variabile e' positiva
        public static bool IsPositive(int x)
        {
                Console.WriteLine("Test di positivita' "+x);
                return x>0;
        }
}
Test di positivita' -1
Test di positivita' 1
Test di positivita' 2
```

Secondo caso: usando l'operatore *&&* il metodo *IsPositive* è chiamata una sola volta.
```
using System;
class hello
{
        static void Main(string[] args)
        {
                Console.Clear();
                int x=-1;int y=1;int z=2;
                if(IsPositive(x) && IsPositive(y) && IsPositive(z))
                        {Console.WriteLine("Tutti positivi");}
                Console.ReadKey();
        }
        // testa se una variabile e' positiva
              public static bool IsPositive(int x)
        {
                Console.WriteLine("Test di positivita' "+x);
                return x>0;
```

```
        }
}
Test di positivita' -1
```

Selezione multipla

```
switch (variabile) {
        Case costante1:
                sequenza istruzioni;
                break;
        Case costante2:
                sequenza istruzioni;
                break;
        Default:
                sequenza istruzioni;
}
```

Il C# non prevede il fall-through automatico tra le clausole dello *switch,* ma se fosse necessario le regola si evita in questo modo.

```
switch(variabile)
        {
        default:
                Console.WriteLine("x è diverso da 0 e 1");
                break;
        case 0:
                Console.WriteLine("x è nulla");
                goto case 1;
        case 1:
                Console.WriteLine("x è 0 oppure 1");
                break;
        case 2:
                goto default;
}
```

Operatore ?

Operatore ternario (richiede tre operandi) che sostituisce l'istruzione *if-else*.

esp1 ? esp2 : esp3

Si valuta *esp1* se è vera si valuta *esp2*, se *esp1* è falsa si valuta *esp3*.

```
int x=10;int y;
y = x>9 ? 100: 200;                /* stampa 100 */
```

ITERAZIONE

Il numero d'iterazioni non è noto a priori

Ciclo a condizione iniziale

```
while (cond) istruzione;
```

Ciclo a condizione finale

```
Do {
        Istruzione;
} while (condizione);
```

Attenzione il C# itera quando è vera ed esce quando è falsa.

Il numero d'iterazioni è noto a priori

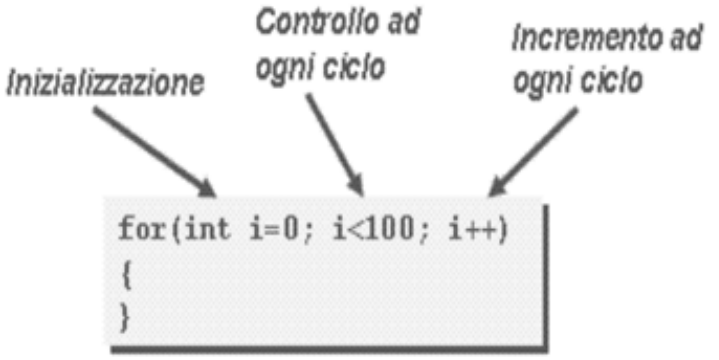
Ciclo enumerativo

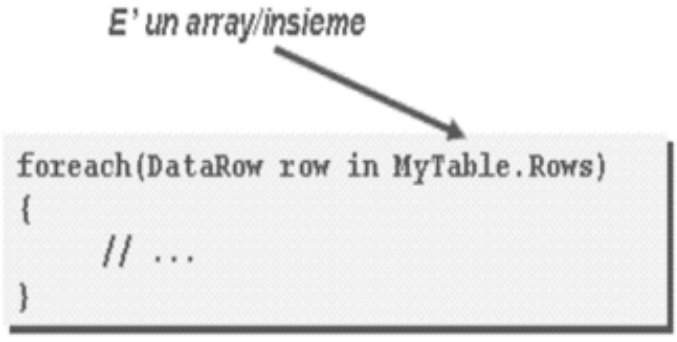
```
for (inizializzazione;condizione;in(dec)cremento) istruzione;
```

C# fornisce un'istruzione che permette di scorrere gli elementi di collezioni o array senza far uso di espressioni booleane, scorrendo gli elementi in modo sequenziale.
foreach (tipo elemento) istruzione;

Esempio.

```
using System;
class hello
{
        static int[] vettore = new int[5];
        static void Main(string[] args)
        {
                Console.Clear();
                stampa();
                Console.WriteLine();
                stampaf();
                Console.ReadKey();
        }
        public static void stampa()
        {       for(int i=0;i<5;i++)
                { vettore[i]=i;
                 Console.WriteLine("vettore[{0}]={1}",i,vettore[i]); }
        }
        public static void stampaf()
        {  foreach (int i in vettore) Console.WriteLine("vettore[]={0}",i); }
}
```





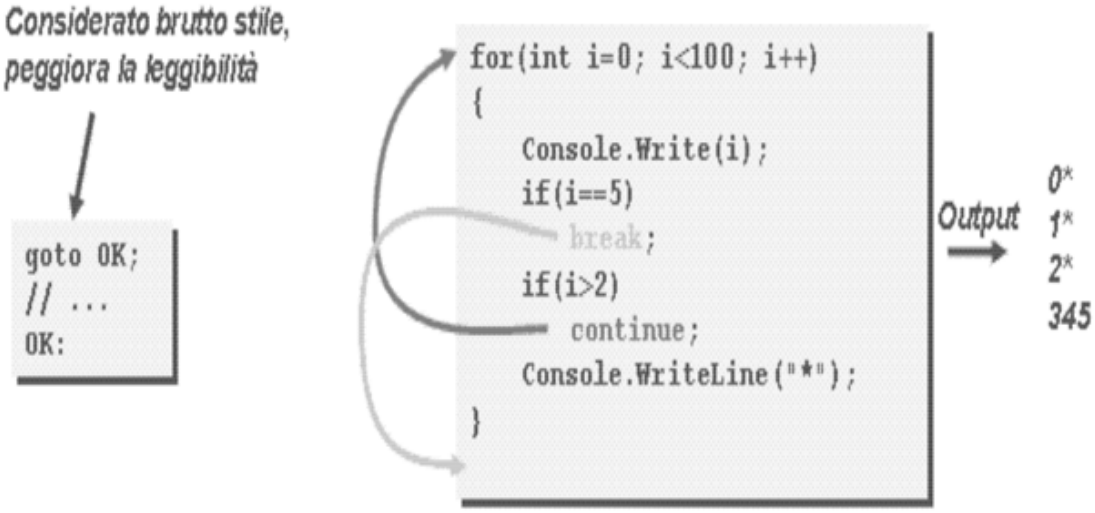
Senza fine
while (true);
do { … } while(true);
for (;;)

break

continue

goto

throw



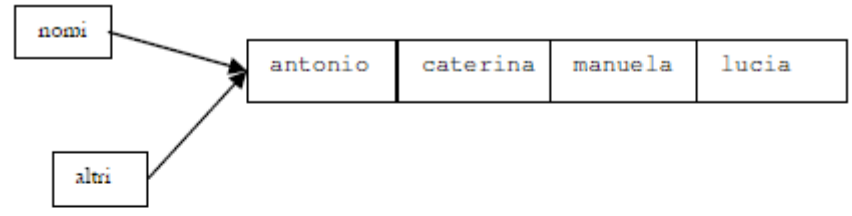
ARRAY

COLLEZIONE DI OGGETTI

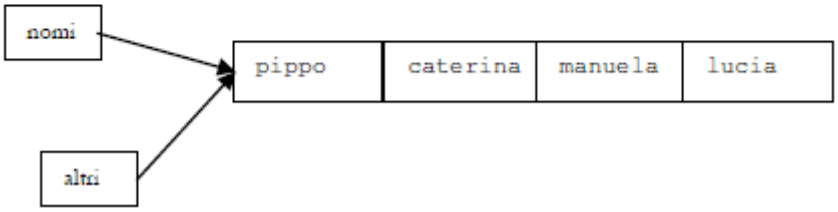
Sono strutture dati fondamentali per memorizzare insiemi di oggetti in memoria.
Le collezioni mantengono una lista dinamica di oggetti, il numero di elementi non ha limiti intrinseci, gli elementi possono essere di tipi eterogenei.
La ricerca, l'ordinamento, la rimozione di un elemento, liste, code, pile, tabelle hash usano il *namespace System.Collections*.
Le classi sono basate su interfacce che standardizzano i metodi per trattare questi insiemi di oggetti.
Per esempio, ogni collezione di oggetti implementa l'interfaccia *Collection*, che espone la proprietà *Count* per ottenere il numero di elementi di una collezione ed il metodo *CopyTo* per copiare i suoi elementi in un array.
L'interfaccia *ICollection* deriva da *IEnumerable*, che espone il metodo *GetEnumerator*, che permette d'iterare, in modo sequenziale, gli elementi di una collezione.

VETTORE

È una sequenza di elementi, tutti dello stesso tipo, che può essere sia un tipo riferimento sia un tipo valore; l'indice dell'array inizia sempre da zero.
È possibile inizializzare la dimensione di un array utilizzando *new*.
byte[] vettore = new byte [5]; oppure *byte[] vettore = new byte [] {1,2,3,4,5};*
La classe *System.Array* fornisce metodi per usare gli array: creare, manipolare, ricercare, ordinare; la classe *Array* è la classe base per tutti gli array, per cui sono istanze della classe *System.Array*, quindi la variabile vettore è un'istanza di un tipo riferimento.
int[] vettore=new int[10];
int vettore2=vettore;
Visualizzare il nome della classe base della variabile *vettore*.
Console.WriteLine(vettore.GetType().BaseType); *// stampa System.Array*
Questa assegnazione è in realtà solo un riferimento all'array, per cui modificando il primo elemento di uno dei due array, la variazione si rifletterà anche sul primo elemento dell'altro array.



Dopo l'assegnazione: Altri[0]="pippo";



string[] nomi={"antonio","caterina","manuela","lucia"};
string[] altri=nomi;
altri[0]= "pippo"; *// cambia il primo elemento dell'array altri*

Console.WriteLine(nomi[0]); // anche il primo elemento di nomi sarà uguale a pippo
La classe *System.Array* fornisce parecchi metodi, per esempio la proprietà *Length*.
string[] stringhe={"Hello","World"};
int lunghezza=vettore.Length; // ritorna lunghezza=2
C# permette di definire due tipi di array multidimensionali.

Esempio.
```
using System;
class hello
{
	static int[] vettore = new int[5];
	static void Main(string[] args)
	{
		Console.Clear();
		stampa();
		Console.ReadKey();
	}
	public static void stampa()
	{	for(int i=0;i<5;i++)
		{
			vettore[i]=i;
			Console.WriteLine("vettore[{0}]={1}",i,vettore[i]);
		}
	}
}
vettore[0]=0
vettore[1]=1
vettore[2]=2
vettore[3]=3
vettore[4]=4
```

Esempio: stampare prima i numeri pari e poi quelli dispari di un vettore.
```
using System;
using System.Collections.Generic;
using System.Text;
namespace Yield
{
   class Yield
   {
      public static class NumberList
      {
         // Creare un vettore di numeri interi.
         public static int[] ints = { 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377 };
         // Definire una proprietà che restituisce soltanto i numeri pari.
         public static IEnumerable<int> GetEven()
         {
            // Utilizzare yield per restituire i numeri pari dell'elenco.
            foreach (int i in ints)
               if (i % 2 == 0)
                  yield return i;
         }
         // Definire una proprietà che restituisce soltanto i numeri dispari.
         public static IEnumerable<int> GetOdd()
         {
```

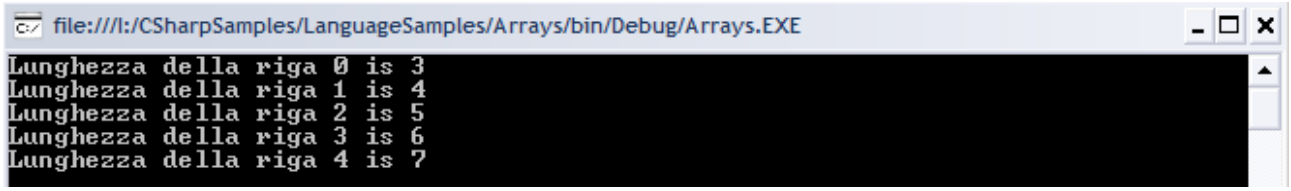
```
            // Utilizzare yield per restituire soltanto i numeri dispari.
            foreach (int i in ints)
                if (i % 2 == 1) yield return i;
        }
    }
    static void Main(string[] args)
    {
        Console.WriteLine("Numeri pari");
        foreach (int i in NumberList.GetEven()) Console.WriteLine(i);
        Console.WriteLine();
        // Visualizzare i numeri dispari.
        Console.WriteLine("Numeri dispari");
        foreach (int i in NumberList.GetOdd()) Console.WriteLine(i);
        Console.ReadKey();
    }
  }
}
```

MATRICE

Definisce una matrice con più righe aventi lo stesso numero di colonne.

```
using System;
class DeclareArraysSample
{
   public static void Main()
   {
      Console.Clear();
      // Matrice unidimensionale
      int[] numbers = new int[5];
      // Matrice multidimensionale
      string[,] names = new string[5,4];
      // Matrice di matrici
      byte[][] scores = new byte[5][];
      // Creare la matrice di matrici
      for (int i = 0; i < scores.Length; i++)
         {scores[i] = new byte[i+3];}
      // Stampare la lunghezza di ciascuna riga
      for (int i = 0; i < scores.Length; i++)
      {
         Console.WriteLine("Lunghezza della riga {0} is {1}", i, scores[i].Length);
      }
```

```
        Console.ReadKey();
    }
}
```



La proprietà *Rank* restituisce la dimensione dell'array.

```
int[] mono=new int[3];
int[,] bidim=new int[5,3];
int[,,] tridim=new int[3,4,5];
Console.WriteLine("mono ha dimensione {0}",mono.Rank);          // 1 dimensione
Console.WriteLine("bidim ha dimensione {0}",bidim.Rank);        // 2 dimensioni
Console.WriteLine("tridim ha dimensione {0}",tridim.Rank);      // 3 dimensioni
```

La proprietà *Length* restituisce il numero totale di elementi dell'array.

```
string[,] names = new string[5,4];                              // 20 elementi
```

Il metodo *GetLength (int dim)* ha come argomento l'indice della dimensione di cui si vuole la lunghezza.

```
for(int i=0;i<numeri.GetLength(0);i++)
        for(int j=0;j<numeri.GetLength(1);j++)
                numeri[i,j]=i*j;
```

I metodi *GetLowerBound* e *GetUpperBound* restituiscono i limiti inferiore e superiore di una dimensione dell'array.

```
string[,] s=new string[3,5];
for(int i=0;i<numeri.Rank;i++)
{Console.WriteLine("rank{0}:indice inf={1} e sup={2}", i, numeri.GetLowerBound(i),
numeri.GetUpperBound(i)); }                                     // stampano [0,2] e [0,4]
```

Si può creare un'istanza di un array con il metodo statico *CreateInstance* che specifica il tipo degli elementi, la lunghezza, il numero di dimensioni ed i limiti inferiori.

```
int[] lunghezze=new int[2]{3,4};
int[] limitiInf=new int[2]{1,2};
Array arr=Array.CreateInstance(typeof(string),lunghezze,limitiInf);
for(int i=0; i<arr.Rank;i++)
        Console.WriteLine("{0}\t{1}\t{2}",i,arr.GetLowerBound(i),arr.GetUpperBound(i));
// limiti       inf     sup
// 0            1       3
// 1            2       5
```

Nell'array creato in questo modo non è possibile usare l'operatore d'indicizzazione [], la classe *Array* fornisce i metodi *GetValue* e *SetValue*.

```
using System;
class ConsoleAdder
{       public static void Main()
        {Console.Clear();
        Array myArray=Array.CreateInstance( typeof(String), 2, 4 );
        myArray.SetValue( "Un", 0, 0 );
        myArray.SetValue( "array", 0, 1 );
        myArray.SetValue( "di", 0, 2 );
```

```
        myArray.SetValue( "stringhe", 0, 3 );
        myArray.SetValue( "disposte", 1, 0 );
        myArray.SetValue( "su", 1, 1 );
        myArray.SetValue( "due", 1, 2 );
        myArray.SetValue( "righe", 1, 3 );
        // Stampa gli elementi dell'array.
        for ( int i = myArray.GetLowerBound(0); i <= myArray.GetUpperBound(0); i++ )
        for ( int j = myArray.GetLowerBound(1); j <= myArray.GetUpperBound(1); j++ )
        Console.WriteLine( "\t[{0},{1}]:\t{2}", i, j, myArray.GetValue( i, j ) );
        Console.ReadKey();
        }
}
```

Il metodo statico *BinarySearch* restituisce l'indice dell'elemento se esiste, altrimenti ritorna un numero negativo.

```
int Array.BinarySearch(Array arr, Object obj)
using System;
class ConsoleAdder
{       public static void Main()
        {       Console.Clear();
                int[] altriNumeri={ 1,2,3,4,5,6,7,8 };
                int nIndex=Array.BinarySearch(altriNumeri,4);
                Console.WriteLine("4 si trova all'indice {0}",nIndex);
                nIndex=Array.BinarySearch(altriNumeri,10);
                if(nIndex<0)
                Console.WriteLine("Il numero 10 non c'è");
                Console.ReadKey();
        }
}
```

Il metodo statico *Array.Clear* cancella tutti o un intervallo di elementi di un array, cancellare significa impostare a 0 dei numeri, a *false* dei booleani, a *null* altri tipi di oggetti.

```
public static void Clear(Array array, int index, int length);
int[] numeri={ 1,2,3,4,5};
Array.Clear(numeri,2,2);
```

Cancella gli elementi 3,4, in pratica 2 elementi a partire dall'indice 2.

Il metodo statico *Copy* copia l'array sorgente nell'array destinazione con la possibilità di specifiacre il numero di elementi da copiare.

```
int[] altriNumeri={ 1,2,3,4,5};
int[] arrCopia=new int[altriNumeri.Length];
Array.Copy(altriNumeri,arrCopia,altriNumeri.Length);
```

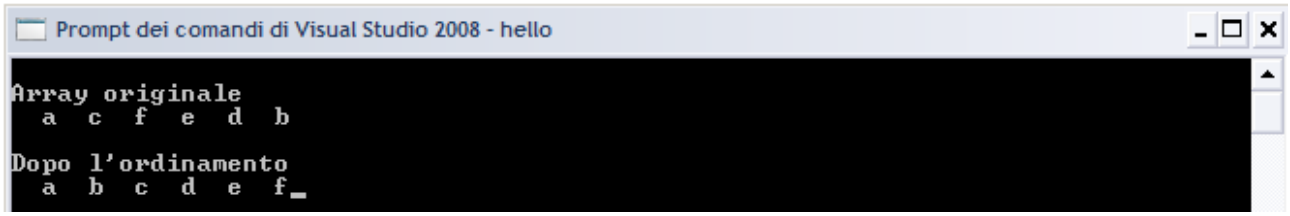
```
for(int i=0;i<arrCopia.Length;i++)
        Console.Write("{0,3}",arrCopia[i]);
```

Il metodo d'istanza *CopyTo (Array,int)* effettua la copia degli elementi dell'array sorgente, li inserisce nell'array destinazione a partire da un dato indice.

```
Array myArray=Array.CreateInstance( typeof(String), 4 );
myArray.SetValue( "Un", 0 );
myArray.SetValue( "array", 1 );
myArray.SetValue( "di", 2 );
myArray.SetValue( "stringhe", 3 );
Array destArray=Array.CreateInstance(typeof(string),8);
myArray.CopyTo(destArray,myArray.Length);                    // indice = 4
Console.WriteLine("L'array destinazione contiene:");
for(int i=0;i<destArray.Length;i++)
        { Console.WriteLine("destArray[{0}]={1}",i,destArray.GetValue(i)); }
```

Il metodo statico *Sort* ordina in modo crescente gli elementi di un array.

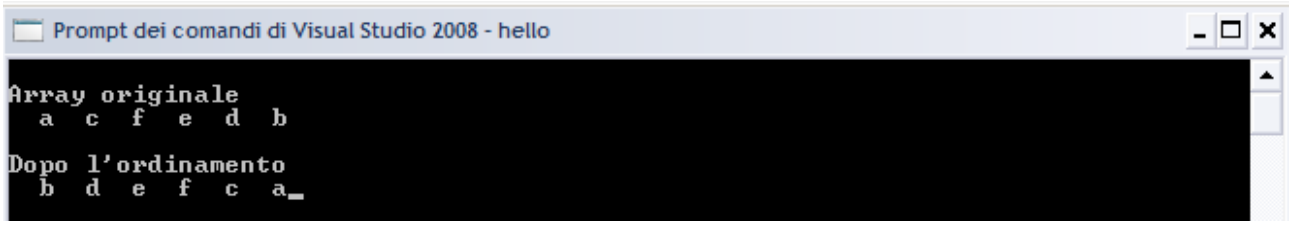
```
using System;
class ConsoleAdder
{       public static void Main()
        {       Console.Clear();
                string[] str={"a","c","f","e","d","b"};
                Console.WriteLine("\nArray originale");
                for(int i=0;i<str.Length;i++)
                        Console.Write("{0,3}",str[i]);
                Console.WriteLine();
                Array.Sort(str);
                Console.WriteLine("\nDopo l'ordinamento");
                        for(int i=0;i<str.Length;i++)
                Console.Write("{0,3}",str[i]);
                Console.ReadKey();
        }
}
```



Il metodo statico *Reverse* ordina in modo decrescente gli elementi di un array.

```
using System;
class ConsoleAdder
{       public static void Main()
        {       Console.Clear();
                string[] str={"a","c","f","e","d","b"};
                Console.WriteLine("\nArray originale");
                for(int i=0;i<str.Length;i++)
                        Console.Write("{0,3}",str[i]);
                Console.WriteLine();
                Array.Reverse(str);
                Console.WriteLine("\nDopo l'ordinamento");
                        for(int i=0;i<str.Length;i++)
```

```
            Console.Write("{0,3}",str[i]);
            Console.ReadKey();
        }
}
```



Il metodo *IndexOf* (*LastIndexOf*) restituisce la prima (ultima) posizione di un elemento in un array.

```
byte[] myArray={0,1,0,0,1,0,1,0};
Console.WriteLine("\n\nmyArray contiene:");
for(int i=0;i<myArray.Length;i++)
        Console.Write("{0,3}",myArray[i]);
int index=Array.IndexOf(myArray,(byte)1);
Console.WriteLine("\nIl primo 1 si trova all'indice {0}",index);
index=Array.LastIndexOf(myArray,(byte)1);
Console.WriteLine("\nL'ultimo 1 si trova all'indice {0}",index);
```

La classe *ArrayList*

Ha dimensioni che possono crescere nel tempo, all'atto della creazione è specificata la capacità iniziale, per default è fissata a 16.

```
ArrayList al=new ArrayList(10);          // capacità iniziale 10 elementi
ArrayList al2=new ArrayList();           // capacità iniziale default 16 elementi
```

È fondamentale distinguere la capacità, *Capacity*, dal numero di elementi effettivamente presenti, *Count*; in genere, la capacità sarà sempre maggiore del numero di oggetti effettivamente contenuti.

```
ArrayList al=new ArrayList(20);
int cap=al.Capacity;                     // restituisce 20;
int n=al.Count;                          // restituisce 0, nessun elemento è stato inserito
```

Per minimizzare lo spreco di memoria si usa il metodo *TrimToSize*, che imposta la capacità uguale al numero di oggetti.

```
al.TrimToSize();
```

È possibile inizializzare un *ArrayList* con un altro oggetto che implementa l'interfaccia *ICollection*, per esempio se si ha un array d'interi, si può creare un *ArrayList* che li contenga tutti i suoi oggetti.

```
int[] arr=new int[50];
for(int i=0;i<50;i++)
        arr[i]=i;
ArrayList vettore=new ArrayList(arr);
Console.WriteLine("L'arraylist vettore contiene");
foreach(int i in vettore)
        Console.Write(" "+vettore[i]);
```

L'ArrayList tratta gli elementi come riferimenti, quindi è possibile inserire qualsiasi tipo di oggetto, e nel caso in cui si voglia memorizzare un elemento di tipo valore, esso subirà il boxing automatico, per leggere un elemento, invece, bisogna effettuare l'unboxing in modo esplicito.

```
int i=vettore[0];                // errore, non si può convertire implicitamente object in int
```

string s=(string)vettore[0]; // eccezione InvalidCastException
int i=(int)vettore[0]; //OK

Il metodo *Add* permette di aggiungere un oggetto alla fine e restituisce l'indice della posizione d'inserimento.
int indice=vettore.Add("hello");

Il metodo *Insert* permette d'inserire un oggetto in una specifica posizione.
Auto auto=new Auto();
vettore.Insert(1, auto);

Il metodo *RemoveAt* rimuove l'oggetto alla posizione specificata.
vettore.RemoveAt(0); // rimuove il primo elemento

Il metodo *Remove* specifica direttamente l'oggetto da rimuovere, per cui è fatta una ricerca lineare per trovare l'oggetto; non restituisce nessun valore, non lancia un'eccezione se l'oggetto non è stato trovato.
vettore.Remove(auto); // ricerca e rimuove l'oggetto auto

Il metodo *Contains* effettua la ricerca e restituisce un booleano per indicare se l'oggetto è stato trovato.
if(vettore.Contains(auto)) vettore.Remove(auto);

Il metodo *Clear* rimuove tutti gli oggetti.
vettore.Clear();

La classe *ArrayList* permette di lavorare con più elementi contemporaneamente, per esempio, dato un insieme di oggetti contenuti in una collezione, in altre parole in un oggetto che implementa l'interfaccia *ICollection*, è possibile aggiungerli in un colpo solo, con il metodo *AddRange*, oppure inserirli in una data posizione con il metodo *InsertRange*, o ancora rimuoverli con il metodo *RemoveRange*.
vettore.AddRange(new string[]{"a","b","c"}); // aggiunge un array di stringhe
vettore.InsertRange(vettore.Count-1, new int[]{1,2,3});// aggiunge un array d'interi in coda
vettore.RemoveRange(2,3); //rimuove 3 elementi a partire dall'indice 2

Il metodo *SetRange* permette di ricopiare su un intervallo lo stesso numero di nuovi oggetti.
Vettore.SetRange(0,new int[]{3,4,5}); // sovrascrive i primi tre elementi con gli interi 3,4,5

Jagged Arrays (ortogonali)

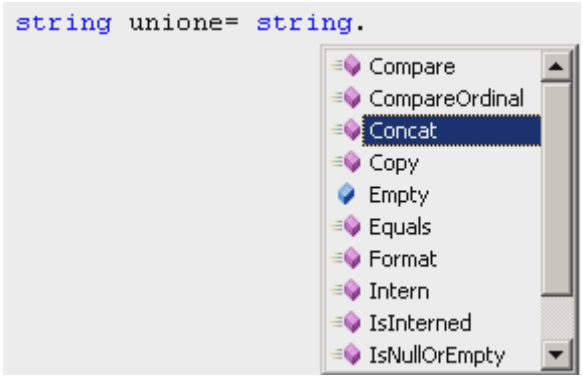
Definisce una matrice con più righe, ma ognuna di esse può avere diverso numero di colonne.
[1]
[2 3]
[4 5 6]
Si dichiara un array di array, usando la coppia di "[] []" per ogni dimensione.
int[][] jagged;
jagged=new int[3][]; // un array a due dimensioni
jagged[0]=new int[1]; // la prima riga ha lunghezza 1
jagged[1]=new int[2]; // la seconda riga ha lunghezza 2
jagged[2]=new int[3]; // la terza riga ha lunghezza 3
Accesso.
jagged[0][0]=1;

Console.WriteLine("jagged[0][0]={0}",jagged[0][0]);
jagged[1][0]=2;
Console.WriteLine("jagged[1][0]="+jagged[1][0]);

STRING

Il metodo statico *String.Format* permette di formattare una stringa in formato standard o personalizzato e non solo per l'output su console, ma anche per visualizzare una stringa in una casella di testo, oppure salvarla su un file di testo.
In .NET una sequenza di caratteri unicode è un'istanza della struttura *System.Char*.
In C# il tipo *System.String* è un tipo riferimento derivato dal *System.Object*, ma non è possibile costruire una stringa con l'operatore *new*, per cui la stringa è trattata come se fosse un tipo primitivo e quindi è sufficiente un'assegnazione di un *literal* ad una variabile di tipo stringa.
string str2="CIAO";
Le stringhe sono immutabile, quindi non è possibile modificare una stringa o uno dei suoi caratteri a run-time.
Per esempio, il metodo *ToLower* (*ToUpper*) della classe *String* prende come parametro una stringa e la converte in minuscolo (maiuscolo).
Console.WriteLine(str2.ToLower()); *// stampa ciao*
In pratica, è creata una nuova stringa "*ciao*", che sarà distrutta dal GC dopo essere stata stampata a video, senza toccare l'originale.

Appena si scrive il punto dopo la parola chiave *string*, si aprirà un menu che darà la possibilità di scegliere uno dei tanti metodi che ci mette a disposizione il tipo *string*.



Esaminare una stringa
Per ottenere la lunghezza di una stringa si usa la proprietà *Length;* la proprietà *Chars* restituisce i caratteri che costituiscono la stringa; l'operatore [] accede al carattere che si trova all'indice i specificato.
string str="pippo";
int lung=str.Length;
for(int i=0;i<lung;i++)
{ Console.WriteLine("Carattere {0}={1}", i, str[i]);}
(i>= *lung)* o i negativo sarà generata un'eccezione *IndexOutRangeException*.

Il metodo *TocharArray* permette di ottenere tutti o parte dei caratteri di una stringa.
string str="pippo";
char[] chars=str.ToCharArray();
foreach(char c in chars)
{Console.WriteLine(c);}

Il metodo *IndexOf* restituisce l'indice della prima occorrenza; *LastIndexOf* restituisce

l'indice dell'ultima occorrenza; restituiscono -1 se il valore cercato non è trovato.

```
string str="hello world";
Console.WriteLine(str.IndexOf("world"));          // stampa 6
Console.WriteLine(str.IndexOf("l",5));            //stampa 9
Console.WriteLine(str.IndexOf('o'));              //stampa 4
Console.WriteLine(str.LastIndexOf('l'));          //stampa 9
Console.WriteLine(str.LastIndexOf("or",5,3));     //stampa -1
```

I metodi *IndexOfAny* e *LastIndexOfAny* restitruiscono la posizione all'interno di una stringa di uno o più caratteri contenuti in un array di char.

```
string str="ricerca nella stringa";
string seek="abc";
char[] chars=seek.ToCharArray();
int j=str.IndexOfAny(chars);
// restituisce 2, indice della prima c di 'ricerca'
Console.WriteLine("str.IndexOfAny({0})={1}",seek,j);
j=str.LastIndexOfAny(chars);
// restituisce 20, indice della ultima a della stringa str
Console.WriteLine("str.LastIndexOfAny({0})={1}",seek,j)
```

Esiste un metodo che prende in input un altro parametro intero, che indica l'indice di partenza della ricerca.

```
j=str.IndexOfAny(chars,10);
// restituisce 12, indice della a di 'nella'
Console.WriteLine("str.IndexOfAny({0},{1})={2}",seek,10,j);
```

Esiste un altro metodo che prende un terzo parametro intero, che indica il numero di caratteri a partire dall'indice di partenza in cui cercare uno dei caratteri.

```
j=str.IndexOfAny(chars,10,3);
// restituisce ancora 12, come nel precedente
Console.WriteLine("str.IndexOfAny({0},{1},{2})={3}",seek,10,3,j);
j=str.LastIndexOfAny(chars,10);
// restituisce 6, indice della a finale di 'ricerca', che è l'ultima occorrenza trovata
// nei primi 10 caratteri
Console.WriteLine("str.LastIndexOfAny({0},{1})={2}",seek,10,j);
j=str.LastIndexOfAny(chars,10,3);
//restituisce -1, non trova nessuno dei caratteri 'abc' negli ultimi 3 caratteri
// dei primi 10 della stringa
Console.WriteLine("str.LastIndexOfAny({0},{1},{2})={3}",seek,10,3,j);
```

Confronto tra stringhe

Il metodo *Equals* è fornito sia in versione statica sia in una versione d'istanza; ritorna *true* per stringhe uguali.

```
string s1="hello";
string s2="world";
string s3="he"+"llo";
Console.WriteLine(s1.Equals(s2));          // false
Console.WriteLine(s1.Equals(s3));          // true
Console.WriteLine(String.Equals(s1,s2));   //f alse
Console.WriteLine(s1==s3);                 // true
```

Se lo scopo del confronto non è quello di determinare l'uguaglianza di due stringhe, ma si desidera un ordinamento alfabetico, si usano i seguenti metodi.

Compare è un metodo statico che determina come due stringhe, A e B, devono essere

ordinate una rispetto all'altra, il valore di ritorno può essere:
1. nullo se A = B;
2. negativo A > B;
3. positivo A < B.

Può essere utilizzato con un terzo parametro booleano per specificare se il confronto deve essere fatto in modo case-sensitive.

```
Console.WriteLine(String.Compare("A","B"));        // -1, A è minore di B
Console.WriteLine(String.Compare("A","A"));        // 0, A è uguale ad A
Console.WriteLine(String.Compare("B","A"));        // 1, B è maggiore di A
Console.WriteLine(String.Compare("A",null));       // 1, A è maggiore di null
```

CompareOrdinal è un metodo statico uguale al precedente, la differenza è che effettua un confronto carattere per carattere, restituisce *false* se le stringhe sono scritte con caratteri diversi.

```
bool b=String.CompareOrdinal("Strass", "Stra@");        // restituisce false
b=String.Compare("Strass", "Stra@");                    // restituisce true
```

CompareTo è un metodo d'istanza che restituisce un valore intero con lo stesso significato dei casi precedenti.

```
string str="b";
Console.WriteLine(str.CompareTo("a"));        // stampa 1
Console.WriteLine(str.CompareTo("c"));        // stampa -1
Console.WriteLine(str.CompareTo("b"));        // stampa 0
```

I metodi *StartsWith* (*EndsWith*) restituiscono un valore booleano ad indicare se la prima (ultima) parte di una stringa coincide con una sotto stringa passata come parametro.

```
str="hello world";
Console.WriteLine(str.StartsWith("hello"));        // true
Console.WriteLine(str.EndsWith("world"));          // true
```

Altre operazioni

Il metodo *SubString* restituisce una sotto stringa della stringa originale, a partire da una certa posizione e per un certo numero di caratteri.

```
str="hello world";
Console.WriteLine(str.Substring(6));        // stampa world
Console.WriteLine(str.Substring(6,2));      // stampa wo
```

Il metodo *Split* suddivide una stringa in diverse sotto stringhe, delimitate da uno o più caratteri, ed inserisce le sotto stringhe in un array di stringhe.

Per esempio, suddividere la stringa *str* in un array di stringhe, usando come separatori i caratteri caricati nell'array *sep*.

```
str="parole,separate,da;virgola;o;punto;e;virgola";\
char[] sep=new char[]{',';'};
string[] parole=str.Split(sep);
foreach(string parola in parole)
        { Console.WriteLine(parola); }
```

Concat è il metodo che permette di concatenare stringhe.

```
string concat=String.Concat(parole);
Console.WriteLine(concat);        // stampa paroleseparatedavirgolaopuntoevirgola
```

Join inserisce un elemento separatore tra le stringhe da concatenare.

```
string join=String.Join(" ",parole);
Console.WriteLine(join);        // parole separate da virgola o punto e virgola
```

I metodi *Remove, Replace, Insert* lavorano su una stringa, restituendo una nuova stringa in cui si è rimossa, rimpiazzata, inserita una sotto stringa o un insieme di caratteri.

```
str="hello world";
Console.WriteLine(str.Remove(4,7));                 // stampa hell
Console.WriteLine(str.Replace("world","universe")); // stampa hello universe
Console.WriteLine(str.Replace('e','a'));            // stampa hallo world
Console.WriteLine(str.Insert(6,"beautiful "));      // stampa hello beautiful world
```

I metodi *PadLeft, PadRight* formattano una stringa, aggiungendo un numero di caratteri specificati a sinistra, destra; se non si specifica il carattere è usato lo spazio.

```
Console.WriteLine(str.PadLeft(15,'_'));             // stampa ____hello world
Console.WriteLine(str.PadRight(15,'_'));            // stampa hello world____
```

I metodi *Trim,TrinLeft, TrimRight*, rimuovono tutte le occorrenze di un dato insieme di caratteri che si trovano a sinistra e a destra della stringa, oppure solo dalla sinistra, solo dalla destra.

```
str="_-_-_hello world_-_-_";
char[] chars=new char[]{'_','-'};
Console.WriteLine(str.TrimStart(chars));            // stampa hello world_-_-_
Console.WriteLine(str.TrimEnd(chars));              // stampa _-_-_hello world
Console.WriteLine(str.Trim(chars));                 // stampa hello world
```

Il metodo *ToString* serve a fornire una rappresentazione testuale del contenuto di un oggetto, è *virtual* nella classe *System.Object*, quindi ogni classe può fornire un override di esso.
Per esempio, i tipi numerici predefiniti.

```
int i=100;
string str=i.ToString();                            // restituisce “100”
```

Esempio, progettare la classe *Studente*.

```
using System;
namespace TestObject
{      class Studente
       {       int matricola;
               string cognome;
               string nome;
               public Studente(int m, string n, string c)
               {       matricola=m;
                       cognome=c;
                       nome=n;
               }
               static void Main()
               {       Console.Clear();
                       Studente studente=new Studente(7777,"Piero", "Bianchi");
                       Console.WriteLine(studente);
                       Console.ReadKey();
               }
       }
}
```

La chiamata *Console.WriteLine(studente)* invoca il metodo *studente.ToString()* e stampa *TestObject.Studente*
Per stampare il nome, il cognome e il numero di matricola, bisogna scrivere nella classe *Studente* un override del metodo *ToString*.

```
using System;
```

```
namespace TestObject
{       class Studente
        {       int matricola;
                string cognome;
                string nome;
                public Studente(int m, string n, string c)
                {       matricola=m;
                        cognome=c;
                        nome=n;
                }
                public override string ToString()
                {return "Studente "+matricola+" - "+cognome+" "+nome;}
                static void Main()
                {       Console.Clear();
                        Studente studente=new Studente(7777,"Piero", "Bianchi");
                        Console.WriteLine(studente);
                        Console.ReadKey();
                }
        }
}
Studente 7777 - Bianchi Piero
```

STRING BUILDER

È contenuta nel *namespace System.Text* e permette di effettuare dinamicamente operazioni di modifica sulle stringhe, nel senso che è possibile aggiungere, rimuovere, sostituire, inserire caratteri anche dopo la creazione della stringa.
Un oggetto *StringBuilder* è un array di caratteri che rappresenta una stringa.
Il costruttore di default riserva sedici caratteri, il valore della capacità e quello della lunghezza della stringa si ottengono in questo modo.

```
StringBuilder sb=new StringBuilder();
Console.WriteLine(sb.Capacity);                 // stampa il valore 16
Console.WriteLine(sb.Length);                   // stampa il valore 0
```

La proprietà *MaxCapacity*, di sola lettura, indica il limite massimo fino al quale la stringa può crescere, per default vale *Int32.MaxValue* (2^{32} - 1 = 4.294.967.295)
Oltre al costruttore di default ci sono altri costruttori.

```
sb=new StringBuilder(100);              // imposta la capacità iniziale a 100 caratteri
sb=new StringBuilder("hello");          // inizializza con i caratteri di 'hello'
sb=new StringBuilder(100,500);
// capacità iniziale 100, che può crescere fino ad un massimo di 500
sb=new StringBuilder("hello",10);
// inizializza con i caratteri di 'hello' e capacità iniziale 10
```

Metodi

La proprietà *Chars* permette di visualizzare e modificare i caratteri, è anche l'indicizzatore della classe.

```
StringBuilder sb=new StringBuilder("pippo");
Console.WriteLine("sb contiene i caratteri");
for(int i=0;i<sb.Length;i++)
        { Console.WriteLine(sb[i]); }
sb[0]='b';
Console.WriteLine(sb.ToString());
```

Per ottenere l'oggetto *string* costruito e gestito dall'oggetto *StringBuilder* è utilizzato il

metodo *ToString* che restituisce un riferimento alla stringa e, dopo la sua chiamata, se si modifica ancora la stringa è creato un altro array di caratteri.

Metodo	Descrizione
Append	Appende un oggetto convertito in stringa in coda all'array di caratteri, aumentando la sua capacità se necessario
Insert	Inserisce un oggetto convertito in stringa in una data posizione dell'array di caratteri, aumentando la sua capacità se necessario.
Replace	Sostituisce un carattere con un altro, oppure una stringa con un'altra all'interno del vettore di caratteri.
Remove	Rimuove un intervallo di caratteri dal vettore di caratteri.
AppendFormat	Appende gli oggetti specificate utilizzando una stringa di formato. E' uno dei metodi più utili della classe StringBuilder.

StringBuilder sb=new StringBuilder("hello");
sb.Append(" world");
sb.Insert(6,"beautiful ");
sb.Replace("world","universe");
sb.Remove(5,sb.Length-5);
sb.AppendFormat(" {0} {1}","da","Antonio");
Console.WriteLine(sb.ToString());
I metodi della classe *StringBuilder* restituiscono un riferimento allo stesso oggetto *StringBuilder* su cui sono chiamati, questo permette di effettuare modifiche in sequenza.
StringBuilder sb2=new StringBuilder();
sb2.AppendFormat("\n\t{0}+{1}","Io","Programmo").Replace('+',' ').Append(" C# ");
Console.WriteLine(sb2.ToString().TrimEnd());

TABELLE HASH

Sono strutture dati che memorizzano coppie chiave-valore e organizzate in base al codice hash della chiave.
La costruzione e la popolazione di una tabella hash avviene grazie alla classe *Hashtable*, basta istanziare la classe ed usare il metodo *Add*, se l'elemento è già presente, è generata l'eccezione *ArgumentException*.
Hashtable rubrica=new Hashtable();
rubrica.Add("Antonio","05 12345");
rubrica.Add("Caterina","09 41234");
rubrica.Add("Daniele","32 8765");
rubrica.Add("Rosita","09 09876");
In questo modo la ricerca di un numero di telefono è fatta con un'indicizzazione della *Hashtable* con il nome da cercare.
string numero=(string)rubrica["Caterina"];

È possibile aggiungere elementi con l'indicizzatore, se l'elemento è già presente, è sostituito con quello nuovo.
rubrica["Pippo"]="0912354";

Il metodo *Remove* rimuove un elemento.
rubrica.Remove("Pippo");

Il metodo *Clear* svuota la tabella.
rubrica.Clear();

La proprietà *Count* ritorna il numero di elementi.

```
int count=rubrica.Count;
Console.WriteLine("La rubrica contiene {0} nomi",count);
```

La proprietà *Keys* contiene la collezione delle chiavi presenti nella tabella, la proprietà *Values* contiene i valori.

```
Console.WriteLine("La rubrica contiene questi nomi:");
foreach(string str in rubrica.Keys)
        { Console.WriteLine(str);}
```

Le coppie nome valore sono memorizzate in un oggetto *Hashtable* come istanze della struttura *DictionarEntry*, che possiede i campi *Key* e *Value*, per cui per iterare tutti gli elementi con un ciclo *foreach* bisogna specificare come tipo degli elementi *DictionarEntry*.

```
Console.WriteLine("Contenuto della rubrica");
foreach(DictionaryEntry entry in rubrica)
        { Console.WriteLine("{0}\t\t{1}",entry.Key,entry.Value);}
```

Il metodo *Equals* confronta gli oggetti da memorizzare o già memorizzati.

Il metodo *GetHashCode* restituisce un valore intero.

```
public struct Point
{       public int x;
        public int y;
        public override int GetHashCode()
        { return x ^ y;}
}
```

Il metodo restituisce lo XOR fra le coordinate x e y.
È possibile richiamare il metodo direttamente.

```
public override int GetHashCode()
{ return base.GetHashCode(); }
```

CODA

La classe *Queue* è una struttura di tipo array.
Il metodo *Enqueue* effettua l'inserimento.

```
Queue codaMessaggi=new Queue();
codaMessaggi.Enqueue("Messaggio1");
codaMessaggi.Enqueue("Messaggio2");
codaMessaggi.Enqueue("Messaggio3");
foreach(string str in codaMessaggi)
        { Console.Write("{0}\t",str);}
```

Il metodo *Dequeue* effettua la rimozione.

```
string msg=(string)codaMessaggi.Dequeue();
Console.WriteLine(msg);
```

Il metodo *Peek* esamina l'elemento in testa alla coda senza rimuoverlo.

```
Console.WriteLine("Peek");
msg=(string)codaMessaggi.Peek();
Console.WriteLine(msg);
```

PILA

La classe *Stack* è una struttura di tipo array.
Il metodo *Push* effettua l'inserimento.

```
Stack pila=new Stack();
```

```
pila.Push("primo");
pila.Push("secondo");
pila.Push("terzo");
```

Stampare gli elementi di una pila, usando l'enumeratore.

```
IEnumerator e=pila.GetEnumerator();
while(e.MoveNext())
        { Console.WriteLine("{0}",e.Current); }
```

Il metodo *Pop* effettua la rimozione.

Il metodo *Peek* esamina l'elemento in testa alla pila senza rimuoverlo.

SEQUENZE DI BIT

La classe *BitArray* permette di gestire un vettore di bit, ognuno dei quali è rappresentato da un valore *true* o *false*.
Per costruire una *BitArray* ci sono diversi costruttori.

```
BitArray ba1=new BitArray(8);                   // lungo 8 con tutti i bit a false, default
BitArray ba2=new BitArray(new bool[]{true,false,true,false});// inizializzazione
BitArray ba3=new BitArray(8,true);              // lungo 8 con tutti i bit a true
```

È possibile modificare i bit con l'operatore d'indicizzazione.

```
ba1[0]=false;
ba1[1]=true;
```

Operazioni di algebra booleana AND, OR, XOR, NOT, per esempio il metodo OR, *ba1* è modificato perché contiene il risultato dell'operazione.

```
// ba1 0011
// ba2 1100
BitArray bOR=ba1.Or(ba3);
        Console.WriteLine("Dopo ba1 OR ba2");           // Dopo ba1 OR ba2
foreach(bool b in ba1)
        { Console.Write(b?"1":"0"); }                   // 1111
```

Un array statico d'interi è più efficiente in termini di prestazioni, ma la classe *BitArray* permette di gestire vettori di bit anche di lunghezza variabile, basta modificare la proprietà *Length* assegnando la nuova lunghezza, i bit aggiunti sono posti a *false*.

```
BitArray ba3=new BitArray(4,true);
foreach(bool b in ba3)
        Console.Write(b?"1":"0")                        // stampa 1111
Console.WriteLine("Raddoppio la lunghezza di ba3\n");
ba3.Length*=2;
foreach(bool b in ba3)
        Console.Write(b?"1":"0");                       // stampa 11110000
```

STRUCT

INTRODUZIONE

È simile ad una *class* in quanto può contenere campi, proprietà e metodi: la differenza è che la *struct* è di tipo valore quindi memorizzata sullo stack, mentre le istanze di una classe sono memorizzate nello heap, inoltre la *struct* non supporta l'ereditarietà.
Il compilatore genera sempre un costruttore di default che inizializza tutti i membri della *struct* al valore di default.

CLASS VS STRUCT

Quello che èvalido per le *class* è valido anche per le *struct*.

```
public struct ComplexNumber
{       public float Real;
        public float Complex
        public ComplexNumber(float real, float complex)
        {       this.Real = real;
                this.Complex = complex;
        }
        public static ComplexNumber operator+(ComplexNumber a, ComplexNumber b)
        { return new ComplexNumber(a.Real+b.Real, a.Complex+b.Complex); }
}
```

Classi.

- ✓ Possono definire data member, proprietà, metodi.
- ✓ Supportano costruttori e l'inizializzazione dei membri.
- ✓ Supportano il metodo *Finalize*.
- ✓ Supportano l'ereditarietà.
- ✓ RT.

Strutture.

- ✓ Possono definire data member, proprietà, metodi.
- ✓ Non supportano costruttori di default e l'inizializzazione dei membri.
- ✓ Non supportano il metodo *Finalize*.
- ✓ Non supportano l'ereditarietà.
- ✓ VT.

```
using System;
        class TheClass
                { public int x; }
        struct TheStruct
                { public int x; }
        class TestClass
        {
                public static void structtaker(TheStruct s)
                        { s.x = 5; }
                public static void classtaker(TheClass c)
                        { c.x = 5;  }
                public static void Main()
                {
                        TheStruct a = new TheStruct();
                        TheClass b = new TheClass();
                        Console.Clear();
```

```
            a.x = 1;
        b.x = 1;
        structtaker(a);
        classtaker(b);
        Console.WriteLine("a.x = {0}", a.x);
        Console.WriteLine("b.x = {0}", b.x);
        Console.ReadKey();
        }
}
```

FUNCTION/METODI

INTRODUZIONE

È una porzione di applicazione costituita da un insieme d'istruzioni che risolvono un problema eventualmente restituendo un risultato.
Può essere invocata in altri punti del codice.
Può a sua volta richiamare un'altra funzione/metodo, restando in attesa che questa termini: funzione/metodo invocante e funzione/metodo invocata oppure funzione/metodo chiamante e funzione/metodo chiamata.
Può a sua volta richiamare direttamente o indirettamente se stessa: funzione/metodo ricorsiva.

```
using System;
class hello
{     static int quadrato (int a)            // tipo del valore di ritorno e nome
      {   a=a*a;
          return a;                          // valore restituito
      }
      static void Main(string[] args)
       {   int n;
           Console.Clear();
          Console.Write("Inserisci il numero n: ");
          n = Convert.ToInt32(Console.ReadLine());
          Console.WriteLine("Il quadrato di {0} vale {1} ",n,quadrato(n));
           Console.ReadKey();
      }

}
```

Passaggio parametri
È necessario specificare il corretto numero ed il corretto tipo dei parametri formali al momento del richiamo della funzione/metodo.
Per default, in C# i parametri sono passati per valore.

Passare un VT by value
public void MyMethod(int a)
La variabile *a* è passata per valore: default; *a* è copiata all'interno della funzione/metodo e quindi riassegnarla, non influenzerà il chiamante.

```
using System;
class hello
{     static void quadrato (int a)
      {  a=a*a;  }
      static void Main(string[] args)
       {    int n;
            Console.Clear();
          Console.Write("Inserisci il numero n: ");
          n = Convert.ToInt32(Console.ReadLine());
          quadrato(n);
          Console.WriteLine("Il quadrato vale {0} ",n);
           Console.ReadKey();
      }
}
```

Inserisci il numero n: 2
Il quadrato vale 2

Passare un VT by reference

public void MyMethod(ref(out) int a)

La parola chiave *ref* (*out*) indica che la variabile *a* è passata by reference; *ref* (*out*) deve essere specificata anche nell'invocazione della funzione/metodo.

Un puntatore alla variabile *a* è copiato all'interno della funzione/metodo che potrà variare il valore di *a*.

```
using System;
class hello
{     static void quadrato (ref int a)
      {  a=a*a;  }
      static void Main(string[] args)
       {    int n;
            Console.Clear();
           Console.Write("Inserisci il numero n: ");
           n = Convert.ToInt32(Console.ReadLine());
           quadrato(ref n);
           Console.WriteLine("Il quadrato vale  {0} ",n);
            Console.ReadKey();
     }
}
```

Inserisci il numero n: 2
Il quadrato vale 4

```
using System;
class hello
{     static void quadrato (int a, out int b)
      {  b=a*a; }
      static void Main(string[] args)
       {    int n,q;
            Console.Clear();
           Console.Write("Inserisci il numero n: ");
           n = Convert.ToInt32(Console.ReadLine());
           quadrato(n,out q);
           Console.WriteLine("Il quadrato vale  {0} ",q);
            Console.ReadKey();
     }
}
```

Inserisci il numero n: 2
Il quadrato vale 4

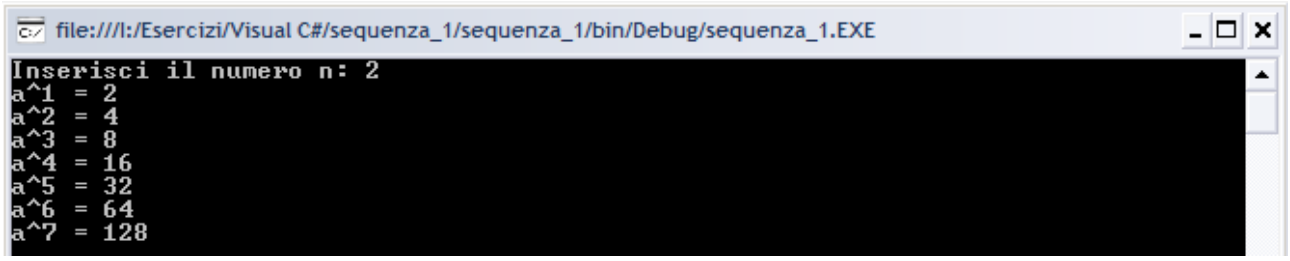
A volte è utile progettare una funzione/metodo con più di un valore di ritorno, *out* permette di specificare che quel parametro è in uscita; quindi non necessita d'inizializzazione prima dell'invocazione della funzione/metodo stessa, ma richiede un'assegnazione all'interno della funzione/metodo.

```
using System;
class hello
{     static void potenze (int a, out int d, out int t, out int q)
      {  d=a*a; t=d*a; q=t*a; }
      static void Main(string[] args)
       {    int n, due,tre,quattro;
            Console.Clear();
           Console.Write("Inserisci il numero n: ");
           n = Convert.ToInt32(Console.ReadLine());
```

```
        potenze (n, out due,out tre,out quattro);
        Console.WriteLine("a^2 = {0} ",due);
        Console.WriteLine("a^3 = {0} ",tre);
        Console.WriteLine("a^4 = {0} ",quattro);
         Console.ReadKey();
    }
}
Inserisci il numero n: 2
a^2 = 4
a^3 = 8
a^4 = 16
```

Se si volesse calcolare la potenza di un numero fino ad un valore arbitrario dell'esponente, la funzione/metodo dovrebbe avere una lista lunghissima di parametri, la soluzione è la parola chiave *params*, la quale specifica che il parametro formale di quella funzione/metodo è un array di parametri formali.

```
using System;
class hello
{     static void potenze (int a, params int[] pot)
      {  foreach (int potenza in pot)
        {   int p=(int)Math.Pow(a,potenza);
            Console.WriteLine("a^{0} = {1} ",potenza,p);
        }
    }
     static void Main(string[] args)
      {int n;
       Console.Clear();
             Console.Write("Inserisci il numero n: ");
             n = Convert.ToInt32(Console.ReadLine());
             potenze (n,1,2,3,4,5,6,7);
        Console.ReadKey();
    }
}
```



Passare un RT by value

public void MyMethod(DataTable dt)

La variabile *dt* è passata per valore: default; la *DataTable* non è copiata perché non è un VT, ma al suo posto è copiato il reference (puntatore).

La *DataTable* non può essere riassegnata:

dt = new Data Table(); *// non influirebbe il chiamante*

La *DataTable* può essere cambiata solo se uno dei suoi metodi/proprietà ne modifica il contenuto; per esempio *dt.Clear().*

Passare un RT by reference

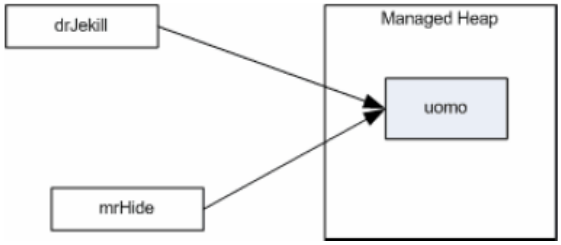
public void MyMethod(ref(out) DataTable dt)

La parola chiave *ref* (*out*) indica che la variabile *dt* è passata by reference.

La *DataTable* non è copiata perché non è un VT, ma al suo posto è copiato il reference

(puntatore).
La *DataTable* può essere riassegnata.
dt = new Data Table("NewTable");
La classe può essere cambiata anche tramite i suoi metodi/proprietà che ne modificano il contenuto; per esempio: *dt.Clear().*

RT
La copia di un tipo *reference* implica la duplicazione del solo *reference*.
Le modifiche su due *reference* modificheranno l'oggetto cui puntano.
Esempio della classe *Uomo*.
Uomo drJekill=new Uomo();
Uomo mrHide=drJekill;
La variabile *drJekill* fa riferimento ad un nuovo oggetto di tipo *Uomo*, la variabile *mrHide*, per come è inizializzata, fa riferimento alla stessa copia di *Uomo*: in pratica in RAM in un'area chiamata **managed heap**, si ha una sola copia della classe *Uomo*.



Si verifica quanto detto con la seguente applicazione.

```
using System;
class Uomo
{
public float fTemperatura;
public Uomo()
{
fTemperatura=27.7f;
}
}
class TestRiferimenti
{
static void Main()
{       Console.Clear();
        Uomo drJekill=new Uomo();
        Uomo mrHide=drJekill;
        Console.WriteLine("Temperatura di drJekill:  "+drJekill.fTemperatura);
        Console.WriteLine("Temperatura di mrHide:    "+mrHide.fTemperatura);
        drJekill.fTemperatura=37.7f;
        Console.WriteLine("Temperatura di drJekill:  "+drJekill.fTemperatura);
        Console.WriteLine("Temperatura di mrHide:    "+mrHide.fTemperatura);
        Console.ReadKey();
}
}
```

C# permette di considerare il valore di qualsiasi tipo come un oggetto, in altre parole ogni tipo deriva direttamente o indirettamente dalla classe *Object*.

METODO MAIN

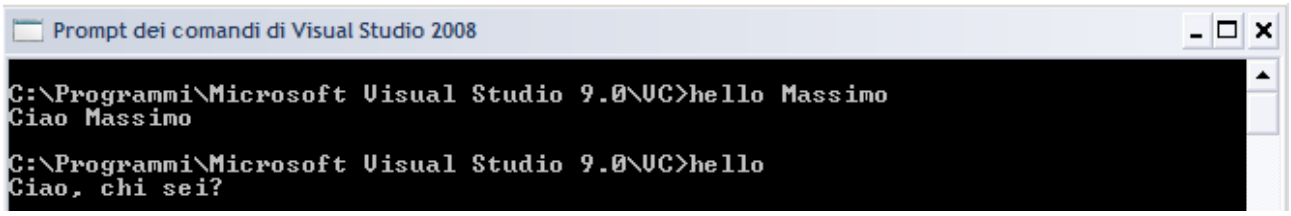
È implementato in modi diversi.

1. Restituisce un valore intero, codice di terminazione.
2. Restituisce nessun valore.
3. Accetta parametri in input dalla linea di comando.

```
static void Main() {…}                  // nessun argomento e nessun valore
static void Main(string[] args) {…}     // argomenti e nessun valore
static int Main() {…}                   // nessun argomento e valore intero
static int Main(string[] args) {…}      // argomenti e valore intero
```

Esempio.

```
using System;
class hello
   {    static void Main(string[] args)
        {   if(args.Length>0) { Console.WriteLine("Ciao {0}",args[0]); }
            else Console.WriteLine("Ciao, chi sei?");
            Console.ReadKey();
   }
}
```



Primo caso: l'array *args* contiene la stringa "*Massimo*", quindi la proprietà *Length* ritorna 1, per cui il metodo *WtiteLine* stampa "*Ciao*" seguito da *args[0*].

MULTITHREAD

INTRODUZIONE

L gestione dei processi è gestita dal CLR, l'accesso a queste funzionalità è semplice, il namespace *System.Threading* fornisce metodi per creare, eseguire, gestire e sincronizzare thread.

Esempio.

```
using System;
using System.Threading;
class App
{   // Codice del metodo eseguito nel primo Thread
    public static void MyThreadMethod1()
    { for (int i = 0; i <= 4; i++)   { Console.WriteLine("Thread 1");  }  }
    // Codice del metodo eseguito nel secondo Thread
    public static void MyThreadMethod2()
    { for (int j = 0; j <= 4; j++) { Console.WriteLine("Thread 2"); }
}
static void Main(string[] args)
{      // Crea il primo thread
       Thread thread1 = new Thread(new ThreadStart(MyThreadMethod1));
       // Avvia il primo thread
       thread1.Start();
       // Crea il secondo thread
       Thread thread2 = new Thread(new ThreadStart(MyThreadMethod2));
         // Avvia il secondo thread
       thread2.Start();
       // Durante l'esecuzione dei 2 thread esegue un terzo ciclo
       for ( int i = 0; i<=5; i++ )
         { Console.WriteLine("Main Thread");   }
       Console.ReadKey();
       }
}
```

```
Thread 1
Thread 1
Thread 1
Thread 1
Thread 1
Main Thread
Thread 2
Thread 2
Thread 2
Thread 2
Thread 2
Main Thread
Main Thread
Main Thread
Main Thread
Main Thread
```

REMOTING

INTRODUZIONE

Fornisce una struttura che consente agli oggetti d'interagire tra di loro anche se risiedono su diversi application domain.
La comunicazione avviene attraverso canali.
Prima che i messaggi tra le applicazioni remote siano inviati sui canali sono codificati nei seguenti modi.

- ✓ In binario per le applicazioni critiche.
- ✓ In **XML** (*eXtensible Markup Language***)** dove è essenziale interoperabilità, i messaggi XML utilizzano sempre **SOAP** (*Simple Object Access Protocol*).

I servizi di remoting offerti dal framework nascondono al programmatore la complessità dei metodi di chiamata remoti.
Quando un client attiva un oggetto remoto ne riceve un proxy con il quale interagisce.
Per prima cosa si registra un canale di comunicazione (HTTP, TCP), poi diventa possibile registrare gli oggetti remoti.
Per la registrazione di un oggetto remoto servono i seguenti dati.

- ✓ Il nome del tipo di oggetto remoto.
- ✓ L'oggetto **URI** (*Uniform Resource Identifier*) che i client utilizzeranno per individuare l'oggetto.
- ✓ La modalità oggetto richiesta per l'attivazione da server, che può essere SingleCall o Singleton.

SigleCall
Un oggetto SingleCall crea un'istanza di classe per ogni chiamata del client, anche se le chiamate provengono dallo stesso client.
L'invocazione successiva, sarà sempre servita da una differente istanza del server anche se la precedente non è stata ancora riciclata dal sistema.

Singleton
Un oggetto Singleton invece non presenta mai più di una istanza contemporaneamente. Se una istanza è già esistente la richiesta è soddisfatta da quella istanza.
Se l'istanza non esiste, alla prima richiesta è creata dal server e tutte le successive richieste sono soddisfatte da quella istanza.

Esempio, progetto composto dai seguenti file, inserire all'interno di *Esplora soluzioni* cartella *Riferimenti* la DLL seguente *System.Runtime.Remoting.dll.*
File program.cs

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
namespace RemotingSamples
{  public class Sample
{  static int Main(string [] args)
{     // Crea e registra un nuovo canale Tcp
      TcpChannel chan = new TcpChannel(8085);
      ChannelServices.RegisterChannel(chan);
      // Il metodo RegisterWellKnownServiceType permette di registrare l'oggetto per la
      // futura attivazione [PARAMETRI (Tipo, URI, metodo di attivazione)]
```

```
    RemotingConfiguration.RegisterWellKnownServiceType (Type.GetType("Esempio
Remoting. HelloServer,object"),
    "SayHello", WellKnownObjectMode.SingleCall);
    Console.ReadKey();return 0;
   }
  }
}
```

File objects.cs

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
namespace RemotingSamples
{
  public class HelloServer : MarshalByRefObject
  { public HelloServer()
    { Console.WriteLine("Il server è attivato."); }
    public String HelloMethod(String name)
    { Console.WriteLine("Hello.HelloMethod : {0}", name);
      return "Hi there " + name;
    }
  }
}
```

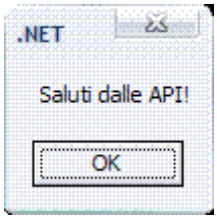
File client.cs

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
namespace RemotingSamples
{
   public class Client
   {
       static int Main(string[] args)
      {
          TcpChannel chan = new TcpChannel();
          ChannelServices.RegisterChannel(chan);
          // Il client localizza l'oggetto remoto passando tipo e URL
          HelloServer obj =
(HelloServer)Activator.GetObject(typeof(RemotingSamples.HelloServer) ,
"tcp://localhost:8085/SayHello");
          if (obj == null)
             System.Console.WriteLine("Non riesco a trovare il server locale.");
          else Console.WriteLine(obj.HelloMethod("Pippo"));
          Console.ReadKey(); return 0;
      }
   }
}
```

ACCESSO ALLE API DI WINDOWS

INTRODUZIONE

```
using System;
using System.Runtime.InteropServices;
class MainApp
{
   [DllImport("user32.dll", EntryPoint = "MessageBox", SetLastError = true, CharSet =
CharSet.Auto)]
   public static extern int MessageBox(int hWnd, String strMessage, String strCaption, uint
uiType);
   static void Main(string[] args)
   {  MessageBox(0, "Saluti dalle API!", ".NET", 0);
   Console.ReadKey();
   }
}
```



CLASSI

Per progettare la classe *Veicolo*, bisogna scrivere il codice per definire la classe stessa e i suoi membri; per membri della classe s'intendono i suoi campi che mantengono lo stato dell'oggetto ed i suoi metodi che definiscono il suo comportamento.
Per creare nuove classi, C# fornisce la parola chiave *class*.
[modificatore] class NomeClasse [:ClasseBase]
{ [membri della classe] }
La sintassi crea un nuovo tipo o meglio la sua rappresentazione, per creare un'istanza di esso, in pratica un oggetto di classe *NomeClasse* si usa la parola chiave *new*.
NomeClasse oggetto=new NomeClasse();
La definizione dei membri della classe avviene all'interno delle parentesi graffe che delimitano il corpo della classe stessa.
Una classe può essere parziale, ossia essere definita in file differenti.

File part1.cs
```
public partial class MyClass
{ … }
```

File part2.cs
```
public partial class MyClass
{ … }
```

Esempio.
```
using System;
public class Test : IDisposable
{   // metodo costruttore per la classe
    public Test()
    { Console.WriteLine("Chiamo il metodo costruttore....");   }
    // metodo Finalization  per la classe
    ~Test()
    { Console.WriteLine("Chiamo il metodo Finalization ....");  }
    public void Hello()
    { Console.WriteLine("Hello, world!");  }
    //metodo Dispose  per la classe
    public void Dispose()
    {   Console.WriteLine("Chiamo il metodo Dispose ...");
        // Soppressione della chiamata al metodo Finalization
        GC.SuppressFinalize(this);
    }
}
public class TestMain
{   static void Main(string[] args)
    {   using (Test t = new Test())
        { t.Hello(); }
        Console.ReadKey();
    }
}
Chiamo il metodo costruttore....
Hello, world!
Chiamo il metodo Dispose ...
```

Modificatori di accesso
Una dichiarazione di classe può essere dichiarata con uno dei seguenti modificatori.

Modificatore d'accesso	Descrizione
public	I membri pubblici sono visibili a qualunque altra classe e relativi membri.
protected	I membri protected sono accessibili oltre che dalla classe stessa, anche dalle classi figlie derivate da essa.
private	I membri privati sono visibili solo all'interno della classe stessa.
internal	I membri internal sono visibili all'interno dello stesso assembly in cui è definita la classe.
protected internal	I mebri protected internal sono accessibili alle classi dello stesso assembly della classe, ed alle classi derivate da essa (anche definite in altri assembly).

private
Non si può usarlo per una classe o ad elementi contenuti in un *namespace* perché non avrebbe senso in quanto in questo modo la classe non sarebbe utilizzabile da nessuno, ma è possibile dichiarare una classe *private* se è innestata in un'altra.
Applicato ad un membro di una classe, ad esempio, ad un suo campo, indica che questo membro è visibile e utilizzabile solo da istanze della classe stessa, mentre il suo accesso sarà impedito ad oggetti di classi diverse.

```
class MiaClasse
{ private int unValore; }
```

Non ha nessun utilizzo, allora bisogna aggiungere un metodo pubblico in questo modo.

```
class MiaClasse
{       private int unValore;
        public int GetValore()
        { return unValore; }
}
```

Composizioni di classi
Un oggetto è un agglomerato di diversi oggetti, ognuno di classe diversa; questi oggetti saranno istanze di classi implementate da noi o da altri ed inseriti come campi della classe contenitrice.

Esempio, implementare una classe *Motore* ed una classe *Serbatoio*.

```
class Motore
{       private int cilindrata;
        public Motore(int cc)
        { cilindrata=cc;}
        public void Accendi()
        { Console.WriteLine("Vrooom!!");}
}
class Serbatoio
{       float capacita;
        float carburante;
        public Serbatoio(float cap)
        {       capacita=cap;
                carburante=0.0f;
        }
        public float Capacità
        {       get
```

```
            { return this.capacita;}
        }
        public float Carburante
        {       get
                { return carburante;}
                set
                { if(value>0)
                {
                        if(carburante+value>capacita) carburante=capacita;
                        else carburante+=value;
                }
                }
        }
}
```

Implementare la classe *Auto*.

```
class Auto
{       private int numeroRuote;
        private string modello;
        private Motore motore;
        private Serbatoio serbatoio;
        public Auto(string mod, int cilindrata)
        {       numeroRuote=4;
                modello=mod;
                motore=new Motore(cilindrata);
                serbatoio=new Serbatoio(50.0f);
        }
        public bool Accensione()
        {       if(serbatoio.Carburante>0)
                { this.motore.Accendi();
                 return true;
                }
                else return false;
                }
        public void RiempiSerbatoio()
        { serbatoio.Carburante=serbatoio.Capacità; }
}
```

Classi nested

È possibile dichiarare ed implementare classi all'interno di altre dichiarazioni di classi, in pratica nidificare le dichiarazioni di classe.

La classe annidata è definita all'interno di un'altra classe e la definizione del tipo interno è vincolata al *namespace* della classe esterna.

```
class Auto
{       class Motore
        {…}
        class Serbatoio
        {…}
        //corpo della classe Auto
}
```

La classe annidata è in tutto e per tutto un membro della classe in cui è dichiarata e, dunque, è possibile applicare ad essa i modificatori di membro.

```
class MiaClasse
{       public MiaClasse()
        { }
```

```
        //classe innestata privata
        private class ClassePrivata
        { }
        //classe innestata pubblica
        public class ClassePubblica
        { }
}
class TestClass
{
        public static void Main()
        {       MiaClasse obj=new MiaClasse();
                MiaClasse.ClassePubblica obj2=new MiaClasse.ClassePubblica();
                //la seguente linea darebbe errore
                //MiaClasse.ClassePrivata obj2=new MiaClasse.ClassePrivata();
        }
}
```

Campi di classe

Rappresentano i membri che contengono i dati di un'istanza della classe stessa.
Un campo può essere di tipo qualsiasi,: oggetto o di tipo valore.

```
public class Veicolo
{
        public int ruote;
        public float velocita;
        public int direzione=90;
        private bool motoreAcceso=false;        // campi valore inizializzato
        private AutoRadio radio;                // campo di classe
}
```

Per istanziare l'oggetto si usa un metodo della classe, chiamato costruttore, anche quando non è esplicitamente definito, ogni classe ne fornisce uno standard che è il costruttore di default e non accetta nessun parametro in ingresso.

```
Veicolo auto=new Veicolo();
```

Per accedere ai membri di una classe si usa l'operatore dot.

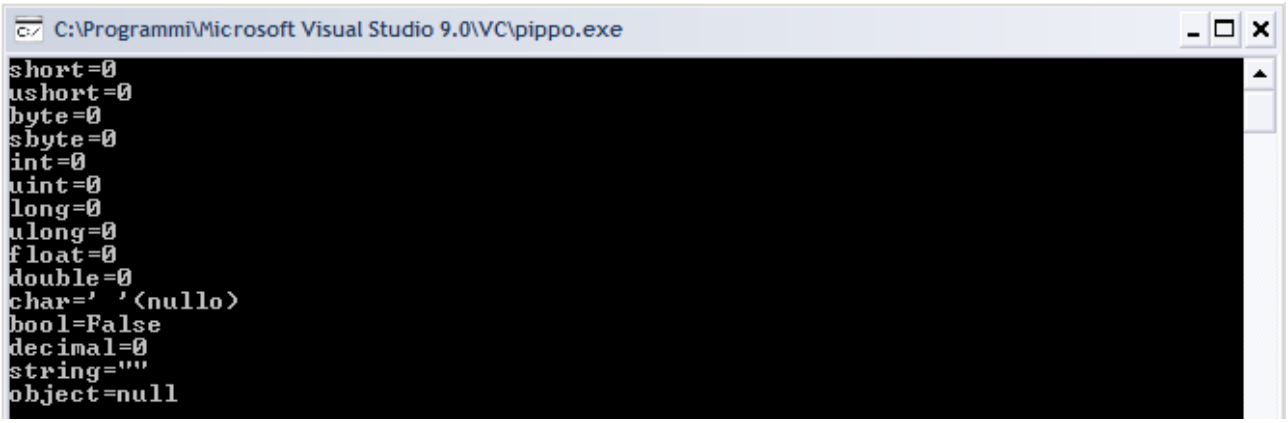
```
auto.ruote=4;
```

I campi di una classe assumono valori di default se non sono inizializzati, questo avviene per assicurarsi che questi campi posseggano comunque un valore dopo la creazione di un oggetto.
Valori assunti dalle variabili di campo dei tipi primitivi.

Tipo del campo	Valore di default
byte, sbyte	(byte)0
short, ushort	(short)0
int, uint	0
long, ulong	0L
float	0.0F
double	0.0D
char	'\u0000' (carattere null)
bool	false
decimal	0
string	"" (stringa vuota)
tipi riferimento	null

Progettare una classe che possegga i campi di tutti i tipi e stamparne i valori, senza averli inizializzati.

```
using System;
class DefaultVal
{       public short s;
        public ushort us;
        public byte b;
        public sbyte sb;
        public int i;
        public uint ui;
        public long l;
        public ulong ul;
        public float f;
        public double d;
        public char c;
        public bool bo;
        public decimal dec;
        public string str;
        public object obj;
        static void Main(string[] args)
                {Console.Clear();
                DefaultVal test=new DefaultVal();
                Console.WriteLine("short={0}",test.s);
                Console.WriteLine("ushort={0}",test.us);
                Console.WriteLine("byte={0}",test.b);
                Console.WriteLine("sbyte={0}",test.sb);
                Console.WriteLine("int={0}",test.i);
                Console.WriteLine("uint={0}",test.ui);
                Console.WriteLine("long={0}",test.l);
                Console.WriteLine("ulong={0}",test.ul);
                Console.WriteLine("float={0}",test.f);
                Console.WriteLine("double={0}",test.d);
                if(test.c=='\u0000') Console.WriteLine("char='\u0000'(nullo)");
                Console.WriteLine("bool={0}",test.bo);
                Console.WriteLine("decimal={0}",test.dec);
                Console.WriteLine("string=\"{0}\"",test.str);
                if(test.obj==null) Console.WriteLine("object=null");
                Console.ReadKey();
                }
}
```



Campi costanti

È possibile che il valore di un campo di una classe debba rimanere costante durante

l'esecuzione dell'applicazione.
```
class Veicolo
{ private const LUNGHEZZA_TARGA=7;
  private string targa;
}
```
Il campo *LUNGHEZZA_TARGA* non può essere variato a run-time, inoltre deve essere inizializzato con un valore costante, quindi non è possibile assegnargli il valore di un'altra variabile.

Campi a sola lettura

Anziché assegnare un valore costante a compile-time, a volte è utile assegnare un valore a run-time, come risultato di un'espressione, ma non variarlo più, in pratica creare un campo scrivibile una volta sola e poi usarlo solo in lettura; per fare questo si usa la parola chiave *readonly*, ma può essere assegnato solo all'interno dei costruttori della classe, in altre parole al momento di creazione dell'oggetto.
```
class MiaClasse
{ private readonly OrarioDiCreazione; }
```

Metodi e proprietà

Gli oggetti istanziati in un'applicazione devono comunicare tra di loro, per esempio l'oggetto *Veicolo*, come fa ad andare in moto, accelerare e frenare.
Allora, la classe *Veicolo* deve mettere a disposizione degli altri oggetti, delle funzioni richiamabili dall'esterno, in modo da poter spedire alle sue istanze dei messaggi.
Queste funzioni sono chiamate metodi della classe e sono così definite.
```
modificatori tipo_ritorno (lista_parametri)
{ // corpo del metodo }
```
NomeMetodo è quello con cui sarà richiamato all'interno del codice; il *tipo_ritorno* determina il valore di ritorno con l'istruzione *return*; *lista_parametri* con i rispettivi tipi sono usati nel corpo del metodo stesso.
```
public class Veicolo
{
        public int ruote;
        public float velocita;
        public int direzione;
        private bool motoreAcceso;
        public float GetVelocita(void)
        {
                return velocita;
        }
        public void AccendiMotore(void)
        {
                if(!motoreAcceso) motoreAcceso=true;
        }
        public void SpegniMotore(void)
        {
                if(motoreAcceso) motoreAcceso=false;
        }
}
```
Per usare un metodo, bisogna chiamarlo tramite un oggetto della classe che lo definisce.
```
nomeoggetto.NomeMetodo(argomenti);
```

Esempio, chiamare il metodo *AccendiMotore(void)* su un'istanza della classe *Veicolo*, prima bisogna costruire l'istanza e poi spedirgli il messaggio di accensione.
```
public class TestVeicolo
```

```
{
        static void Main()
        {
        Veicolo auto=new Veicolo();          //costruzione di veicolo
        auto.AccendiMotore();                //chiamata del metodo AccendiMotore();
        }
}
```

In C# i metodi devono essere membri di una classe, non esistono funzioni globali.
Un tipo particolare di metodi sono le proprietà di una classe: le proprietà sono dei metodi che permettono l'accesso in R/W agli attributi di una classe.
Le proprietà permettono di accedere ad un campo in maniera intuitiva, ad esempio con un'assegnazione, così come si farebbe con un campo pubblico, ma con la differenza che le proprietà non denotano delle locazioni di memorizzazione dei dati, come i campi, e permettono contemporaneamente di fare controlli sul valore che si vuole assegnare o leggere.
Inoltre, le proprietà mantengono il codice coerente e compatto visto che la lettura e la scrittura sono implementate in blocchi adiacenti.

Esempio, aggiungere alla classe *Veicolo* la proprietà *Direzione*, per impostare la direzione di moto che varia da 0 a 359 gradi.

```
public int Direzione
{
        get                                  // lettura del campo, restituisce un valore
        {                                    // del tipo dichiarato dalla proprietà
                return direzione;
        }
        set                                  // scrittura del campo, è sempre passato
        {                                    // un parametro implicito di nome value
                if (direzione<0) direzione+=360;
                direzione=value%360;
        }
}
```

Una proprietà può provvedere accessibilità in lettura (*get*) scrittura (*set*) o entrambi.
Si può usare una proprietà per ritornare valori calcolati o eseguire una validazione.

```
Proprietà tradizionale          Indexer
public class MyClass            public class MyClass
{                               {
private string_Name;            private string[] val=new string[] {"uno", "due", "tre"};
public string Name              public string this [long i]
{                               {
get {return_Name};              get {return val [i]};
set{_Name = value};             set {val [i] = value};
}                               }
MyClass c = new MyClass();      MyClass c = new MyClass();
c.Name = "Raffaele";            Console.Writeline (c[1]);   //due
```

Accessibilità differenziata nelle proprietà e indexer.

- ✓ *get* e *set* possono avere accessibilità diversa.
- ✓ La loro accessibilità deve essere più restrittiva di quella indicata nella proprietà.
- ✓ Non può essere usato nelle interfacce.

```
public string Nome
{       get return_Nome;
        private set _Nome = value;
}
```

Overloading dei metodi

Una classe può avere più metodi con lo stesso nome, ma con numero e/o tipi di parametri diversi, non possono esistere due metodi che differiscono del solo parametro di ritorno, oppure che differiscono per un parametro dichiarato *ref* e poi *out*.

public int quadrato(int a)
*{ return a*a; }*

Questo metodo non accetta un parametro attuale in virgola mobile.

double d=quadrato(2.1);

Per due motivi.

1. Il metodo *quadrato* non accetta come parametro un tipo *double*.
2. Il tipo di ritorno è *int*.

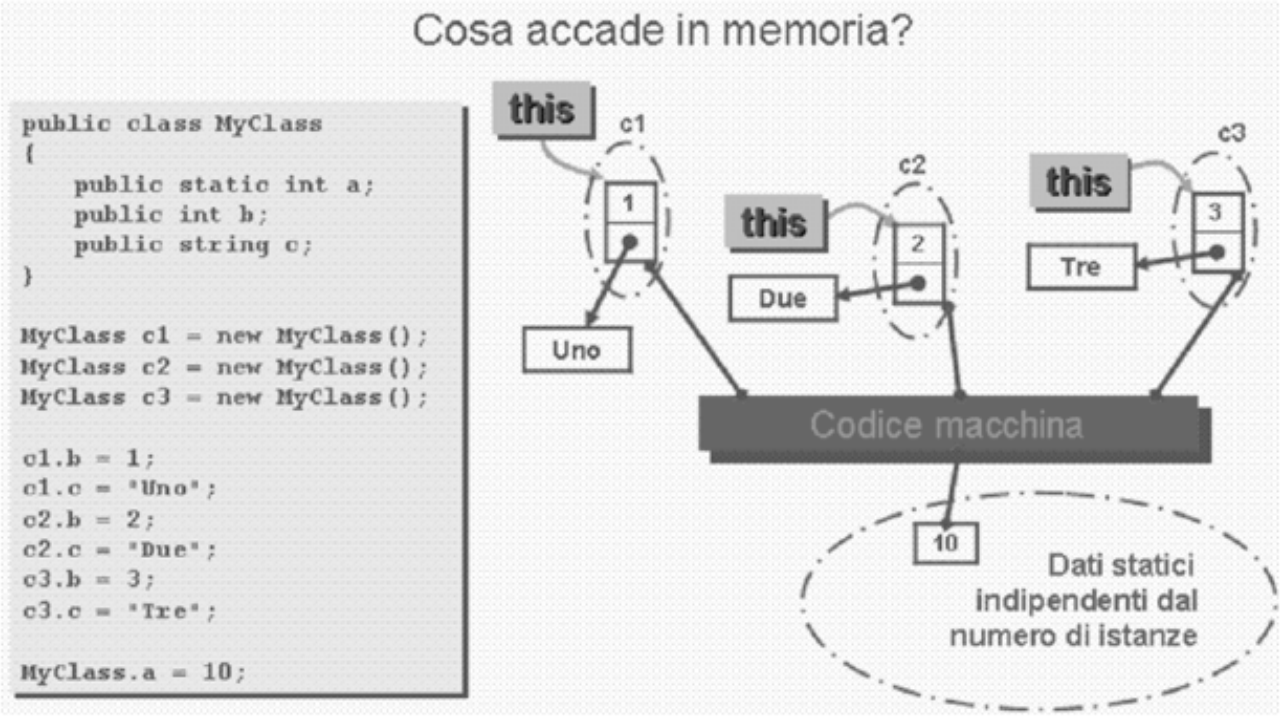
public double quadrato(double d)
*{ return d*d; }*

Con questa dichiarazione, il compilatore usa il metodo appropriato: questa tecnica è chiamata overloading, in altre parole sovraccarico dei metodi.

La parola chiave *this*

I dati della classe sono marcati con *static* e descrivono le informazioni comuni a tutti gli oggetti di una classe.
Ciò che è marcato *static* può essere utilizzato senza la necessità d'istanziare oggetti.

```
public class MyClass
{
        public static int a;
        public int b;
}
        // …
MyClass c1 = new MyClass();
MyClass c2 = new MyClass();
```



La parola chiave *this* è usata all'interno di un qualsiasi metodo di una classe per riferirsi all'istanza stessa in quel momento in esecuzione, è in pratica come se l'oggetto attivo voglia usare se stesso, specificandolo in modo esplicito per evitare fraintendimenti.

```
Public class MyClass
{
        public int a;
        public void Abc (int a)
```

```
        {
                this.a = a;
        }
}
```

Costruttore

Il costruttore di una classe è un metodo, quindi può avere parametri, che permette di stabilire come un oggetto dev'essere creato, in pratica di controllare la sua inizializzazione. È obbligatorio che il costruttore si chiami come la classe e che non sia specificato nessun tipo di ritorno, neppure *void*, perché quello che restituisce un costruttore è proprio l'oggetto che deve costruire.

Il costruttore di default è il costruttore che non prende alcun parametro.

```
Public class Persona
{       public string Nome;
        public int Eta;
        public Persona(string Nome, int Eta)
        {       this.Nome = Nome
                this.Eta = Eta;
        }
}
Public class Persona
{       public string Nome;
        public int Eta;
        public Persona()
        {       this.Nome = "Carlo";
                this.Eta = 25;
        }
}
```

Il costruttore può avere una visibilità ristretta, in questo caso l'istanza dovrà essere creata da un metodo o da un'altra classe: *factory*.

Può esistere un costruttore statico che permette di definire lo stato dei membri statici, è chiamato in modo implicito e quindi è senza parameri e senza parola chiave di accessibilità, subito prima che qualsiasi membro statico o d'istanza sia usato.

```
public class Persona
{       // …
        private Persona ()
        { // … }
        public Persona CreatePersona ()
        { // … }
}
public class Persona
{       // …
        static Persona ()
        { // … }
}
```

Distruttore
È un metodo invocato quando un oggetto è rimosso dalla memoria, ma non è immediato.
Il GC è preposto alla distruzione non deterministica degli oggetti nel managed heap e rende disallocabili gli oggetti di cui l'applicazione non mantiene più alcun reference diretto o indiretto.
L'implementazione di un distruttore è un metodo con lo stesso nome della classe, preceduto dal carattere ~ senza parametri.
~NomeClasse()
{ //effettuo la pulizia delle risorse }
Il distruttore è un alias per il metodo *Finalize* che è chiamato dal GC quando e se vorrà.
Se non esiste, l'oggetto non va nella coda di finalizzazione.
Il pattern *Idisposable* è la soluzione ideale per poter disallocare risorse.

```
public class Persona
{
        Persona ( )
        {  // … }
}
```

Membri *static*
Quando si crea un'istanza di una classe, in altre parole quando si crea un oggetto, esso possiede uno stato proprio, in pratica un proprio insieme dei campi definiti dalla classe di appartenenza.

```
public class Persona
{       private int eta;
        private string nome;
        public string Nome
        {       get{return nome;}
                set{nome=value;}
        }
}
```

Ogni oggetto della classe *Persona* possiede un proprio nome ed una propria età, per accedere a questi campi bisogna creare delle istanze della classe.

```
Persona p1=new Persona();
Persona p2=new Persona();
p1.nome="antonio";
p2.nome="caterina";
```

In alcuni casi è necessario avere dei membri che non siano associati ad ogni singola istanza, ma che siano membri comuni a tutte le istanze di una classe: sono i membri statici, chiamati anche membri di classe, al contrario dei precedenti che sono membri d'istanza.

Campi *static*
Se si volesse contare il numero d'istanze della classe *Persona* create durante il run dell'applicazione.

```
public class Persona
{public static int numeroPersone;                // contatore
// …
}
```

L'accesso ai membri *static* non avviene mediante un'istanza, in quanto sono membri della classe, ma tramite il nome della classe.

```
Persona.numeroPersone=1;
```

In conclusione, la notazione è la seguente.
NomeClasse.nomeMembroStatic

L'unica eccezione è quando si accede ad un membro *static* dall'interno di un'istanza della classe stessa, allora si può omettere il nome della classe.
I campi *static* di una classe sono inizializzati la prima volta che si crea un oggetto della classe stessa, oppure la prima volta che è usato un membro *static* della classe, e se non inizializzati in modo esplicito, essi assumono i valori di default.

```
public class Persona
{
        public static int numeroPersone;
        public Persona()
        {
                Persona.numeroPersone++;            // una persona creata
        }
        ~public Persona()
        {
                Persona.numeroPersone--             ;// una persona distrutta
        }
        static void Main()
        {
        Persona p1=new Persona();
        Console.WriteLine("1. Numero di persone viventi: {0}",Persona.numeroPersone);
        Persona p2=new Persona();
        Console.WriteLine("2. Numero di persone viventi: {0}",Persona.numeroPersone);
        }
}
```

Metodi *static*
È un metodo che appartiene ad un'intera classe e non ad una singola istanza, per esempio il metodo seguente.
Console.WriteLine();
Si può quindi progettare un metodo *static* per accedere ad un campo *static* di una classe, ma all'interno di questo metodo si può solo accedere ai membri *static* della classe o di altre classi, quindi non si può usare la parola chiave *this*.

Esempio, modificare la classe *Persona* rendendo *private* il campo *static numeroPersone* ed aggiungere un metodo che restituisca il valore del campo per evitare che dall'esterno si possa dare un valore arbitrario al campo con un'assegnazione tipo la seguente.
Persona.numeroPersone=100000;
Metodo per la lettura del campo.

```
public static int GetNumeroPersone()
{ return Persona.numeroPersone;
// non è possibile accedere ad un membro non static
}
```

Costruttori *static*
Non possono avere parametri e sono invocati una sola volta, prima dell'uso della classe.

```
public class Persona
{
        public static int numeroPersone;
        static Persona()
        {
                numeroPersone=0;
        }
        public Persona()
        { }
```

}
È possibile implementare contemporaneamente anche un costruttore non *static* senza parametri anche se esiste già un costruttore *static*: non c'è conflitto perché il costruttore *static* non può essere chiamato dal codice, ma è invocato solo dal CLR.

Overloading degli operatori
È possibile aggiungere ad una classe un altro tipo di membri, in altre parole dei membri di overload degli operatori.
Questa operazione aggiunge la possibilità di utilizzare i classici operatori con le istanze della classe.

Esempio, classe che rappresenta i numeri complessi: a + ib.

```
Class NumeroComplesso
{       private float re;
        private floate im;
        public float Re
        {       get
                { return re;}
                set
                { re=value;}
        }
        public float Im
        {       get
                { return im; }
                set
                { im=value; }
        }
        //formatta un numero complesso nella maniera classica a+ib
        public override string ToString()
        { return re+"+i"+im; }
}
```

Per i numeri complessi sono definite le operazioni: +, -, * e /.

```
public static NumeroComplesso Somma(NumeroComplesso z1, NumeroComplesso z2)
{ NumeroComplesso s=new NumeroComplesso();
  s.Re=z1.Re+z2.Re;
  s.Im=z1.Im+z2.Im;
  return s;
}
```

z1 e *z2* sono due istanze di *NumeroComplesso*, si calcola la somma in questo modo.

```
NumeroComplesso somma=NumeroComplesso.Somma(z1,z2);
```

L'overload, invece, permette la seguente sintassi.

```
NumeroComplesso somma=z1+z2;
```

Questo è possibile perché un operatore non è altro che un metodo della classe che ha come nome il simbolo dell'operatore stesso preceduto dalla parola chiave *operator*, che indica che si sta sovraccaricando un operatore, inoltre è necessario che il metodo sia *static*.

```
public static NumeroComplesso operator +(NumeroComplesso z1, NumeroComplesso z2)
{ NumeroComplesso s=new NumeroComplesso();
  s.Re=z1.Re+z2.Re;
  s.Im=z1.Im+z2.Im;
  return s;
}
```

Esempio, fare la somma: 5 + (3 + i4) = 8 + i4, il risultato è ancora un numero complesso, basta implementare un altro overload dell'operatore +.

```
public static NumeroComplesso operator +(double d, NumeroComplesso z)
{ NumeroComplesso s=new NumeroComplesso();
  s.Re=d+z.Re;
  s.Im=z.Im;
  return s;
}
```

Se si volesse fare questa somma, invece: (6 + i) +7, è necessario un secondo overload con i parametri scambiati di posto.

```
public static NumeroComplesso operator +(NumeroComplesso z,double d)
{ NumeroComplesso s=new NumeroComplesso();
  s.Re=d+z.Re;
  s.Im=z.Im;
  return d*z;
}
```

Esempio, fare il prodotto scalare di due vettori: [a,b,c] * [d,e,f] = a*d + b*c + c*f

```
public static double operator * (Vettore x,Vettore y)
{ return x.a*y.a + x.b*y.b + x.c* y.c; }
```

Gli operatori unari sovraccaricabili sono i seguenti.

```
+ - ! ~ ++ -- true false
```

Gli operatori binari sovraccaricabili sono i seguenti.

```
+ - / * % & | ^ << >> == != >= > <= <
```

Gli operatori di confronto sono sovraccaricabili a coppie, per esempio == e *!=*.

Indicizzatori

Sono membri che permettono di accedere ad un oggetto in modo analogo agli array, in pratica usando l'operatore d'indicizzazione *[i].*

Esempio, accedere alle righe di un testo come se fossero elementi di un array.

```
class Documento
{
        private string[] righe;
        public Documento(int n)
        { righe=new string[n]; }
        public string this[int i]
        {       get
                { if(i<righe.Length)
                  return righe[i];
                  else return "";
                }
                set
                { if(i<righe.Length)
                  righe[i]=value;
                }
        }
}
```

È possibile impostare o ricavare le righe dell'oggetto *Documento*.

```
Documento doc=new Documento(3);
doc[0]="prima";
doc[1]="seconda";
doc[2]="terza";
Console.WriteLine(doc[0]);
```

È possibile utilizzare come indice un tipo non numerico.

Esempio ricavare l'indice di una riga di testo del documento, si usa un indicizzatore con parametro *string* (è la riga da cercare) e che restituisce come valore *int* (indice della riga trovata, oppure -1).

```
public int this[string str]
{       get
        {       for(int i=0;i<righe.Length;i++)
                {
                        if(righe[i]==str) return i;
                }
                return -1;
        }
}
```

Si usa in questo modo.

```
int i=doc["seconda"];
Console.WriteLine("doc[\"seconda\"]={0}",i);// stampa 1
```

È possibile implementare indicizzatori con più indici, come negli array multi dimensionali.

```
public char this[int riga,int colonna]
{       get
        {       if(riga<righe.Length)
                        {string str=righe[riga];
                        if(colonna<str.Length) return str[colonna];
                        }
                return '\u0000';
        }
        set
        {       if(riga<righe.Length)
                {       string str=righe[riga];
                        if(colonna<str.Length)
                        {righe[riga]=str.Substring(0,colonna)+value;
                        if(colonna<str.Length-1) righe[riga]+=str.Substring(colonna+1);
                        }
                }
        }
}
```

Si usa in questo modo.

```
Console.WriteLine(doc[2]); //stampa terza riga
doc[2,3]='r';//cambia il carattere di indice 3
Console.WriteLine(doc[0]);//stampa la terza riga modificata
```

EREDITARIETÀ

Per derivare una nuova classe da una classe esistente si usa la sintassi seguente.

```
class Auto: Veicolo                 // il nome della classe è seguito da : e la classe base
{ … }
```

Significa: "*Auto* deriva da *Veicolo*", si può creare un'istanza della classe *Auto* ed utilizzare i membri ereditati dalla classe *Veicolo*.

```
Auto auto=new Auto();
auto.Accelera();
auto.Frena();
```

Per regolare l'accesso ai membri della classe madre, per esempio facendo in modo che alcuni membri siano ereditati dalle classi figlie, mentre altri devono restare a disposizione solo della classe madre: si usano i modificatori di accesso.

Esempio.
```
class Veicolo
{ private int numeroRuote;
  private float velocita;
}
```
Se si derivasse la classe *Auto*, entrambi i campi non sono visibili (*private*), allora si preferisce dichiarare dei membri nascosti al mondo esterno, eccetto che alle classi derivate (*protected*).
```
class Veicolo
{ protected int numeroRuote;
  protected float velocita;
}
```
Inizializzazione.
```
class Auto: Veicolo
{       public Auto()
        {       numeroRuote=4;
                velocita=0;
        }
}
```

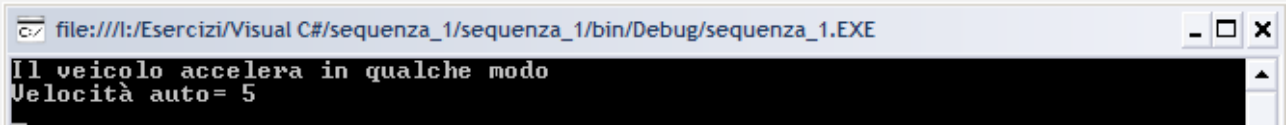
Upcasting
Se l'*Auto* è di tipo *Veicolo* si possono scrivere le seguenti assegnazioni.
```
Veicolo veicolo;
Auto auto=new Auto();
veicolo=auto;
veicolo.Accelera();
```
L'assegnazione della terza linea converte un'istanza di una classe derivata nel tipo da cui deriva: questa operazione è l'upcasting, è sempre lecita e permessa perché la classe madre è più generica della classe figlia.

Downcasting
È l'operazione inversa, in altre parole la conversione da un oggetto della classe madre ad uno della classe figlia, e questo non è sempre lecito.
È eseguibile con l'operatore di cast oppure *as*.
```
Veicolo nuovoVeicolo=new Veicolo();
Auto auto=(Auto)nuovoVeicolo;           // downcasting non valido, genera un'eccezione
auto=nuovoVeicolo as Auto;              // downcasting non valido, auto restituisce null
if(auto==null) Console.WriteLine("nuovoVeicolo non e un'auto");
```
Usando l'operatore di cast, se il downcasting non è possibile, sarà generata un'eccezione *InvalidCastException*, mentre con *as* sarà restituito null.
È buona regola di programmazione verificare sempre il tipo degli oggetti con *is*.
```
if(nuovoVeicolo is Auto)
{ auto=(Auto)nuovoVeicolo;              // il cast è possibile
  Console.WriteLine("nuovoVeicolo è un'auto");
}
```

Hiding
Esempio, all'interno della classe *Veicolo* si è implementato il metodo *Accelera*, si derivi la classe *Auto* e *Bicicletta*.
È ovvio che il metodo *Accelera* di *Bicicletta* è diverso nel funzionamento da *Auto*.
In questo caso si deve implementare più volte il metodo per ogni classe, oppure scrivere il codice in questo modo.
```
using System;
```

```
class veloce
{
   class Veicolo
   {   protected int numeroRuote;
       protected float velocita;
       protected float velocitaMax;
       public void Accelera()
       { Console.WriteLine("Il veicolo accelera in qualche modo"); }
   }
   class Auto : Veicolo
   {
       public Auto()
       {  numeroRuote = 4;
          velocita = 0;
          velocitaMax = 180.0f;
       }
       public void Accelera()
       {
          if (velocita + 5 < velocitaMax) velocita += 5;
          Console.WriteLine("Velocità auto= " + velocita);
       }
   }
   public static void Main()
{  Console.Clear();
   Veicolo veicolo=new Auto();
   Auto auto=new Auto();
   veicolo.Accelera();                    // classe Veicolo
   auto.Accelera();                       // classe Auto
   Console.ReadKey();
}
}
```

Il metodo *Accelera* della classe *Auto* nasconde, **hide**, il metodo omonimo della classe madre, quindi per un oggetto *Auto* sarà invocato *Accelera* della relativa classe, al contrario avverrà per un *Veicolo*.



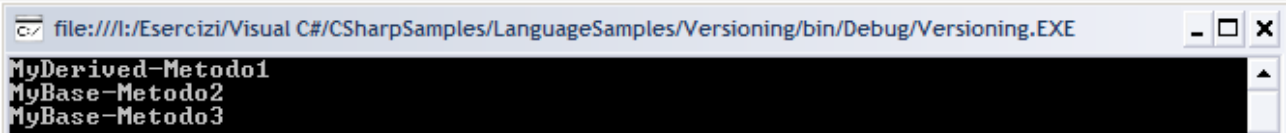
POLIMORFISMO

Effettuando l'overriding dei metodi ereditati e prevedendo già in fase d'implementazione della classe madre quali dei suoi metodi saranno sottoposti a sovraccarico nelle classi derivate, si ottiene il polimorfismo in due modi.

1. Dichiarare il metodo della classe base con la parola chiave *virtual*.
2. Dichiarare il metodo corrispondente delle classi figlie con la parola chiave *ovveride*.

```
using System;
public class MyBase
{
  public virtual string Meth1()
  { return "MyBase-Metodo1"; }
  public virtual string Meth2()
  { return "MyBase-Metodo2"; }
  public virtual string Meth3()
```

```
  { return "MyBase-Metodo3"; }
}
class MyDerived : MyBase
{
  // Esegue l'override del metodo virtuale Meth1 utilizzando la parola chiave override
  public override string Meth1()
  { return "MyDerived-Metodo1";}
  // Nascondere in modo esplicito il metodo virtuale Meth2 usando la parola chiave new
  public new string Meth2()
  { return "MyDerived-Metodo2"; }
  // Poiché nella seguente dichiarazione non è specificata una parola chiave, sarà
  // generato un avviso per segnalare al programmatore che il metodo nasconde il
  // membro ereditato MyBase.Meth3()
  public string Meth3()
  {return "MyDerived-Metodo3"; }
  public static void Main()
  {  Console.Clear();
     MyDerived mD = new MyDerived();
     MyBase mB = (MyBase) mD;
     Console.WriteLine(mB.Meth1());
     Console.WriteLine(mB.Meth2());
     Console.WriteLine(mB.Meth3());
     Console.ReadKey();
  }
}
```



Versionamento

Non è possibile sapere quali metodi sono sottoposti ad overriding, specialmente se si eredita una classe perché non è detto che i metodi siano stati dichiarati *virtual* nella classe base.

Allora, si deve dichiarare il nuovo metodo con la parola chiave *new*, in questo modo si dichiara esplicitamente che il nuovo metodo *Accelera* nasconde il metodo della classe madre, in altre parole si crea una nuova versione del metodo.

```
class Auto:Veicolo
{…
        public new void Accelera()
        {       if(velocita+5<velocitaMax)
                velocita+=5;
                Console.WriteLine("velocità auto= "+velocita);
        }
}
```

È possibile usare la parola chiave *new* anche sui campi, statici e non, di una classe, mentre non è possibile applicare ai campi la parola chiave *virtual*; e non è possibile usare contemporaneamente sullo stesso metodo *override* e *new*.

La parola chiave *new* può essere applicata alla dichiarazione di una classe, in questo modo maschera i metodi ereditati dalla propria classe base, oppure può essere applicata ad una classe nested con lo stesso nome di una classe innestata nella classe madre.

```
using System;
public class ClasseBase
```

```
{
        public class ClasseInnestata
        { public int x = 200; }
}
public class ClasseDerivata : ClasseBase
{
        new public class ClasseInnestata
        // la classe nested nasconde la classe della classe base
        { public int x = 100; }
        public static void Main()
        {       ClasseInnestata S1 = new ClasseInnestata ();
                // Crea un oggetto della classe nascosta
                ClasseBase.ClasseInnestata S2 = new ClasseBase.ClasseInnestata ();
                Console.Clear();
                Console.WriteLine(S1.x);
                Console.WriteLine(S2.x);
                Console.ReadKey();
        }
}
100
200
```

Chiamare i metodi della classe

Per accedere a membri dichiarati nella classe madre si usa la parola chiave *base*.

```
class Prodotto
{       protected decimal prezzo;
        public Prodotto(decimal prezzo)
        {this.prezzo=prezzo;}
        public virtual decimal CalcolaPrezzo()
        {return prezzo;}
}
class ProdottoScontato:Prodotto
{       private decimal sconto=10;//sconto 10%
        public ProdottoScontato(decimal prezzo):base(prezzo)
        { … }
        public override decimal CalcolaPrezzo()
        {return (1-sconto/100)*base.CalcolaPrezzo();}
}
```

Sia la classe *Prodotto* sia la classe *ProdottoScontato* hanno un metodo per calcolare il prezzo, e nel caso della seconda il prezzo è calcolato sulla base di quello restituito dalla classe *Prodotto*, con la chiamata seguente.

```
base.CalcolaPrezzo();
```

Si ottiene il prezzo della classe Prodotto e quindi si applica lo sconto, se s'istanziano due prodotti e si calcola il prezzo si ottiene il seguente output.

```
Prodotto p1=new Prodotto(100);
ProdottoScontato p2=new ProdottoScontato(100);
Console.WriteLine("Prezzo p1={0}",p1.CalcolaPrezzo());
Console.WriteLine("Prezzo p2={0}",p2.CalcolaPrezzo());
Prezzo p1=100
Prezzo p2=90.0
```

Classe astratta

È una classe che non può essere istanziata, in altre parole non è possibile creare un oggetto di questa classe, è utile, quindi, come template per altre classi che saranno

derivate da essa per illustrare un comportamento comune che esse dovranno avere.
C# permette la creazione con il modificatore *abstract*.

```
public abstract class Figura
{}
```

Una classe astratta può contenere metodi anch'essi *abstract*, dei quali non è fornito il corpo e quindi il funzionamento interno.
Viceversa, una classe che possiede dei metodi astratti deve obbligatoriamente essere dichiarata astratta.

```
public abstract class Figura
{       protected float fArea;
        public abstract void CalcolaArea();
        public virtual string GetTipoFigura()
        {return "Figura generica";}
}
```

Una classe che deriva da una classe astratta, deve fornire un'implementazione dei suoi metodi astratti, a meno che sia anch'essa una classe astratta, questo implica che i metodi sono implicitamente dei metodi virtuali e, quindi, non è possibile dichiarare un metodo *abstract virtual*, mentre i metodi delle classi figlie che implementano i metodi astratti dovranno essere qualificati con il modificatore *override*.

```
public class Quadrato
{       float lato;
        public override void CalcolaArea()
        {fArea= lato*lato;}
        public override string GetTipoFigura()
        {return "Quadrato";}
}
```

È possibile dichiarare anche delle proprietà astratte, senza l'implementazione *get/set*.

```
public abstract float Area
{       get;
        set;
}
```

Il codice definisce una proprietà *Area* astratta, ed una classe che deriva dalla classe che la contiene deve fornire un'implemetazione per *get/set*.

```
public override float Area
{       get
        {return fArea;}
        set
        {fArea=value;}
}
```

Classe sealed

È una classe che non può essere derivata da una classe figlia: si blocca la catena di ereditarietà della classe, affinché la classe non sia estesa in modo accidentale.
Per esempio, la classe *System.String* è una classe *sealed*, per evitare che si possa ereditare da essa una diversa implementazione delle stringhe.
Dichiarazione della classe.

```
class Triangolo
{ //… }
sealed class TriangoloScaleno:Triangolo
{ }
//errore di compilazione: si deriva una classe sealed
class TriangoloScalenoIsoscele:TriangoloScaleno
{ }
```

Una classe *abstract* non può essere *sealed* perché non è istanziabile e non è estendibile

da una classe figlia.
È possibile applicare il modificatore *sealed* ad un metodo, in questo modo non può avere override in classi derivate ed ha senso, quindi, solo se è esso stesso un override di un metodo di una classe base.

INTERFACCE

È un contratto applicabile ad una classe: che s'impegna a rispettare, in pratica implementa tutti i metodi, le proprietà, gli eventi e gli indicizzatori esposti dall'interfaccia.
Il concetto è simile a quello di classe astratta, ma nel caso dell'interfaccia non è possibile fornire alcuna implementazione dei membri, non è possibile contenere campi, inoltre, il tipo di relazione di una classe derivante da una classe astratta è sempre una relazione di generalizzazione, invece, nel caso dell'interfaccia la relazione è d'implementazione.
Per esempio, un *Triciclo* potrebbe derivare dalla classe astratta *VeicoliAPedali* e potrebbe implementare un'interfaccia che espone i metodi per guidarlo, interfaccia implementabile anche da altri tipi, ad esempio dalla classe *Moto*, che non è un *VeicoliAPedali*, ma deve essere guidabile, quindi si ha l'interfaccia *IGuidabile*.
Un'interfaccia è dichiarata con la parola chiave *interface*.

```
[modificatore_accesso] interface InomeInterfaccia [: ListaInterfacceBase]
{       //corpo dell'interfaccia }
```

Tutti i membri esposti da un'interfaccia sono *public virtual*, in quanto servono solo da segnaposto per quelle che saranno le loro implementazioni.

Per esempio, definire l'interfaccia *Atleta*.

```
public interface IAtleta
{       void Corre();
        void Salta();
}
```

Se una classe deve implementare questa interfaccia, deve fornire un'implementazione *public* dei due metodi.

```
class Calciatore:IAtleta
{       private string nome,cognome, squadra;
        public void Corre()
        {Console.WriteLine("Sto correndo");}
        public void Salta()
        {Console.WriteLine("Sto saltando");}
}
```

Una classe può implementare più interfacce: un oggetto è una sola cosa, ma si comporta in modi diversi, in modo polimorfico.

Per esempio, confrontare due calciatori, si può implementare contemporaneamente l'interfaccia *IAtleta* e l'interfaccia *IComparable* che espone il seguente metodo.

```
int CompareTo(object obj);
```

Ora, è possibile confrontare i due calciatori.

```
class Calciatore:IAtleta, IComparable
{       private string nome, cognome, squadra;
        public void Corre()
        {Console.WriteLine("Sto correndo");}
        public void Salta()
        {Console.WriteLine("Sto saltando");}
        public int CompareTo(object obj)
        {       if(obj is Calciatore)
                {       Calciatore c=(Calciatore)obj;
                        if(nome==c.nome && cognome==c.cognome && squadra==c.squadra)
```

```
                return 0;
            }
            return -1;
        }
}
```
C# consente l'ereditarietà multipla delle interfacce, in altre parole l'interfaccia può derivare da più interfacce contemporaneamente.

Per esempio, interfaccia *IAtletaUniversale*.
```
interface INuotatore
{void Nuota();}
interface ISciatore
{void Scia();}
interface ITennista
{       void Dritto();
        void Rovescio();
}
interface IAtletaUniversale: INuotatore, ISciatore,ITennista,IAtleta
{       void Mangia();
        void Dormi();
}
```
Una classe che implementa l'interfaccia *IAtletaUniversale*, deve non solo fornire un corpo per i metodi Mangia e *Dormi*, ma anche tutti quelli delle interfacce da cui *IAtletaUniversale* deriva.
```
class AtletaCompleto:IAtletaUniversale
{       public void Mangia()
        {Console.WriteLine("Mangio");}
        public void Dormi()
        {Console.WriteLine("Dormo");}
        public void Nuota()
        {Console.WriteLine("Nuoto");}
        public void Scia()
        {Console.WriteLine("Scio");}
        public void Dritto()
        {Console.WriteLine("Colpisco con il dritto");}
        public void Rovescio()
        {Console.WriteLine("Colpisco di rovescio");}
        public void Corre()
        {Console.WriteLine("Corro");}
        public void Salta()
        {Console.WriteLine("Salto");}
}
```
Se una classe deriva da una classe base ed implementa qualche interfaccia, si deve dichiarare prima la classe seguita dall'elenco delle interfacce.
Potrebbe accadere che più interfacce espongano uno stesso metodo, per esempio, le interfacce *ITennista* e *ISciatore* hanno il metodo *Esulta*, allora occorre risolvere l'ambiguità nella classe che implementa le due interfacce.
```
class AtletaCompleto:IAtletaUniversale
{ …
        void ITennista.Esulta()
        {Console.WriteLine("Sono un grande tennista!");}
        void ISciatore.Esulta()
        {Console.WriteLine("Sono il miglior sciatore!");}
}
```

L'implementazione dei metodi non può essere *public*, per cui se si vuole usare i due metodi, prima bisogna effettuare un cast dell'oggetto verso una delle due interfacce.

```
AtletaCompleto atleta=new AtletaCompleto();
atleta.Esulta;                          //Errore, la classe non ha un metodo Esulta
((ISciatore)atleta).Esulta();           //OK esulta da sciatore
((ITennista)atleta).Esulta();           //OK esulta da tennista
```

Perché usare le interfacce

Visto che interfaccia è uguale a contratto, per definire un punto di contatto tra due sviluppatori senza che il legame sia troppo stretto.
Preziose in .NET perché non è disponibile la multiple inheritance (eredità).
Per caricare dinamicamente degli assembly e realizzare un sistema a "plug-in".

FINALIZE

Gli oggetti possono usare memoria e/o risorse.
Alla memoria ci pensa il GC, alle risorse ci deve pensare il programmatore.

- ✓ Handle grafici, di file, delle porte di comunicazione.
- ✓ Connessioni di database.
- ✓ Altre risorse del mondo *unmanaged*.

Questo meccanismo può essere chiamato finalizzazione di un oggetto, per cui un oggetto che necessita di risorse *unmanaged* deve supportarlo.
Quando il GC stabilisce che un oggetto può essere rimosso dalla memoria, invoca il metodo *Finalize* ereditato dalla classe *System.Object*, ma è *protected* quindi non invocabile da oggetti di una classe differente.

```
class MiaClasse
{       protected override void Finalize()
        {
                //libera le risorse unmanaged
        }
}
```

Non si può fornire un override del metodo *Finalize*: errore.
Allora, invece d'implementare l'override in modo esplicito, si usa la sintassi del distruttore.

```
class MiaClasse
{       ~MiaClasse()
        {
                //libera le risorse unmanaged
        }
}
```

Il compilatore produce lo stesso codice con l'aggiunta della gestione di eccezioni nella finalizzazione stessa.
Attenzione: il distruttore influenza le prestazioni del GC.
Allora, è meglio liberare le risorse in modo esplicito, tramite un metodo ad hoc, usando il pattern *Dispose*.

DISPOSE

Il pattern *Dispose* permette di definire le funzionalità da fornire e da implementare in una classe per ottenere in maniera deterministica ed esplicita la distruzione delle sue istanze e la liberazione delle risorse occupate.
Per fare questo si usa l'interfaccia *IDisposable* che espone un solo metodo.

```
public void IDisposable
        { void Dispose(); }
```

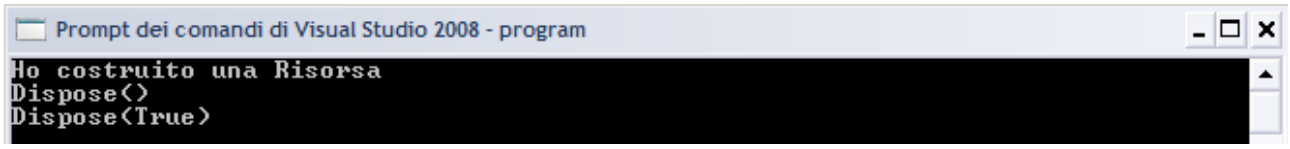
Esempio, implementare un metodo pubblico *Dispose*, sarà richiamato ogni volta che è

necessario liberare le risorse associate all'oggetto; in genere, il pattern *Dispose* è usato insieme al metodo *Finalize*.

```
using System;
using System.IO;
using System.Windows.Forms;
using System.Drawing;
public class Risorsa: IDisposable
{       // handle di una risorsa unmanaged, ad esempio di un file
        private IntPtr handle;
        // un'Icon è una risorsa managed
        private Icon icona;
        private bool disposed = false;
        public Risorsa(IntPtr handle,string iconfile)
        {       this.handle = handle;
                this.icona=new Icon(iconfile);
        }
        // implementazione di IDisposable
        public void Dispose()
        {       Console.WriteLine("Dispose()");
                Dispose(true);
                // informa il CLR di non invocare il distruttore
                // perchè le risorse sono state liberate con il Dispose precedente
                GC.SuppressFinalize(this);
        }
        // se disposing è true, sono liberate sia risorse managed sia unmanaged
        // con disposing = false, il metodo è richiamato internamente dal CLR,
        // esattamente dal distruttore, che avrà già liberato le risorse managed,
        // dunque qui sono ripulite solo quelle unmanaged
        private void Dispose(bool disposing)
        {       Console.WriteLine("Dispose({0})",disposing);
                if(!this.disposed)
                {       if(disposing)
                        {       // l'icona è una risorsa managed
                                icona.Dispose();
                        }
                // libera opportunamente le risorse unmanaged
                CloseHandle(handle);
                handle = IntPtr.Zero;
                }
        // per evitare che Dispose sia eseguito più volte
        disposed = true;
        }
        // distruttore richiamato in modo automatico dal GC
        [System.Runtime.InteropServices.DllImport("Kernel32")]
        private extern static Boolean CloseHandle(IntPtr handle);
        // il distruttore è invocato automaticamente
        // e solo se non è stato invocato il metodo Dispose
        ~Risorsa()
        {       Console.WriteLine("~Risorsa()");
                // è false perché in caso di risorse managed si occupa il GC
                Dispose(false);
        }
        public static void Main()
        {       Console.Clear();
```

```
            FileStream fs=File.OpenRead(@"I:\pippo.txt");
            // il CLR invoca automaticamente il metodo res.Dispose()
            using(Risorsa res=new Risorsa(fs.Handle,@"I:\max.ico"))
            {Console.WriteLine("Ho costruito una {0}",res.ToString());}
            Console.ReadKey();
        }
}
```

Perché implementare un'interfaccia solo per aggiungere ad una classe un metodo *public*, basta inserire un metodo uguale in qualsiasi classe e chiamarlo quando si vuole.
Il motivo è la parola chiave *using* perché garantisce ed automatizza la chiamata del metodo *Dispose* quando l'oggetto non serve più e deve essere distrutto.
Se si provasse ad utilizzare la classe *Risorsa* con il metodo precedente si ottiene.



Se non s'invocasse esplicitamente il *Dispose* o non si usa il blocco *using* è il CLR che si occupa d'invocare il distruttore.

```
    public static void Main()
    {       Console.Clear();
            FileStream fs=File.OpenRead(@"I:\pippo.txt");
            Risorsa res=new Risorsa(fs.Handle,@"I:\max.ico");
            {Console.WriteLine("Ho costruito una {0}",res.ToString());}
    }
```

DELEGATI

È un meccanismo per il reindirizzamento delle chiamate ad un metodo verso un altro metodo, invocando, quindi, un *delegate* s'invoca in modo indiretto il metodo che il *delegate* maschera al suo interno, anche senza sapere quale metodo il *delegate* contenga.
I *delegate* sono l'equivalente .NET dei puntatori a funzione del C/C++ *unmanaged*, ma hanno il grosso vantaggio di essere tipizzati (type-safe).
In altre parole, è possibile usare un metodo come se fosse un oggetto qualsiasi e dunque passarlo come parametro ad un altro metodo.
Un'istanza di un *delegate* incapsula uno o più metodi, ognuno noto come **callable entity**.
L'oggetto *delegate* può essere passato al codice che richiama il metodo in esso incapsulato senza sapere in compile-time quale metodo sarà invocato.
L'unica cosa che richiede al metodo che incapsula è di essere compatibile con il tipo del *delegate*.

Esempio.

```
using System;
delegate void SimpleDelegate();
class MyClass
{   static void A()
    {  System.Console.WriteLine("Test.A");  }
    static void Main(string[] args)
```

```
    {     SimpleDelegate d = new SimpleDelegate(A);
          d();
          Console.ReadKey();
      }
}
```

Esempio, progettare un'applicazione bancaria con la classe *Banca*, all'interno della quale sono contenuti i conti correnti dei clienti.

Dichiarazione di un *delegate*

È uguale a una dichiarazione di variabile, specifica i parametri d'ingresso ed il tipo di ritorno del metodo che può essere sia statico sia d'istanza, che il delegato stesso maschera.

[modificatore] delegate tipo_ritorno NomeDelegate([parametri]);

Un delegato è dietro le quinte una classe derivata dalla classe *System.Delegate*.

La seguente dichiarazione di delegato è usata per riferirsi a metodi che hanno come parametro d'ingresso un parametro di tipo *String* e non hanno valore di ritorno.

public delegate void SaldoDelegate(ContoCorrente c);

Lo scopo è quello di fornire un metodo per la visualizzazione del saldo di un numero di conto corrente: a video, su file, con l'invio di un e-mail.

La classe *ContoCorrente*, per esempio, può avere due metodi diversi che rispettano lo stesso metodo specificato dal delegato.

```
public void VisualizzaSaldo(ContoCorrente conto)
// visualizza a video il saldo del conto
{Console.WriteLine("Saldo del conto {0}: {1}",conto.numeroConto,conto.saldoAttuale);}
public static void SalvaSaldoSuFile(ContoCorrente conto)
{       StreamWriter sw=null;
        try
        {
        sw=File.CreateText(@"c:\temp\saldo_"+conto.numeroConto+".txt");
        sw.WriteLine("Saldo del conto {0}: {1}", conto.numeroConto,conto.saldoAttuale);
        }
        catch(IOException ioe)
        {Console.WriteLine(ioe.Message);}
        finally
        {if(sw!=null) sw.Close();}
}
```

Istanziazione e invocazione di un *delegate*

Come per qualsiasi oggetto, l'istanziazione avviene con la parola chiave *new* e passando come argomento un metodo che rispetti sia come numero sia come tipo tutti i parametri.

Per creare, per esempio, un'istanza del *SaldoDelegate*, delegato del metodo *VisualizzaSaldo* basta il codice seguente.

SaldoDelegate sd1=new SaldoDelegate(conto.VisualizzaSaldo);

Nessuna differenza se il metodo è statico.

SaldoDelegate sd2=new SaldoDelegate(ContoCorrente.SalvaSaldoSuFile);

Aggiungere alla classe *ContoCorrente* il metodo *ElaboraSaldo* che invocherà il delegato passato come argomento.

```
public void ElaboraSaldo(SaldoDelegate sd,ContoCorrente conto,string data)
{       saldoAttuale=conto.CalcolaSaldo(data);
        sd(conto);
}
```

Quando si passa al metodo *ElaboraSaldo* l'istanza di delegato *sd1*, sarà indirettamente invocato il metodo *VisualizzaSaldo*, mentre con *sd2* il metodo statico *SalavaSaldoSuFile*.

È possibile che un delegato incapsuli più metodi contemporaneamente, si parla di *delegate multicast* che derivano da una classe apposita, la *System.MulticastDelegate*.
La lista d'invocazione di un *delegate multicast* è creata usando gli overload degli operatori + e += forniti dalla classe *MulticastDelegate*.
Per esempio, creato il delegato *sd1* si può aggiungere un delegato alla lista d'invocazione e ottenere un delegato *sd3* con l'istruzione seguente.
sd3 = sd1 + new SaldoDelegate(ContoCorrente.SalvaSaldoSuFile);
Oppure.
sd1 += new SaldoDelegate(ContoCorrente.SalvaSaldoSuFile);
È possibile l'operazione inversa di rimozione della lista d'invocazione, grazie agli operatori – e -=.
La lista d'invocazione si ottiene con il metodo *GetInvocationList*, restituisce un array di oggetti *Delegate*.

```
Console.WriteLine("InvocationList di sd1");
foreach(Delegate d in sd1.GetInvocationList())
        { Console.WriteLine("Delegate {0}",d.Method); }
```

EVENTI

Il .NET Framework nasconde la complessità di basso livello dei messaggi grazie al concetto di evento.
Un oggetto genera degli eventi per un altro oggetto per cui intraprende delle azioni: è il modello del produttore-consumatore.
Tipicamente sono usati per gestire nelle Windows Forms le notifiche dai controlli al contenitore: la form.
Si parla di:

- ✓ Publisher spara eventi a tutti i subscriber.
- ✓ Subscriber colui che è interessato a ricevere gli eventi.

Generare un evento
Prima di generare un evento si deve definire una classe che contenga le informazioni correlate ad esso.
Il .NET Framework fornisce una classe base da cui derivare gli eventi: la classe *System.EventArgs*.

Esempio, progettare un'applicazione automobilistica per monitorare i giri del motore, avvisando l'utente con un allarme quando il numero di giri è superiore ad un valore limite e quando il motore è spento perché il numero di giri è troppo basso.

```
public class MotoreFuoriGiriEventArgs: EventArgs
{       private int rpm;
        public int Rpm
        {       set
                {rpm=value;}
                get
                {return rpm;}
        }
        public string Message
        {       get
                {
                        string msg=String.Format("Il numero dei giri motore è {0}/min",rpm);
                        return msg;
                }
        }
}
```

Si deve dichiarare un delegato che serva come gestore dell'evento.
public delegate void MotoreFuoriGiriEventHandler(object sender,
MotoreFuoriGiriEventArgs e);
public event EventHandler MotoreSpentoEvent;
Se l'evento non ha dati aggiuntivi si può usare il delegato standard *EventHandler*.
È fornito per primo il parametro *sender* affinché il gestore possa ricavare anche l'oggetto generatore dell'evento, il secondo parametro *args* contiene l'evento con i dati che lo caratterizzano, *args* è un'istanza di una classe derivata di *EventArgs.*
Se non c'è bisogno di passare parametri si usa *EventArgs*.
La parola chiave *event* aggiunge un nuovo campo alla classe che genera l'evento.
public event MotoreFuoriGiriEventHandler FuoriGiriEvent;
public event EventHandler MotoreSpentoEvent;
I due campi identificano i due eventi che possono essere generati dalla classe.
Aggiungere alla classe generatrice dell'evento, il metodo *OnNomeEvento*.
È in questo metodo che è generato l'evento, invocando il delegato identificato dalla parola chiave *event* e creando l'oggetto *EventArgs* contenente dati aggiuntivi.
//genera l'evento fuori giri
protected virtual void OnFuoriGiri(MotoreFuoriGiriEventArgs ev)
{FuoriGiriEvent(this,new MotoreFuoriGiriEventArgs(rpm));}
//genera l'evento motore spento
protected virtual void OnMotoreSpento()
{MotoreSpentoEvent(this,new EventArgs());}
Il metodo è dichiarato *protected virtual*, in questo modo esso potrà essere ridefinito in una classe derivata.

```
public class Motore
{       private int rpm;
        private const int maxRpm=7000;
        private const int minRpm=700;
        private bool acceso;
        public delegate void MotoreFuoriGiriEventHandler(object sender,
        MotoreFuoriGiriEventArgs e);
        public event MotoreFuoriGiriEventHandler FuoriGiriEvent;
        public event EventHandler MotoreSpentoEvent;
        public Motore()
        {       rpm=0;
                acceso=false;
        }
        public int Rpm
        {       get
                {return rpm;}
        }
        public void Accendi()
        {       acceso=true;
                this.rpm=800;
        }
        protected virtual void OnFuoriGiri(MotoreFuoriGiriEventArgs ev)
        {FuoriGiriEvent(this,new MotoreFuoriGiriEventArgs(rpm));}
        protected virtual void OnMotoreSpento()
        {MotoreSpentoEvent(this,new EventArgs());}
        public void AumentaRpm()
        {       if(acceso)
                        this.rpm+=100;
                if(rpm>maxRpm)
                        OnFuoriGiri(new MotoreFuoriGiriEventArgs(rpm));
```

```
        }
        public void DiminuisciRpm()
        {       if(acceso)
                        this.rpm-=100;
                if(rpm<minRpm)
                {       this.acceso=false;
                        OnMotoreSpento();
                }
        }
}
```

Consumare un evento

Per consumare un evento bisogna prevedere un metodo che sia richiamato al verificarsi dell'evento e registrare tale metodo come gestore dell'evento.

Progettare la classe *Auto* che contenga un campo di tipo *Motore*, che può andare fuori giri o spegnersi, quindi generare gli eventi relativi.

I metodi che si occupano della gestione degli eventi devono avere gli stessi parametri dei delegati che definiscono gli eventi stessi, in questo caso *MotoreFuoriGiriEventHandler* e *EventHandler*.

```
private void motore_FuoriGiriEvent(object sender, MotoreFuoriGiriEventArgs e)
{       Console.WriteLine("Evento da {0}:\n{1}",sender.ToString(),e.Message);
        Frena();
}
private void motore_MotoreSpento(object sender, EventArgs e)
{Console.WriteLine("Evento da {0}:\nMotore Spento",sender.ToString());}
```

Questi metodi stampano il tipo di evento generato, il messaggio contenuto nell'istanza *MotoreFuoriGiriEventArgs* nel primo caso ed il tipo dell'oggetto che ha generato l'evento che è un'istanza della classe *Motore*.

Ora bisogna registrare i due gestori associandoli agli eventi relativi.

Per aggiungere un gestore all'evento si usano gli operatori + e +=, al contrario – e -= per toglierlo.

Se la classe *Auto* contiene un campo motore di classe *Motore* la registrazione.

```
motore.FuoriGiriEvent+=new
Motore.MotoreFuoriGiriEventHandler(motore_FuoriGiriEvent);
motore.MotoreSpentoEvent+=new EventHandler(motore_MotoreSpento);
```

Nella prima istruzione, il delegato è creato specificando il nome della classe *Motore*.

Aggiungere due metodi che aumentano o diminuiscono i giri del motore.

```
public void Accelera()
{motore.AumentaRpm();}
public void Frena()
{motore.DiminuisciRpm();}
```

Creare una sequenza di chiamate per portare il motore fuori giri o a spegnersi.

```
Auto auto=new Auto();
Console.WriteLine("RPM: {0}", auto.RpmMotore);
while(auto.RpmMotore<7000)
        {auto.Accelera();}
Console.WriteLine("RPM: {0}", auto.RpmMotore);
auto.Accelera();
```

In questo punto del codice, il motore supererà la soglia massima e genera l'evento *FuoriGiri*, di cui la classe *Auto* riceverà la notifica mediante la chiamata al gestore.

```
while(auto.RpmMotore>700)
        {auto.Frena();}
Console.WriteLine("RPM: {0}", auto.RpmMotore);
auto.Frena();
```

In questo punto del codice, il motore si spegne e genera l'evento di cui la classe *Auto* riceverà la notifica mediante la chiamata al gestore

Anonymous methods

Semplificano l'uso dei delegate.
Prima.

```
button1.Click += new EventHandler(button1_Click);
…
private void button1_Click(object sender,EventArgs e)
  {MessageBox.Show((sender as Control).Name); }
```

Dopo.

```
button2.Click += delegate(object sender, EventArgs e)
   {MessageBox.Show("Click!"); }
  MyDlg del = new MyDlg(Method);
```

Anonymous methods senza "signature"
È possibile omettere la signature.

- ✓ Solo se non ha parametri di tipo "out".
- ✓ Può tornare un valore, può non essere *void*.
- ✓ I parametri in ingresso saranno ignorati.
- ✓ L'invocazione deve essere coerente con la dichiarazione del delegato.

```
Delegate void OneDelegate(int x);
...
OneDelegate del = delegate;
{Console.WriteLine("Special"); }
del (20);
```

Generics

L'obiettivo è quello di rendere generico il tipo per: *class, struct, interface, delegate* e metodi.

Esempio, progettare un metodo che scambi i valori di due variabili.

```
using System;
class hello
{       static void Swap(ref int x, ref int y)
        { int Temp = x; x = y; y = Temp;  }
        public static void Main()
        {       Console.Clear();
                int a=1, b=3;
                Console.WriteLine("a = {0}, b = {1}",a,b);        // stampa a = 1, b = 3b
                Swap(ref a, ref b);
                Console.Write("a = {0}, b = {1}",a,b);            // stampa a = 3, b = 1
                Console.ReadKey();
        }
}
using System;
class hello
{       static void Swap(ref string x, ref string y)
        { string Temp = x; x = y; y = Temp;  }
        public static void Main()
        {       Console.Clear();
                string a="a", b="b";
                Console.WriteLine("a = {0}, b = {1}",a,b);        // stampa a = a, b = b
```

```
            Swap(ref a, ref b);
            Console.Write("a = {0}, b = {1}",a,b);          // stampa a = b, b = a
            Console.ReadKey();
        }
}
```
Questi metodi sono limitati ai soli tipi *int* e *string.*

Soluzione uno.
```
using System;
class hello
{       static void Swap(ref object x, ref object y)
        { object Temp = x; x = y; y = Temp;  }
        public static void Main()
        {       Console.Clear();
                object a=1, b=3;
                // object a="a", b="b";
                Console.WriteLine("a = {0}, b = {1}",a,b);
                Swap(ref a, ref b);
                Console.Write("a = {0}, b = {1}",a,b);
                Console.ReadKey();
        }
}
```

Soluzione due: *Generics*
```
using System;
class hello
{       static void Swap<T>(ref T x, ref T y)
        { T Temp = x; x = y; y = Temp;  }
        public static void Main()
        {       Console.Clear();
                int a=1, b=3;
                // string a="a", b="b";
                Console.WriteLine("a = {0}, b = {1}",a,b);
                Swap(ref a, ref b);
                Console.Write("a = {0}, b = {1}",a,b);
                Console.ReadKey();
        }
}
```

Generics controllo del tipo

Il namespace *System.Collections.Generic*.

- ✓ *LinkedList<T>*
- ✓ *List<T>*
- ✓ *Queue<T>*
- ✓ *Stack<T>*
- ✓ *Dictionary<T,V>*

Si possono applicare dei vincoli ai tipi utilizzabili come parametri nelle dichiarazioni.
```
public class Auto<T, U>
        where T: Icollection, Ienumerable
        where U: class
        {  // …  }
```
Esistono vincoli speciali.

- ✓ class vincola ai soli RT.
- ✓ classi, interfacce, *delegate* e array.

- ✓ *struct* ed *enum* vincola ai soli VT.
- ✓ *new* obbliga l'esistenza del costruttore di default.

Ma anche limitazioni.

- ✓ Non devono essere marcati come *sealed*.
- ✓ Non devono essere uno di (o derivato da) questi tipi: *System.Array, System.delegate, System.Enum*, *System.ValueType, System.Objetc*.

Una classe che usa i *generics* non può derivare da *System.Attribute*.
Un metodo dichiarato con *extern* (*PInvoke*) non può usare i *generics*.

Generic *delegate*

Le firme di *delegate* possono avvalersi dei generics.
delegate int CompareFunc<T>(T a, T b);
È possibile associarle esplicitamente.
compareFunc<int>comparedesc2 = new CompareFunc
<int>(MyCompareFunc);
static int MyCompareFunc(int a, int b)
{return a > b ? 0 ; 1; }
O in forma anonima.
CompareFunc<int>compareDesc = delegate(int a, int b)
{return a > b ? 0 ; 1; }

PREDICATE

È una *delegate* generica.
Public delegate bool Predicate<T> (T obj)
È utilizzata in combinazione ad alcuni metodi degli insiemi generici, permettendo di specificare dei criteri di selezione.
List<int> v2 = v.FindAll(delegate(int) { return i < 34;) ; }

ACTION

È una *delegate* generica.
Public delegate bool Action<T> (T obj)
È utilizzata in combinazione ad alcuni metodi degli insiemi generici, permettendo di specificare l'azione sull'elemento corrente.
Public delegate bool Action<T> (T obj)

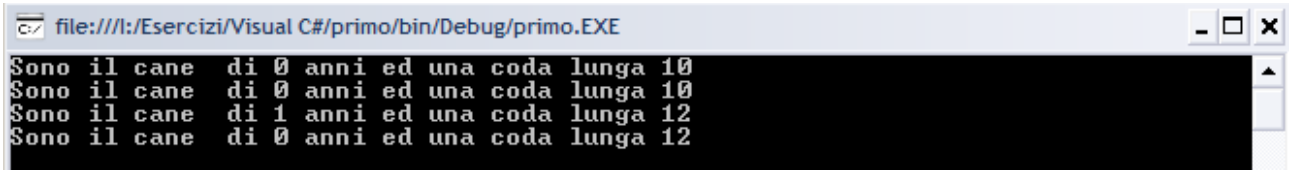
CLONAZIONE

Il metodo *protected MemberwiseClone*, della classe *System.Object*, permette di ottenere una copia dell'oggetto sul quale è invocato, è chiamata **shallow copy**.
È fatta la copia bit a bit di tutti i campi non statici di un VT, mentre per i campi di RT è copiato solo il riferimento e non l'oggetto cui si riferisce il campo stesso.
La copia dell'oggetto restituita, quindi ha dei campi che si riferiscono agli oggetti cui si riferiscono anche i campi dell'oggetto originale.

Esempio, progettare la classe *Cane*, con due attributi VT ed uno di tipo *Coda* che è un RT.

```
using System;
class Cane
{       public string nome;
        public int età;
        public Coda coda;
        public Cane(string nome,int età)
        {coda=new Coda(10);}
        public Cane Copia()
        {return (Cane)this.MemberwiseClone();}
```

```
        public override string ToString()
        {return "Sono il cane "+nome+" di "+età+" anni ed una coda lunga "+coda.Length ;}
}
class Coda
{       private int length;
        public Coda(int l)
        {this.length=l;}
        public int Length
        {       get
                        {return length;}
                set
                        {length=value;}
        }
}
public class TestObjectCloning
{       public static void Main()
        {       Cane fido=new Cane("Fido",2);
                Console.Clear();
                Console.WriteLine(fido.ToString());
                Cane cloneDiFido=fido.Copia();
                Console.WriteLine(cloneDiFido.ToString());
                fido.età+=1;
                fido.coda.Length+=2;
                Console.WriteLine(fido.ToString());
                Console.WriteLine(cloneDiFido.ToString());
                Console.ReadKey();
        }
}
```



Quando si varia il campo *età* dell'oggetto *fido*, questo cambiamento non si riflette sull'istanza *cloneDiFido* perché le due istanze possiedono due copie separate del campo, mentre l'assegnazione ad un campo di RT come il seguente

fido.coda.Length+=2;

si ripercuote su entrambe le istanze, perché il metodo *MemberwiseClone* ha copiato solo il riferimento alla stessa unica istanza della classe *Coda*.

Nel caso in cui servisse una copia totale dell'oggetto, **deep copy**, è necessario che la classe implementi l'interfaccia *ICloneable*, che espone l'unico metodo *object Clone*, che realizza sia la deep copy sia la shadow copy.

La classe *Cane*.

```
public object Clone()
{       Cane clone=new Cane(this.nome,this.età);
        clone.coda=new Coda(this.coda.Length);
        return clone;
}
```

Le due code sono due istanze distinte della classe *Coda* e quindi i cambiamenti sulla coda di *fido* non influenzano la coda del clone2.

XML (*EXTENSIBLE MARKUP LANGUAGE*)

C# supporta la generazione automatica di documentazione usando i tag XML: è un metalinguaggio utilizzato per creare nuovi linguaggi, atti a descrivere documenti strutturati inseriti nei commenti.

```
using System;
/// <summary>
/// Documentazione di riepilogo a livello di classe.</summary>
/// <remarks>
/// È possibile associare commenti più estesi a un tipo o un membro
/// tramite il tag remarks</remarks>
public class SomeClass
{
  /// <summary>
  /// Archivio della proprietà del nome</summary>
  private string myName = null;
  /// <summary>
  /// Costruttore di classe.</summary>
  public SomeClass()
  {
     //  aggiungere qui la logica del costruttore
  }
  /// <summary>
  /// Proprietà del nome </summary>
  /// <value>
  /// Un tag value viene utilizzato per descrivere il valore della proprietà</value>
  public string Name
  {
    get
    {
      if ( myName == null )
      {
        throw new Exception("Name is null");
      }
      return myName;
    }
  }
  /// <summary>
  /// Descrizione di SomeMethod.</summary>
  /// <param name="s"> Descrizione di parametro relativa a s</param>
  /// <seealso cref="String">
  /// È possibile utilizzare l'attributo cref in qualsiasi tag per fare riferimento a un tipo o un
membro.
  /// Il compilatore verificherà l'esistenza del riferimento. </seealso>
  public void SomeMethod(string s)
  {
  }
  /// <summary>
  /// Altro metodo. </summary>
  /// <returns>
  /// I risultati restituiti sono descritti tramite il tag returns.</returns>
  /// <seealso cref="SomeMethod(string)">
  /// l'utilizzo dell'attributo cref per fare riferimento a un metodo specifico </seealso>
  public int SomeOtherMethod()
```

```
  {
    return 0;
  }
  /// <summary>
  /// Punto d'ingresso dell'applicazione.
  /// </summary>
  /// <param name="args"> Elenco di argomenti della riga di comando</param>
  public static int Main(String[] args)
  {
    //  aggiungere il codice per l'avvio dell'applicazione
     return 0;
  }
}
```

Compilare via riga di comando con la seguente opzione.

```
C:\Programmi\Microsoft Visual Studio 9.0\VC>csc hello.cs /doc:c:\hello.xml
Compilatore Microsoft (R) Visual C# 2008 versione 3.5.21022.8
per Microsoft (R) .NET Framework versione 3.5
Copyright (C) Microsoft Corporation. Tutti i diritti riservati.
```

È generato il seguente file: hello.xml.

```
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>provaXML</name>
    </assembly>
    <members>
        <member name="T:SomeClass">
            <summary>
            Documentazione di riepilogo a livello di classe.</summary>
            <remarks>
            È possibile associare commenti più estesi a un tipo o un membro
            tramite il tag remarks</remarks>
        </member>
        <member name="F:SomeClass.myName">
            <summary>
            Archivio della proprietà del nome</summary>
        </member>
        <member name="M:SomeClass.#ctor">
            <summary>
            Costruttore di classe.</summary>
        </member>
        <member name="M:SomeClass.SomeMethod(System.String)">
            <summary>
            Descrizione di SomeMethod.</summary>
            <param name="s"> Descrizione di parametro relativa a s</param>
            <seealso cref="T:System.String">
            È possibile utilizzare l'attributo cref in qualsiasi tag per fare riferimento a un tipo o
un membro.
            Il compilatore verificherà l'esistenza del riferimento. </seealso>
        </member>
        <member name="M:SomeClass.SomeOtherMethod">
            <summary>
            Altro metodo. </summary>
            <returns>
            I risultati restituiti sono descritti tramite il tag returns.</returns>
```

```
        <seealso cref="M:SomeClass.SomeMethod(System.String)">
        l'utilizzo dell'attributo cref per fare riferimento a un metodo specifico </seealso>
      </member>
      <member name="M:SomeClass.Main(System.String[])">
        <summary>
        Punto d'ingresso dell'applicazione.
        </summary>
        <param name="args"> Elenco di argomenti della riga di comando</param>
      </member>
      <member name="P:SomeClass.Name">
        <summary>
        Proprietà del nome </summary>
        <value>
        Un tag value viene utilizzato per descrivere il valore della proprietà</value>
      </member>
    </members>
</doc>
```

GESTIONE DELLE ECCEZIONI

INTRODUZIONE

Durante l'esecuzione di un'applicazione si potrebbero verificare degli errori anche se la compilazione fosse andata bene, per esempio gli input dell'utente potrebbero essere sbagliati oppure un file potrebbe non trovarsi nella posizione in cui è cercato.
In C# gli errori in esecuzione sono gestiti con il meccanismo delle **eccezioni**: ad ogni blocco d'istruzioni, nel quale potrebbe verificarsi un errore, si associa un blocco d'istruzioni che lo gestisce.
Un'eccezione è un oggetto, derivato, in modo diretto o indiretto dalla classe *System.Exception*.
Una volta che l'eccezione si è verificata, in pratica è stata lanciata in qualche punto dell'applicazione, è necessario catturarla e svolgere un'azione di recupero.
Non è obbligatorio implementare il codice per la gestione delle eccezioni.
Per fare questo, le parti di codice in cui è prevedibile che un'eccezione si possa verificare, sono racchiuse in blocchi.

```
try
{
        // codice che può generare un'eccezione: serve ad indicare quali sono gli statement
        // di cui si vuole intercettare l'errore, chiamate annidate comprese
}
catch(TipoEccezione ex)
{
        // gestione dell'eccezione: serve per catturare uno specifico errore, più si è specifici
        // e maggiore è la possibilità di recuperare l'errore in modo soft
}
finally
{
        // libera le risorse o svolge altre azioni: serve ad indicare gli statement che si
        // vogliono eseguire sia in caso di errore sia di normale esecuzione
}
```

Se all'interno del blocco *try* si verifica un'eccezione del tipo previsto dall'istruzione *catch*, il controllo dell'applicazione passa al blocco *catch* dove sono intraprese le azioni per risolvere il problema.
Se all'interno del blocco *try* non si verifica un'eccezione o si è raggiunta la fine del blocco *catch*, il flusso dell'applicazione prosegue nel blocco *finally* dove sono liberate le risorse usate nel *try* precedente, o sono eseguite le funzioni previste sia in caso di errore sia in caso di esecuzione normale.
I blocchi *catch* possono essere più di uno in cascata per gestire più eccezioni.

Esempio: aprire un file.

```
public void TestTryCatchFinally()
{
        StreamReader sr=null;
        try
        {
                sr=File.OpenText(@"C:\nomefile.txt");
                string str=sr.ReadLine().ToLower;
                Console.WriteLine(str);
        }
        catch(FileNotFoundException fnfEx)              // il file non esiste
```

```
        {
                Console.WriteLine(fnfEx.Message);
        }
        catch(NullReferenceException flEx)              // il file è vuoto
        {
                Console.WriteLine(flEx.Message);
        }
        finally
        {
                if(sr!=null) sr.Close();
        }
}
```

Se il file non è trovato l'esecuzione salta nel primo blocco *catch*
Se il file è vuoto, quando si legge una riga con *sr.ReadLine,* è restituito il valore *null*, per cui nel convertire la stringa in minuscolo sarà generata l'eccezione *NullReferenceException,* gestita nel secondo blocco *catch*.
Alla fine si passa dal blocco *finally* che chiude l'oggetto *StreamReader*.
È possibile omettere sia il blocco *catch* sia il blocco *finally*, ma non entrambi contemporaneamente.
Solo il blocco *finally* è utilizzato per garantire che siano eseguite certe istruzioni prima di uscire da un metodo, ad esempio il blocco *try* potrebbe contenere più istruzioni *return*, quindi più punti di uscita, quando s'incontra il primo *return* l'applicazione prima di terminare il metodo passa dal blocco *finally*.

```
public int TestTryFinally()
{
        int i=2;
        try
        {
                switch(i)
                {
                        case 2:return 4;
                        case 3:return 9;
                        default:return i*i;
                }
        }
        finally
        {
                Console.WriteLine("Prima di uscire passa dal finally");
        }
}
```

Il meccanismo per cui un'eccezione è generata, è chiamato **throwing**.
Quando si verifica un'eccezione, è eseguita un'istruzione del tipo:
throw new TipoEccezione();
che ha la funzione di generare e lanciare un'eccezione specifica: se è eseguita da una parte di codice all'interno di un blocco *try* e se l'eccezione è stata prevista in un blocco *catch*, entra in funzione la cattura dell'eccezione.

```
try
{
        throw new Exception("Genero un'eccezione");
}
catch(Exception ex)
{
        Console.WriteLine(ex.Message);
}
```

Nell’esempio, l’eccezione è catturata ed è stampato il messaggio *"Genero un'eccezione"*. È possibile utilizzare *throw* anche al di fuori del blocco *try*, in pratica in una parte qualunque di codice, in questo caso al verificarsi di un’eccezione e non trovando il blocco *catch* per gestirla, il CLR risalirà lo stack delle chiamate fino a trovarne uno, o al limite fino a quando sarà restituito un messaggio di eccezione non gestita.

Esempio: metodo che divide due interi.

```
public int Dividi(int a,int b)
{
	if(b==0) throw new DivideByZeroException("Eccezione generata da Dividi(a,b)");
	return a/b;
}
```

Nel metodo *Dividi* l’eccezione può essere generata, ma non è gestita, per cui usando questo metodo da qualche altra parte nel codice, si deve gestire l’eccezione.

```
public void ChiamaDividi()
{
	try
	{
		Dividi(5,0);
	}
	catch(DivideByZeroException ex)
	{
		Console.WriteLine(ex.Message);
	}
}
```

Il *throw* può essere eseguito anche all’interno di un blocco *catch*, in questo caso si parla di eccezione rilanciata, esattamente è rilanciata al metodo che ha chiamato quello in cui si è verificata.

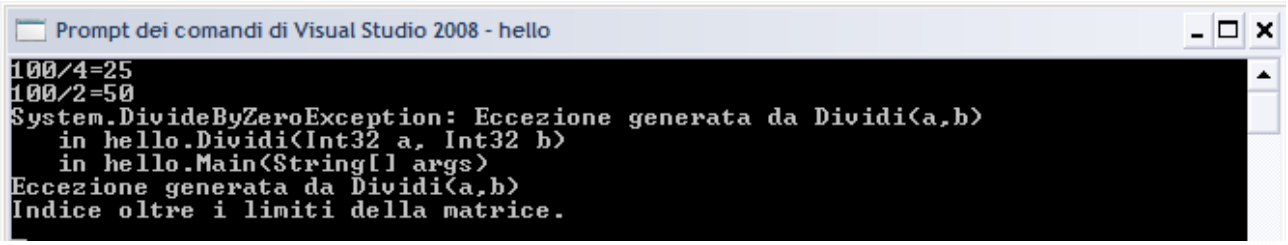
```
public void ChiamaDividi()
{
	try
	{
		Dividi(5,0);
	}
	catch(DivideByZeroException ex)
	{
		Console.WriteLine("Rilancio l'eccezione
		"+ex.GetType()+Environment.NewLine+ex.Message);
		throw ex;
	}
}
```

Il metodo *GetType* restituisce un’istanza della classe *System.Type*, usato per ottenere informazioni a run-time sul tipo corrente di oggetto, per esempio il *namespace*, i metodi di una classe, il nome completo, quindi è un punto di accesso alla tecnologia *Reflection*.

È possibile scrivere blocchi di *catch* anche senza specificare il tipo di eccezione tra parentesi, in questo caso sono catturate tutti i tipi di eccezioni generate, ma non si possono avere dettagli sull’eccezione verificatasi.

```
try
{
	Dividi(5,0);
}
catch
{
	Console.WriteLine("Si è verificata un'eccezione");
```

```
}
```
In questo caso è ancora possibile rilanciare l'eccezione catturata, con *throw* senza specificare niente.
```
try
{
        Dividi(5,0);
}
catch
{
        throw;
}
```
È possibile scrivere blocchi annidati di *try*, in questo modo un'eccezione generata all'interno di un blocco *try*, nel caso in cui il corrispondente *catch* non la gestisca, sarà propagata al blocco *try* esterno.
```
using System;
class hello
{
      static void Main(string[] args)
       { int[] arr=new int[]{4,2,0};
         int dividendo=100;
        Console.Clear();
         for(int i=0;i<4;i++)
         {
                  try
                  {
                           try
                           {
                           Console.WriteLine("{0}/{1}={2}",dividendo,arr[i],Dividi(dividendo,arr[i]));
                           }
                           catch(DivideByZeroException de)
                           {
                                    Console.WriteLine(de.ToString());
                                    Console.WriteLine(de.Message);
                           }
                  }
                  catch(IndexOutOfRangeException ie)
                  {
                           Console.WriteLine(ie.Message);
                  }
         }
           Console.ReadKey();
       }
       public static int Dividi(int a,int b)
         {
                  if(b==0) return a/b;
         }
}
```



Nel codice si hanno due *try* annidati, in quello più interno è eseguita una divisione con il metodo *Dividi*, l'eccezione *DivideByZeroException* è gestita dal corrispondente *catch*; ma il divisore è preso dall'array *arr* all'interno del ciclo *for*, alla fine l'indice supererà i limiti dell'array generando l'eccezione *IndexOutOfRangeException*, che non essendo gestita nel blocco *try* interno, sarà propagata a quello più esterno dove esiste un *catch* apposito.

Esempio: legge una stringa e la converte in un numero intero.

```
using System;
class hello
{
    static void Main(string[] args)
    {
       Console.Clear();
      try
      {     Console.Clear();
            Console.Write("Inserisci un numero: ");
            int divisore = int.Parse(Console.ReadLine());
            int risultato = 10/divisore;
            Console.WriteLine(risultato);
      }
      catch (Exception e)
      {
            Console.WriteLine("Errore : {0}", e.Message );
      }
            Console.ReadKey();
    }
}
```

L'esempio stampa un messaggio in corrispondenza di un'eccezione: l'oggetto di tipo *Exception* passato all'istruzione *catch* rappresenta l'errore; per dare maggiori informazioni sul tipo di errore è stata usata la proprietà *Message* della classe *Exception*.

Checked ed unchecked

È possibile eseguire un'istruzione o un blocco d'istruzioni che contengono operazioni aritmetiche, in un contesto controllato, *checked*, o non controllato, *unchecked*.

Esempio: operazione di somma in un contesto *checked*, a b1 è assegnato il valore massimo di un byte, ciò provoca un'eccezione di overflow se si somma il valore uno.

```
public void TestChecked()
      {
            byte b1=Byte.MaxValue;
            byte b2;
      try
      {
            b2=checked((byte)(b1+1));
      }
      catch(System.OverflowException oex)
```

```
        {
                Console.WriteLine(oex.Message);
        }
}
```

Se, invece, si usa un contesto *unchecked* l'operazione di somma sarà eseguita e b2 assumerà il valore zero perché i bit sono troncati.

```
public void TestUnchecked()
{
        byte b1=Byte.MaxValue; //
        byte b2;
        b2=unchecked((byte)(b1+1));
        Console.WriteLine("b2={0}",b2);
}
```

LA CLASSE SYSTEM.EXCEPTION

È la classe base per ogni tipo di eccezione con le seguenti proprietà.

Proprietà	Descrizione
`public string Message`	Restituisce una stringa che descrive l'eccezione.
`public string Source`	Restituisce o imposta il nome dell'applicazione o dell'oggetto che ha generato l'eccezione.
`public string StackTrace`	Restituisce la rappresentazione dello stack di chiamate al momento dell'eccezione.
`public string HelpLink`	Restituisce o imposta il link alla documentazione sull'eccezione generata.
`public Exception InnerException`	Restituisce l'istanza di System.Exception che ha causato l'eccezione corrente, cioè se un'eccezione A è stata lanciata da una precedente eccezione B, allora la proprietà InnerException di A restituirà l'istanza B.
`public MethodBase TargetSite`	Restituisce il metodo in cui è stata generata l'eccezione.

```
public void PrintExceptionInfo(Exception ex)
{
Console.WriteLine("Message: {0}",ex.Message);
Console.WriteLine("Source: {0}",ex.Source);
Console.WriteLine("StackTrace: {0}",ex.StackTrace);
Console.WriteLine("HelpLink: {0}",ex.HelpLink);
Console.WriteLine("InnerException: {0}",ex.InnerException);
Console.WriteLine("Method Name: {0}",ex.TargetSite.Name);
}
```

ECCEZIONI PERSONALIZZATE

Ogni programmatore può creare le proprie classi di eccezioni, o derivarle da una esistente, per rappresentare una situazione che si può generare nell'applicazione.

Esempio: l'eccezione di I/O si può derivare da *IOException*.

```
class EccezionePersonalizzata: Exception
{
```

```
}
```
Da questa il compilatore fornisce, di default, un costruttore senza parametri.
```
try
{
        throw new EccezionePersonalizzata();
}
catch(EccezionePersonalizzata ep)
{
        Console.WriteLine(ep.Message);
}
```
Che stampa la seguente stringa.
Exception of type *EccezionePersonalizzata was throw.*
Se si desidera fornire maggiori dettagli, ad esempio s'implementa la classe.
```
class MiaException:InvalidOperationException
{
        public MiaException():base()
        {
        }
        public MiaException(string msg):base(msg)
        {
        }
        public MiaException(string msg,Exception inner):base(msg,inner)
        {
        }
}
```
Testando questa nuova eccezione.
```
try
{
        try
        {
                int b=0;
                int a=1/b;
        }
        catch(DivideByZeroException ex)
        {
                throw new MiaException("Operazione impossibile",ex);
        }
}
catch(MiaException mp)
{
        Console.WriteLine("Message {0}",mp.Message);
        Console.WriteLine("InnerException: {0}",mp.InnerException.Message);
}
```

GESTIONE FILE

FILE E DIRECTORY

Il namespace *System.IO* fornisce due coppie di classi per la gestione di file e directory, per i file si hanno le classi *File* e *FileInfo*, mentre per directory le classi *Directory* e *DirectoryInfo*.
La classe *File* contiene solo metodi statici.

Metodo	Descrizione
Open	Apre un file esistente e restituisce un oggetto FileStream.
Create	Crea un file con nome e percorso specificati..
Delete	Elimina il file specificato.
Copy	Copia un file esistente in un'altra posizione.
Move	Sposta un file esistente in un'altra posizione.
Exists	Verifica se un determinato file esiste.
OpenRead	Apre un file esistente in lettura.
OpenWrite	Apre un file esistente in scrittura.
CreateText	Crea un file di testo con nome e percorso specificati.
OpenText	Apre un file di testo in lettura.
AppendText	Apre un file di testo per appendere del testo a quello esistente.

Questi metodi hanno diversi overload , quindi è possibile usarli in modi diversi.

```
File.Create (@"c:\ova.txt");
File.Copy (@"c:\prova.txt","c:\prova2.txt");
File.Move (@"c:\prova.txt","c:\prova3.txt");
File.Delete (@"c:\prova.txt");
```

Questi metodi possono generare eccezioni.

```
try
{
        File.Delete (@"c:\temp\prova.txt");
}
        catch(IOException ioe)
{
        //gestione dell'eccezione
}
```

Prima di eliminare un file è buona regola verificare se esiste.

```
If (!File.Exists(txtOrig.Text)) File.Delete(txtOrig.Text);
```

La classe *Directory* contiene questi metodi.

Metodo	Descrizione
CreateDirectory	Crea una directory con nome e percorso specificati.
Delete	Elimina lsa directory specificata.
Exists	Copia una directory esistente in un'altra posizione.
Move	Sposta una directory esistente in un'altra posizione.
GetFileSystemEntries	Restituisce i file e le directory contenuti nella directory specificata
GetFiles, GetDirectories	Restituiscono i nomi dei file o delle sottodirectory di una directory specificata.
GetDirectoryRoot	Restituisce la radice del percorso specificato.
GetCurrentDirectory	Restituisce la directory di lavoro corrente dell'applicazione.
GetLogicalDrives	Restituisce i drive logici presenti nel sistema.
GetParent	Restituisce la directory madre del percorso specificato
AppendText	Apre un file di testo per appendere del testo a quello esistente.

Sia la classe *File* sia la classe *Directory* permettono di ottenere ed impostare i tempi di creazione o di accesso ad un dato elemento.

```
string creazione=File.GetCreationTime(@"c:\pippo.txt").ToString();
File.SetLastWriteTime (@"c:\pippo.txt",new DateTime(2000,8,2);
string path=@"c:\nuova cartella";
Directory.Create(path);
Directory.SetLastAccessTime(path,new DateTime(1981,11,25));
```

Le due classi *FileInfo* e *DirectoryInfo* forniscono anche metodi che permettono di ottenete altre informazioni sui file, come gli attributi, la data di creazione, la data di lettura e di modifica.
A differenza delle classi File e Directory possono essere istanziate.
Il metodo statico per creare una directory:

```
Directory.CreateDirectory(@"c:\tmp");
```

è equivalente al metodo *Create* della classe *DirectoryInfo*.

```
DirectoryInfo di=new DirectoryInfo(txtOrigine2.Text);
di.Create();
```

Il primo metodo è più veloce perché non ha bisogno d'istanziare un oggetto, ma nel secondo l'oggetto creato è disponibile per eseguire altre operazioni sulla stessa directory.

Esempio.

```
DirectoryInfo di=new DirectoryInfo(path);
StringBuilder sb=new StringBuilder();
sb.Append("Data di creazione: "+di.CreationTime+"\n");
sb.Append("Ultimo accesso: "+di.LastAccessTime+"\n");
sb.Append("Ultima scrittura: "+di.LastWriteTime+"\n");
sb.AppendFormat("{0} contiene\n{1} file e {2} subdirectory",
di.FullName,
di.GetFiles().Length,
di.GetDirectories().Length);
MessageBox.Show(sb.ToString());
```

FILE BINARI

La scrittura e lettura di file sono effettuate per mezzo di classi che rappresentano il concetto di stream.
Un oggetto *FileStream* crea uno stream, è possibile effettuare operazioni di apertura, scrittura, lettura e chiusura del file.
Per istanziare un *FileStream* si può utilizzare uno dei costruttori della classe:
string path=@"c:\pippo.exe";
FileStream fs=new FileStream(path, FileMode.Open, FileAccess.Read);
oppure ricorrere ai metodi statici della classe *File* o utilizzare un oggetto *FileInfo*:
FileStream fs2=File.OpenRead(path);
FileInfo fi=new FileInfo(path);
Fi.OpenWrite();
Quando si crea un FileStream si specificano le seguenti informazioni.

1. Il nome del file.
2. La modalità *FileMode*.
3. Il tipo di accesso *FileAccess*.
4. La modalità di condivisione *FileShare*.

Questi parametri sono rappresentati da tre tipo enumerativi.

Enumerazione	Valori	Descrizione
FileMode	Append, Create, CreateNew, Open, OpenOrCreate, Truncate	Specifica la modalità di apertura o di creazione di un file.
FileAccess	Read, Write,ReadWrite	Specifica le modalità di accesso in lettura, scrittura o entrambe. Per default l'accesso è ReadWrite.
FileShare	Inheritable, None, Read, ReadWrite, Write	Specifica il tipo di accesso che altri FileStream possono avere sullo stesso file, per default è Read, cioè di sola lettura.

Leggere dati per mezzo di un *FileStream,* due metodi.
1. Legge il byte successivo.
int nuovoByte=fs.ReadByte();
Se si è raggiunta la fine del file, il metodo *ReadByte* restituisce il valore -1
2. Legge blocchi di byte di lunghezza predefinita e li inserisce in un array, si usa il metodo *Read* i cui parametri sono: l'array, l'offset dalla posizione corrente nel file dal quale effettuare la lettura, il numero di byte da leggere e restituisce il numero di byte effettivamente letti.
byte[] bytesLetti=new byte[16];
int nBytesLetti=fs.Read(bytesLetti, 0, 16);

Scrivere dati per mezzo di un *FileStream,* due metodi.
1. Scrive un byte sullo stream.
byte unByte=255;
fs.WriteByte(unByte);
2. Scrive blocchi di byte di lunghezza predefinita.

```
byte[] bytesDaScrivere=new byte[16];
for(int i=0;i<16;i++)
{
bytesDaScrivere[i]=i;
}
fs.Write(bytesDaScrivere, 0, 16);
```

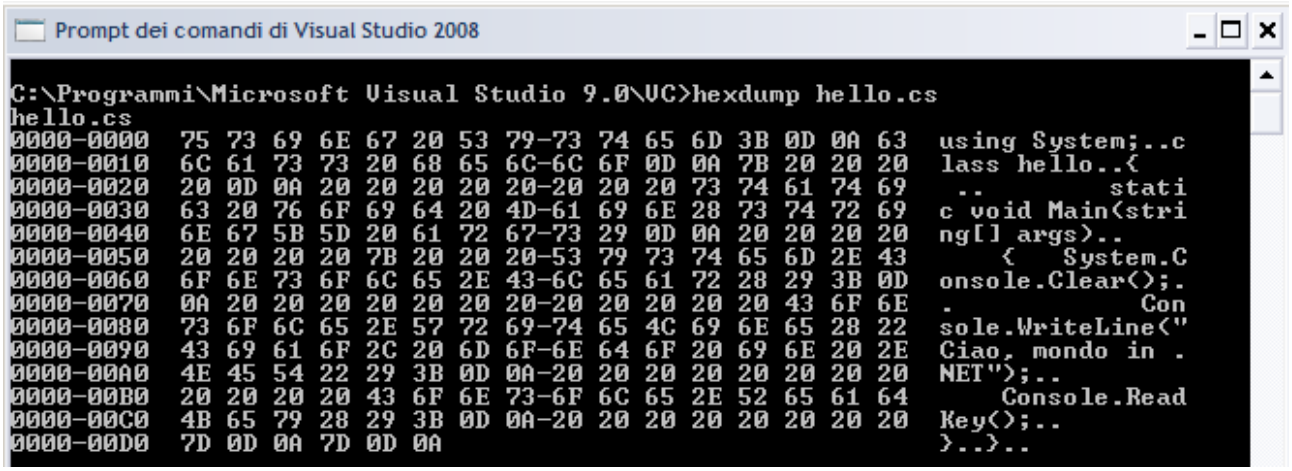
Chiusura di un *FileStream*.
fs.Close();

Altri metodi.
int dim=fs.Length; *// dimensione del file*
fs.Seek(100); *// posiziona al byte 100;*
fs.Lock(); *// blocca l'accesso*
fs.Unlock(); *// e lo sblocca*

Esempio: progettare il dump di un file.

```
using System;
using System.IO;
class HexDump
{
    public static int Main(string[] astrArgs)
    {
        Console.Clear();
        if (astrArgs.Length == 0)
        {
           Console.WriteLine("Sintassi: HexDump file1 ");
            return 1;
        }
        foreach (string strFileName in astrArgs) DumpFile(strFileName);
        return 0;
    }
    protected static void DumpFile(string strFileName)
    {
        FileStream fs;
        try
        {
            fs = new FileStream(strFileName, FileMode.Open,
                        FileAccess.Read, FileShare.Read);
        }
        catch (Exception exc)
        {
            Console.WriteLine("HexDump: {0}", exc.Message);
            return;
        }
        Console.WriteLine(strFileName);
        DumpStream(fs);
        fs.Close();
    }
    protected static void DumpStream(Stream stream)
    {
        byte[] abyBuffer = new byte[16];
        long   lAddress  = 0;
        int    iCount;
        while ((iCount = stream.Read(abyBuffer, 0, 16)) > 0)
        {   Console.WriteLine(ComposeLine(lAddress, abyBuffer, iCount));
            lAddress += 16;
        }
    }
    public static string ComposeLine(long lAddress, byte[] abyBuffer, int iCount)
    {
```

```
        string str =String.Format("{0:X4}-{1:X4}  ",(uint) lAddress / 65536, (ushort) lAddress);
        for (int i = 0; i < 16; i++)
        {     str += (i < iCount) ?
              String.Format("{0:X2}", abyBuffer[i]) : "  ";
              str += (i == 7 && iCount > 7) ? "-" : " ";
        }
        str += " ";
        for (int i = 0; i < 16; i++)
        {     char ch = (i < iCount) ? Convert.ToChar(abyBuffer[i]) : ' ';
             str += Char.IsControl(ch) ? "." : ch.ToString();
        }
        return str;
    }
}
```

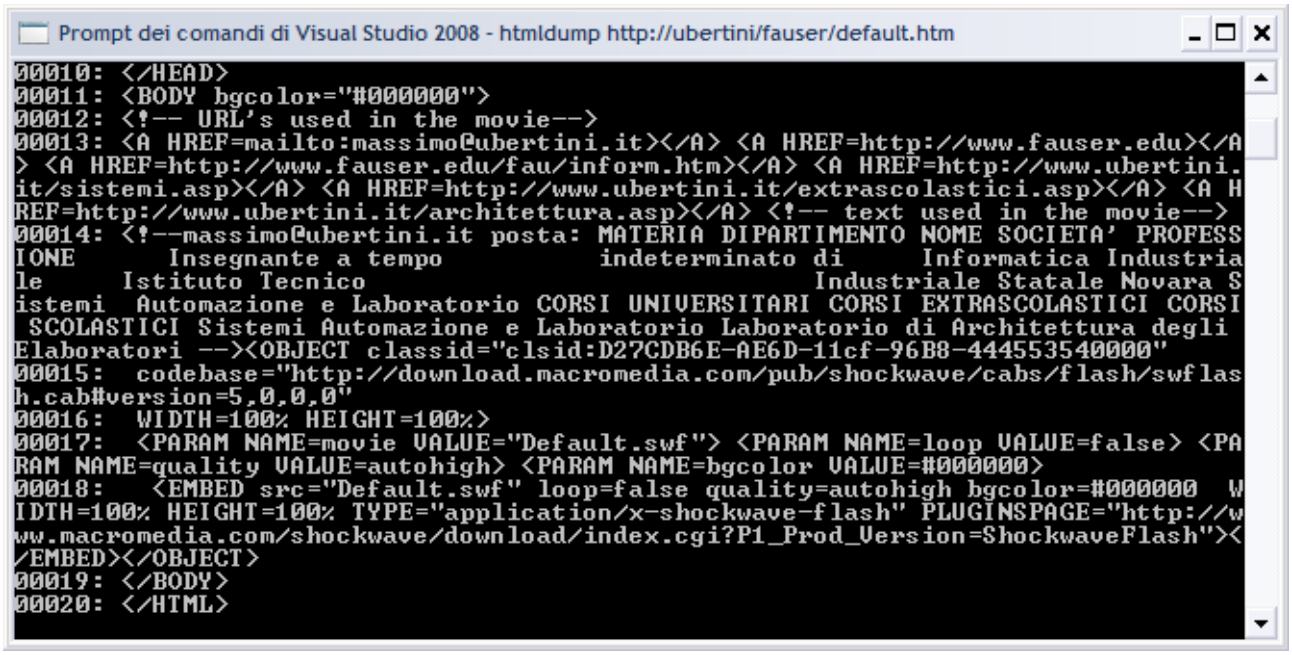


Esempio: progettare il dump di un **URI** (*Uniform Resource Identifier*).

```
using System;
using System.IO;
using System.Net;
class HtmlDump
{
    public static int Main(string[] astrArgs)
    {   Console.Clear();
        if (astrArgs.Length == 0)
        {
            Console.WriteLine("Sintassi: HtmlDump URI");
            return 1;
        }
        WebRequest webreq;
        WebResponse webres;
        try
        {
            webreq = WebRequest.Create(astrArgs[0]);
            webres = webreq.GetResponse();
        }
        catch (Exception exc)
        {
            Console.WriteLine("HtmlDump: {0}", exc.Message);
            return 1;
        }
```

```
        if (webres.ContentType.Substring(0, 4) != "text")
        {
            Console.WriteLine("HtmlDump: URI must be a text type.");
            return 1;
        }
        Stream      stream = webres.GetResponseStream();
        StreamReader strrdr = new StreamReader(stream);
        string      strLine;
        int         iLine = 1;
        while ((strLine = strrdr.ReadLine()) != null)
            Console.WriteLine("{0:D5}: {1}", iLine++, strLine);
        stream.Close();
        Console.ReadKey();
        return 0;
    }
}
```

C:\Programmi\Microsoft Visual Studio 9.0\VC>htmldump http://ubertini/fauser/default.htm


Prompt dei comandi di Visual Studio 2008 - htmldump http://ubertini/fauser/default.htm
00010: </HEAD>
00011: <BODY bgcolor="#000000">
00012: <!-- URL's used in the movie-->
00013: <A HREF=mailto:massimo@ubertini.it></A> <A HREF=http://www.fauser.edu></A> <A HREF=http://www.fauser.edu/fau/inform.htm></A> <A HREF=http://www.ubertini.it/sistemi.asp></A> <A HREF=http://www.ubertini.it/extrascolastici.asp></A> <A HREF=http://www.ubertini.it/architettura.asp></A> <!-- text used in the movie-->
00014: <!--massimo@ubertini.it posta: MATERIA DIPARTIMENTO NOME SOCIETA' PROFESSIONE Insegnante a tempo indeterminato di Informatica Industriale Istituto Tecnico Industriale Statale Novara Sistemi Automazione e Laboratorio CORSI UNIVERSITARI CORSI EXTRASCOLASTICI CORSI SCOLASTICI Sistemi Automazione e Laboratorio Laboratorio di Architettura degli Elaboratori -->
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
00015: codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=5,0,0,0"
00016: WIDTH=100% HEIGHT=100%>
00017: <PARAM NAME=movie VALUE="Default.swf"> <PARAM NAME=loop VALUE=false> <PARAM NAME=quality VALUE=autohigh> <PARAM NAME=bgcolor VALUE=#000000>
00018: <EMBED src="Default.swf" loop=false quality=autohigh bgcolor=#000000 WIDTH=100% HEIGHT=100% TYPE="application/x-shockwave-flash" PLUGINSPAGE="http://www.macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash"></EMBED></OBJECT>
00019: </BODY>
00020: </HTML>
```

## SCRIVERE E LEGGERE TIPI PRIMITIVI

La classe *FileStream* permette di leggere e scrivere byte nudi, se, invece, si vuole lavorare con i tipi di dati primitivi, la Base Class Library fornisce le classi *BinaryReader* e *BinariWriter*.

Per costruire un oggetto *BinaryWriter*, bisogna passare al suo costruttore un parametro FileStream, con il quale si è creato il file.

```
FileStream fs=new FileStream("file.dat",FileMode.Create);
BinaryWriter bw=new BinaryWriter(fs);
```

Il metodo *Write* dispone di molti overload che permettono di scrivere un valore di un tipo primitivo qualsiasi ed array di byte o char.

```
bool bVal=true;
int nVal=-100;
uint unVal=uint.MaxValue;
byte byteVal=8;
sbyte sbyteVal=-8;
```
```

```
char chVal='a';
byte[] bytes=new byte[4]{20,30,40,50};
char[] chars=new char[5]{'h','e','l','l','o'};
string str="world";
double dVal=0.123d;
float fVal=3.14f;
long lVal=long.MaxValue;
ulong ulVal=ulong.MinValue;
decimal decVal=1000000000M;
short sVal=-32768;
ushort usVal=32767;
bw.Write(bVal);
bw.Write(nVal);
bw.Write(unVal);
bw.Write(byteVal);
bw.Write(sbyteVal);
bw.Write(bytes);
bw.Write(bytes,1,2);
bw.Write(chVal);
bw.Write(chars);
bw.Write(chars,0,3);
bw.Write(str);
bw.Write(dVal);
bw.Write(fVal);
bw.Write(lVal);
bw.Write(ulVal);
bw.Write(decVal);
bw.Write(sVal);
bw.Write(usVal);
bw.Close();
```

Leggere i dati salvati precedentemente.

```
FileStream fs=new FileStream(@"C:\file.dat",FileMode.Open);
BinaryReader br=new BinaryReader(fs);
```

I dati devono essere letti nello stesso ordine in cui sono stati scritti.

```
bool bVal=br.ReadBoolean();
int nVal=br.ReadInt32();
uint unVal=br.ReadUInt32();
byte byteVal=br.ReadByte();
sbyte sbyteVal=br.ReadSByte();
byte[] bytes=br.ReadBytes(4);
bytes=br.ReadBytes(2);
char chVal=br.ReadChar();
char[] chars=br.ReadChars(5);
chars=br.ReadChars(3);
string str=br.ReadString();
double dVal=br.ReadDouble();
float fVal=br.ReadSingle();
long lVal=br.ReadInt64();
ulong ulVal=br.ReadUInt64();
decimal decVal=br.ReadDecimal();
short sVal=br.ReadInt16();
ushort usVal=br.ReadUInt16();
br.Close();
```

FILE DI TESTO

È possibile lavorare con file di testo continuando ad usare la classe *FileStream*, o le classi *BinaryReader* e *BinaryWriter*; il .NET Framework mette a disposizione altre due classi per scrivere e leggere linee di testo.

1. *StreamReader*
2. *StreamWriter*

```
StreamReader sr=new StreamReader(@"c:\pippo.txt");
```

Oppure con le classi *File* o *FileInfo*.

```
sr=File.OpenText(@"c:\pippo.txt");
FileInfo fi=new FileInfo(@"c:\pippo.txt");
sr=fi.OpenText();
```

Per leggere un file di testo si usa il metodo *ReadToEnd* che ritorna una stringa con l'intero contenuto del file.

```
string strContent=sr.ReadToEnd();
```

Per leggere un file di testo riga per riga si usa il metodo *ReadLine,* raggiunta la fine del file ritorna null.

```
string linea=sr.ReadLine();
```

Per leggere un file di testo carattere per carattere si usano i metodi.

1. *Read,* ritorna -1 raggiunta la fine del file, per cui se si vuole ottenere un carattere bisogna effettuare un cast esplicito.
2. *Peek*, ma non avanza nel file.

```
char c;
int carattere=sr.Read();
if(carattere!=-1) c=(char)carattere;
```

Il metodo *Read* ha un secondo overload che permette di leggere un dato numerico di caratteri ed inserirli in un array.

```
int toRead=16;
char[] charsLetti=new char[toRead];
int nLetti=sr.Read(charsLetti,0, toRead);
str.close();
```

Oppure.

```
using (StreamReader sr = new StreamReader("file.txt"))
{
String line;
while ((line = sr.ReadLine()) != null)
Console.WriteLine(line);
}
```

Per scrivere linee di testo si usa la classe *StreamWriter* con il metodo *Writer* con un parametro stringa, il metodo *WriteLine* aggiunge automaticamente il carattere di fine riga.

```
using System;
using System.IO;
class StreamWriterDemo
{
    public static void Main()
    {
        StreamWriter sw = new StreamWriter("file.txt", true);
        Console.Clear();
        sw.WriteLine("Scrivo il mio file con StreamWriter il {0}",DateTime.Now);
        sw.Close();
    }
}
```

Gli overload permettono di scrivere diversi tipi di dato: carattere per carattere, un array di

caratteri.

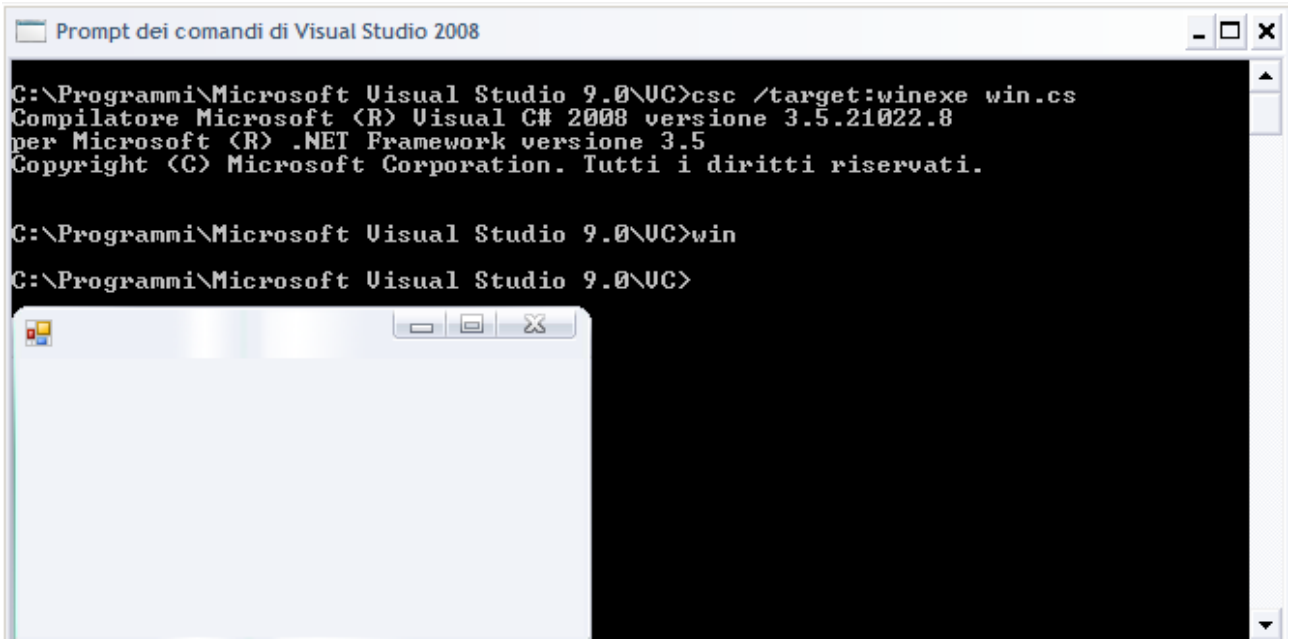
```
char ch=’a’;
sw.Write(ch);
char[] charArray = {'a','b','c','d','e','f','g','h','i','j','k','l','m'};
sw.Write(charArray);
sw.Write(charArray,3,5);//scrive 5 caratteri a partire dall’indice 3
```

APPLICAZIONI WINDOWS

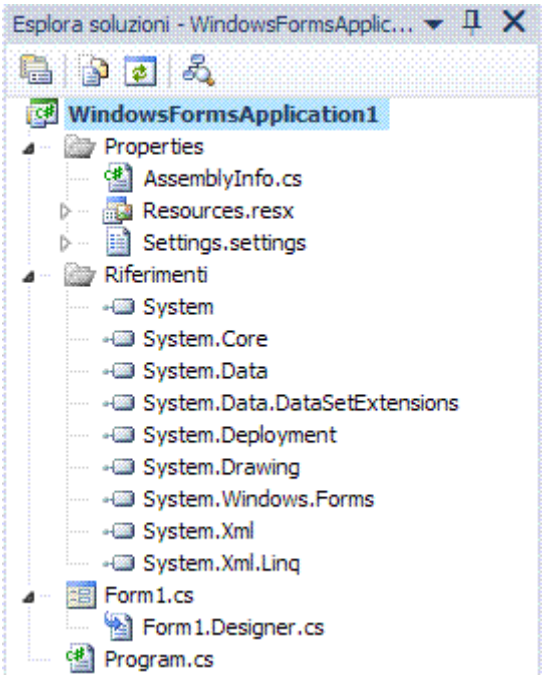
Le applicazioni a finestre: sono quelle Windows, con i bottoni, menu; si progettano mediante l'uso del *namespace System.Windows.Forms*.
Un'applicazione ha almeno una finestra che si crea mediante una classe.
Il ciclo di vita di un'applicazione parte con la creazione della form principale e la sua visualizzazione a schermo intero e termina con la pulitura delle risorse impegnate e la distruzione della form.
Lo startup avviene per mezzo del metodo statico *Run* della classe *Application*, che prende come unico parametro in ingresso un'istanza di una classe derivata da *Form*, questo provoca la visualizzazione e a questo punto l'applicazione resta in attesa dell'interazione con l'utente o di altri eventi.
Esempio da console.

```
using System.Windows.Forms;
class PrimaForm : Form
{
   static void Main()
   { Application.Run(new Form()); }
}
```

La classe *Application* fornisce il metodo *Exit* che provoca la chiusura delle finestre e la fine dell'applicazione, ma è buona regola di programmazione usare il metodo *Close* sulla finestra principale per rilasciare le risorse occupate dall'applicazione stessa.



Esempio da Visual Studio.
Aprire un progetto, *File/Nuovo progetto… (CTRL+N).*
Nella finestra *Nuovo progetto*, selezionare *Tipi di progetto*.
Altri linguaggi/Visual C#/Applicazione Windows Form.
Nel nuovo progetto C# inserisce automaticamente i seguenti file.



Properties
File AssemblyInfo.cs
Sono presenti tutte le informazioni generali dell'applicazione, dal nome del progettista al Copyright ed altre informazioni generali.

Riferimenti
Contiene le librerie esterne che utilizzerà l'applicazione.

File Form1.Designer.cs
Ha il codice seguente già inserito.

```
namespace WindowsFormsApplication1
{
    partial class Form1
    {
        /// <summary>
        /// Variabile di progettazione necessaria.
        /// </summary>
        private System.ComponentModel.IContainer components = null;
        /// <summary>
        /// Liberare le risorse in uso.
        /// </summary>
        /// <param name="disposing">ha valore true se le risorse gestite devono essere eliminate, false in caso contrario.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #region Codice generato da Progettazione Windows Form
        /// <summary>
        /// Metodo necessario per il supporto della finestra di progettazione. Non modificare
```

```
        /// il contenuto del metodo con l'editor di codice.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.Text = "Form1";
        }
        #endregion
    }
}
```

File Form1.cs
È la finestra di progettazione del form.

File Program.cs
Ha il codice seguente già inserito.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    static class Program
    {
        // Punto di ingresso principale dell'applicazione.
        [STAThread]
        static void Main()
        {   Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

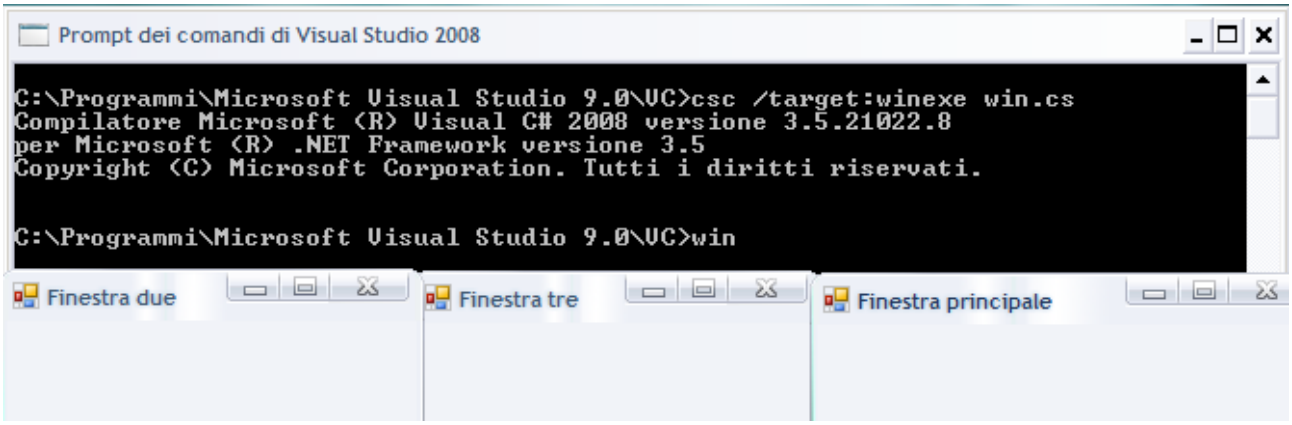
LA CLASSE FORM

```
using System.Windows.Forms;
class PrimaForm: Form
{
        static void Main()
        { PrimaForm f=new PrimaForm();
          f.Show();
        }
}
```

L'istruzione *New* crea un'istanza di *PrimaForm*, ma non è sufficiente a visualizzarla sullo schermo perché bisogna invocare il metodo *Show*, un'alternativa è l'assegnazione del valore *true* alla proprietà *Visible*; il metodo per nascondere, ma non distruggere, è *Hide*.
La finestra creata sarà una finestra senza controllo e senza titolo, ma non si fa in tempo a vederla a monitor perché subito dopo la chiamata del metodo *Show*, l'esecuzione del codice ritorna al *Main* e l'applicazione termina.
Per questo motivo si deve usare *Application.Run* per la form principale dell'applicazione, mentre per le form succesive si usa il metodo *Show*.

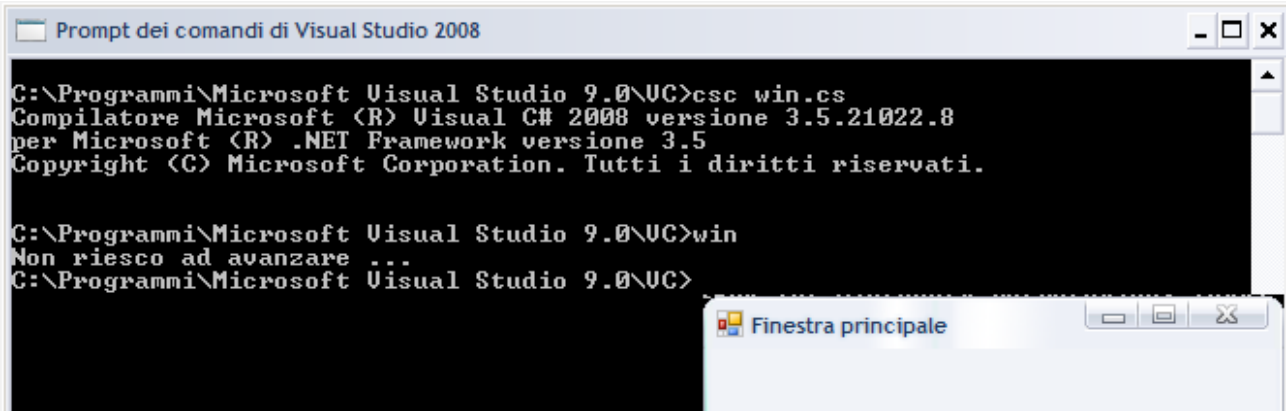
```
using System.Windows.Forms;
```

```
class PrimaForm: Form
{       public PrimaForm()
        {    Form form2=new Form();
             form2.Text="Finestra due";
            form2.Show();
            Form form3=new Form();
            form3.Text="Finestra tre";
             form3.Visible=true;
}
        static void Main()
        {      PrimaForm f = new PrimaForm();
              f.Text="Finestra principale";
              Application.Run(f);
        }
}
```



Chiudendo la form principale, l'applicazione termina chiudendo tutte le form aperte, mentre le form secondarie, che sono indipendenti, possono essere chiuse senza che l'applicazione ne risenta.
In questo caso, invece, la finestra sarà visualizzata, ma una volta chiusa la finestra, l'applicazione resterà bloccata.

```
using System;
using System.Windows.Forms;
class PrimaForm: Form
{
        static void Main()
        { PrimaForm f = new PrimaForm();
          f.Text="Finestra principale";
          Application.Run(f);
         Console.Write ("Non riesco ad avanzare ...");
        }
}
```



Proprietà

```
f.icon=new icon ("1.ico");                          // carica un'icona

f.Size=new Size(300,400);                           // imposta le dimensioni
f.Width=300;form.Height=400;                        // equivalente alla riga precedente

f.StartPosition=FormStartPosition.CenterScreen      //centra la form sullo schermo
f.StartPosition=FormStartPosition.CenterParent;     //centra rispetto alla form madre
f.Location=new Point(10,10);                        // posizione sullo schermo
this.BackColor=Color.Red;                           // colore di sfondo della form

// impostano dimensione e posizione della form
f.Bounds=new Rectangle(10,10, 300,400);             // x, y, larghezza e altezza
Point p=new Point(10,10);
Size s=new Size(300,400);
f.DesktopBounds=new Rectangle(p,s);                 // equivalente

// dimensione massima e minima alla form
f.MinimumSize=new Size(200,150);
f.MaximumSize=new Size(800,600);
```

// l'enumerazione *FormBorderStyle* permette di definire i bordi di una form.

```
public SplashForm()
{
this.FormBorderStyle = FormBorderStyle.None;
this.MaximizeBox = false;
this.MinimizeBox = false;
this.StartPosition = FormStartPosition.CenterScreen;
this.ControlBox = false;
}
```

La proprietà *Opacity* crea un effetto trasparenza.

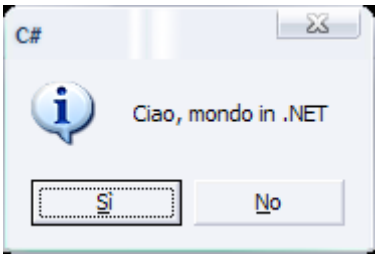
```
using System.Windows.Forms;
class PrimaForm: Form
{
        public void ShowSplash()
        {
                this.Opacity=0.0;
                this.Visible=true;
                for(double i=0;i<=1.0;i+=0.05)
                {
                        this.Opacity=i;
```

```
                    System.Threading.Thread.Sleep(100);//si ferma per 100 millisecondi
                }
        this.Opacity=1.0;
        }
        static void Main()
        {       PrimaForm f = new PrimaForm();
                f.Text="Finestra principale";
                f.ShowSplash();
        }
}
```

Message Box

È la finestra di messaggio per l'utente.

```
using System.Windows.Forms;
class Messaggio
{
        static void Main()
        {
        MessageBox.Show("Ciao, mondo in .NET","C#",MessageBoxButtons.YesNo,
        MessageBoxIcon.Asterisk);
        }
}
```




```
MessageBox.Show (Environment.GetFolderPath (Environment.SpecialFolder.Personal),
"La mia cartella");
```

Del metodo *MessageBox* sono disponibili overload che permettono la personalizzazione.

```
MessageBox.Show
DialogResult MessageBox.Show(string text)
DialogResult MessageBox.Show(string text,string caption)
DialogResult MessageBox.Show(string text,string caption,MessageBoxButtons buttons)
DialogResult MessageBox.Show(string text,string caption,MessageBoxButtons buttons,MessageBoxIcon
icon)
DialogResult MessageBox.Show(string text,string caption,MessageBoxButtons buttons,MessageBoxIcon
icon,MessageBoxDefaultButton defaultButton)
DialogResult MessageBox.Show(string text,string caption,MessageBoxButtons buttons,MessageBoxIcon
icon,MessageBoxDefaultButton defaultButton,MessageBoxOptions options)
```

Membro	Descrizione
OK	Visualizza il pulsante Ok.
OKCancel	Visualizza i pulsanti Ok ed Annulla..
AbortRetryIgnore	Visualizza i tre pulsanti Interrompi, Riprova, Ignora.
RetryCancel	Visualizza i pulsanti Riprova e Annulla.
YesNo	Visualizza i pulsanti Sì e No.
YesNoCancel	Visualizza i pulsanti Sì, No e Annulla.

Inoltre può contenere al massimo tre pulsanti, il tipo enumerativo *MessageBoxButtons* permette di specificare il tipo ed il numero, per default solo il pulsante *OK*, la scelta dell'utente sarà restituita come valore di ritorno del metodo, che è di tipo enumerativo *DialogResult*.

Il parametro *MessageBoxIcon* visualizza un'icona.

Membro	Descrizione
Asterisk	Visualizza una i blu minuscola in un fumetto bianco.
Error	Visualizza una croce bianca in un cerchio rosso.
Exclamation	Visualizza un punto esclamativo nero in un triangolo giallo.
Hand	Visualizza una croce bianca in un cerchio rosso.
Information	Visualizza una i blu minuscola in un fumetto bianco.
None	Non visualizza nessuna icona.
Question	Visualizza un punto interrogativo blu in un fumetto bianco.
Stop	Visualizza una croce bianca in un cerchio rosso.
Warning	Visualizza un punto esclamativo nero in un triangolo giallo.

Esempio: visualizzare in sequenza tutte le icone.

```
using System;
using System.Windows.Forms;
public class TestMessageBoxIcon
{
        public static void Main()
        {
            Array icons=Enum.GetValues(typeof(MessageBoxIcon));
            foreach(MessageBoxIcon icon in icons)
                {
        MessageBox.Show("Visualizzo un'icona "+icon,"MessageBoxIcon "+icon,
        MessageBoxButtons.OK,icon);
                }
        }
}
```

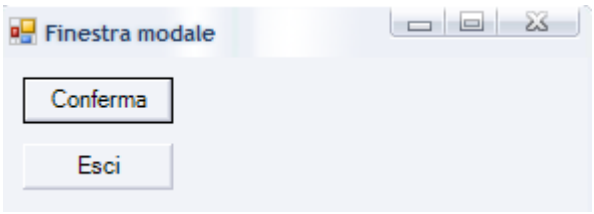
Se in una *MessageBox* sono visualizzati più pulsanti, è possibile specificare quale fra essi deve possedere il focus tramite*MessageBoxDefaultButton* che è un tipo enumerativo che può assumere i valori *Button1*, *Button2* e *Button3*.

Finestre di dialogo

Sono finestre modali, alla chiusura ritornano *DialogResult*.

```
using System.Windows.Forms;
using System.Drawing;
class PrimaForm: Form
{
        static void Main()
        {
        PrimaForm form1 = new PrimaForm();
         form1.Text="Finestra modale";            // Crea 2 Button da usare
        Button button1 = new Button ();
        Button button2 = new Button ();
        button1.Text = "Conferma";
        button1.Location = new Point (10, 10);
        button2.Text = "Esci";
        button2.Location = new Point (button1.Left, button1.Height + button1.Top + 10);
        form1.AcceptButton = button1;
        form1.CancelButton = button2;
        form1.Controls.Add(button1);
        form1.Controls.Add(button2);
```

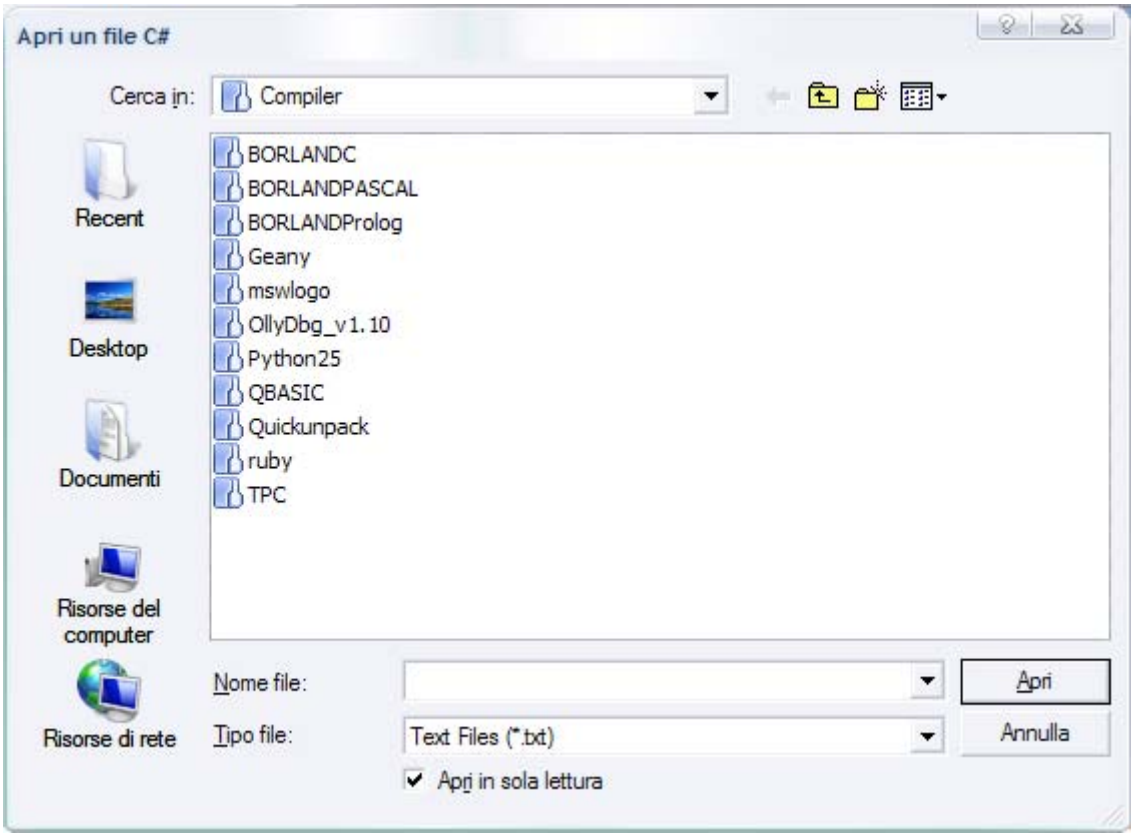
```
        form1.ShowDialog();
        }
}
```



Common Dialog
Sono le classi seguenti.

Metodo	Descrizione
OpenFileDialog	Rappresenta la finestra di dialogo per la scelta di un file da aprire.
SaveFileDialog	Rappresenta la finestra di dialogo per il salvataggio di un file.
PrintDialog	Rappresenta la finestra di dialogo per l'impostazione delle proprietà di stampa.
PrintPreviewDialog	Rappresenta la finestra di dialogo per la visualizzazione dell'anteprima di stampa di un documento.
PageSetupDialog	Rappresenta la finestra di dialogo per l'impostazione del formato di stampa, come la dimensione e i margini delle pagine.
FontDialog	Visualizza e permette di scegliere uno dei font installati nel sistema operativo.
ColorDialog	Visualizza e permette la selezione di un colore standard del sistema e la definizione di colori personalizzati.

```
using System.Windows.Forms;
class PrimaForm: Form
{
        static void Main()
        {
        OpenFileDialog dlg=new OpenFileDialog();
        dlg.Title="Apri un file C#";
        dlg.Filter="C# File (*.cs)|*.cs|Text Files (*.txt)|*.txt|All Files (*.*)|*.*";
        dlg.FilterIndex=2;
        dlg.InitialDirectory=@"C:\compiler";
        dlg.ShowReadOnly=true;
        dlg.ReadOnlyChecked=true;
        dlg.CheckFileExists=false;
        if(dlg.ShowDialog()==DialogResult.OK)
        {
        MessageBox.Show("Apertura file "+dlg.FileNames[0]);
        }
        }
}
```



Per selezionare ed aprire file, è usata la classe *OpenFileDialog*, la proprietà *Title* imposta il titolo della barra del titolo, la proprietà *Filter* è un filtro per la tipologia di file.

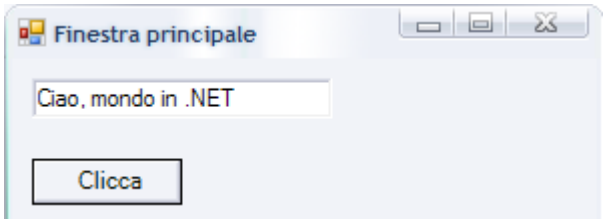
CONTROLLI

Derivano dalla classe: *System.Windows.Forms.Control*.

Controlli di testo

Esempio: progettare una form con una casella di testo ed un pulsante.

```
using System.Windows.Forms;
using System.Drawing;
class PrimaForm: Form
{
        static void Main()
        { PrimaForm f = new PrimaForm();
        f.Text="Finestra principale";
        // crea la casella di testo
        TextBox txt=new TextBox();
        txt.Location=new Point(10,10);
        txt.Size=new Size(150,30);
        f.Controls.Add(txt);
        // crea il bottone
        Button bt=new Button();
        bt.Text="Clicca";
        bt.Location=new Point(10,50);
        bt.Parent=f;
        bt.Parent=f;
         Application.Run(f);
        }
}
```



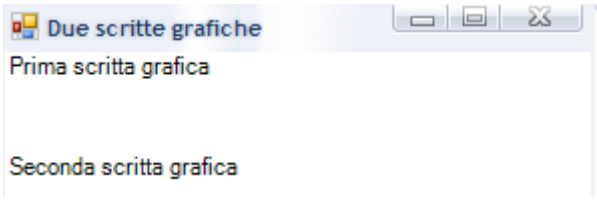
Per rimuovere il controllo basta invocare il metodo *Remove*.
f.Controls.Remove(txt);
Proprietà della classe Control.

Proprietà	Descrizione
Anchor	Specifica quali lati del controllo sono ancorati al suo contenitore.
BackColor	Imposta o ricava il colore di sfondo del controllo.
BackgroundImage	Imposta o ricava l'immagine di sfondo mostrata sulla superficie del controllo.
Bottom	Ricava la distanza fra il lato inferiore del controllo e il lato superiore del contenitore.
Bounds	Imposta o ricava la posizione e le dimensioni del controllo.
Cursor	Imposta o ricava il puntatore del mouse da visualizzare quando esso si trova sulla superficie del controllo.
Dock	Specifica quale lato del controllo è legato al corrispondente lato del contenitore, ad esempio Imposta o ricavando il valore a DockStyle.Fill, il controllo riempie sempre l'area del contenitore.
Enabled	Specifica se il controllo è abilitato o meno.
Focused	Indica se il controllo ha il focus.
Font	Imposta o ricava il tipo di carattere per il testo mostrato dal controllo.
ForeColor	Imposta o ricava il colore del testo mostrato dal controllo.
Heigth	Imposta o ricava l'altezza del controllo
Ieft	Imposta o ricava la coordinata x del lato sinistro del controllo.
Location	Imposta o ricava la posizione dell'angolo superiore sinistro del controllo.
Name	Imposta o ricava il nome del controllo, per default esso è una stringa vuota.
Parent	Imposta o ricava il controllo contenitore del controllo corrente.
Right	Ricava la distanza fra il lato destro del controllo e il lato sinistro del suo contenitore.
Size	Imposta o ricava le dimensioni del controllo.
TabIndex	Imposta o ricava il valore di tab order del controllo. Il tab order determina l'ordine con cui i controlli ricevono il focus alla pressione del tasto TAB.
TabStop	Specifica se il controllo può ricevere il focus alla pressione del tasto TAB.
Tag	Imposta o ricava un oggetto associato al controllo e che può contenere dati su di esso.
Text	Imposta o ricava il testo associato al controllo. Ad esempio per una form è il testo sulla barra del titolo, per una TextBox è il testo contenuto in essa.
Top	Imposta o ricava la coordinata y del lato superiore del controllo.
Visible	Specifica se il controllo viene visualizzato sul contenitore.
Width	Imposta o ricava la larghezza del controllo.

Esempio: progettare una form con due scritte di tipo grafico.

```
using System;
using System.Drawing;
using System.Windows.Forms;
class PrimaForm
{
    public static void Main()
    {
        Form form      = new Form();
        form.Text      = "Due scritte grafiche";
```

```
        form.BackColor = Color.White;
        form.Paint    += new PaintEventHandler(PaintHandler1);
        form.Paint    += new PaintEventHandler(PaintHandler2);
        Application.Run(form);
    }
    static void PaintHandler1(object objSender, PaintEventArgs pea)
    {
        Form     form = (Form)objSender;
        Graphics grfx = pea.Graphics;
        grfx.DrawString("Prima scritta grafica", form.Font,
                        Brushes.Black, 0, 0);
    }
    static void PaintHandler2(object objSender, PaintEventArgs pea)
    {
        Form     form = (Form)objSender;
        Graphics grfx = pea.Graphics;
        grfx.DrawString("Seconda scritta grafica", form.Font,
                        Brushes.Black, 0, 50);
    }
}
```



Esempio: progettare una form che aspetta.

```
using System.Threading;
using System.Windows.Forms;
class PrimaForm
{
    public static void Main()
    {
        Form form = new Form();
        form.Show();
        Thread.Sleep(2500);
        form.Text = "Thread e aspetta ...";
        Thread.Sleep(2500);
    }
}
```

Ricercare una stringa in una casella di testo.

```
string searchString = "hello";
int index = txt.Text.IndexOf(searchString, 0);
if (index != -1)
txt.Select(index, searchString.Length);
```

Casella di testo multi linea.

```
txt.Multiline = true;
txt.ScrollBars = ScrollBars.Vertical;
txt.WordWrap = true;
txt.Lines = new String[]{"linea1","linea2","linea3"};
```

Casella di testo con password.
txt.PasswordChar='';*

Casella di testo **RTF** (*Rich Text Format*).
```
rtf = new RichTextBox();
rtf.Dock = DockStyle.Fill;
//carica il file, se non esiste genera un'eccezione
rtf.LoadFile("C:\\documento.rtf");
//cerca la stringa hello
rtf.Find("hello");
//colora la seleziona e cambia il font
rtf.SelectionFont = new Font("Verdana", 12, FontStyle.Bold);
rtf.SelectionColor = Color.Red;
//salva il file
rtf.SaveFile("C:\\documento.rtf", RichTextBoxStreamType.RichText);
```

Controllo *Label*.
```
Label lab=new Label();
lab.Text="etichetta";
```

Label con immagine.
```
Label labImage=new Label();
ImageList immagini=new ImageList();
immagini.Images.Add(Image.FromFile("immagine.bmp");
labImage.ImageList=immagini;
labImage.ImageIndex=0;
labImage.Text="Icona";
```

Controlli di comando
La funzionalità di un pulsante è quella di eseguire un'azione quando l'utente clicca su di esso, per ottenere una notifica di questo evento bisogna associare al pulsante un gestore.
```
Class TestButton:Form
{
public TestButton()
{
Button bt=new Button();
bt.Text="Clicca";
bt.Location=new System.Drawing.Point(10,10);
bt.Parent=this;
bt.Click+=new EventHandler(bt_Click);
}
// installa un nuovo gestore per l'evento click
private void bt_Click(object sender, EventArgs e)
{
Button button=(Button)sender;
MessageBox.Show("Hai cliccato "+button.Text);
}
}
```

La classe *NotifyIcon* aggiunge un'icona all'area di notifica.
```
notify=new NotifyIcon();
notify.Text="Esempio di NotifyIcon";                    // tooltip
notify.Icon=new Icon(@"c:\icon.ico");
```

notify.Visible=true;

Controlli di selezione
Permettono di effettuare delle scelte tra più valori.

Controllo *CheckBox*, proprietà: *Text, Checked, AutoCheck*.
```
CheckBox chk=new CheckBox();
chk.Text="Disabilita";
```
Checked rappresenta lo stato: true o false.
```
chk.CheckedChanged+=new EventHandler(chk_CheckedChanged);
…
private void chk_CheckedChanged(object sender, EventArgs e)
{
CheckBox c=(CheckBox)sender;
c.Text=(c.Checked?"Checked":"Unchecked");
}
```
AutoCheck uguale a false.
```
chk.Click+=new EventHandler(chk_Click);
…
private void chk_Click(object sender, EventArgs e)
{
CheckBox c=(CheckBox)sender;
c.Checked=!c.Checked;
}
```
Può assumere un terzo stato indeterminato.
```
chk.ThreeState=true;
chk.CheckState=CheckState.Indeterminate;
```

Controllo *RadioButton* permette di scegliere una sola opzione fra tante.
```
radio1=new RadioButton();
radio1.Text="Scelta1";
if(radio1.Checked)
MessageBox.Show("Hai fatto la Scelta1");
```

Controllo *ListBox* permette di visualizzare una lista di elementi.
```
ListBox lb = new ListBox();
lb.Size = new System.Drawing.Size(200, 100);
lb.Location = new System.Drawing.Point(10,10);
lb.MultiColumn = true;                  // elementi su più colonne
lb.SelectionMode = SelectionMode.MultiExtended;
```
Per aggiungere elementi.
```
public void FillList(ListBox lb)
{
for(int i=0;i<10;i++)
lb.Items.Add("Elemento "+i);
}
lb.SetSelected(1, true);//seleziona l’elemento di indice 1
lb.SetSelected(3, false);//deseleziona l’elemento di indice 3
```
Ricercare un elemento.
```
string str="elemento3";
int x=lb.FindString(str);
if(x>-1)
{
MessageBox.Show(str+" si trova all'indice "+x);
```

```
}
else MessageBox.Show(str+" non trovato");
```

Controllo *ComboBox* rappresenta una casella combinata.

```
ComboBox cbo=new ComboBox();
cbo.DropDownStyle=ComboBoxStyle.Simple;
```

Aggiungere un elemento.

```
cbo.Items.Add("nuovo elemento");
cbo.Items.AddRange(new string[]{"a","b","c"});
c.Remove("nuovo elemento");
cbo.Items.RemoveAt(0);              //rimuove il primo elemento
```

L'elemento selezionato è restituito.

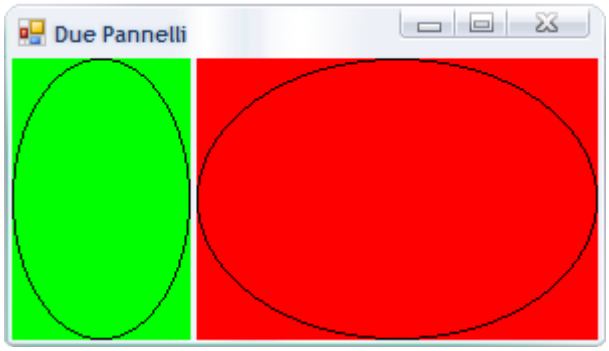
```
object selected=cbo.SelectedItem;
int indexSel=cbo.SelectedIndex;
```

Panel

Sono utilizzati per suddividere una form in base alla funzione.

```
using System;
using System.Drawing;
using System.Windows.Forms;
class TwoPanelsWithSplitter: Form
{
    public static void Main()
    {
        Application.Run(new TwoPanelsWithSplitter());
    }
    public TwoPanelsWithSplitter()
    {
        Text = "Due Pannelli";
        Panel panel1     = new Panel();
        panel1.Parent    = this;
        panel1.Dock      = DockStyle.Fill;
        panel1.BackColor = Color.Lime;
        panel1.Resize   += new EventHandler(PanelOnResize);
        panel1.Paint    += new PaintEventHandler(PanelOnPaint);
        Splitter split   = new Splitter();
        split.Parent     = this;
        split.Dock       = DockStyle.Right;
        Panel panel2     = new Panel();
        panel2.Parent    = this;
        panel2.Dock      = DockStyle.Right;
        panel2.BackColor = Color.Red;
        panel2.Resize   += new EventHandler(PanelOnResize);
        panel2.Paint    += new PaintEventHandler(PanelOnPaint);
    }
    void PanelOnResize(object obj, EventArgs ea)
    {
        ((Panel) obj).Invalidate();
    }
    void PanelOnPaint(object obj, PaintEventArgs pea)
    {
        Panel    panel = (Panel) obj;
        Graphics grfx  = pea.Graphics;
        grfx.DrawEllipse(Pens.Black, 0, 0,
```

```
                panel.Width - 1, panel.Height - 1);
    }
}
```

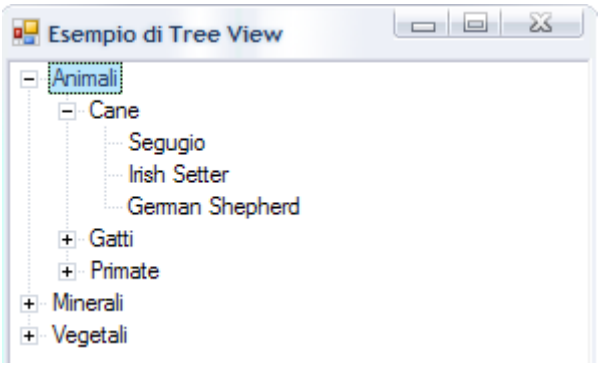


TreView

Consente di visualizzare una gerarchia di nodi, nello stesso modo in cui file e cartelle sono visualizzati nel riquadro sinistro della funzionalità Esplora risorse.

```
using System;
using System.Drawing;
using System.Windows.Forms;
class SimpleTreeView: Form
{
    public static void Main()
    {
        Application.Run(new SimpleTreeView());
    }
    public SimpleTreeView()
    {
        Text = "Esempio di Tree View";
        TreeView tree = new TreeView();
        tree.Parent = this;
        tree.Dock = DockStyle.Fill;
        tree.Nodes.Add("Animali");
        tree.Nodes[0].Nodes.Add("Cane");
        tree.Nodes[0].Nodes[0].Nodes.Add("Segugio");
        tree.Nodes[0].Nodes[0].Nodes.Add("Irish Setter");
        tree.Nodes[0].Nodes[0].Nodes.Add("German Shepherd");
        tree.Nodes[0].Nodes.Add("Gatti");
        tree.Nodes[0].Nodes[1].Nodes.Add("Calico");
        tree.Nodes[0].Nodes[1].Nodes.Add("Siamese");
        tree.Nodes[0].Nodes.Add("Primate");
        tree.Nodes[0].Nodes[2].Nodes.Add("Chimpanzee");
        tree.Nodes[0].Nodes[2].Nodes.Add("Ape");
        tree.Nodes[0].Nodes[2].Nodes.Add("Human");
        tree.Nodes.Add("Minerali");
        tree.Nodes[1].Nodes.Add("Calcio");
        tree.Nodes[1].Nodes.Add("Zinco");
        tree.Nodes[1].Nodes.Add("Iron");
        tree.Nodes.Add("Vegetali");
        tree.Nodes[2].Nodes.Add("Carote");
        tree.Nodes[2].Nodes.Add("Asparagi");
        tree.Nodes[2].Nodes.Add("Broccoli");
    }
```

}



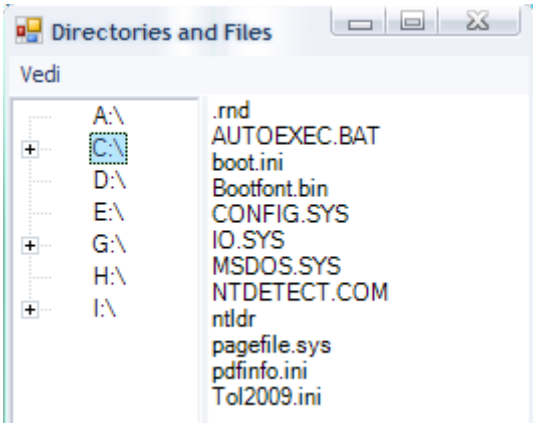
ListView
Permette di rappresentare una lista di elementi con testo ed eventuale icona.
Quattro modi di visualizzazione.
ListView lv=new ListView();
lv.View=View.Details;

Esempio: visualizzare directory e file.

```
using System;
using System.Drawing;
using System.IO;
using System.Windows.Forms;
class DirectoriesAndFiles: Form
{
    DirectoryTreeView dirtree;
    Panel             panel;
    TreeNode          tnSelect;
    public static void Main()
    {
        Application.Run(new DirectoriesAndFiles());
    }
    public DirectoriesAndFiles()
    {
        Text = "Directories and Files";
        BackColor = SystemColors.Window;
        ForeColor = SystemColors.WindowText;
        panel = new Panel();
        panel.Parent = this;
        panel.Dock = DockStyle.Fill;
        panel.Paint += new PaintEventHandler(PanelOnPaint);
        Splitter split = new Splitter();
        split.Parent = this;
        split.Dock = DockStyle.Left;
        split.BackColor = SystemColors.Control;
        dirtree = new DirectoryTreeView();
        dirtree.Parent = this;
        dirtree.Dock = DockStyle.Left;
        dirtree.AfterSelect +=
                new TreeViewEventHandler(DirectoryTreeViewOnAfterSelect);
        Menu = new MainMenu();
        Menu.MenuItems.Add("Vedi");
```

```
        MenuItem mi = new MenuItem("Refresh",
                        new EventHandler(MenuOnRefresh), Shortcut.F5);
        Menu.MenuItems[0].MenuItems.Add(mi);
    }
    void DirectoryTreeViewOnAfterSelect(object obj, TreeViewEventArgs tvea)
    {
        tnSelect = tvea.Node;
        panel.Invalidate();
    }
    void PanelOnPaint(object obj, PaintEventArgs pea)
    {
        if (tnSelect == null) return;
        Panel        panel     = (Panel) obj;
        Graphics     grfx      = pea.Graphics;
        DirectoryInfo dirinfo   = new DirectoryInfo(tnSelect.FullPath);
        FileInfo[]    afileinfo;
        Brush        brush     = new SolidBrush(panel.ForeColor);
        int          y         = 0;
        try
        {
            afileinfo = dirinfo.GetFiles();
        }
        catch
        {
            return;
        }
        foreach (FileInfo fileinfo in afileinfo)
        {
            grfx.DrawString(fileinfo.Name, Font, brush, 0, y);
            y += Font.Height;
        }
    }
    void MenuOnRefresh(object obj, EventArgs ea)
    {
        dirtree.RefreshTree();
    }
}
using System;
using System.Drawing;
using System.IO;
using System.Windows.Forms;
class DirectoryTreeView: TreeView
{
    public DirectoryTreeView()
    {
        Width *= 2;
        ImageList = new ImageList();
        RefreshTree();
    }
    public void RefreshTree()
    {
        BeginUpdate();
        Nodes.Clear();
        string[] astrDrives = Directory.GetLogicalDrives();
```

```
        foreach (string str in astrDrives)
        {
            TreeNode tnDrive = new TreeNode(str, 0, 0);
            Nodes.Add(tnDrive);
            AddDirectories(tnDrive);
            if (str == "C:\\") SelectedNode = tnDrive;
        }
        EndUpdate();
    }
    void AddDirectories(TreeNode tn)
    {
        tn.Nodes.Clear();
        string          strPath = tn.FullPath;
        DirectoryInfo   dirinfo = new DirectoryInfo(strPath);
        DirectoryInfo[] adirinfo;
        try
        {
            adirinfo = dirinfo.GetDirectories();
        }
        catch
        {
            return;
        }
        foreach (DirectoryInfo di in adirinfo)
        {
            TreeNode tnDir = new TreeNode(di.Name, 1, 2);
            tn.Nodes.Add(tnDir);
        }
    }
    protected override void OnBeforeExpand(TreeViewCancelEventArgs tvcea)
    {
        base.OnBeforeExpand(tvcea);
        BeginUpdate();
        foreach (TreeNode tn in tvcea.Node.Nodes)
            AddDirectories(tn);
        EndUpdate();
    }
}
```



MENU

Ci sono due diversi tipi di menu.

1. Classici: *MainMenu.*
2. Contestuali: *ContextMenu*.

Per costruire un *MainMenu* ci sono due costruttori.

MainMenu();
MainMenu(MenuItem[]);

Associare ad un form un menu.

MainMenu menuPrincipale=new MainMenu(menuItems);
miaForm.Menu= menuPrincipale;

Impostazione di un singolo menu.

MenuItem mnuFileNuovo=new MenuItem();
mnuFileNuovo.Text="&Nuovo";
MenuItem mnuFileApri=new MenuItem("Apri…");
MenuItem mnuSeparatore=new MenuItem("-");

Menu che appare sotto la barra del titolo.

MenuItem mnuFile=new MainMenu("&File",new MenuItem[]{mnuFileNuovo,mnuFileApri});

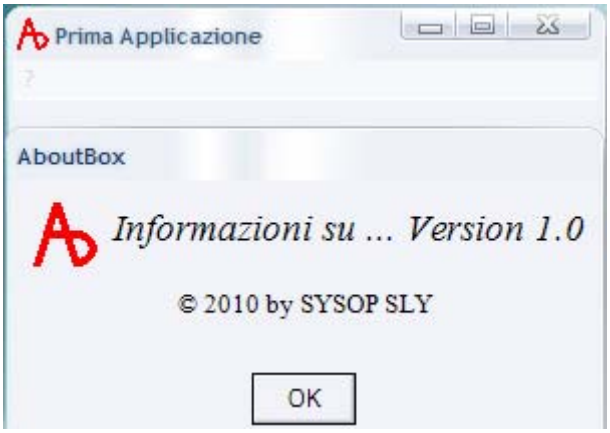
I menu creati non eseguono nessuna azione, occorre implementare il metodo di gestione dell'evento.

mnuFileApri.Click+= new System.EventHandler(this. mnuFileApri_Click);
private void mnuFileApri_Click(object sender, System.EventArgs e)
{
OpenFileDialog fd = new OpenFileDialog();
fd.ShowDialog();
}

Esempio.

```
using System;
using System.Drawing;
using System.Windows.Forms;
class AboutBox: Form
{
    public static void Main()
    {
        Application.Run(new AboutBox());
    }
    public AboutBox()
    {
        Text = "Prima Applicazione";
        Icon = new Icon(GetType(), "AboutBox.AforAbout.ico");

        Menu = new MainMenu();
        Menu.MenuItems.Add("?");
        Menu.MenuItems[0].MenuItems.Add("Informazioni su  ...",
                                new EventHandler(MenuAboutOnClick));
    }
    void MenuAboutOnClick(object obj, EventArgs ea)
    {
        AboutDialogBox dlg = new AboutDialogBox();
        dlg.ShowDialog();
    }
}
class AboutDialogBox: Form
{
    public AboutDialogBox()
    {
```

```
        Text = "AboutBox";
        StartPosition   = FormStartPosition.CenterParent;
        FormBorderStyle = FormBorderStyle.FixedDialog;
        ControlBox      = false;
        MaximizeBox     = false;
        MinimizeBox     = false;
        ShowInTaskbar   = false;
        Label label1     = new Label();
        label1.Parent    = this;
        label1.Text      = " Informazioni su ... Version 1.0 ";
        label1.Font      = new Font(FontFamily.GenericSerif, 14,
                               FontStyle.Italic);
        label1.AutoSize  = true;
        label1.TextAlign = ContentAlignment.MiddleCenter;
        Icon icon = new Icon(GetType(), "AboutBox.AforAbout.ico");
        PictureBox picbox = new PictureBox();
        picbox.Parent     = this;
        picbox.Image      = icon.ToBitmap();
        picbox.SizeMode   = PictureBoxSizeMode.AutoSize;
        picbox.Location   = new Point(label1.Font.Height / 2,
                              label1.Font.Height / 2);
        label1.Location  = new Point(picbox.Right,label1.Font.Height / 2);
        int iClientWidth = label1.Right;
        Label label2     = new Label();
        label2.Parent    = this;
        label2.Text = "\x00A9 2010 by SYSOP SLY ";
        label2.Font      = new Font(FontFamily.GenericSerif, 10);
        label2.Location  = new Point(0, label1.Bottom +
                               label2.Font.Height);
        label2.Size      = new Size(iClientWidth, label2.Font.Height);
        label2.TextAlign = ContentAlignment.MiddleCenter;
        Button button   = new Button();
        button.Parent   = this;
        button.Text     = "OK";
        button.Size     = new Size(4 * button.Font.Height,
                            2 * button.Font.Height);
        button.Location = new Point((iClientWidth - button.Size.Width) / 2,
                            label2.Bottom + 2 * button.Font.Height);
        button.DialogResult = DialogResult.OK;
        CancelButton = button;
        AcceptButton = button;
        ClientSize = new Size(iClientWidth,
                          button.Bottom + 2 * button.Font.Height);
    }
}
```



Esempio: finestre di dialogo Carattere e Colore.

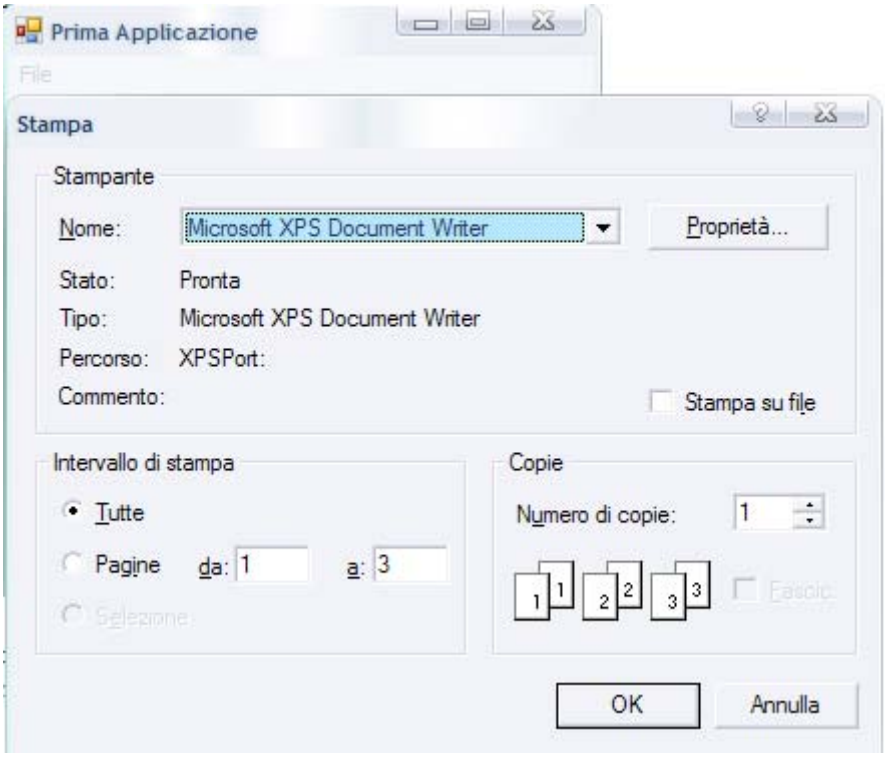
```
using System;
using System.Drawing;
using System.Windows.Forms;
class BetterFontAndColorDialogs:Form
{
    protected ColorDialog clrdlg = new ColorDialog();
    public static void Main()
    {
        Application.Run(new BetterFontAndColorDialogs());
    }
    public BetterFontAndColorDialogs()
    {
        Text = "Prima Applicazione";
        Menu = new MainMenu();
        Menu.MenuItems.Add("&Formato");
        Menu.MenuItems[0].MenuItems.Add("&Carattere...",
                                new EventHandler(MenuFontOnClick));
        Menu.MenuItems[0].MenuItems.Add("&Colore di Background...",
                                new EventHandler(MenuColorOnClick));
    }
    void MenuFontOnClick(object obj, EventArgs ea)
    {
        FontDialog fontdlg = new FontDialog();
        fontdlg.Font  = Font;
        fontdlg.Color = ForeColor;
        fontdlg.ShowColor = true;
        fontdlg.ShowApply = true;
        fontdlg.Apply += new EventHandler(FontDialogOnApply);
        if(fontdlg.ShowDialog() == DialogResult.OK)
        {
            Font = fontdlg.Font;
            ForeColor = fontdlg.Color;
            Invalidate();
        }
    }
    void MenuColorOnClick(object obj, EventArgs ea)
    {
        clrdlg.Color = BackColor;
        if (clrdlg.ShowDialog() == DialogResult.OK)
            BackColor = clrdlg.Color;
```

```
    }
     void FontDialogOnApply(object obj, EventArgs ea)
    {
        FontDialog fontdlg = (FontDialog) obj;
        Font = fontdlg.Font;
        ForeColor = fontdlg.Color;
        Invalidate();
    }
   protected override void OnPaint(PaintEventArgs pea)
    {
        Graphics grfx = pea.Graphics;
        grfx.DrawString("Modifica il tipo di carattere ed il colore background!", Font,
                    new SolidBrush(ForeColor), 0, 0);
    }
}
```

Esempio: finestra di dialogo Stampa.

```
using System;
using System.Drawing;
using System.Drawing.Printing;
using System.Windows.Forms;
class PrintDialogHelloWorld: Form
{
    const int iNumberPages = 3;
    int       iPagesToPrint, iPageNumber;
    public static void Main()
    {
        Application.Run(new PrintDialogHelloWorld());
    }
    public PrintDialogHelloWorld()
    {
        Text = "Prima Applicazione";
        Menu = new MainMenu();
        Menu.MenuItems.Add("&File");
        Menu.MenuItems[0].MenuItems.Add("S&tampa...",
                            new EventHandler(MenuFilePrintOnClick));
    }
    void MenuFilePrintOnClick(object obj, EventArgs ea)
    {
        // Crea il documento e la dialog box di stampa
        PrintDocument prndoc = new PrintDocument();
        PrintDialog   prndlg = new PrintDialog();
        prndlg.Document = prndoc;
        // Insieme delle pagine
        prndlg.AllowSomePages = true;
        prndlg.PrinterSettings.MinimumPage = 1;
        prndlg.PrinterSettings.MaximumPage = iNumberPages;
        prndlg.PrinterSettings.FromPage = 1;
        prndlg.PrinterSettings.ToPage = iNumberPages;
        // Se la dialog box returna OK, stampa
        if(prndlg.ShowDialog() == DialogResult.OK)
        {
            prndoc.DocumentName = Text;
            prndoc.PrintPage += new PrintPageEventHandler(OnPrintPage);
```

```
            // Determina quale pagina stampare
            switch (prndlg.PrinterSettings.PrintRange)
            {
            case PrintRange.AllPages:
                iPagesToPrint = iNumberPages;
                iPageNumber = 1;
                break;
            case PrintRange.SomePages:
                iPagesToPrint = 1 + prndlg.PrinterSettings.ToPage -
                                prndlg.PrinterSettings.FromPage;
                iPageNumber = prndlg.PrinterSettings.FromPage;
                break;
            }
            prndoc.Print();
        }
    }
    void OnPrintPage(object obj, PrintPageEventArgs ppea)
    {
        Graphics grfx  = ppea.Graphics;
        Font     font  = new Font("Times New Roman", 360);
        string   str   = iPageNumber.ToString();
        SizeF    sizef = grfx.MeasureString(str, font);
        grfx.DrawString(str, font, Brushes.Black,
                (grfx.VisibleClipBounds.Width - sizef.Width) / 2,
                (grfx.VisibleClipBounds.Height - sizef.Height) / 2);
        iPageNumber += 1;
        iPagesToPrint -= 1;
        ppea.HasMorePages = iPagesToPrint > 0;
    }
}
```

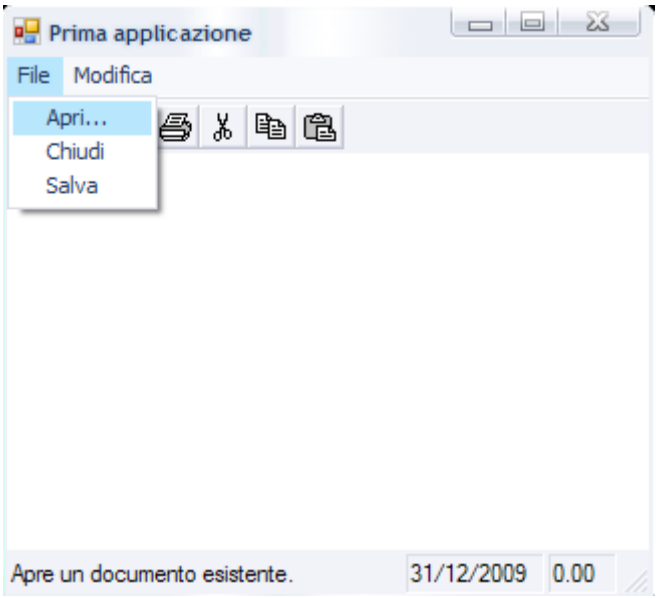


Esempio: barra dei menu, barra degli strumenti e barra di stato.

```
using System;
using System.Drawing;
using System.Windows.Forms;
class MenuHelpFirstTry: Form
{
   StatusBarPanel sbpMenuHelp,sbpDate, sbpTime;
    string        strSavePanelText;
    public static void Main()
    {
        Application.Run(new MenuHelpFirstTry());
    }
    public MenuHelpFirstTry()
    {
        Text = "Prima applicazione";
        // inizio barra degli strumenti
 Bitmap bm = new Bitmap("I:\\Esercizi\\Visual C#\\Toolbars and Status
Bars\\MenuHelpFirstTry\\StandardButtons.bmp");
        ImageList imglst = new ImageList();
        imglst.Images.AddStrip(bm);
        imglst.TransparentColor = Color.Cyan;
        ToolBar tbar = new ToolBar();
        tbar.Parent = this;
        tbar.ImageList = imglst;
        tbar.ShowToolTips = true;
        string[] astr = { "Nuovo", "Apri", "Salva", "Stampa", "taglia", "Copia", "Incolla" };
        for (int i = 0; i < 7; i++)
        {
            ToolBarButton tbarbtn = new ToolBarButton();
            tbarbtn.ImageIndex = i;
            tbarbtn.ToolTipText = astr[i];
            tbar.Buttons.Add(tbarbtn);
        }
        // fine barra degli strumenti
        BackColor = SystemColors.Window;
        ForeColor = SystemColors.WindowText;
        // crea la barra di stato con un solo pannello
        StatusBar sb = new StatusBar();
        sb.Parent = this;
        sb.ShowPanels = true;
        sbpMenuHelp = new StatusBarPanel();
        sbpMenuHelp.Text = "Barra di stato";
        sbpMenuHelp.BorderStyle = StatusBarPanelBorderStyle.None;
        sbpMenuHelp.AutoSize = StatusBarPanelAutoSize.Spring;
        // data e ora nella barra di stato
        sbpDate = new StatusBarPanel();
        sbpDate.AutoSize = StatusBarPanelAutoSize.Contents;
        sbpDate.ToolTipText = "Data Corrente: ";
        sbpTime = new StatusBarPanel();
        sbpTime.AutoSize = StatusBarPanelAutoSize.Contents;
        sbpTime.ToolTipText = "Ora corrente: ";
        sb.Panels.AddRange(new StatusBarPanel[] { sbpMenuHelp, sbpDate, sbpTime });
        Timer timer = new Timer();
        timer.Tick += new EventHandler(TimerOnTick);
```

```
        timer.Interval = 1000;
        timer.Start();
        // inizia la costruzione del menu
        Menu = new MainMenu();
        EventHandler ehSelect = new EventHandler(MenuOnSelect);
        // menu file
        MenuItem mi = new MenuItem("File");
        mi.Select += ehSelect;
        Menu.MenuItems.Add(mi);
        mi = new MenuItem("Apri...");
        mi.Select += ehSelect;
        Menu.MenuItems[0].MenuItems.Add(mi);
        mi = new MenuItem("Chiudi");
        mi.Select += ehSelect;
        Menu.MenuItems[0].MenuItems.Add(mi);
        mi = new MenuItem("Salva");
        mi.Select += ehSelect;
        Menu.MenuItems[0].MenuItems.Add(mi);
        // menu modifica
        mi = new MenuItem("Modifica");
        mi.Select += ehSelect;
        Menu.MenuItems.Add(mi);
        mi = new MenuItem("Taglia");
        mi.Select += ehSelect;
        Menu.MenuItems[1].MenuItems.Add(mi);
        mi = new MenuItem("Copia");
        mi.Select += ehSelect;
        Menu.MenuItems[1].MenuItems.Add(mi);
        mi = new MenuItem("Incolla");
        mi.Select += ehSelect;
        Menu.MenuItems[1].MenuItems.Add(mi);
    }
    protected override void OnMenuStart(EventArgs ea)
    { strSavePanelText = sbpMenuHelp.Text;}
    protected override void OnMenuComplete(EventArgs ea)
    { sbpMenuHelp.Text = strSavePanelText;}
    void MenuOnSelect(object obj, EventArgs ea)
    {
        MenuItem mi = (MenuItem) obj;
        string   str;
        switch (mi.Text)
        {
        case "File":   str = "Comandi per lavorare con il documento.";break;
        case "Apri...":   str = "Apre un documento esistente.";break;
        case "Chiudi":  str = "Chiude il documento attivo.";break;
        case "Salva":   str = "Salva il documento attivo.";break;
        case "Modifica":   str = "Comandi per modificare il documento";  break;
        case "Taglia": str = "Taglia la selezione e la inserisce negli Appunti.";break;
        case "Copia": str = "Copia la selezione e la inserisce negli Appunti."; break;
        case "Incolla":  str = "Inserisce il contenuto degli Appunti." ;break;
        default: str = "";break;
        }
        sbpMenuHelp.Text = str;
    }
```

```
    private void InitializeComponent()
    {
        this.SuspendLayout();
        // MenuHelpFirstTry
        this.ClientSize = new System.Drawing.Size(292, 268);
        this.Name = "MenuHelpFirstTry";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
        this.ResumeLayout(false);
        // Data e Ora
        this.ClientSize = new System.Drawing.Size(292, 268);
        this.Name = "DateAndTimeStatus";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
        this.ResumeLayout(false);
    }
    void TimerOnTick(object obj, EventArgs ea)
    {
        DateTime dt = DateTime.Now;
        sbpDate.Text = dt.ToShortDateString();
        sbpTime.Text = dt.ToShortTimeString();
    }
}
```



MOUSE

```
using System;
using System.Drawing;
using System.Windows.Forms;
class BlockOut: Form
{
    bool      bBlocking, bValidBox;
    Point     ptBeg, ptEnd;
    Rectangle rectBox;
    public static void Main()
    {
        Application.Run(new BlockOut());
    }
    public BlockOut()
```

```
{
    Text = "Rettangoli con il mouse";
    BackColor = SystemColors.Window;
    ForeColor = SystemColors.WindowText;
}
protected override void OnMouseDown(MouseEventArgs mea)
{
    if (mea.Button == MouseButtons.Left)
    {
        ptBeg = ptEnd = new Point(mea.X, mea.Y);
        Graphics grfx = CreateGraphics();
        grfx.DrawRectangle(new Pen(ForeColor), Rect(ptBeg, ptEnd));
        grfx.Dispose();
        bBlocking = true;
    }
}
protected override void OnMouseMove(MouseEventArgs mea)
{
    if (bBlocking)
    {
        Graphics grfx = CreateGraphics();
        grfx.DrawRectangle(new Pen(BackColor), Rect(ptBeg, ptEnd));
        ptEnd = new Point(mea.X, mea.Y);
        grfx.DrawRectangle(new Pen(ForeColor), Rect(ptBeg, ptEnd));
        grfx.Dispose();
        Invalidate();
    }
}
protected override void OnMouseUp(MouseEventArgs mea)
{
    if (bBlocking && mea.Button == MouseButtons.Left)
    {
        Graphics grfx = CreateGraphics();
        rectBox = Rect(ptBeg, new Point(mea.X, mea.Y));
        grfx.DrawRectangle(new Pen(ForeColor), rectBox);
        grfx.Dispose();
        bBlocking = false;
        bValidBox = true;
        Invalidate();
    }
}
protected override void OnPaint(PaintEventArgs pea)
{
    Graphics grfx = pea.Graphics;
    if (bValidBox)
        grfx.FillRectangle(new SolidBrush(ForeColor), rectBox);
    if (bBlocking)
        grfx.DrawRectangle(new Pen(ForeColor), Rect(ptBeg, ptEnd));
}
Rectangle Rect(Point ptBeg, Point ptEnd)
{
    return new Rectangle(Math.Min(ptBeg.X, ptEnd.X),
                         Math.Min(ptBeg.Y, ptEnd.Y),
                         Math.Abs(ptEnd.X - ptBeg.X),
```

```
                    Math.Abs(ptEnd.Y - ptBeg.Y));
    }
}
```

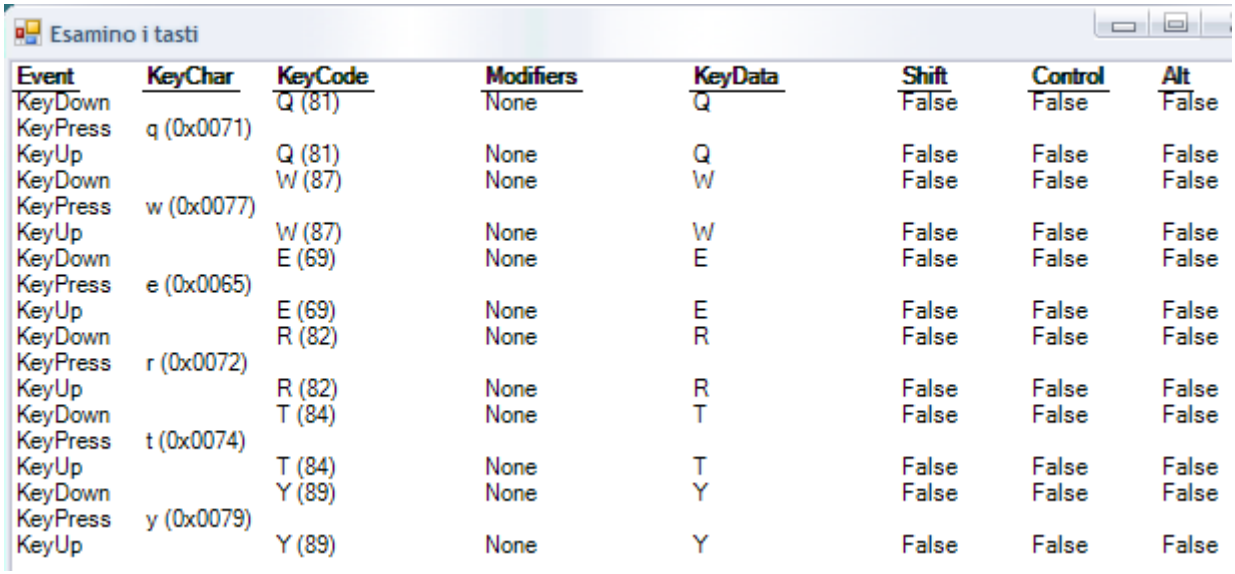
TASTIERA

```
using System;
using System.Drawing;
using System.Windows.Forms;
public class KeyExamine: Form
{
    public static void Main()
    {
        Application.Run(new KeyExamine());
    }
    enum EventType
    {
        None,
        KeyDown,
        KeyUp,
        KeyPress
    }
    struct KeyEvent
    {
        public EventType evttype;
        public EventArgs evtargs;
    }
    const int  iNumLines    = 25;
    int        iNumValid    = 0;
    int        iInsertIndex = 0;
    KeyEvent[] akeyevt = new KeyEvent[iNumLines];
    int xEvent, xChar, xCode, xMods, xData,
       xShift, xCtrl, xAlt, xRight;
    public KeyExamine()
    {
        Text = "Esamino i tasti";
        BackColor = SystemColors.Window;
        ForeColor = SystemColors.WindowText;
        xEvent = 0;
        xChar  = xEvent + 5 * Font.Height;
        xCode  = xChar  + 5 * Font.Height;
        xMods  = xCode  + 8 * Font.Height;
        xData  = xMods  + 8 * Font.Height;
```

```
        xShift = xData  + 8 * Font.Height;
        xCtrl  = xShift + 5 * Font.Height;
        xAlt   = xCtrl  + 5 * Font.Height;
        xRight = xAlt   + 5 * Font.Height;
        ClientSize  = new Size(xRight, Font.Height * (iNumLines + 1));
        FormBorderStyle = FormBorderStyle.Fixed3D;
        MaximizeBox = false;
    }
    protected override void OnKeyDown(KeyEventArgs kea)
    {
        akeyevt[iInsertIndex].evttype = EventType.KeyDown;
        akeyevt[iInsertIndex].evtargs = kea;
        OnKey();
    }
    protected override void OnKeyUp(KeyEventArgs kea)
    {
        akeyevt[iInsertIndex].evttype = EventType.KeyUp;
        akeyevt[iInsertIndex].evtargs = kea;
        OnKey();
    }
    protected override void OnKeyPress(KeyPressEventArgs kpea)
    {
        akeyevt[iInsertIndex].evttype = EventType.KeyPress;
        akeyevt[iInsertIndex].evtargs = kpea;
        OnKey();
    }
    void OnKey()
    {
        if(iNumValid < iNumLines)
        {
            Graphics grfx = CreateGraphics();
            DisplayKeyInfo(grfx, iInsertIndex, iInsertIndex);
            grfx.Dispose();
        }
        else
        {
            ScrollLines();
        }
        iInsertIndex = (iInsertIndex + 1) % iNumLines;
        iNumValid = Math.Min(iNumValid + 1, iNumLines);
    }
    protected virtual void ScrollLines()
    {
        Rectangle rect = new Rectangle(0, Font.Height,
                                       ClientSize.Width,
                                       ClientSize.Height - Font.Height);
        Invalidate(rect);
    }
    protected override void OnPaint(PaintEventArgs pea)
    {
        Graphics grfx = pea.Graphics;
        BoldUnderline(grfx, "Event",     xEvent, 0);
        BoldUnderline(grfx, "KeyChar",   xChar,  0);
        BoldUnderline(grfx, "KeyCode",   xCode,  0);
```

```
        BoldUnderline(grfx, "Modifiers", xMods,  0);
        BoldUnderline(grfx, "KeyData",   xData,  0);
        BoldUnderline(grfx, "Shift",     xShift, 0);
        BoldUnderline(grfx, "Control",   xCtrl,  0);
        BoldUnderline(grfx, "Alt",       xAlt,   0);
        if(iNumValid < iNumLines)
        {
            for (int i = 0; i < iNumValid; i++)
                DisplayKeyInfo(grfx, i, i);
        }
        else
        {
            for (int i = 0; i < iNumLines; i++)
                DisplayKeyInfo(grfx, i,(iInsertIndex + i) % iNumLines);
        }
    }
    void BoldUnderline(Graphics grfx, string str, int x, int y)
    {
        Brush brush = new SolidBrush(ForeColor);
        grfx.DrawString(str, Font, brush, x, y);
        grfx.DrawString(str, Font, brush, x + 1, y);

        // Underline
        SizeF sizef = grfx.MeasureString(str, Font);
        grfx.DrawLine(new Pen(ForeColor), x, y + sizef.Height,
                                x + sizef.Width, y + sizef.Height);
    }
    void DisplayKeyInfo(Graphics grfx, int y, int i)
    {
        Brush br = new SolidBrush(ForeColor);
        y = (1 + y) * Font.Height;    // Convert y in coordinate pixel
        grfx.DrawString(akeyevt[i].evttype.ToString(),
                    Font, br, xEvent, y);
        if(akeyevt[i].evttype == EventType.KeyPress)
        {
            KeyPressEventArgs kpea =
                            (KeyPressEventArgs) akeyevt[i].evtargs;
            string str = String.Format("\x202D{0} (0x{1:X4})",
                                kpea.KeyChar, (int) kpea.KeyChar);
            grfx.DrawString(str, Font, br, xChar, y);
        }
        else
        {
            KeyEventArgs kea = (KeyEventArgs) akeyevt[i].evtargs;
            string str = String.Format("{0} ({1})",
                                kea.KeyCode, (int) kea.KeyCode);
            grfx.DrawString(str, Font, br, xCode, y);
            grfx.DrawString(kea.Modifiers.ToString(), Font, br, xMods, y);
            grfx.DrawString(kea.KeyData.ToString(), Font, br, xData, y);
            grfx.DrawString(kea.Shift.ToString(), Font, br, xShift, y);
            grfx.DrawString(kea.Control.ToString(), Font, br, xCtrl,  y);
            grfx.DrawString(kea.Alt.ToString(), Font, br, xAlt,   y);
        }
    }
```

}


Esamino i tasti
Event KeyChar KeyCode Modifiers KeyData Shift Control Alt
KeyDown Q (81) None Q False False False
KeyPress q (0x0071)
KeyUp Q (81) None Q False False False
KeyDown W (87) None W False False False
KeyPress w (0x0077)
KeyUp W (87) None W False False False
KeyDown E (69) None E False False False
KeyPress e (0x0065)
KeyUp E (69) None E False False False
KeyDown R (82) None R False False False
KeyPress r (0x0072)
KeyUp R (82) None R False False False
KeyDown T (84) None T False False False
KeyPress t (0x0074)
KeyUp T (84) None T False False False
KeyDown Y (89) None Y False False False
KeyPress y (0x0079)
KeyUp Y (89) None Y False False False

MODULO 3

F#

Programmazione funzionale
F# Interactive
Compilazione CLI
MDE

PROGRAMMAZIONE FUNZIONALE

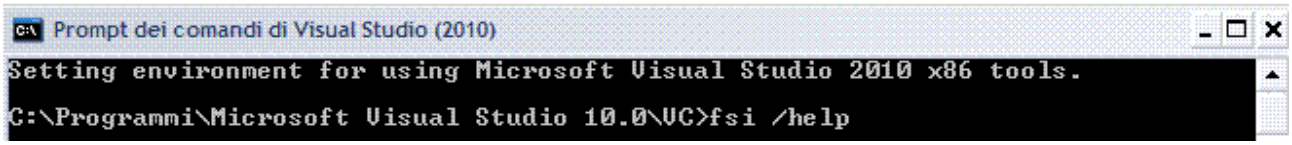
INTRODUZIONE

È un linguaggio che fornisce il supporto sia alla programmazione funzionale sia alla programmazione OO e imperativa.
Il codice scritto con F# è compilato in IL così come avviene per ogni altro linguaggio supportato dall'ambiente .NET, pertanto non vi è alcuna differenza con un eseguibile scritto in C# o VB.NET.
Il linguaggio F# è fortemente tipizzato e usa la *type inference,* inferenza dei tipi.
Ciò vuol dire che lo sviluppatore non è tenuto a definire i tipi degli oggetti, ma questi sono automaticamente dedotti dal compilatore.
F# è un linguaggio lazy, il che indica che non è eseguita alcuna computazione fin quando non è espressamente richiesta.
È possibile definire una serie numerica infinita senza bloccare l'esecuzione, dato che ogni elemento sarà effettivamente prodotto solo quando dovrà essere usato all'interno di un'istruzione dell'applicazione.
La ricorsione è il concetto alla base del paradigma di programmazione funzionale.

F# INTERACTIVE

INTRODUZIONE

È una finestra all'interno della quale è possibile digitare codice.



Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>fsi /help
Microsoft (R) F# 2.0 Interactive - build 4.0.30319.1
Copyright (c) Microsoft Corporation. Tutti i diritti riservati.
Sintassi: fsi.exe <opzioni> [script.fsx [<argomenti>]]
- FILE DI INPUT -
--use:<file> Usa il file specificato all'avvio come input iniziale
--load:<file> #load file specificato all'avvio
--reference:<file> Fa riferimento a un assembly (forma breve: -r)
-- ... Tratta gli argomenti rimanenti come argomenti della riga di comando accessibili mediante fsi.CommandLineArgs
- GENERAZIONE CODICE -
--debug[+|-] Crea informazioni di debug (forma breve: -g)
--debug:{full|pdbonly} Specificare il tipo di debug: full, pdbonly ('full' è l'opzione predefinita e consente di collegare un debugger a un programma in esecuzione).
--optimize[+|-] Abilita le ottimizzazioni (forma breve: -O)
--tailcalls[+|-] Abilita o disabilita le chiamate tail
--crossoptimize[+|-] Abilita o disabilita le ottimizzazioni tra i moduli
- ERRORI E AVVISI -
--warnaserror[+|-] Segnala tutti gli avvisi come errori
--warnaserror[+|-]:<warn;...> Segnala determinati avvisi come errori
--warn:<n> Imposta un livello di avviso (0-4)
--nowarn:<warn;...> Disabilita messaggi di avviso specifici
- LINGUAGGIO -
--checked[+|-] Genera controlli dell'overflow
--define:<string> Definisce simboli di compilazione condizionale (forma breve: -d)
--mlcompatibility Ignora avvisi di compatibilità ML
- VARIE -
--nologo Non visualizza il messaggio di copyright del compilatore
--help Visualizza questo messaggio relativo all'uso (forma breve: -?)
- AVANZATE -
--codepage:<n> Specificare la tabella codici utilizzata per leggere i file di origine
--utf8output Genera messaggi con codifica UTF-8
--fullpaths Messaggi di output con percorsi completi
--lib:<dir;...> Specifica una directory per il percorso di inclusione utilizzato per risolvere assembly e file di origine (forma breve: -I)
--noframework Per impostazione predefinita, non fa riferimento agli assembly CLI predefiniti
--exec Uscita da fsi dopo il caricamento dei file o l'esecuzione dello script .fsx specificato nella riga di comando
--gui[+|-] Esegue interazioni in un ciclo di eventi di

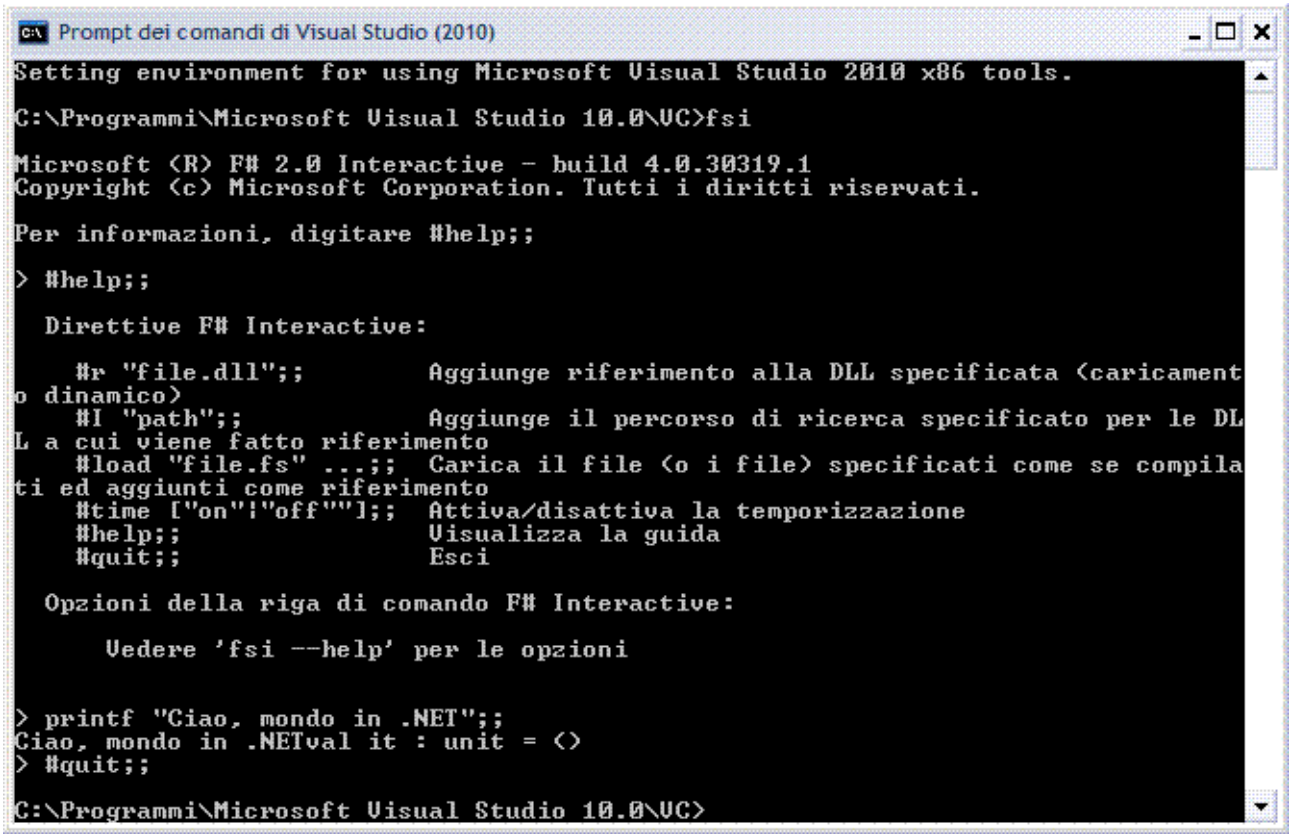
Windows Forms (abilitata per impostazione predefinita)

--quiet *Impedisci scrittura fsi su stdout*

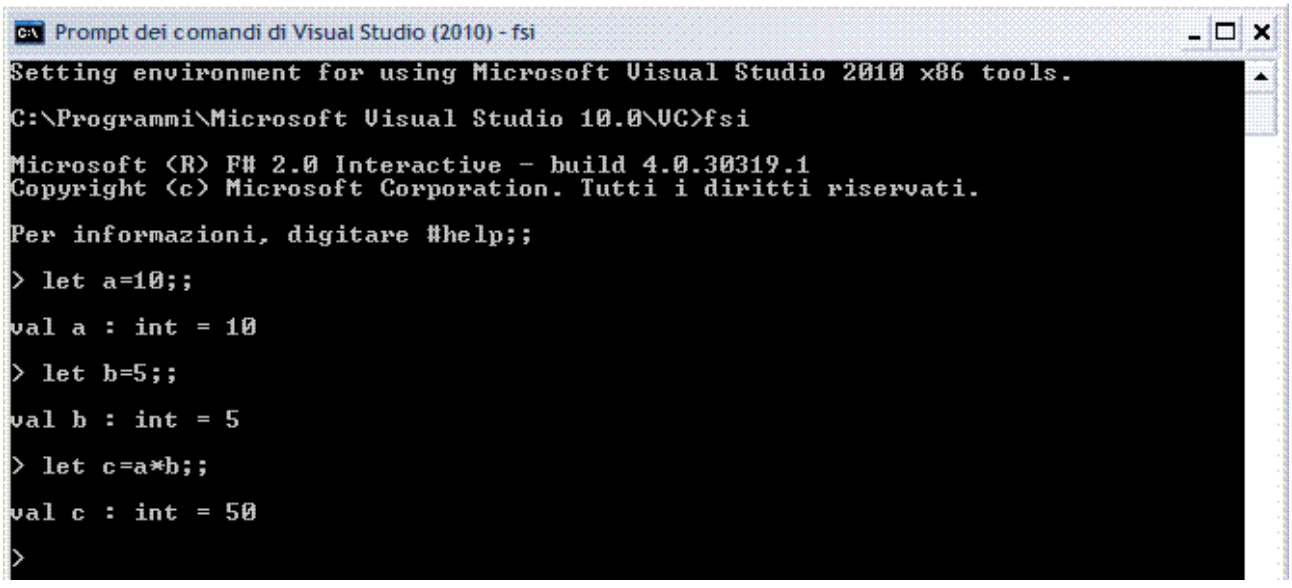
--readline[+|-] *Supporta completamento con tasto TAB in console (abilitata per impostazione predefinita)*

APPLICAZIONE CONSOLE INTERACTIVE

Terminare ogni istruzione con ";;".

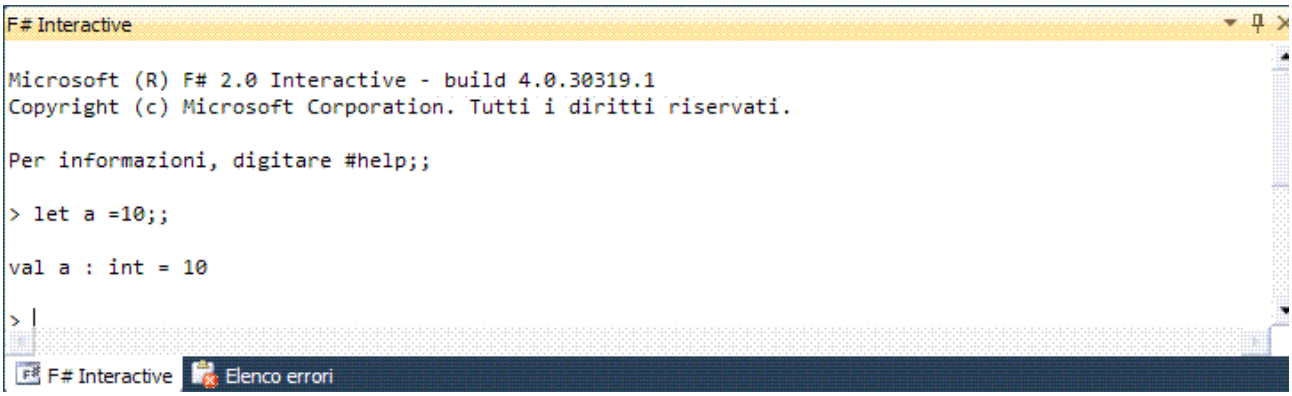


Esempio, moltiplicare due numeri interi.
La prima istruzione F# è quella che permette di legare un valore ad un identificatore.
Per fare questo si usa la parola chiave *let*.



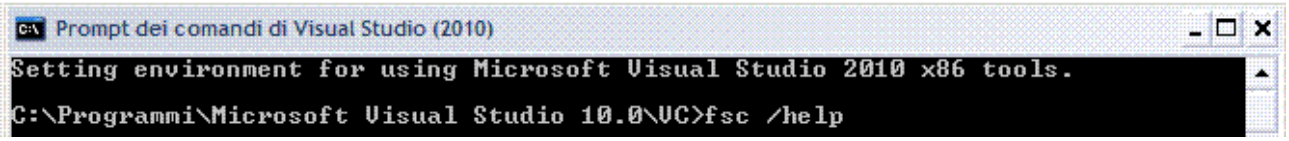
APPLICAZIONE MDE INTERACTIVE

Fare clic sul menu ***Visualizza/Altre finestre/F# Interactive (CTRL+ALT+F).***



COMPILAZIONE CLI (*COMMAND LINE INTERFACE*)

INTRODUZIONE



Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>fsc /help
Compilatore Microsoft (R) F# 2.0 - build 4.0.30319.1
Copyright (c) Microsoft Corporation. Tutti i diritti riservati.

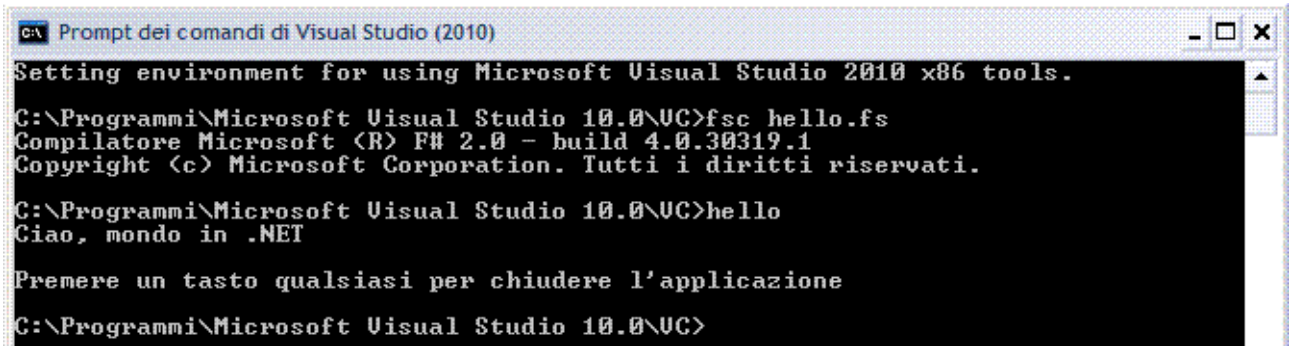
- FILE DI OUTPUT -
--out:<file> Nome del file di output (forma breve: -o)
--target:exe Compila un file eseguibile da console
--target:winexe Compila un file eseguibile Windows
--target:library Compila una libreria (forma breve: -a)
--target:module Compila un modulo che può essere aggiunto ad altro assembly
--delaysign[+|-] Ritarda la firma dell'assembly utilizzando solo la parte pubblica della chiave con nome sicuro
--doc:<file> Scrive lo xmldoc dell'assembly nel file specificato
--keyfile:<file> Specifica un file di chiave con nome sicuro
--keycontainer:<string> Specifica un contenitore di chiavi con nome sicuro
--platform:<string> Limita le piattaforme in cui è possibile eseguire il codice: x86, Itanium, x64 o anycpu. Il valore predefinito è anycpu.
--nooptimizationdata Include solo informazioni di ottimizzazione essenziali per l'implementazione dei costrutti inline. Impedisce l'incorporamento tra moduli ma migliora la compatibilità binaria.
--nointerfacedata Non aggiunge una risorsa all'assembly generato contenente metadati specifici di F#
--sig:<file> Stampa l'interfaccia dedotta dell'assembly in un file
- FILE DI INPUT -
--reference:<file> Fa riferimento a un assembly (forma breve: -r)
- RISORSE -
--win32res:<file> Specifica un file di risorse Win32 (.res)
--win32manifest:<file> Specifica un file manifesto Win32
--nowin32manifest Non include il manifesto Win32 predefinito
--resource:<resinfo> Incorpora la risorsa gestita specificata
--linkresource:<resinfo> Collega la risorsa specificata all'assembly in cui il formato di resinfo è <file>[,<nome stringa>[,public|private]]
- GENERAZIONE CODICE -
--debug[+|-] Crea informazioni di debug (forma breve: -g)
--debug:{full|pdbonly} Specificare il tipo di debug: full, pdbonly ('full' è l'opzione predefinita e consente di collegare un debugger a un programma in esecuzione).
--optimize[+|-] Abilita le ottimizzazioni (forma breve: -O)
--tailcalls[+|-] Abilita o disabilita le chiamate tail
--crossoptimize[+|-] Abilita o disabilita le ottimizzazioni tra i moduli

- ERRORI E AVVISI -
--warnaserror[+|-] Segnala tutti gli avvisi come errori
--warnaserror[+|-]:<warn;...> Segnala determinati avvisi come errori
--warn:<n> Imposta un livello di avviso (0-4)
--nowarn:<warn;...> Disabilita messaggi di avviso specifici
- LINGUAGGIO -
--checked[+|-] Genera controlli dell'overflow
--define:<string> Definisce simboli di compilazione condizionale (forma breve: -d)
--mlcompatibility Ignora avvisi di compatibilità ML
- VARIE -
--nologo Non visualizza il messaggio di copyright del compilatore
--help Visualizza questo messaggio relativo all'uso (forma breve: -?)
- AVANZATE -
--codepage:<n> Specificare la tabella codici utilizzata per leggere i file di origine
--utf8output Genera messaggi con codifica UTF-8
--fullpaths Messaggi di output con percorsi completi
--lib:<dir;...> Specifica una directory per il percorso di inclusione utilizzato per risolvere assembly e file di origine (forma breve: -I)
--baseaddress:<address> Indirizzo di base della libreria da compilare
--noframework Per impostazione predefinita, non fa riferimento agli assembly CLI predefiniti
--standalone Collega in modo statico la libreria F# e tutte le DLL da questa dipendenti a cui viene fatto riferimento nell'assembly in fase di generazione
--staticlink:<file> Collega in modo statico l'assembly specificato e tutte le DLL da questo dipendenti a cui viene fatto riferimento. Utilizzare un nome di assembly, ad esempio lib, non un nome di DLL.
--pdb:<string> Nome del file di debug di output
--simpleresolution Risolve i riferimenti ad assembly mediante regole Mono basate su directory invece che con la risoluzione MSBuild. Impostazione predefinita: false (ad eccezione dei casi in cui fsc.exe viene eseguito su Mono)

APPLICAZIONE CONSOLE

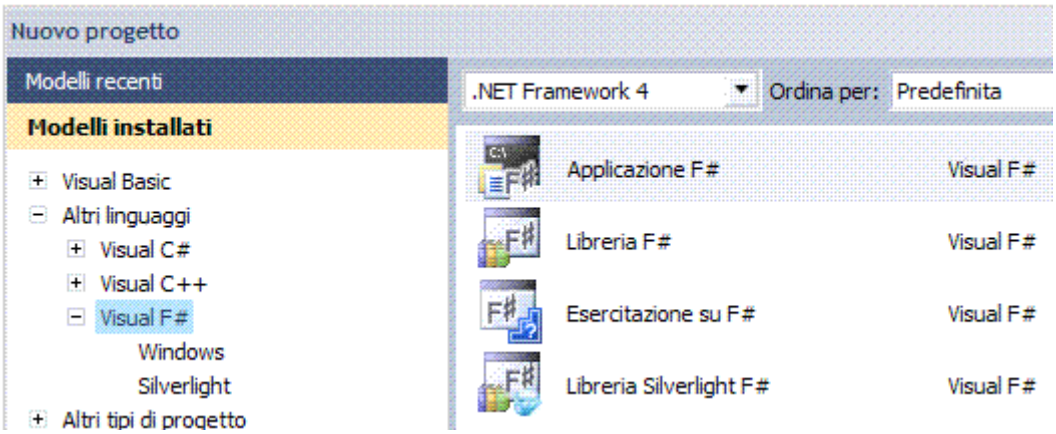
Sono semplici da costruire, non hanno grafica e hanno interfaccia **CUI** (*Character User Interface*) che permette all'utente d'interagire con la tastiera e una finestra.

```
// Nome dell'applicazione: hello.fs
// Programmatore:
// Descrizione:
#light
open System
Console.WriteLine("Ciao, mondo in .NET")
Console.WriteLine()
Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione")
let a= Console.ReadKey()
```

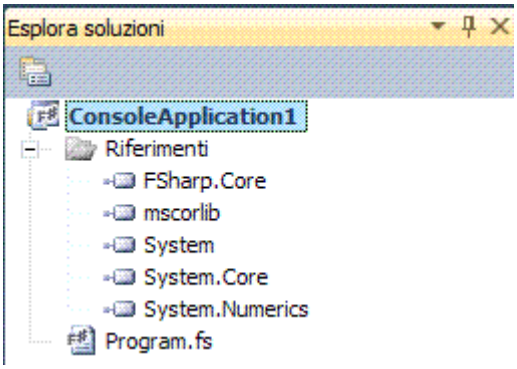

Prompt dei comandi di Visual Studio (2010)
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
C:\Programmi\Microsoft Visual Studio 10.0\VC>fsc hello.fs
Compilatore Microsoft (R) F# 2.0 - build 4.0.30319.1
Copyright (c) Microsoft Corporation. Tutti i diritti riservati.
C:\Programmi\Microsoft Visual Studio 10.0\VC>hello
Ciao, mondo in .NET
Premere un tasto qualsiasi per chiudere l'applicazione
C:\Programmi\Microsoft Visual Studio 10.0\VC>

INTRODUZIONE

Per creare un nuovo progetto, fare clic su ***File/Nuovo Progetto… (CTRL+N)***.
Sarà visualizzata la finestra di dialogo ***Nuovo Progetto***.



Nel nuovo progetto F# inserisce automaticamente i seguenti file.




```
// Nome dell'applicazione: hello.fs
// Programmatore:
// Descrizione:
#light
open System
Console.Clear()
Console.WriteLine("Ciao, mondo in .NET")
Console.WriteLine()
Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione")
let a= Console.ReadKey()
```

Esercitazione su F#

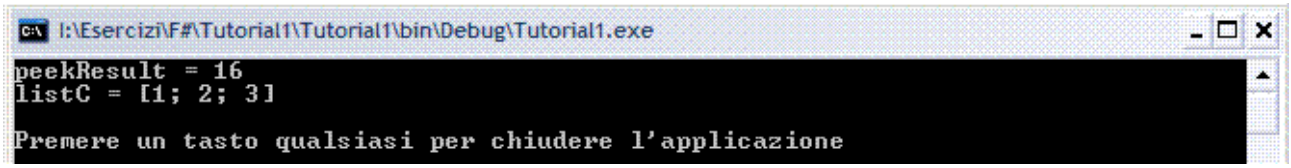
File TUTORIAL.FS

```
// File di esercitazione su F#
// Questo file contiene codice di esempio che illustra primitive del linguaggio F#.
//   - Calcoli semplici
//   - Funzioni su Integer
//   - Tuple
//   - Valori booleani
//   - Stringhe
//   - Elenchi
//   - Matrici
//   - Altre raccolte
//   - Funzioni
//   - Tipi: unioni
//   - Tipi: record
//   - Tipi: classi
//   - Tipi: interfacce
//   - Tipi: classi con implementazioni di interfaccia
//   - Stampa
// aprire alcuni spazi dei nomi standard
#light
open System
Console.Clear()
// Calcoli semplici
// ---------------------------------------------------------------
// Di seguito sono disponibili alcuni calcoli semplici. Si noti in che modo è possibile
// documentare il codice con commenti
//. Passare il puntatore del mouse sui riferimenti a una variabile per
// visualizzare la relativa documentazione.
/// Un Integer costante estremamente semplice
let int1 = 1
/// Un secondo Integer costante semplice
let int2 = 2
/// Aggiungere due Integer
let int3 = int1 + int2
// Funzioni su Integer
// ---------------------------------------------------------------
/// Funzione su Integer
let f x = 2*x*x - 5*x + 3
/// Risultato di un calcolo semplice
let result = f (int3 + 4)
/// Altra funzione su Integer
let increment x = x + 1
/// Calcolare il fattoriale di un Integer
let rec factorial n = if n=0 then 1 else n * factorial (n-1)
/// Calcolare il massimo comun divisore di due Integer
let rec hcf a b =                    // avviso: 2 parametri separati da spazi
    if a=0 then b
    elif a<b then hcf a (b-a)          // avviso: 2 argomenti separati da spazi
    else hcf (a-b) b
    // nota: gli argomenti di una funzione sono in genere separati da spazi
    // nota: 'let rec' definisce una funzione ricorsiva
// Tuple
```

```
// ----------------------------------------------------------
// Tupla di Integer semplice
let pointA = (1, 2, 3)
// Semplice tupla di un Integer, una stringa e un numero a virgola mobile con due posizioni
decimali
let dataB = (1, "fred", 3.1415)
/// Funzione che scambia l'ordine di due valori in una tupla
let Swap (a, b) = (b, a)
// Valori booleani
// ----------------------------------------------------------
/// Valore booleano semplice
let boolean1 = true
/// Altro valore booleano semplice
let boolean2 = false
/// Calcolare un nuovo valore booleano mediante operatori AND, OR e NOT
let boolean3 = not boolean1 && (boolean2 || false)
// Stringhe
// ----------------------------------------------------------
/// Stringa semplice
let stringA  = "Hello"
/// Altra stringa semplice
let stringB  = "world"
/// "Hello world" calcolato mediante il concatenamento di stringhe
let stringC  = stringA + " " + stringB
/// "Hello world" calcolato mediante una funzione di libreria .NET
let stringD = String.Join(" ",[| stringA; stringB |])
  // Provare a digitare di nuovo la riga indicata sopra per vedere IntelliSense in azione
  // Si noti che è possibile riattivare premendo CTRLl-J sugli identificatori (parziali)
// Elenchi funzionali
// ----------------------------------------------------------
/// Elenco vuoto
let listA = [ ]
/// Elenco con 3 Integer
let listB = [ 1; 2; 3 ]
/// Elenco con 3 Integer. Si noti che :: è l'operazione 'cons'
let listC = 1 :: [2; 3]
/// Calcolare la somma di un elenco di Integer mediante una funzione ricorsiva
let rec SumList xs =
    match xs with
    | []    -> 0
    | y::ys -> y + SumList ys
/// Somma di un elenco
let listD = SumList [1; 2; 3]
/// Elenco di Integer compresi tra 1 e 10 (estremi inclusi)
let oneToTen = [1..10]
/// Quadrati dei primi 10 Integer
let squaresOfOneToTen = [ for x in 0..10 -> x*x ]
// Matrici modificabili
// ----------------------------------------------------------
/// Creare una matrice
let arr = Array.create 4 "hello"
arr.[1] <- "world"
arr.[3] <- "don"
/// Calcolare la lunghezza della matrice mediante un metodo di istanza sull'oggetto matrice
```

```
let arrLength = arr.Length
// Estrarre una sottomatrice mediante una notazione di sezionamento
let front = arr.[0..2]
// Altre raccolte
// ---------------------------------------------------------
/// Dizionario con chiavi Integer e valori stringa
let lookupTable = dict [ (1, "One"); (2, "Two") ]
let oneString = lookupTable.[1]
// Per alcune altre strutture di dati comuni, vedere:
//   System.Collections.Generic
//   Microsoft.FSharp.Collections
//   Microsoft.FSharp.Collections.Seq
//   Microsoft.FSharp.Collections.Set
//   Microsoft.FSharp.Collections.Map
// Funzioni
// ---------------------------------------------------------
/// Funzione che esegue il quadrato del relativo input
let Square x = x*x
// Mappare una funzione in un elenco di valori
let squares1 = List.map Square [1; 2; 3; 4]
let squares2 = List.map (fun x -> x*x) [1; 2; 3; 4]
// Pipeline
let squares3 = [1; 2; 3; 4] |> List.map (fun x -> x*x)
let SumOfSquaresUpTo n =
  [1..n]
  |> List.map Square
  |> List.sum
// Tipi: unioni
// ---------------------------------------------------------
type Expr =
  | Num of int
  | Add of Expr * Expr
  | Mul of Expr * Expr
  | Var of string

let rec Evaluate (env:Map<string,int>) exp =
    match exp with
    | Num n -> n
    | Add (x,y) -> Evaluate env x + Evaluate env y
    | Mul (x,y) -> Evaluate env x * Evaluate env y
    | Var id    -> env.[id]

let envA = Map.ofList [ "a",1 ;
                        "b",2 ;
                        "c",3 ]

let expT1 = Add(Var "a",Mul(Num 2,Var "b"))
let resT1 = Evaluate envA expT1
// Tipi: record
// ---------------------------------------------------------
type Card = { Name  : string;
              Phone : string;
              Ok    : bool }
let cardA = { Name = "Alf" ; Phone = "(206) 555-0157" ; Ok = false }
```

```
let cardB = { cardA with Phone = "(206) 555-0112"; Ok = true }
let ShowCard c =
  c.Name + " Phone: " + c.Phone + (if not c.Ok then " (unchecked)" else "")
// Tipi: classi
// ------------------------------------------------------------
/// Vettore bidimensionale
type Vector2D(dx:float, dy:float) =
    // Lunghezza precalcolata del vettore
    let length = sqrt(dx*dx + dy*dy)
    /// Spostamento lungo l'asse X
    member v.DX = dx
    /// Spostamento lungo l'asse Y
    member v.DY = dy
    /// Lunghezza del vettore
    member v.Length = length
    // Scalare di nuovo il vettore in base a una costante
    member v.Scale(k) = Vector2D(k*dx, k*dy)
// Tipi: interfacce
// ------------------------------------------------------------
type IPeekPoke =
    abstract Peek: unit -> int
    abstract Poke: int -> unit
// Tipi: classi con implementazioni di interfaccia
// ------------------------------------------------------------
/// Widget che conta il numero di volte in cui ne viene eseguito il poke
type Widget(initialState:int) =
    /// Stato interno del widget
    let mutable state = initialState
    // Implementare l'interfaccia IPeekPoke
    interface IPeekPoke with
        member x.Poke(n) = state <- state + n
        member x.Peek() = state

    /// È stato eseguito il poke del widget?
    member x.HasBeenPoked = (state <> 0)
let widget = Widget(12) :> IPeekPoke
widget.Poke(4)
let peekResult = widget.Peek()
// Stampa
// ------------------------------------------------------------
// Stampare un Integer
printfn "peekResult = %d" peekResult
// Stampare un risultato mediante %A per la stampa generica
printfn "listC = %A" listC
Console.WriteLine()
Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione")
let a= Console.ReadKey()
```



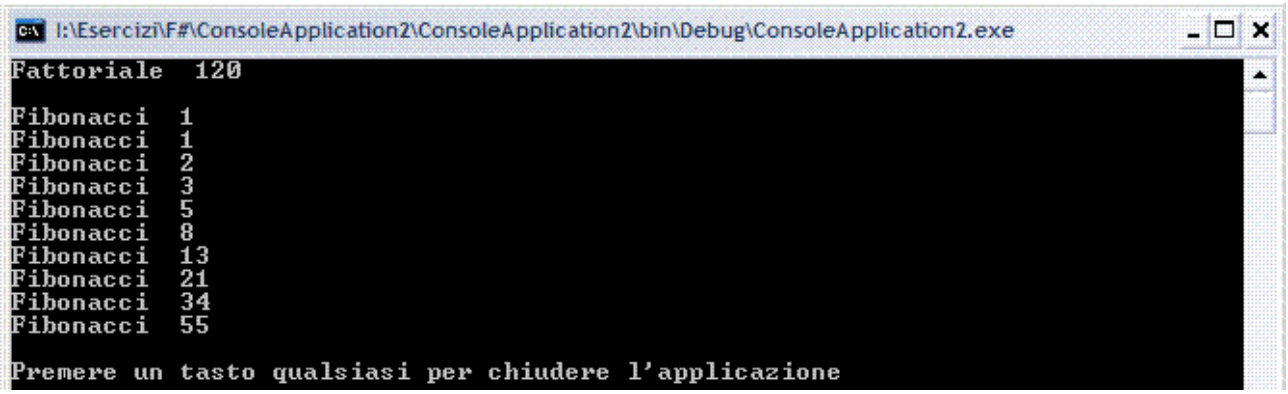
RICORSIONE

Calcolare il fattoriale di un numero e la sequenza di Fibonacci.

```
let rec fattoriale n =
    match n with
    | 0 -> 1
    | _ -> n * fattoriale (n-1)
```

La parola chiave *rec* aggiunge alla definizione di funzione la chiamata ricorsiva. L'espressione che segue è nota come **pattern matching** e si tratta di un costrutto potente e flessibile, dato che permette di confrontare non solo valori costanti, ma in generale può accettare funzioni anche complesse.
La sua sintassi prevede la parola chiave *match* seguita dall'identificatore da confrontare e l'altra parola chiave *with*.
Le regole di confronto, una per riga sono precedute dal simbolo "|".
Dopo la freccia "->" va scritto il valore da restituire in caso di *match*.
Nel calcolo del fattoriale il caso base sarà costituito dal valore zero ed in questo caso, come da definizione, si restituisce uno.
In tutti gli altri casi, il carattere "_" indica proprio qualunque valore, il valore da restituire sarà uguale al prodotto di *n*, il valore corrente, per il fattoriale di *(n-1).*

```
#light
open System
Console.Clear()
// fattoriale
let rec fattoriale n =
    match n with
    | 0 -> 1
    | _ -> n * fattoriale (n-1)
// Fibonacci *)
let rec fib n =
    match n with
    | 0 | 1 -> n
    | _ -> fib (n - 1) + fib (n - 2)
Console.WriteLine("Fattoriale  {0}",fattoriale (5))
Console.WriteLine()
for i in 1..10 do Console.WriteLine("Fibonacci  {0}",fib (i))
Console.WriteLine()
Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione")
let a= Console.ReadKey()
```



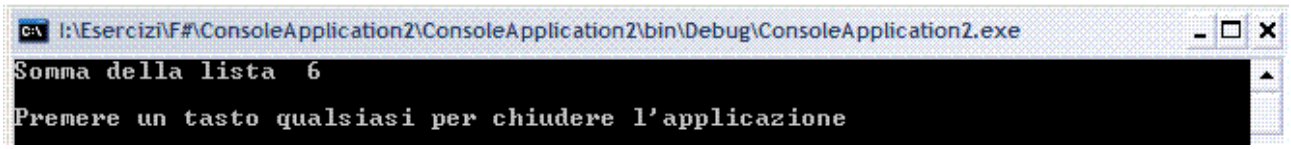
LISTE

La lista è una struttura dati che può contenere un numero arbitrario di elementi, anche zero, la lista vuota, rappresentata come *[]*.
È possibile aggiungere un elemento ad una lista usando l'operatore di concatenazione "::". Dato che la lista è una struttura dati immutabile, in questo caso sarà creata una nuova avente il nuovo elemento in testa.
Gli elementi di una lista devono essere racchiusi fra parentesi quadre e separati con il ";" ad esempio, la lista che contiene i numeri da 1 a 5 è scritta [1; 2; 3; 4; 5].
È possibile crearne una specificando il range dei valori ammissibili, ad esempio la lista definita come [1 .. 10] conterrà tutti gli interi compresi fra 1 e 10.
Analogamente, per creare una lista contenente tutte le lettere dell'alfabeto basterà scrivere ['a' .. 'z'] i caratteri vanno racchiusi fra apici.
Le operazioni sulle liste avvengono nella maggior parte dei casi per pattern matching, operando sulla testa della lista, un elemento e iterando sulla coda, il resto, quindi zero o più elementi.

Esempio, dati in input una lista di numeri restituire la somma.
Il caso base è costituito dalla lista vuota, che ovviamente ha somma pari a zero, mentre nel caso induttivo si deve separare la testa della lista *x* dalla coda *xs*.
Il valore da restituire sarà pari alla somma di *x* e l'applicazione ricorsiva della funzione sull'input *xs*.
Il comportamento della funzione *sommaLista* applicata all'input [1; 2; 3] è il seguente.

sommaLista [1; 2; 3] = 1 + sommaLista [2; 3] =
1 + 2 + sommaLista [3] = 1 + 2 + 3 + sommaLista []
= 1 + 2 + 3 + 0 = 6

```
#light
open System
Console.Clear()
let rec sommaLista lst =
    match lst with
    | [] -> 0
    | x::xs -> x + sommaLista xs
Console.WriteLine("Somma della lista  {0}",sommaLista [1; 2; 3])
Console.WriteLine()
Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione")
let a= Console.ReadKey()
```



MODULO 4

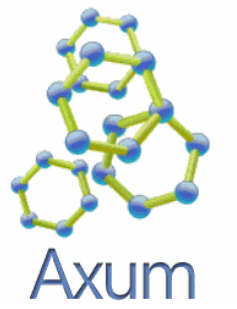
AXUM

Programmazione parallela
Compilazione CLI
MDE

PROGRAMMAZIONE PARALLELA

INTRODUZIONE

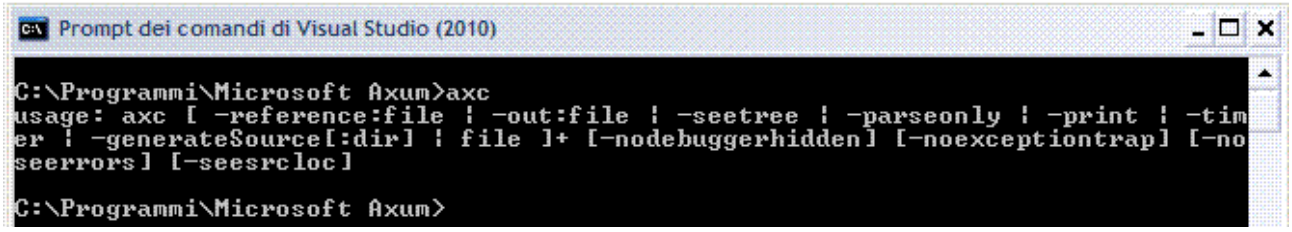
Regno di Axum antico regno situato nell'Etiopia settentrionale (I-VII secolo d.C.).
Fondato da popolazioni provenienti dall'Arabia meridionale, aveva il suo centro principale ad Axum, città nota ora per i suoi resti archeologici.
Il regno di Axum aveva buoni legami commerciali sia con il mondo greco-romano, sia con il subcontinente indiano, e nel III e nel VI secolo la sua influenza giungeva fino allo Yemen.
All'inizio del IV secolo la popolazione si convertì alla fede copta cristiana e la Bibbia fu tradotta nell'idioma locale (gecez), ma il contatto con il resto del mondo cristiano venne interrotto con la conquista musulmana dell'Africa settentrionale (VII secolo).



Le applicazioni “data-intensive” beneficiano del parallelismo.
Il punto è che per ottenere delle buone performance non è sufficiente avere un grande numero di core: per avere un sistema equilibrato è necessario che ci sia una adeguata larghezza di banda per l’accesso alla memoria, un bassa latenza nelle comunicazioni, una altrettanto adeguata larghezza di banda per l’I/O.
Quello che si verifica sperimentalmente è che se ho una sistema che ha come collo di bottiglia nella larghezza di banda per l’accesso alla memoria, risulterà del tutto inutile l’aggiunta di nuovi core.

COMPILAZIONE CLI (*COMMAND LINE INTERFACE*)

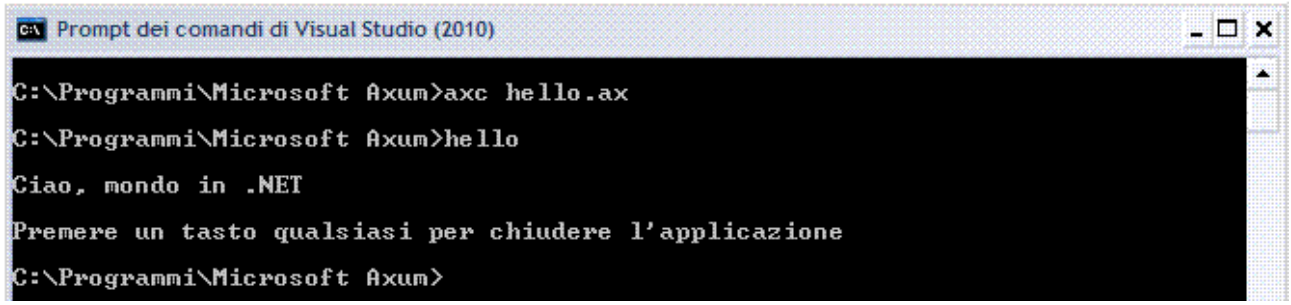
INTRODUZIONE



APPLICAZIONE CONSOLE

Sono semplici da costruire, non hanno grafica e hanno interfaccia **CUI** (*Character User Interface*) che permette all'utente d'interagire con la tastiera e una finestra.

```
// Nome dell'applicazione: hello.ax
// Programmatore:
// Descrizione:
using System;
agent Program : Microsoft.Axum.ConsoleApplication
{ override int Run(String[] args)
        { Console.WriteLine();
         Console.WriteLine("Ciao, mondo in .NET");
         Console.WriteLine();
         Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione");
        Console.ReadKey();
        }
}
```



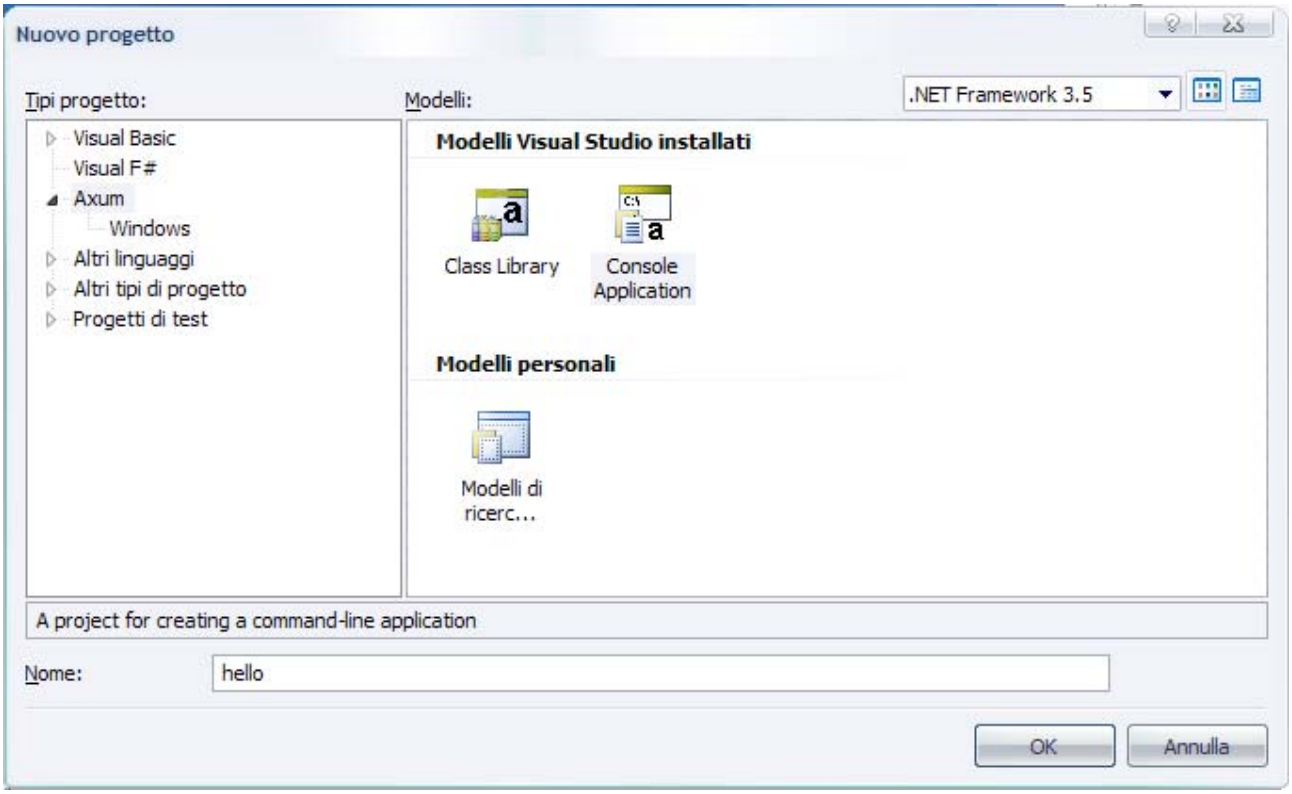
L'applicazione inizia con la direttiva e la parola chiave *agent*.
Il concetto di un agente deriva da ciò che è noto in informatica come Actor Model, un modello in cui sono rappresentate delle entità, dette attori, che comunicano fra loro scambiando dei messaggi
L'applicazione in Axum è basata sulla definizione degli agenti e sull'organizzare dell'interazione tra di loro.
La programmazione Agent-based è diversa dall'OOP, prima di tutto, a differenza di oggetti, gli agenti non forniscono metodi pubblici, non si possono, in un agente, modificare tutti i suoi campi, non è possibile chiamare un metodo su un agente, invece, si può inviare un messaggio e mettere l'agente in attesa di risposta.
Axum è fornito con una libreria di supporto di classe, che comprende un agente chiamato *ConsoleApplication*, che implementa le applicazioni console, l'avvio, i parametri della riga

di comando e lo shutdown.
Quando si deriva da *ConsoleApplication*, è necessario eseguire l'override del metodo principale e quindi inserire il codice dell'applicazione.
Essendo un linguaggio .NET, Axum può utilizzare le librerie del .NET Framework, nell'esempio, si chiama *WriteLine* della classe *System.Console.*

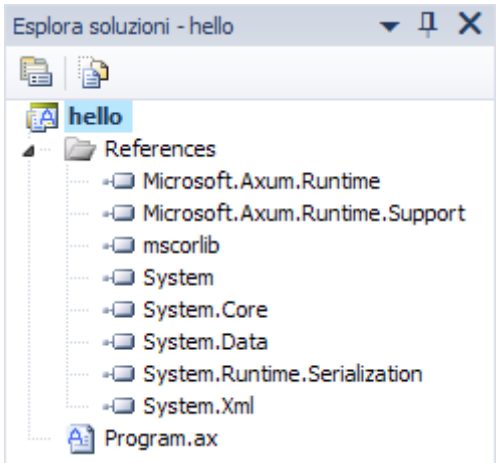
MESSAGE-PASSING

Per rendere l'esempio precedente più interessante, s'introduce il concetto di canali e s'implementa un agente che invia un messaggio alla porta del canale.
Gli agenti sono componenti che eseguono operazioni sui dati, i dati normalmente entrano ed escono attraverso un canale, quindi per lavorare con diversi tipi di dati, un canale ha uno o più porte.

Esempio, utilizzare un canale per inviare e ricevere messaggi.
Per creare un nuovo progetto, fare clic su ***File/Nuovo Progetto… (CTRL+N)***.
Sarà visualizzata la finestra di dialogo ***Nuovo Progetto***.



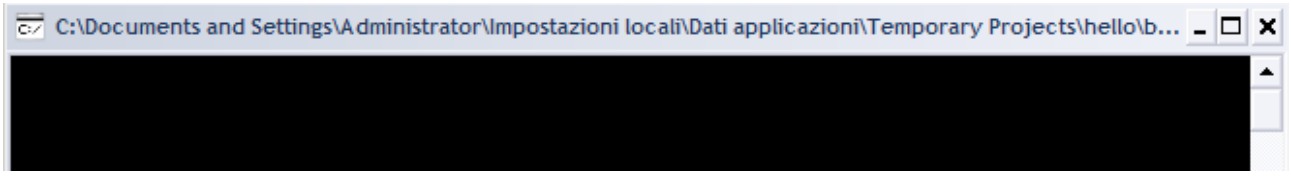
Selezionare ***ConsoleApplication***, ***Nome:*** *hello*, nel nuovo progetto Axum inserisce automaticamente i seguenti file.



File PROGRAM.AX
Ha il codice seguente già inserito.

```
using System;
using Microsoft.Axum;
using System.Concurrency.Messaging;
namespace ConsoleApplication1
{
   public domain Program
   {
      agent MainAgent : channel Microsoft.Axum.Application
      {
         public MainAgent()
         {
            String [] args = receive(PrimaryChannel::CommandLine);
            // TODO: Add work of the agent here.
            PrimaryChannel::Done <-- Signal.Value;
         }
      }
   }
}
```

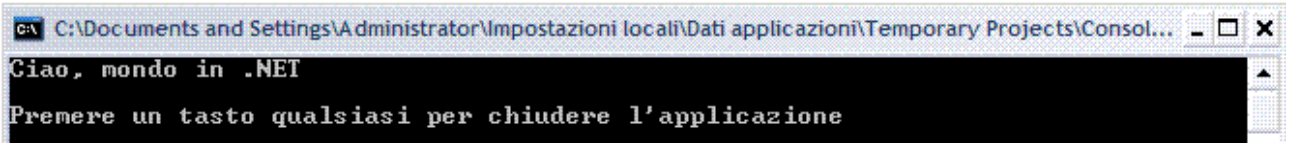
Lanciando prima la compilazione e poi l'esecuzione si ottiene la seguente finestra.



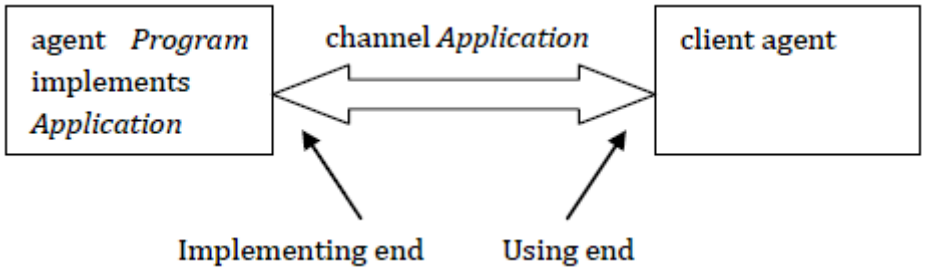
Modificare il codice nel modo seguente.

```
using System;
using Microsoft.Axum;
using System.Concurrency.Messaging;
namespace ConsoleApplication1
{
   public domain Program
   {
      agent MainAgent : channel Microsoft.Axum.Application
      {
         public MainAgent()
         {
            { // riceve dalla linea di comando gli argomenti dalla porta CommandLine
               String [] args = receive(PrimaryChannel::CommandLine);
               Console.Clear();
               Console.WriteLine("Ciao, mondo in .NET");
               Console.WriteLine();
               Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione");
               Console.ReadKey();
               // invia un messaggio alla porta
               PrimaryChannel::Done <-- Signal.Value;
            }
```

```
        }
    }
  }
}
```



L'applicazione è basata su di un *MainAgent*, che implementa il *channel Microsoft.Axum.Application*; è quello che consente di scambiare messaggi con la console.
L'implementazione di un canale è diversa, sintatticamente e semanticamente, da un agente di base.
Quando un agente implementa un canale, la parola chiave *channel* dopo i due punti nella dichiarazione dell'agente, si "attacca" per implementare il fine di quel canale e diventa il "server" di messaggi su quel canale.
L'altra estremità del *channel*, conosciuta come *using end*, è visibile solo per il "cliente" o il componente, in genere un altro agente, per l'altra estremità del canale.
I canali hanno due ends: *implementing end* e *using end*.
Esistono canali per trasmettere messaggi tra gli agenti.
Canali che definiscono quali tipi di dati possono andare in I/O, ma a differenza degli agenti, non eseguono alcuna trasformazione di tali dati.
In figura: le due estremità di un canale.



Nell'esempio lo *using* end di *Application* è implementato nell'Axum run-time.
Il run-time istanzia l'implementazione del canale *Microsoft.Axum.Application*, invia i parametri della riga di comando alla porta *CommandLine* del canale e poi attende un messaggio sulla porta *Done*.
Quando il messaggio è ricevuto, l'applicazione termina.
Il *MainAgent* attende un messaggio che arriva sulla porta *CommandLine* dichiarazione di ricezione e poi i segnali di completamento con l'invio di un messaggio alla porta *Done* (operatore <-), ha una proprietà *PrimaryChannel* per accedere al canale.
I doppi due punti "::" sono usati per accedere alla porta del canale.
La ricezione di un messaggio è un'operazione di blocco, significa che le istruzioni di ricezione attendono fino a quando arriva il messaggio a quella porta.
D'altra parte, l'invio è asincrono, il mittente del messaggio non aspetta che arrivi a destinazione.

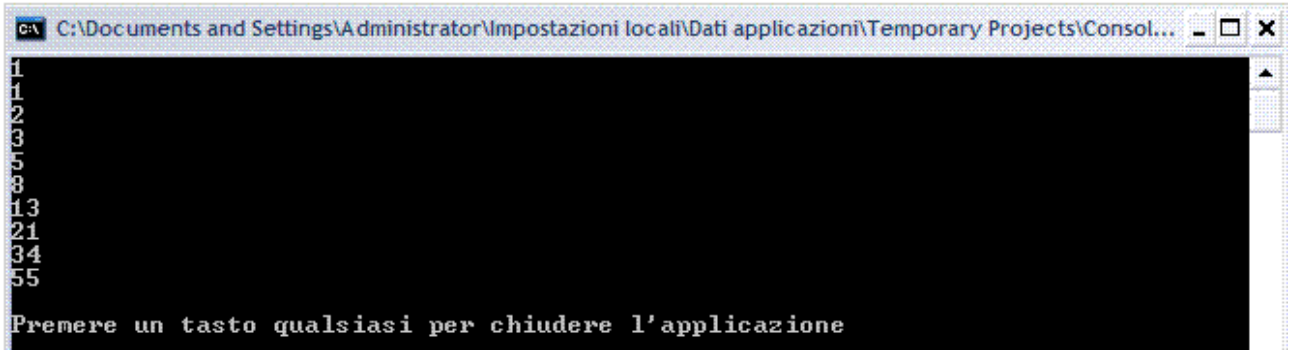
Programmazione asincrona con i messaggi
I messaggi sono il principale mezzo di comunicazione tra agenti, nel suo insieme, ci si riferisce a questo come *orchestration*.
Axum offre due distinti approcci all'*orchestration.*
1. Controlflow.
2. Dataflow.

Spesso, i due approcci sono combinati.
In Axum, i messaggi sono inviati e ricevuti dagli *interaction point*, un punto d'interazione da cui ha origine un messaggio è chiamata *source* e la destinazione è chiamato *target*.
Un punto d'interazione può essere sia *source* sia *target*, il che significa che è possibile inviare e ricevere messaggi.
Questo permette la composizione multipla di punti d'interazione nelle reti dataflow: è un costrutto che riceve i dati ed esegue una trasformazione su di essi e produce un risultato.
Utilizzando una rete dataflow può essere vantaggioso se alcuni nodi della rete sono indipendenti l'uno dall'altro e quindi in grado di permettere l'esecuzione concorrente.
A differenza del controlflow che si basa su istruzioni condizionali, cicli e le chiamate ai metodi, dataflow fonda la sua logica sulla trasmissione, il filtraggio, il bilanciamento del carico e l'unione di messaggi che passano attraverso la rete.
È un approccio diverso e complementare per la gestione messaggi.

FIBONACCI

È una rete dataflow che calcola la sequenza di Fibonacci.

```
using System;
using Microsoft.Axum;
using System.Concurrency.Messaging;
namespace hello
{   public domain Program
   {   agent MainAgent : channel Microsoft.Axum.Application
      { function int Fibonacci(int n)
        {  if( n<=1 ) return n;
            return Fibonacci(n-1) + Fibonacci(n-2);
        }
         int numCount = 10;
        void ProcessResult(int n)
        {  Console.WriteLine(n);
           if( --numCount == 0 )
            {    Console.WriteLine();
               Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione");
               Console.ReadKey();
                PrimaryChannel::ExitCode <-- 0;
             }
         }
         public MainAgent()
         {   var numbers = new OrderedInteractionPoint<int>();
            Console.Clear();
            // crea la pipeline
            numbers ==> Fibonacci ==> ProcessResult;
            // invia i messaggi
           for( int i=1; i<=numCount; i++ ) numbers <-- i;
         }
      }
   }
}
```



Fibonacci è un metodo che non modifica nessun stato al di fuori di sé, in altre parole, non lascia alcun effetto collaterale della sua esecuzione.
Per esempio, se si tenta di modificare *numCount* membro o inviare un messaggio nella funzione *Fibonacci*, il compilatore genera un errore.
Il costruttore di *MainAgent* : la prima istruzione crea l'istanza seguente.
OrderedInteractionPoint<int>();
Che è un punto d'interazione che agisce sia come source sia come target.
La parola *Ordered* significa che l'ordine dei messaggi è preservato, i messaggi sono in fila nell'ordine del loro arrivo, e lasciano la fila nello stesso ordine.
Successivamente, l'agente imposta la rete di dataflow utilizzando l'operatore di *forwarding* ==>.
La dichiarazione seguente.

numbers ==> Fibonacci ==> ProcessResult;

Deve essere intesa come: ogni volta che un messaggio arriva al punto d'interazione *numbers*, lo trasmette ad una trasformazione punto d'interazione attuato dalla funzione *Fibonacci*, quindi trasmettere il risultato al metodo *ProcessResult*.
Si nota che nelle reti di dataflow i messaggi da un nodo all'altro avanzano come in una pipeline, per cui la pipeline è la forma più semplice di una rete di dataflow.

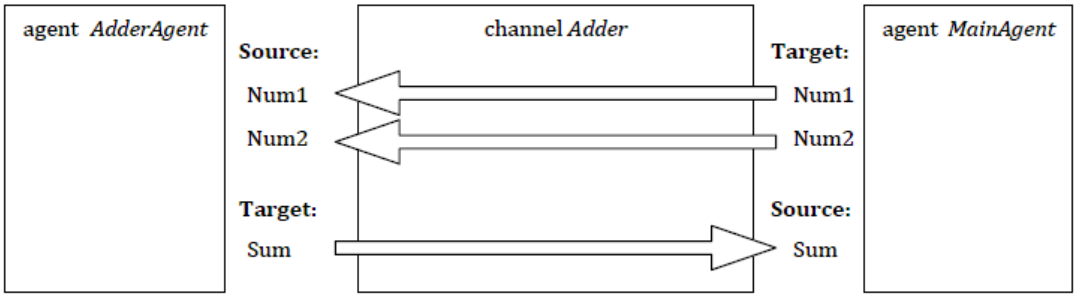
PROGRAMMAZIONE CON GLI AGENTI

Le reti di dataflow funzionano bene per "dati in dati out", ma non si specifica esattamente come i dati viaggiano attraverso la rete e non consentono ai diversi tipi di dati di entrare o uscire dalla rete.

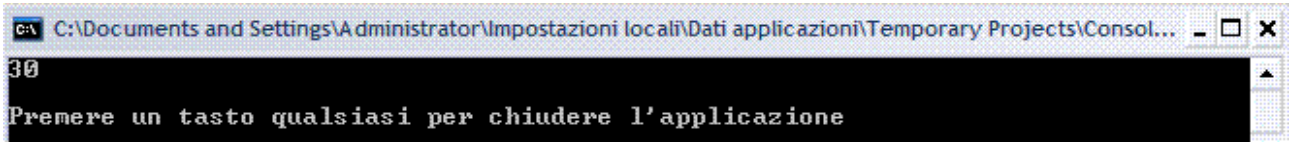
Canali e Porte

Due agenti comunicano su un canale, ma sono disaccoppiati gli uni dagli altri: non conoscono l'altra implementazione, il "contratto" tra di essi è specificato dal canale.
Prendendo in prestito un'analogia dell'OOP, il canale funge da interfaccia e l'agente di come la classe implementa l'interfaccia.
Quando si utilizza un canale, s'inviano i dati alle porte d'ingresso e si ricevono i dati dalle porte di uscita.
In pratica le porte di ingresso sono *target* e le porte di uscita sono *source*.

Esempio, si consideri un canale *Adder* che prende due numeri e produce la somma.
L'utente del canale manda i numeri per le porte d'ingresso *Num1* e *Num2* e riceve il risultato sulla porta di uscita *Sum*.
Il canale è utilizzato da un agente *MainAgent* ed è implementato da un agente *AdderAgent*.



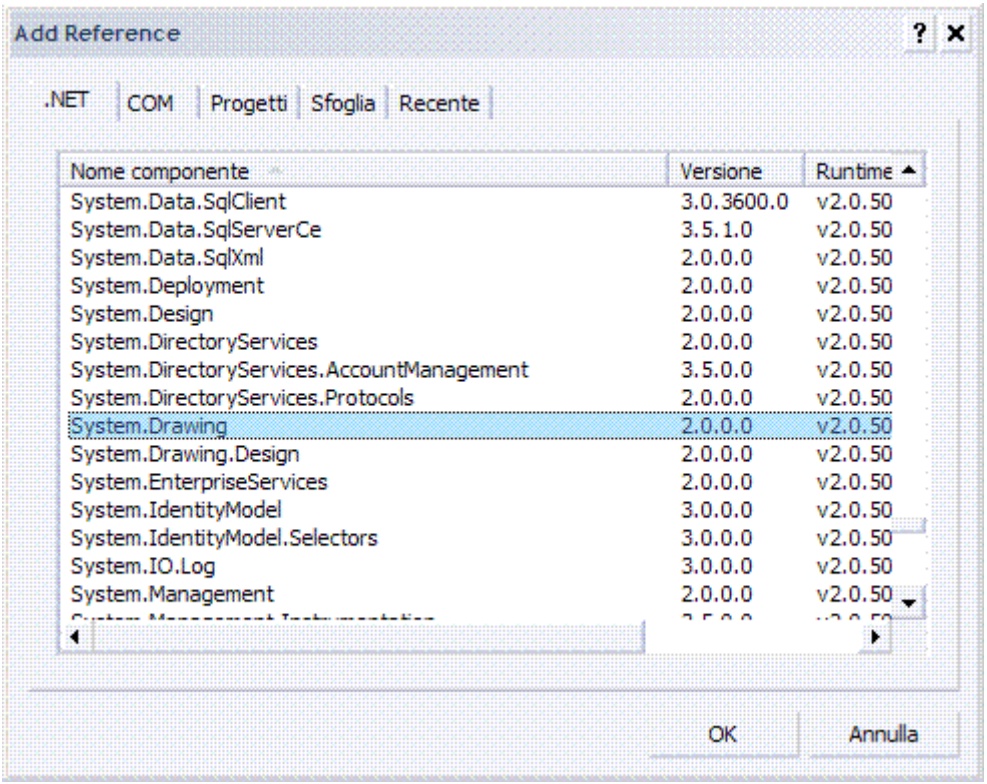

```
using System;
using System.Concurrency;
using Microsoft.Axum;
channel Adder
{ input int Num1;
  input int Num2;
  output int Sum;
}
agent AdderAgent : channel Adder
{ public AdderAgent()
 { int result = receive(PrimaryChannel::Num1) + receive(PrimaryChannel::Num2);
   PrimaryChannel::Sum <-- result;
 }
}
agent MainAgent : channel Microsoft.Axum.Application
{ public MainAgent()
  { var adder = AdderAgent.CreateInNewDomain();
    adder::Num1 <-- 10;
    adder::Num2 <-- 20;
    var sum = receive(adder::Sum);
    Console.Clear();
    Console.WriteLine(sum);                    // stampa 30
     Console.WriteLine();
     Console.WriteLine("Premere un tasto qualsiasi per chiudere l'applicazione");
     Console.ReadKey();
    PrimaryChannel::ExitCode <-- 0;
  }
}
```



Il canale è definito con la parola chiave *channel* e le sue porte con le parole chiave *input* e *output*.
Nel costruttore della *MainAgent* si è prima creato un'istanza di canale *adder* chiamando il metodo statico *CreateInNewDomain* su *AdderAgent*.
Questo crea un'istanza di due ends del canale *adder* che sono istanze di tipi diversi, crea un'istanza di *AdderAgent* che implementa il canale, quindi restituisce lo using end del canale.
La motivazione per l'uso di agenti in questo esempio si trova nella capacità di sovrapporre l'esecuzione di *MainAgent* e *AdderAgent*.
È possibile fare qualcosa mentre *AdderAgent* esegue il calcolo.

IMAGERESIZER

Fare clic sul menu ***Progetto/Aggiungi riferimento…***

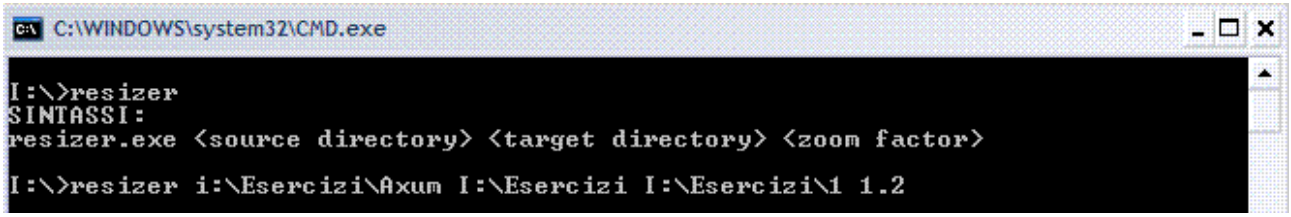


File RESIZER.AX

```
using System;
using System.Concurrency;
using Microsoft.Axum;
using System.IO;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;
namespace ImageResizer
{   schema ImageProcessParam
    {  required string SourceFile;
       required string TargetFile;
       required double ZoomFactor;
    }
    schema ImageWriteParam
    {  required string TargetFile;
       required Bitmap Bitmap;
    }
    channel ChProcessImage
    {  input ImageProcessParam ProcessParam;
       output bool Done;
    }
    reader agent ImageProcessor : channel ChProcessImage
    {   public ImageProcessor()
        {
            // setto la pipeline
```

```
        PrimaryChannel::ProcessParam ==> Resize ==> Save ==>
PrimaryChannel::Done;
    }
    private ImageWriteParam Resize(ImageProcessParam param)
    {   Bitmap sourceBitmap = new Bitmap(param.SourceFile);
        int newWidth = (int)(sourceBitmap.Width * param.ZoomFactor);
        int newHeight = (int)(sourceBitmap.Height * param.ZoomFactor);
        Bitmap targetBitmap = new Bitmap(newWidth, newHeight);
        Graphics g = Graphics.FromImage(targetBitmap);
        g.InterpolationMode = InterpolationMode.HighQualityBicubic;
        g.DrawImage(sourceBitmap, 0, 0, newWidth, newHeight);
        g.Dispose();
        return new ImageWriteParam{TargetFile=param.TargetFile, Bitmap=targetBitmap};
    }
    private bool Save(ImageWriteParam param)
    {   ImageCodecInfo jpegCodec = GetEncoderInfo("image/jpeg");
        if (jpegCodec == null)
        return false;
        EncoderParameters encoderParams = new EncoderParameters(1);
  EncoderParameter qualityParam = new EncoderParameter(Encoder.Quality, (long)85);
        encoderParams.Param[0] = qualityParam;
        param.Bitmap.Save(param.TargetFile, jpegCodec, encoderParams);
        return true;
    }
    private ImageCodecInfo GetEncoderInfo(string mimeType)
    {  // codecs per tutti i formati immagine
        ImageCodecInfo[] codecs = ImageCodecInfo.GetImageEncoders();
        // trova il codec corretto
        for (int i = 0; i < codecs.Length; i++)
        if (codecs[i].MimeType == mimeType)
            return codecs[i];
        return null;
    }
}
reader agent MainAgent : channel Microsoft.Axum.Application
{   public MainAgent()
    {   var args = receive(PrimaryChannel::CommandLine);
        if (args.Length != 3)
        {   Console.WriteLine("SINTASSI:");
Console.WriteLine("resizer.exe <source directory> <target directory> <zoom factor>");
        }
        else
        {  ResizeImages(args);  }
        PrimaryChannel::Done <-- Signal.Value;
    }
    void ResizeImages(string[] args)
    {   var sourceDir = args[0];
        var targetDir = args[1];
        var zoomFactor = Double.Parse(args[2]);
        var sw = Stopwatch.StartNew();
        var di = new System.IO.DirectoryInfo(sourceDir);
        var files = di.GetFiles("*.jpg");
        int numFiles = files.Length;
        var processors = new List<ChProcessImage>();
```

```
            foreach (var file in files)
            {  var param = new ImageProcessParam
               { SourceFile = file.FullName,
               TargetFile = Path.Combine(targetDir, file.Name),
               ZoomFactor = zoomFactor
               };
               var processor = ImageProcessor.CreateInNewDomain();
               processors.Add(processor);
               processor::ProcessParam <-- param;
            }
            receive(processors::Done);
            sw.Stop();
            Console.WriteLine("Elapsed time: {0}", sw.Elapsed);
        }
    }
}
```



APPLICAZIONI DISTRIBUITE

Axum ha un grande supporto per la scrittura di applicazioni distribuite.
In effetti, uno dei motivi per l'adozione di una tale linea di isolamento è così che i domini possono interagire a livello locale o in remoto senza alcun cambiamento nel modello.
Per esempio, il modo in cui il web è programmato con domini, servizi, domande di applicazioni **SOA** (*Service Oriented Architecture*), agenti di gestione dei protocolli.
Per raggiungere un agente all'interno di un dominio, si deve avere un indirizzo, questo è vero in locale e in remoto.
All'interno di un processo, è un po' più facile, perché il nome del tipo di agente si comporta come un indirizzo di default.
Il run-time di Axum attraverso un'interfaccia denominata *IHost*, permette di dare un agente indirizzo all'interno di un dominio, per essere precisi, quello che si associa con un indirizzo è una fabbrica per gli agenti di tipo *hosted*, che è utilizzato per creare istanze agente quando si crea una nuova connessione.
Ogni comunicazione di base o servizio di hosting deve avere la sua implementazione di *IHost*, Axum è dotato di uno per WCF e uno per in-process communication.
L'indirizzo può essere associato ad un'istanza di dominio esistente, nel qual caso ha creato un'istanza agente associata a tale dominio.
Ad esempio, se si sta progettando un Axum-based, è possibile ospitare domini come servizi con il codice seguente.

```
channel Simple
{ input string Msg1;
  output string Msg2;
}
domain ServiceDomain
{ agent ServiceAgent : channel Simple
  { public ServiceAgent ()
    { // …I  }
  }
```

```
}
agent Server : channel Microsoft.Axum.Application
{ public Server ()
 { var hst = new WcfServiceHost(new NetTcpBinding(SecurityMode.None, false));
   hst.Host<ServiceDomain.ServiceAgent>("net.tcp://localhost/Service1");  }
}
```

Ogni volta che un client si connette all'indirizzo *net.tcp://localhost/Service* una nuova istanza di *ServiceAgent* è creata, associata con una nuova istanza *ServiceDomain*.
Se, invece, si voleva creare agenti da associare ad una singola istanza di dominio, si deve passare uno per un *Host*.

```
hst.Host<ServiceDomain.ServiceAgent("net.tcp:...", new ServiceDomain());
```

Vi è un'interfaccia corrispondente per il lato client, chiamata *ICommunicationProvider*.
Questa è utilizzata per creare una nuova connessione ad un servizio di Axum o qualsiasi altro servizio che è scritto in Axum, è un accoppiamento lasco.
Essa, inoltre, deve avere una versione per ogni comunicazione di base e il run-time di Axum è dotato di uno per WCF e uno per in-process communication.
La connessione al servizio è simile a questa.

```
var prov = new WcfCommunicationProvider(new NetTcpBinding(SecurityMode.None,
false));
var chan = prov.Connect<Simple>("net.tcp://localhost/Service1");
```

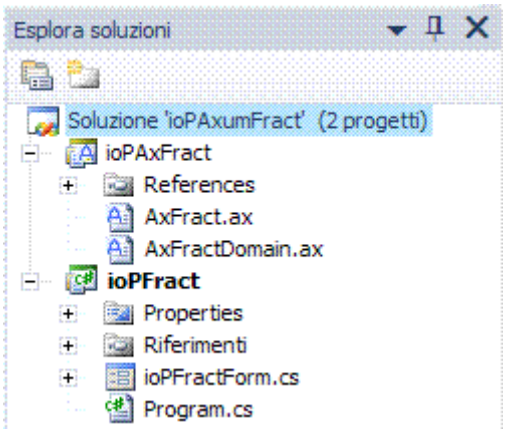
Naturalmente, non è necessario creare un nuovo provider di comunicazione per ogni connessione, o un nuovo host per ogni chiamata *Host*.

FRATTALI

L'obiettivo è quello di confrontare le performance di un'applicazione scritta con Axum con quelle ottenibili utilizzando la corrispondente versione seriale, facendo l'attenzione anche sulle modifiche da apportare al codice per introdurre il parallelismo nelle applicazioni.
L'applicazione è un generatore di frattali di Mandelbrot, composto da un'interfaccia grafica che consente d'inserire i parametri da utilizzare per la generazione dei frattali e di visualizzare il risultato in tempo reale e un'applicazione console che fornirà il servizio di elaborazione dei frattali con l'utilizzo di Axum.

Frattali di Mandelbrot
Un frattale è un oggetto geometrico che si ripete nella sua struttura allo stesso modo su scale diverse, ovvero che non cambia aspetto anche se visto con una lente d'ingrandimento.
Questa caratteristica è chiamata auto similarità: self-similarity.
Il termine fu introdotto da Benoît Mandelbrot, (Varsavia, 20 novembre 1924) è un matematico polacco naturalizzato francese, ad indicare la natura di oggetti matematici di dimensione frazionaria.
I frattali che prendono il suo nome sono basati sulla formula seguente.
$a_{n+1}=a_n^2 + c$
Dove *a* e *c* sono numeri complessi.

Per l'applicazione si utilizza come interfaccia utente un'unica maschera, da cui selezionare il tipo di generazione dei frattali da utilizzare, classica o parallela e i parametri per il calcolo; nella stessa maschera sarà visualizzata l'immagine generata.
Creare un nuovo progetto.



Aggiungere al form i seguenti componenti: tre *TextBox*, due *NumericUpDown*, un *Checkbox*, un *PictureBox*, uno *StatusStrip*, un *Button* e cinque *Label*.
Al form, aggiungere il codice per gestire le interazioni con l'utente e generare le immagini senza utilizzare il parallelismo.
Il metodo *SerialFract* generare l'immagine, al suo interno la procedura *ImpostaUnPixel* provvede ad impostare il colore dei pixel elaborati in base al numero di cicli utilizzato per l'elaborazione; la procedura *GeneraBitmap* provvede a generare una bitmap a partire dall'array di byte contenente il risultato delle elaborazioni precedenti.

```
string SerialFract(int imageWidth, int imageHeight, double xcenter, double ycenter, double
size, int numMaxIterazioni)
      {  Byte[] allbits;
         System.Drawing.Imaging.PixelFormat format =
System.Drawing.Imaging.PixelFormat.Format32bppArgb;
         int pixelFormatSize = Image.GetPixelFormatSize(format) / 8;
         int stride = imageWidth * pixelFormatSize;
         allbits = new byte[stride * imageHeight];
         double left = xcenter - size / (double)2;
         double top = ycenter - size / (double)2;
         double xr = size;
         double yr = size;
         double xs = xr / imageWidth;
         double ys = yr / imageHeight;
         double xc, yc;
         double xsqr, ysqr, x, y;
         int i, j, cnt;
         for (yc = top, j = 0; j < imageHeight; yc += ys, j++)
         {  for (xc = left, i = 0; i < imageWidth; xc += xs, i++)
            {  cnt = 0; x = y = xsqr = ysqr = 0.0;
               while (cnt < numMaxIterazioni && xsqr + ysqr < (double)4)
               {  xsqr = x * x;
                  ysqr = y * y;
                  y *= x;
                  y += y + yc;
                  x = xsqr - ysqr + xc;
                  cnt++;
               }
               cnt *= 10;
      ImpostaUnPixel(i, j, cnt, numMaxIterazioni, imageWidth, pixelFormatSize, ref allbits);
            }
         }
```

```
        return GeneraBitmap(stride, format, ref allbits);
    }
```

Il corpo della funzione è composto da due cicli *for* innestati, che scorrono le ascisse e le ordinate dell'immagine da generare identificando il pixel da elaborare; per ogni pixel è quindi effettuato un ciclo *while* per stabilire il valore da assegnare al pixel stesso, in base al quale nella funzione *ImpostaUnPixel* è scelto il colore da mostrare.

```
void ImpostaUnPixel(int x, int y, int aVal, int numMaxIterazioni, int imageWidth, int
pixelFormatSize, ref byte[] bits)
    {  int irange = numMaxIterazioni;
        int val = (aVal) % irange;
        double range = irange;
        double per50 = range / (double)2;
        double newVal = val;
        double temp = (newVal) / (range / 2);
        byte R = (byte)((double)255 / (temp * temp + (double)1));
        temp = (newVal - per50) / (range / 3);
        byte G = (byte)((double)255 / (temp * temp + (double)1));
        temp = (newVal - range) / (range / 3);
        byte B = (byte)((double)255 / (temp * temp + (double)1));
        int pos = (x + y * imageWidth) * pixelFormatSize;
        bits[pos] = (byte)B;
        bits[pos + 1] = (byte)G;
        bits[pos + 2] = (byte)R;
        bits[pos + 3] = (byte)255;
    }
```

Alla fine dell'elaborazione è richiamata la funzione *GeneraBitmap*, che provvede a salvare nel file FRATTALESER.BMP l'immagine finale.

```
string GeneraBitmap(int stride, System.Drawing.Imaging.PixelFormat format, ref byte[]bits)
    {  // dall'array di byte genera la bitmap
        GCHandle handle = GCHandle.Alloc(bits, GCHandleType.Pinned);
        IntPtr pointer = Marshal.UnsafeAddrOfPinnedArrayElement(bits, 0);
        Bitmap bitmap = new Bitmap(bmpFrattale.Width, bmpFrattale.Height, stride,
format, pointer);
        string nf = "FrattaleSer.bmp";
        bitmap.Save(nf);
        bitmap.Dispose();
        return nf;
    }
```

Metodo utilizzato per creare la bitmap a partire dall'array di byte *bits*: è allocato un handle facendolo puntare all'array già valorizzato, utilizzando il parametro *GCHandleType.Pinned* per evitare che il GC possa spostare la posizione dell'array nell'heap e vanificare quindi il lavoro fatto; è successivamente ricavato l'indirizzo del primo elemento dell'array e passato tale puntatore al costruttore della bitmap.
L'immagine da mostrare è salvata su disco: tale scelta consente di effettuare una comparazione delle immagini prodotte dall'algoritmo seriale con quelle generate dal servizio Axum.
L'algoritmo per la generazione dei frattali appena descritto, essendo composto da due cicli innestati, può essere reso facilmente parallelo in quanto i valori elaborati in ogni ciclo non sono dipendenti da quelli generati nel ciclo precedente; tale indipendenza consente di

suddividere il ciclo più esterno in n blocchi, ognuno dei quali eseguito in parallelo.
Le funzioni di Axum possono essere eseguite parallelamente laddove non modifichino lo stato interno dell'*agent*.
Aggiungere nella soluzione un'applicazione console Axum, l'*agent* principale si limita ad esporre sulla porta 5555 il servizio dell'*agent AxumFract*.
Aggiungere un nuovo domain *AxFractDomain* all'applicazione e iniziare ad implementare gli schemi ed i channel necessari; l'*agent* riceve in ingresso gli stessi parametri della funzione *SerialFract*, con in più il numero di *agent* da generare per l'elaborazione della richiesta; in risposta sarà fornito il nome del file contenente la bitmap appena creata:

```
using System;
using System.IO;
using System.Net;
using Microsoft.Axum;
using System.Collections.Generic;
using System.Text;
using System.Concurrency;
using System.Concurrency.Messaging;
using System.Drawing;
using System.Runtime.InteropServices;
namespace AxFract
{  // parametri per la generazione dell'immagine
   public schema FractParams{
      required int imageW;
      required int imageH;
      required double posX;
      required double posY;
      required double scala;
      required int maxIter;
      required int numProcessi;
   }
   public channel FractChannel {
                input FractParams Params;
                output string NomeFile;
   }
```

Il nome *numProcessi* è improprio, in quanto esso identifica gli *agent* utilizzati, che vivono all'interno dell'unico processo *ioPAxFract*.
L'implementazione dell'*agent* che implementa il channel *FractChannel* per motivi di debug sono mostrati a video i parametri della richiesta; successivamente è allocato l'array di byte necessario per la generazione della bitmap.

```
public reader agent AxumFract : channel FractChannel
      { string GeneraBitmap(int imgWidth, int imgHeight,
System.Drawing.Imaging.PixelFormat format, int stride, ref byte[] bytearray) {
         // dall'array di byte genera la bitmap
         unsafe {
               GCHandle handle = GCHandle.Alloc(bytearray, GCHandleType.Pinned);
                IntPtr pointer = Marshal.UnsafeAddrOfPinnedArrayElement(bytearray, 0);
         Bitmap bitmap = new Bitmap(imgWidth, imgHeight, stride, format, pointer);
               string nome = "frattale.bmp";
               bitmap.Save(nome);
               bitmap.Dispose();
               return nome;
```

```
        }
    }
    public AxumFract()
    {  while (true) {
            var par = receive(PrimaryChannel::Params);
            // per debugging: mostro i parametri da utilizzare
            Console.WriteLine("Ricevuta richiesta");
            Console.WriteLine("imageW = " + par.imageW);
            Console.WriteLine("imageH = " + par.imageH);
            Console.WriteLine("posX = " + par.posX);
            Console.WriteLine("posY = " + par.posY);
            Console.WriteLine("scala = " + par.scala);
            Console.WriteLine("maxIter = " + par.maxIter);
            Console.WriteLine("numProcessi = " + par.numProcessi);
            // imposta i parametri comuni per le elaborazioni di tutti gli agent "worker"
            System.Drawing.Imaging.PixelFormat format =
System.Drawing.Imaging.PixelFormat.Format32bppArgb;
            int pixelFormatSize = Image.GetPixelFormatSize(format) / 8;
            int stride = par.imageW * pixelFormatSize;
            // array di byte per la generazione della bitmap finale
            byte[] allbits = new byte[stride*par.imageH];
```

A questo punto è elaborata la parte relativa alle ordinate y dell'algoritmo seriale.

```
            double top = par.posY - par.scala / (double)2;
            double yr = par.scala;
            double ys = yr / par.imageH;
```

Per poi suddividere il lavoro fra tanti *AxWorkerAgent* quanti richiesti nel parametro *numProcessi*.

```
            int numWorkers = par.numProcessi;
            // Numero di righe per ogni worker
            int numCicli = par.imageH / numWorkers;
            FractWorkerChannel[] workers = new FractWorkerChannel[numWorkers];
            // suddivide il "lavoro" fra numWorkers Agent
            for (int idxAgent = 0; idxAgent < numWorkers; ++idxAgent)
            {   // crea nuovo Agent
                workers[idxAgent] = AxWorkerAgent.CreateInNewDomain();
                // invia la richiesta all'Agent appena creato
                workers[idxAgent]::WrkIn <-- new ValoriWorkerIn { NumWorker = idxAgent,
j = idxAgent * numCicli, valYC = top + (double)(idxAgent * numCicli) * ys, valYS = ys,
        fp = par, NumCicli = numCicli};
                Console.WriteLine("Creato agent" + idxAgent);
            }
```

Le richieste, poiché inviate a diversi *agent*, saranno evase in parallelo; l'ottenimento delle risposte sarà invece sequenzializzato per poter copiare le risposte nell'array finale *allbits*; infine è richiamata la funzione *GeneraBitmap*.

```
            ValoriWorkerOut[] risposte = new ValoriWorkerOut[numWorkers];
            // riceve le risposte degli agent e le memorizza nell'array "finale"
            for(int k = 0; k < numWorkers; ++k) {
                risposte[k] = receive (workers[k]::WrkOut);
```

```
                unsafe {
            risposte[k].slice.CopyTo(allbits, k * numCicli * pixelFormatSize * par.imageW);
                }
            }
            // genera la bitmap a partire dall'array di byte
            PrimaryChannel::NomeFile <-- GeneraBitmap(par.imageW, par.imageH,
format, stride, ref allbits);
```

Nell'implementazione dell'*agent*, si usa il blocco *unsafe* per racchiudere l'utilizzo del metodo *CopyTo* dell'array *slice*: poiché i blocchi copiati di volta in volta sono provenienti da worker diversi, essi saranno disgiunti e quindi non potranno verificarsi problemi di accesso contemporaneo allo stesso elemento.
AxWorkerAgent gli schemi necessari alla definizione del *channel* sono i seguenti.

```
    schema ValoriWorkerIn {
       required int NumWorker;
      required int j;
       required double valYC;
       required double valYS;
       required int NumCicli;
      required FractParams fp;
    }
      schema ValoriWorkerOut {
       required int NumWorker;
       required byte[] slice;
    }
    channel FractWorkerChannel {
      input ValoriWorkerIn WrkIn;
       output ValoriWorkerOut WrkOut;
    }
```

In input sono inoltrati all'*agent*, oltre all'insieme dei parametri della richiesta originaria, anche i valori preelaborati nell'*agent AxumFract*, l'indice *j* da cui iniziare il ciclo e il numero di colonne da elaborare, rappresentato nel campo *NumCicli*.

```
        public AxWorkerAgent() {
            var wrkIn = receive (PrimaryChannel::WrkIn);
            FractParams par = wrkIn.fp;
            System.Drawing.Imaging.PixelFormat format =
System.Drawing.Imaging.PixelFormat.Format32bppArgb;
        int pixelFormatSize = Image.GetPixelFormatSize(format) / 8;
           double left = par.posX - par.scala / (double)2; // 2.0 bug (interpretato 0.2)!!
            double xr = par.scala;
            double xs = xr / par.imageW;
            double xc, yc;
            double xsqr,ysqr,x,y;
            int i,j,cnt;
            byte[] bits = new byte[par.imageW * wrkIn.NumCicli * pixelFormatSize];
                for (yc = wrkIn.valYC, j = 0; j < wrkIn.NumCicli; yc += wrkIn.valYS, j++)
                    {   for (xc = left, i = 0; i < par.imageW; xc += xs, i++)
              { ElaboraUnPixel(new ValoriElaborazione{
                        valI = i, valJ = j,
                        maxIter = par.maxIter,
                        valXC = xc,
```

```
                    valYC = yc,
                    imageWidth = par.imageW,
                    pixelFormatSize = pixelFormatSize}, ref bits  );
            }
        }
        PrimaryChannel::WrkOut <-- new ValoriWorkerOut{ NumWorker =
wrkIn.NumWorker, slice = bits };
    }
```

Il corpo dell'*agent* è in pratica l'equivalente della funzione *SerialFract*, con la differenza che il ciclo esterno è limitato al solo range di valori che l'*agent* deve elaborare; il ciclo di *while* è stato invece spostato all'interno della funzione *ElaboraUnPixel*, in modo da consentire un'ulteriore parallelizzazione dell'elaborazione.
È utilizzato uno schema che non è necessario per la definizione di un channel ma solo al fine di raggruppare velocemente i parametri necessari all'elaborazione della funzione *ElaboraUnPixel*.

```
schema ValoriElaborazione {
    required int valI;
    required int valJ;
     required int maxIter;
     required double valXC;
     required double valYC;
     required int imageWidth;
     required int pixelFormatSize;
    }
```

Occorre implementare nell'interfaccia utente le chiamate al servizio *Axum*.
Aggiungere al progetto ***ioPFract*** i riferimenti al progetto ***ioPAxFract*** e all'assembly *Microsoft.Axum.Runtime*, in modo da poterli richiamare nel codice.
Aggiungere nel form un riferimento ad un'interfaccia *ICommunicationProvider* che nel costruttore del form s'inizializza nel modo seguente.

```
ICommunicationProvider cp;
public ioPFractForm()
{   InitializeComponent();
    cp = new TcpCommunicationProvider();
    stopWatch = new Stopwatch();
}
```

Implementare adesso un metodo *DrawIt*, che s'invoca sul clic del bottone *Aggiorna*, in modo da effettuare l'elaborazione con Axum o con il metodo seriale a seconda della selezione effettuata.

```
private void btnAggiorna_Click(object sender, EventArgs e)
 { DrawIt(chkUsaAxum.Checked); }
 private void DrawIt(bool usaAxum)
 {   posX = Double.Parse(ePosX.Text);
     posY = Double.Parse(ePosY.Text);
     scaleFactor = Double.Parse(eScala.Text);
     stopWatch.Reset();
     stopWatch.Start();
     if (usaAxum)
     {   // inizializza i parametri della chiamata
```

```
            FractParams par = new FractParams
            {   imageH = bmpFrattale.Width,
                imageW = bmpFrattale.Width,
                maxIter = (int)eNumColori.Value,
                posX = posX,
                posY = posY,
                scala = scaleFactor,
                numProcessi = int.Parse(eNumProcessi.Text)
            };
        }
        else
        { bmpFrattale.Load(SerialFract(bmpFrattale.Width, bmpFrattale.Height, posX,
posY, scaleFactor, (int)eNumColori.Value));
            FineElab();
        }
    }
```

Se si è scelto di utilizzare Axum, è creata un'istanza dello schema *FractParams* popolata con gli opportuni parametri valorizzati sul form, dopo è inizializzata la connessione al channel da adoperare per l'elaborazione.
Il metodo *OnAxFractDone* è legato alla ricezione del messaggio di risposta sul *channel*, subito prima di effettuare l'invio del messaggio mediante il metodo *Post*. Nell'implementazione è presente anche l'utilizzo di un oggetto *Stopwatch* per misurare i tempi di elaborazione, che sono poi mostrati nella *StatusStrip* del form.

```
    delegate void VoidDelegate();
    private void OnAxFractDone(string nomeFile)
    {   if (!String.IsNullOrEmpty(nomeFile))
        {  bmpFrattale.Load(nomeFile);
            VoidDelegate del = new VoidDelegate(FineElab);
            this.Invoke(del);
        }
        else MessageBox.Show("Errore nell'elaborazione Axum");
    }
```

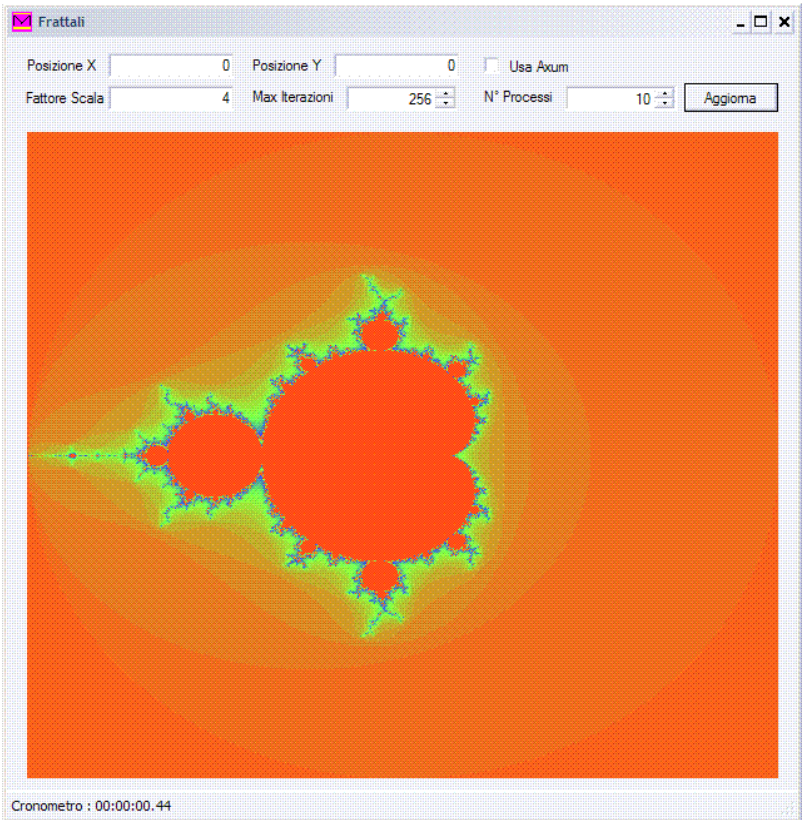
Nel metodo *OnAxFractDone* è caricata la bitmap dal file e invocato il metodo *FineElab* mediante l'utilizzo di un *delegate*.

```
    private void FineElab()
    {   bmpFrattale.Invalidate();
        bmpFrattale.Refresh();
        stopWatch.Stop();
        TimeSpan ts = stopWatch.Elapsed;
        slStatus.Text = String.Format("Cronometro : {0:00}:{1:00}:{2:00}.{3:00}",
            ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10);
    }
```

Tale accorgimento è necessario in quanto in quest'ultimo metodo è invocato il metodo *Refresh()* sul componente *PictureBox* ed essendo in un thread differente rispetto a quello in cui il componente è stato creato, si verificherebbe un'operazione cross-thread.
Per completare il codice dell'interfaccia utente, aggiungere un gestore per l'evento *MouseDown* sul componente *PictureBox*, in modo da centrare l'immagine generata nel punto cliccato, ingrandendo al contempo lo zoom di un fattore pari al 50% del precedente valore.

```
private void bmpFrattale_MouseDown(object sender, MouseEventArgs e)
{   // imposta il nuovo centro in base al punto cliccato
    double left = posX - scaleFactor / 2.0;
    double top = posY - scaleFactor / 2.0;
    posX = scaleFactor * e.X / bmpFrattale.Width + left;
    posY = scaleFactor * e.Y / bmpFrattale.Height + top;
    // aumenta lo zoom (riduce il moltiplicatore)
    scaleFactor = scaleFactor / 1.5;
    ePosX.Text = posX.ToString();
    ePosY.Text = posY.ToString();
    eScala.Text = scaleFactor.ToString();
    DrawIt(chkUsaAxum.Checked);
}
```

Per provare l'applicazione occorre lanciare sia l'applicazione console ***ioPAxFract*** sia l'interfaccia grafica ***ioPFract***.
Mediante l'interfaccia grafica si possono esplorare i frattali generati cliccando di volta in volta nel punto dell'immagine che si desidera ingrandire; nella status status bar si vede indicato il tempo di elaborazione impiegato ad ogni pressione del tasto *Aggiorna*.
L'implementazione con Axum risulta più veloce dell'analoga implementazione seriale, soprattutto all'aumentare del numero massimo d'iterazioni, esempio 5000.
In tali condizioni, su una CPU dual core anche l'elaborazione con un solo processo porta a performance migliori: in questo caso non è il numero di *agent* che fa la differenza, ma la capacità di Axum di parallelizzare l'esecuzione delle funzioni.
Nel caso di un basso numero d'iterazioni, esempio 50 con un fattore di scala elevato, esempio quattro, il miglioramento dei tempi non è così netto, in quanto anche con il metodo seriale l'elaborazione risulta abbastanza veloce.
L'unico caso in cui le prestazioni di Axum risultano equivalenti all'algoritmo sequenziale è nella prima esecuzione, in quanto il servizio ***ioPAxFract*** è inizializzato per la prima volta.



MODULO 5

PYTHON

IronPython on .NET
Console interattiva
Compilazione CLI
MDE

IRON PYTHON ON .NET

INTRODUZIONE

I linguaggi di scripting consentono di ridurre notevolmente le righe di codice necessarie per risolvere determinate classi di problemi.
Python, essendo un linguaggio interpretato dal design curato e moderno, risulta: chiaro, compatto, elegante, versatile, portabile, modulare, estensibile.
La sua semplicità ne ha favorito la diffusione nei contesti più svariati: implementazione rapida di strumenti di supporto, integrazione di sistemi eterogenei, elaborazione del testo, programmazione web e lato server, accesso generico a basi di dati.
L'interprete Python è scritto in ANSI C, è un linguaggio multi paradigma.

- ✓ Imperativo.
- ✓ OOP.
- ✓ Funzionale, non è puro come **LISP** (*LISt Processor*) o Haskell perché non privilegia la ricorsione.

Il controllo dei tipi è forte, strong typing ed è eseguito al run-time: dynamic typing.
In altre parole, una variabile non è altro che un contenitore che nella sua storia può assumere valori sempre dello stesso tipo, al quale è associata un'etichetta, il nome, che, durante l'esecuzione dello script può essere spostata e associata a diversi contenitori anche di tipo diverso.
Usa GC per la gestione automatica della memoria.
Nei linguaggi Pascal e C i blocchi di codice sono indicati con le parentesi oppure con parole chiave, il C usa { }, il Pascal usa *begin* e *end*.
In questi linguaggi è solo una convenzione degli sviluppatori il fatto d'indentare il codice interno ad un blocco, per metterlo in evidenza rispetto al codice circostante.
Python, invece, deriva il suo sistema d'indentazione dal linguaggio Occam: invece di usare parentesi o parole chiave, usa l'indentazione stessa per indicare i blocchi nidificati, si può usare sia una tabulazione, sia un numero arbitrario di spazi bianchi, ma lo standard Python è di quattro spazi bianchi.
Grazie alla sua sintassi, alla tipizzazione dinamica e alla natura di linguaggio interpretato, si presta particolarmente per lo scripting e lo sviluppo rapido di applicazioni multipiattaforma.

Monty Python Flying Circus

Il nome del linguaggio è legato alla passione del suo progettista per la serie televisiva.
Guido Van Rossum, informatico olandese, nel 2001 ha vinto l'Award for the Advancement of Free Software dalla **FSF** (*Free Software Foundation*) alla conferenza FOSDEM a Bruxelles, in Belgio.
Nel dicembre 2005, venne assunto da Google, ha scritto uno strumento web based per controllare il codice per Google in Python.

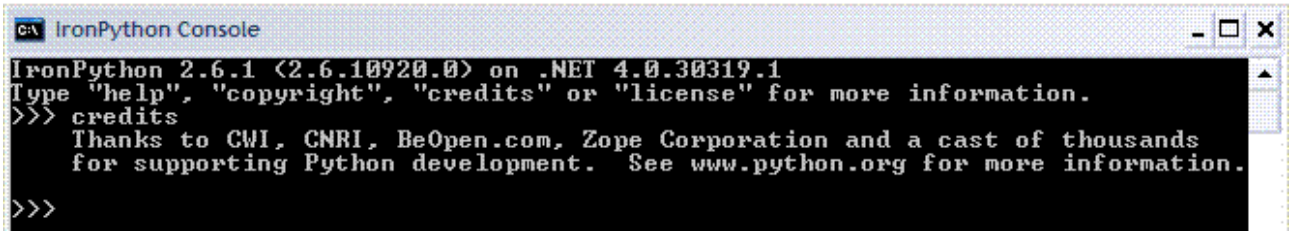
IronPython

È un'implementazione del linguaggio per il CLR, integrato perfettamente con il mondo .NET, rendendo le librerie del Framework utilizzabili da Python, mantenendo allo stesso tempo la totale compatibilità con il linguaggio originale.

CONSOLE INTERATTIVA

INTRODUZIONE

È una caratteristica fondamentale del linguaggio, l'eseguibile IPY.EXE è l'interprete.
Start/Tutti i programmi/Microsoft Visual Studio 2010/IronPython 2.6 for .NET 4.0 IronPython Console



La console interattiva può essere utilizzata come calcolatrice.
>>> 7 + 7
14

Si può eseguire codice più articolato che sarà immediatamente interpretato.
>>> for i in range (1,5):
... print i
1
2
3
4

Per utilizzare le librerie si utilizza la seguente funzione.
>>> import sys

Per conoscere il contenuto del modulo *sys* si utilizza la seguente funzione.
>>> dir(sys)
['__doc__', '__name__', '__package__', '__stderr__', '__stdin__', '__stdout__',
'api_version', 'argv', 'builtin_module_names', 'byteorder', 'call_tracing', 'cal
lstats', 'copyright', 'displayhook', 'dllhandle', 'dont_write_bytecode', 'exc_cl
*ear', 'exc_info', 'exc_traceback', 'exc_type', 'exc_value', 'excepthook', 'exec_*
prefix', 'executable', 'exit', 'flags', 'float_info', 'getcheckinterval', 'getde
faultencoding', 'getfilesystemencoding', 'getrecursionlimit', 'getsizeof', 'gett
race', 'getwindowsversion', 'hexversion', 'last_traceback', 'last_type', 'last_v
*alue', 'maxint', 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path', 'path_*
hooks', 'path_importer_cache', 'platform', 'prefix', 'ps1', 'ps2', 'py3kwarning'
, 'setcheckinterval', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdou
t', 'subversion', 'version', 'version_info', 'warnoptions', 'winver']

Il modulo contiene degli attributi, per esempio *version* che rappresenta la versione dell'ambiente, per visualizzare il valore basta eseguire il comando seguente.
>>> sys.version
'2.6.1 (IronPython 2.6.1 (2.6.10920.0) on .NET 4.0.30319.1)'

Per utilizzare le classi librerie del .NET Framework dalla console interattiva o da uno script, basta utilizzare ancora la funzione *import* ma è necessario utilizzare il nome

completo, per esempio, in caso contrario si otterrebbe un errore di mancata definizione, se si vuol utilizzare la classe *Console* del *namespace System*, scrivere il codice seguente.

```
>>> import System
>>> System.Console.WriteLine ('Ciao, mondo in .NET')
Ciao, mondo in .NET
```

La funzione *import* può essere utilizzata anche per importare il suo contenuto oppure una particolare classe nel *namespace* globale.

```
>>> from System import *
>>> Console.WriteLine ('Ciao, mondo in .NET')
Ciao, mondo in .NET
```

Le funzioni sono oggetti di “prima classe”.

```
>>> def somma (a,b):
...     return(a+b)
>>> somma (7,7)
14
>>>
>>> somma.nuovoattributo = 10
>>> print somma.nuovoattributo
10
```

Normalmente non si usano questi attributi ma possono essere utili.

```
>>> def sommatore (a):
...     if not hasattr(sommatore,"totale"):
...             sommatore.totale=a
...     else:
...             sommatore.totale +=a
...     return sommatore.totale
>>> sommatore(1)
1
>>> sommatore(2)
3
>>> sommatore(5)
8
```

Oggetti richiamati come funzioni: le classi hanno un metodo speciale *__call__* è richiamato quando un oggetto è utilizzato come una funzione, senza specificare il metodo o l'attributo.

```
>>> class ClasseSomma (object):
...     def __call__ (self,a,b):
...             return(a+b)
>>> sommatore = ClasseSomma()
>>> sommatore (7,7)
14
```

Funzioni passate come argomento.

```
>>> def somma (a,b):
...     return (a+b)
>>> altra_somma = somma
>>> altra_somma (7,7)
14
```

Si può passare come argomento anche questo.

```
>>> def quadrato (x):
```

```
...     return(x*x)
>>> def cubo (x):
...     return (x*x*x)
>>> def eleva (a,b,potenza):
...     return (potenza (a+b))
>>> eleva (2,3,quadrato)
25
>>> eleva (2,3,cubo)
125
```

La funzione, *lambda parametri:istruzioni*, permette di creare funzioni anonime molto semplici e richiamate solo in un punto, non può contenere più di un'istruzione, il risultato è restituito senza usare *return*,utile per passare una funzione come argomento.

```
>>> eleva (2,3,lambda x: x*x)
25
>>> eleva (2,3,lambda x: x*x*x)
125
```

Clousures permette di definire funzioni all'interno di altre funzioni e rendono possibile la creazione di decoratori che costituiscono un design pattern della programmazione, sono delle funzioni che aggiungono o modificano il comportamento di altre funzioni.

```
>>> def memo(func):
...     def _memo(*args):
...             if not hasattr(func,'cache'):
...                     func.cache={}
...             if args not in func.cache:
...                     func.cache[args]=func(*args)
...                     return func.cache[args]
...     return _memo
>>> def add (a,b):
...     return (a+b)
```

La funzione *memo* prende come argomento una funzione, si definisce una clousure di nome *_memo*, quest'ultima cerca in un attributo della funzione in input un dizionario “cache” e lo crea se non esiste, in questo dizionario saranno messi da parte i valori di ritorno della funzione *func* con determinati argomenti in input, *memo* restituisce la clousure.
Questo decoratore permette di memorizzare i risultati di una funzione richiamata con determinati argomenti e di restituirne i risultati senza dover richiamare nuovamente la funzione.

```
>>> add = memo (add)
>>> add (7,7)
14
>>> add (7,7)
```

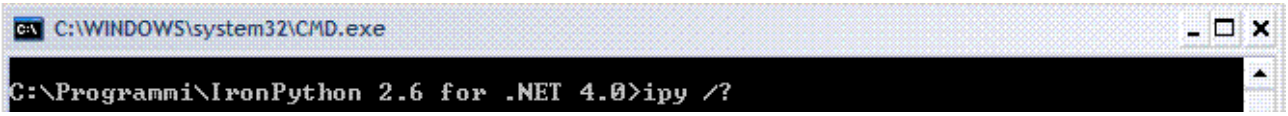
La seconda volta il risultato è prelevato dalla cache.

Uscita dalla console interattiva.

```
>>> quit()
```

COMPILAZIONE CLI (*COMMAND LINE INTERFACE*)

INTRODUZIONE



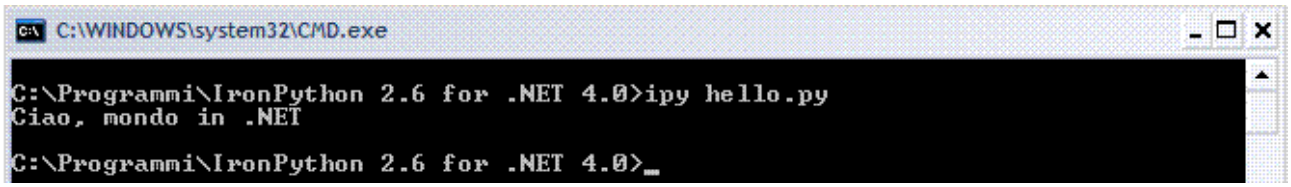

```
C:\Programmi\IronPython 2.6 for .NET 4.0>ipy /?
Usage: ipy.exe Usage: ipy [options] [file.py|- [arguments]]
Options:
 -3                 Warn about Python 3.x incompatibilities
 -c cmd               Program passed in as string (terminates option list)
 -D                 Enable application debugging
 -E                 Ignore environment variables
 -h                 Display usage
 -i                 Inspect interactively after running script
 -m module            run library module as a script
 -O                 generate optimized code
 -OO                  remove doc strings and apply -O optimizations
 -Q arg               Division options: -Qold (default), -Qwarn, -Qwarnall,
-Qnew
 -s                 Don't add user site directory to sys.path
 -S                 Don't imply 'import site' on initialization
 -t                 Issue warnings about inconsistent tab usage
 -tt                Issue errors for inconsistent tab usage
 -u                 Unbuffered stdout & stderr
 -v                 Verbose (trace import statements) (also PYTHONVERBOSE=x)
 -V                 Print the version number and exit
 -W arg               Warning control (arg is action:message:category:module:lineno)
 -x                 Skip first line of the source
 -X:AutoIndent          Enable auto-indenting in the REPL loop
 -X:ColorfulConsole       Enable ColorfulConsole
 -X:CompilationThreshold  The number of iterations before the interpreter starts compiling
 -X:Debug               Enable application debugging (preferred over -D)
 -X:EnableProfiler        Enables profiling support in the compiler
 -X:ExceptionDetail       Enable ExceptionDetail mode
 -X:Frames              Enable basic sys._getframe support
 -X:FullFrames          Enable sys._getframe with access to locals
 -X:GCStress          Specifies the GC stress level (the generation to collect each statement)
 -X:LightweightScopes     Generate optimized scopes that can be garbage collected
 -X:MaxRecursion          Set the maximum recursion level
 -X:MTA               Run in multithreaded apartment
 -X:NoAdaptiveCompilation Disable adaptive compilation
 -X:PassExceptions        Do not catch exceptions that are unhandled by script code
 -X:PrivateBinding        Enable binding to private members
 -X:Python30            Enable available Python 3.0 features
 -X:ShowClrExceptions     Display CLS Exception information
 -X:TabCompletion         Enable TabCompletion mode
 -X:Tracing           Enable support for tracing all methods even before sys.settrace is called
Environment variables:
```

IRONPYTHONPATH Path to search for module
IRONPYTHONSTARTUP Startup module

SCRIPT

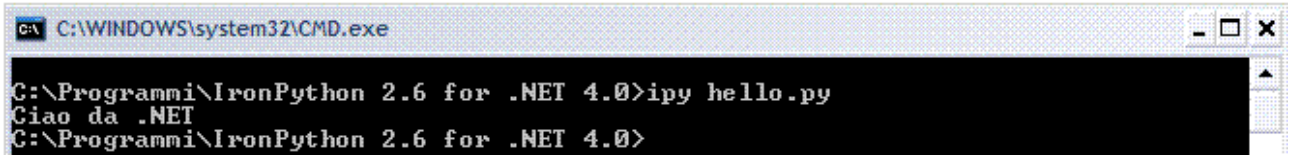
Sono semplici da costruire, non hanno grafica e hanno interfaccia **CUI** (*Character User Interface*) che permette all'utente d'interagire con la tastiera e una finestra.

```
# Nome della applicazione: hello.py
# Programmatore:
# Descrizione:
from System import *
Console.WriteLine ('Ciao, mondo in .NET')
```



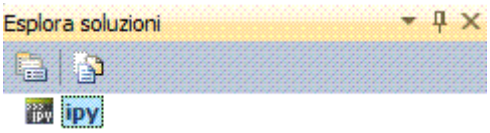
Esempio di utilizzo delle librerie .NET e in particolare i *generics*, per questi è necessario utilizzare le parentesi quadre per il tipo argomento e non < e > come si fa in .NET.

```
from System import *
from System.Collections.Generic import *
lista=List[String]()
lista.Add('Ciao')
lista.Add('da')
lista.Add('.NET')
for elemento in lista:
        Console.Write(elemento+' ')
```

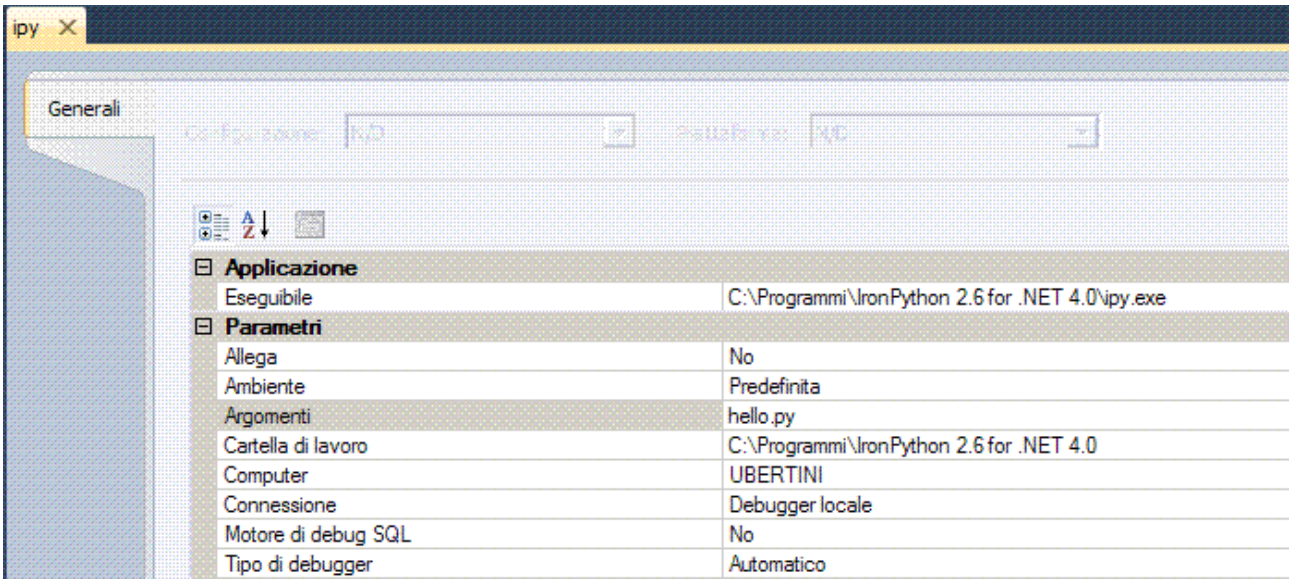


APPLICAZIONE CONSOLE

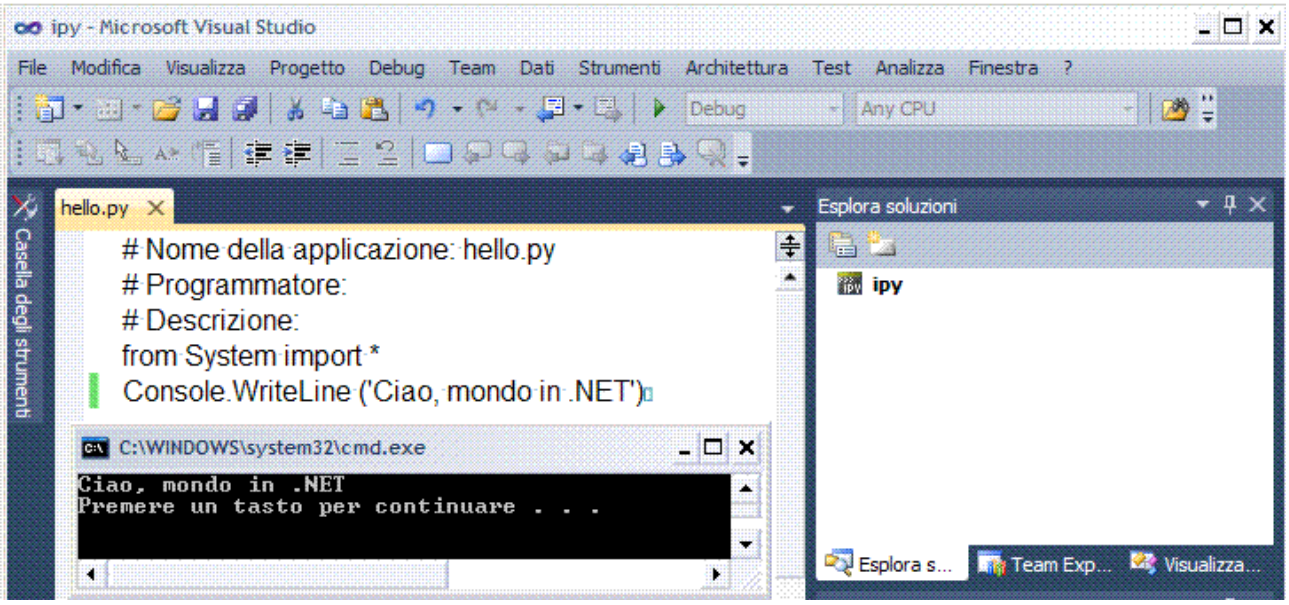
È possibile eseguire e debuggare gli script IronPython da Visual Studio.
Fare clic su ***File/Apri/Progetto/Soluzione… (CTRL+O)/IPY.EXE***



Fare clic con il tasto destro su IPY.EXE in ***Esplora soluzioni*** e selezionare ***Proprietà***.



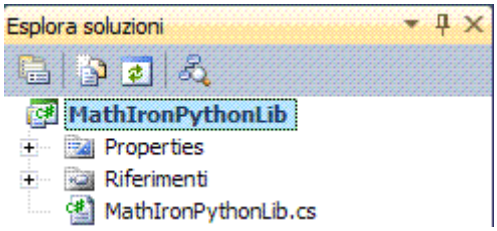
Nella finestra ***Generali*** bisogna inserire nel campo ***Argomenti*** il nome del file da eseguire.
Premere ***F5*** o (***CTRL+F5***).



LIBRERIA DI CLASSI

IronPython può importare direttamente solo alcune delle librerie di .NET, in altre parole quelle standard.
Per le rimanenti, oppure per quelle create dai programmatori in C#, in VB.NET o in un qualunque linguaggio .NET, è necessario qualche operazione in più.

Esempio, creare una libreria in C#, il cui unico metodo calcola il quadrato di un numero intero.
Per creare un nuovo progetto, fare clic su ***File/Nuovo/Progetto… (CTRL+N)***
Altri linguaggi/Visual C#/Libreria di classi/MathIronPythonLib

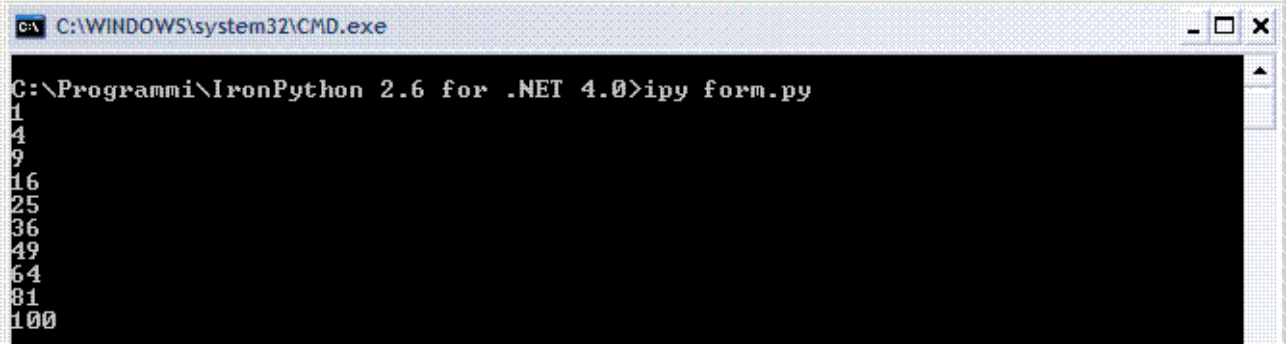


File MATHIRONPYTHONLIB.CS

```
using System;
public class MathIronPythonLib
{
        public int Quadrato(int i)
        { return i*i; }
}
```

Dopo aver salvato la classe, compilare in modo da ottenere la libreria.
Il risultato sarà l'assembly MATHIRONPYTHONLIB.DLL.
Lo script Python, all'interno del quale si utilizza l'assembly creato, utilizza il modulo *clr* e il metodo *AddReferenceToFile*.

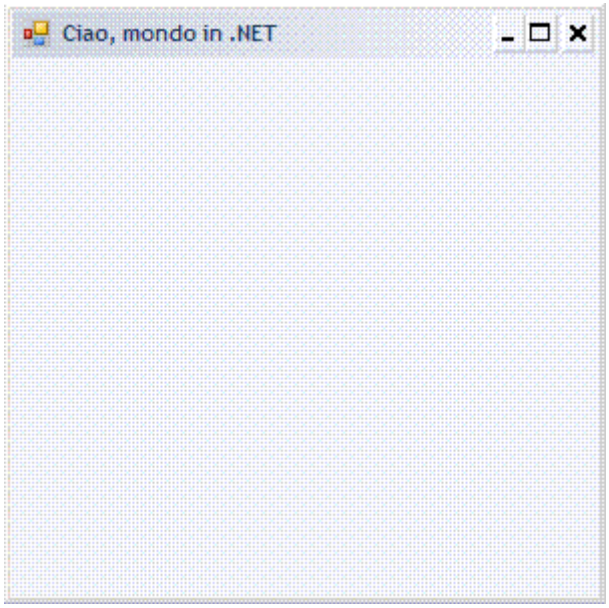
```
import clr
clr.AddReferenceToFile("MathIronPythonLib.dll")
import MathIronPythonLib
m=MathIronPythonLib()
for i in range(1,11):
        print m.Quadrato(i)
```



Applicazione Windows Form

Creare una classe *MainForm* derivata dalla classe base *Form*.

```
import clr
clr.AddReferenceByPartialName("System.Windows.Forms")
clr.AddReferenceByPartialName("System.Drawing")
from System.Windows.Forms import *
from System.Drawing import *
class MainForm(Form):
        def __init__(self):
                self.Text = 'Ciao, mondo in .NET'
Application.Run(MainForm())
```



Il metodo *__init__* è il costruttore della classe, è chiamato automaticamente quando si crea un nuovo oggetto, all'interno del quale è solo inizializzato il testo sulla barra del titolo. Python impone di dire sempre a quale oggetto si fa riferimento, la variabile *self* appare anche come primo argomento di tutti i metodi e rappresenta l'istanza della classe attuale, è il *this* di C#.
Quando si chiama il metodo, non si deve passare un valore a questo parametro, il sistema ci pensa da solo.

Esempio, progettare una finestra con un pulsante.

```
import clr
clr.AddReferenceByPartialName("System.Windows.Forms")
clr.AddReferenceByPartialName("System.Drawing")
from System.Windows.Forms import *
from System.Drawing import *
f = Form(Text="Windows IronPython on .NET", HelpButton=True,MinimizeBox=False, MaximizeBox=False)
f.FormBorderStyle = FormBorderStyle.FixedDialog
f.StartPosition = FormStartPosition.CenterScreen
b1 = Button(Text="Premere il tasto", Location=Point(30,30), Size=Size(100,30))
def push(data, event):
   l = Label(Text="IronPython ready!", ForeColor=Color.Red)
   l.Location = Point(30, 50+f.Controls.Count*25)
```

```
    f.Controls.Add(l)
b1.Click += push
f.Controls.Add(b1)
f.ShowDialog()
```

Esempio, progettare un'applicazione con gestione degli eventi e disegno su un form.

```
import clr
clr.AddReferenceByPartialName("System.Windows.Forms")
clr.AddReferenceByPartialName("System.Drawing")
from System.Windows.Forms import *
from System.Drawing import *
class MainForm(Form):
        def __init__(self):
                        self.Text = 'Ciao, mondo in .NET'
                        self.FormClosing += self.formClosing
                        self.InitializeComponents()
```

Nel costruttore è gestito l'evento *FormClosing* e invocato il metodo *InitializeComponents*, che crea l'interfaccia grafica.

```
        def InitializeComponents(self):
                        bt=Button()
                        bt.Text="Clicca qui"
                        bt.Click += self.bt_Click
                        bt.Location=Point(10,10)
                        bt.Size=Size(80,30)
                        self.Controls.Add(bt)
                        self.MouseDown+=self.formMouseDown
```

Nel metodo *InitializeComponents* è creato un pulsante *bt*, impostate le sue proprietà e il metodo che gestisce l'evento *Click*.
Poi il pulsante è aggiunto alla collezione *Controls* del form.
L'ultima istruzione aggiunge il metodo *formMouseDown* come gestore dell'evento *MouseDown* del form.

```
    def formClosing(self, sender, e):
            result=MessageBox.Show("Vuoi
            uscire?","",MessageBoxButtons.YesNo)
            if result == DialogResult.No:
                e.Cancel = True
                return
```

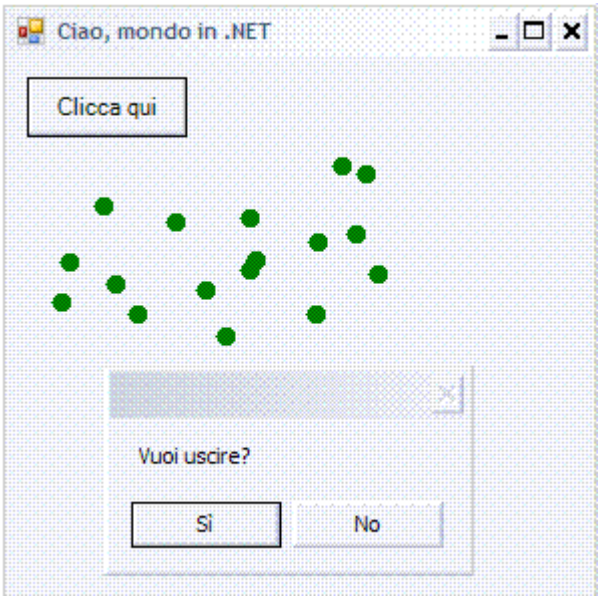
Il metodo *formClosing* gestisce l'evento *FormClosing* che è scatenato quando l'utente tenta di chiudere la finestra.
In questo caso è visualizzata una *MessageBox* con i pulsanti *Sì* e *No*, per confermare o meno l'uscita dall'applicazione.
In caso negativo è impostato il valore *e.Cancel* a *true* per annullare l'evento di chiusura.

```
    def bt_Click(self,sender,e):
            MessageBox.Show("Hai cliccato il pulsante!")
```

Il metodo *bt_Click* gestisce l'evento di clic sul pulsante *bt*, non fa niente di particolare in questo caso, solo visualizzare una *MessageBox* per verificare che in effetti l'evento è gestito.
Il metodo *formMouseDown* fa uso delle funzioni di disegno del *namespace System.Drawing*.

```
    def formMouseDown(self,sender,e):
            gfx=self.CreateGraphics()
            gfx.FillEllipse(Brushes.Green,e.X,e.Y,10,10)
Application.Run(MainForm())
```

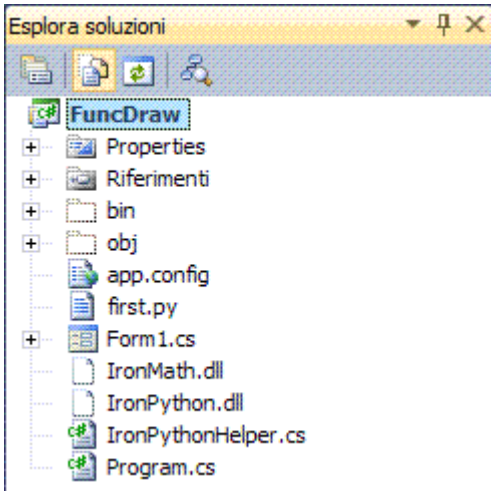
Ad ogni clic sulla superficie del form è disegnato un cerchio verde.



C# E IRONPYTHON ON .NET

È possibile utilizzare l'engine di Python all'interno di un'applicazione .NET e utilizzare quindi le sue funzionalità.

Esempio, progettare un'applicazione che permetta di disegnare il grafico di una funzione a una sola variabile x, sfruttando Python per calcolarne i valori su un dato dominio di x.



La classe che rappresenta e permette di utilizzare il motore dell'interprete Python è *PythonEngine*, le cui istanze possono essere controllate da una qualunque applicazione host scritta in .NET.
Un'istanza PythonEngine permette di effettuare qualunque operazione eseguibile da Python, in questo caso servirà a eseguire un'istruzione di assegnazione alla variabile x, lungo tutto il suo dominio e poi a valutare un'espressione rappresentata da una funzione ad una variabile.
Dopo aver preparato un'interfaccia grafica in C#, che permetta d'inserire la funzione e l'intervallo dei domini e che contenga un pannello sul quale disegnare la funzione.
Creare una classe *Helper* che crei e mantenga l'istanza *PythonEngine*.

File IRONPYTHONHELPER.CS

```
using System;
using System.Collections.Generic;
using System.Text;
using IronPython.Hosting;
using IronPython.Compiler;
namespace FuncDraw
{
   class IronPythonHelper
   {
      private PythonEngine engine;
      public IronPythonHelper()
      { InitializePyEngine(); }
      void InitializePyEngine()
      {
         engine = new PythonEngine();
         // esegui altre inizializzazioni
         engine.Execute("from math import *");
      }
}
```

Nel metodo *InitializePyEngine* sarà possibile eseguire operazioni d'inizializzazione addizionali, come per esempio importare moduli, in questo caso l'importazione del modulo *math* è obbligatorio per calcolare funzioni più complesse, come quelle trigonometriche.
Il processo sarà quello di calcolare il valore di una funzione, su tutti i punti del dominio della variabile x.
Quindi prima di tutto si assegnerà alla variabile x un valore, eseguendo l'istruzione di assegnazione Python del tipo seguente.

X = valore

Per eseguire una qualunque istruzione Python, la classe *PythonEngine* mette a disposizione il metodo *Execute*, con diversi overload.
In questo caso, basterà creare una stringa *x = valore*, dove *valore* sarà di volta in volta un diverso elemento del dominio e poi dare in ingresso la stessa stringa al metodo *PythonEngine.Execute*.
Assegnato il valore alla variabile x si potrà procedere con il valutare la funzione, invocando il metodo *EvaluateAs*, che valuta un'espressione Python restituendo un valore di un tipo predeterminato.
In definitiva tutto ciò che serve è un metodo C# come il seguente.

```
    public double EvaluateExpression(string expr)
    { return engine.EvaluateAs<double>(expr); }
    public double EvaluateFunction(string function,string varName, double varValue)
    {
        try
        {
            engine.Execute(varName + "=" + varValue);
            return engine.EvaluateAs<double>(function);
        }
        catch
        {
            throw;
        }
    }
}
```

File FORM1.CS

Supponendo che la funzione sia rappresentata dalla stringa x * x + 1 e di volerla valutare nel punto di ascissa x = 0, s'invocherà il metodo *EvaluateFunction* nel modo seguente.

```
double y=EvaluateFunction("x*x+1","x",0);
```

In questo caso y assumerà il valore 1.
Per calcolare il valore della funzione su tutti i punti dell'intervallo scelto, basterà un ciclo *for* e per comodità si conserveranno le coppie di punti che costituiscono il grafico della funzione in un oggetto *List<Point>.*
Se *min* e *max* rappresentano rispettivamente l'estremo inferiore e quello superiore del dominio di x e lo step che si desidera utilizzare è di 0.05, il ciclo sarà il seguente.

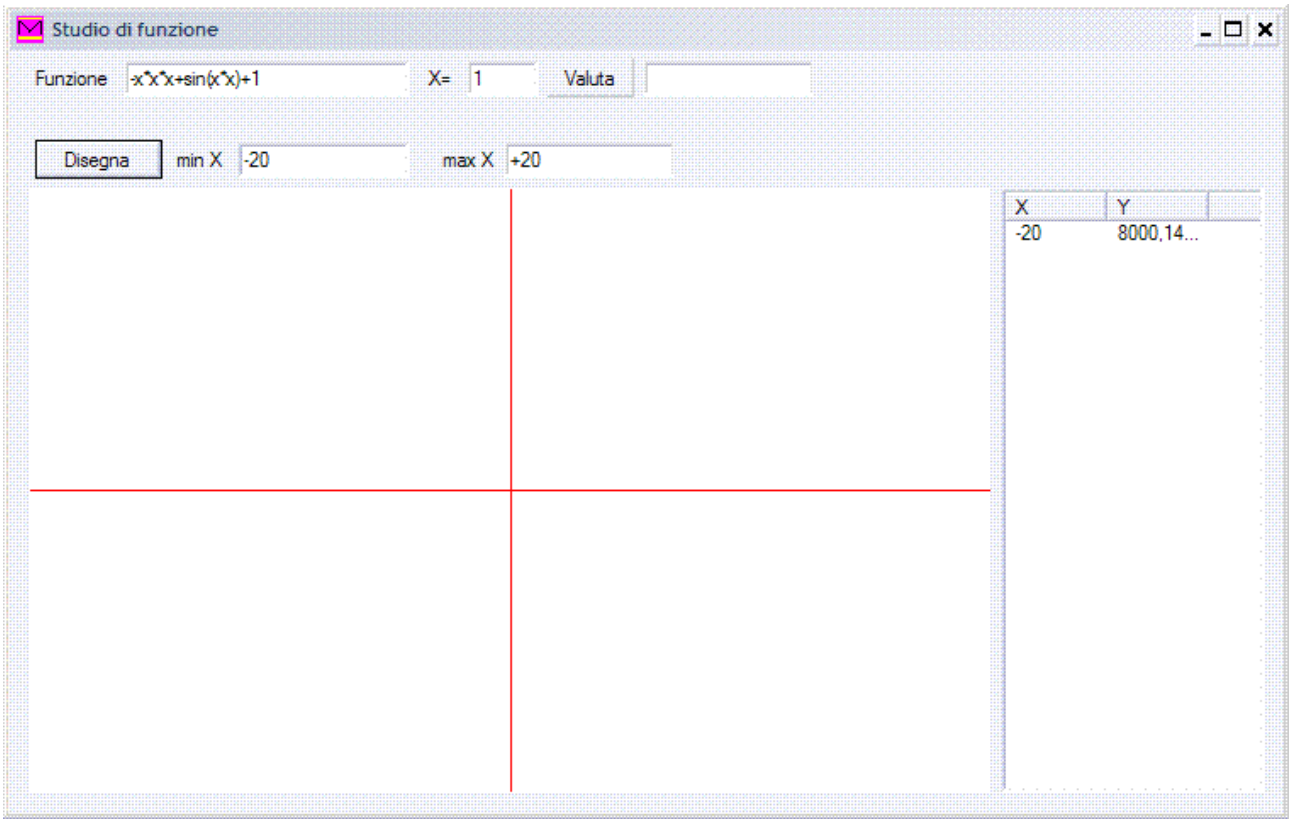
```
python = new IronPythonHelper();
for (double x = min; x < max; x += 0.05)
            {
                y = python.EvaluateFunction(txtExpression.Text, "x", x);
                points.Add(new PointF((float)x, (float)y));
                itm = new ListViewItem(x.ToString());
                itm.SubItems.Add(y.ToString());
                lvwValues.Items.Add(itm);
            }
```

A questo punto non resta che trasformare i valori reali in valori compatibili con le coordinate dello schermo e in particolare del *Panel* utilizzato e disegnare una spezzata formata dall'unione di tutti i punti ottenuti.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace FuncDraw
{
    public partial class Form1 : Form
    {
        private List<PointF> points;
        private PointF[] pointsArray;
        private IronPythonHelper python = new IronPythonHelper();
        private float yscale = 100;
        public Form1()
        {
            points = new List<PointF>();
            InitializeComponent();
        }
        private void btEval_Click(object sender, EventArgs e)
        {
            try
            {
                double x;
                if (double.TryParse(txtValX.Text, out x))
                {
            txtResult.Text = python.EvaluateFunction(txtExpression.Text, "x", x).ToString();
                    errorProvider1.SetError(txtValX,"");
                }
                else errorProvider1.SetError(txtValX, "Inserire un valore numerico");
            }
            catch (Exception exception)
            {
                MessageBox.Show(exception.ToString());
            }
        }
        private void btDraw_Click(object sender, EventArgs e)
        {
            double min,max;
            if(!double.TryParse(txtMin.Text,out min))
            {
                errorProvider1.SetError(txtMin,"Valore non numerico");
                return;
            }
            else errorProvider1.SetError(txtMin,"");
            if(!double.TryParse(txtMax.Text,out max))
            {
                errorProvider1.SetError(txtMax,"Valore non numerico");
                return;
```

```
            }
            else errorProvider1.SetError(txtMax,"");
            lvwValues.Items.Clear();
            points.Clear();
            ListViewItem itm;
            double y;
            lvwValues.SuspendLayout();
            try
            {
                for (double x = min; x < max; x += 0.05)
                {
                    y = python.EvaluateFunction(txtExpression.Text, "x", x);
                    points.Add(new PointF((float)x, (float)y));
                    itm = new ListViewItem(x.ToString());
                    itm.SubItems.Add(y.ToString());
                    lvwValues.Items.Add(itm);
                }
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.ToString());
                return;
            }
            lvwValues.ResumeLayout();
            TranslatePoints();
            panel1.Invalidate();
        }
        private void TranslatePoints()
        {
            float incr = ((float) panel1.Width) / points.Count;
            pointsArray = new PointF[points.Count];
            PointF pt;
            PointF ptO=new PointF(panel1.Width/2,panel1.Height/2);
            for (int i = 0; i < points.Count; i++)
            {
                pt = new PointF();
                pt.X = i*incr;
                pt.Y = (ptO.Y- yscale*((float)points[i].Y)/panel1.Height);
                pointsArray[i]=pt;
            }
        }
        private void panel1_Paint(object sender, PaintEventArgs e)
        {
            Graphics gfx = e.Graphics;
            gfx.DrawLine(Pens.Red,new Point(panel1.Width/2,0),new
Point(panel1.Width/2,panel1.Height));
            gfx.DrawLine(Pens.Red, new Point(0, panel1.Height / 2), new Point(panel1.Width,
panel1.Height / 2));
            if (pointsArray != null && pointsArray.Length > 0)
            {  gfx.DrawLines(Pens.Black,pointsArray);  }
        }
        private void panel1_Resize(object sender, EventArgs e)
        { panel1.Invalidate();  }
    }
```

}


Studio di funzione
Funzione
-x*x*x+sin(x*x)+1
X=
1
Valuta
Disegna
min X
-20
max X
+20
X
Y
-20
8000,14...

MODULO 6

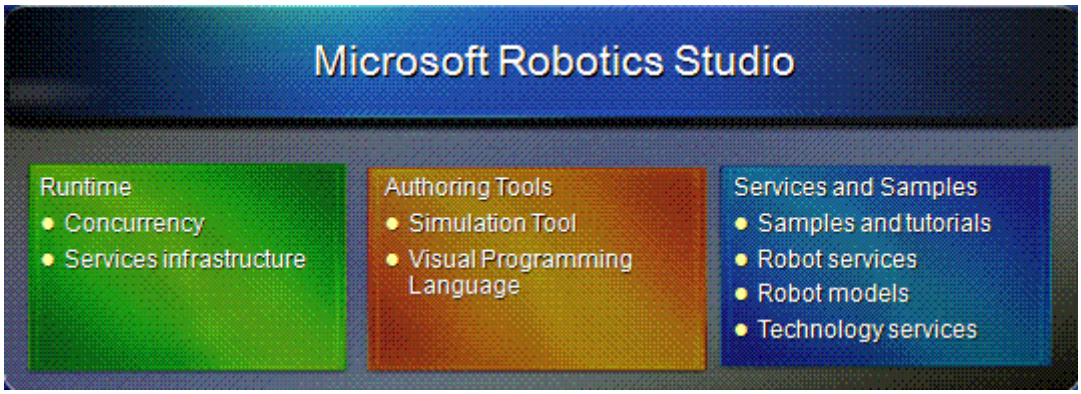
MICROSOFT ROBOTICS DEVELOPER STUDIO

Caratteristiche
Programmazione

CARATTERISTICHE

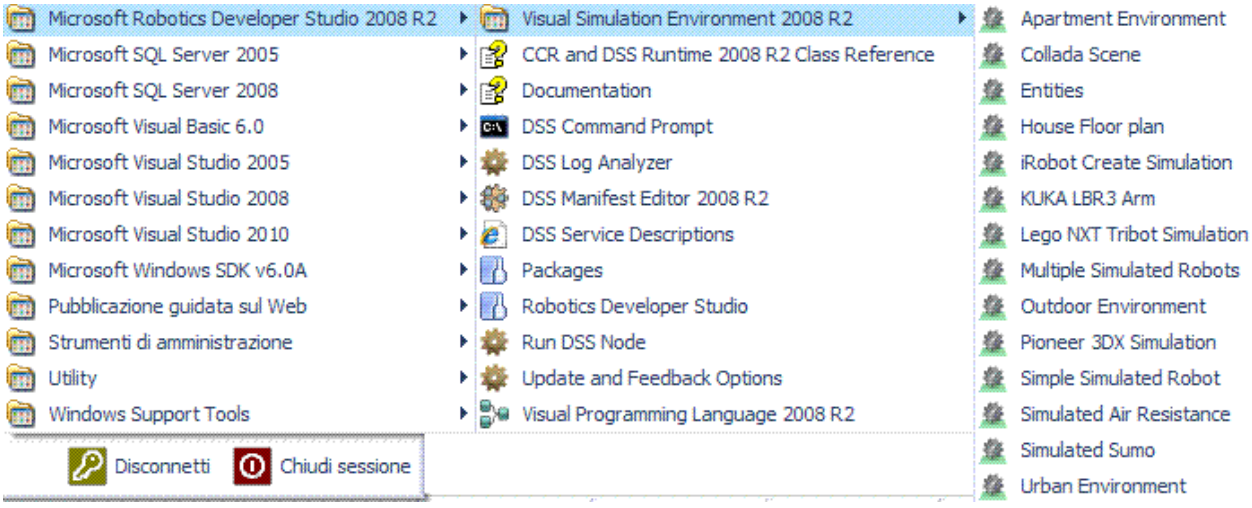
INTRODUZIONE

Ambiente di programmazione dedicato alla cibernetica, riunisce, come in una catena di montaggio, i “pezzi di robot”.
È una piattaforma di sviluppo per la robotica, supporta varie tipologie di utenti e scenari applicativi, inoltre è adattabile ad un’ampia varietà di hardware.



Supporto per LEGO® Mindstorms® NXT.

Primo ambiente **RAD** (*Rapid Application Development*) e visuale per lo sviluppo rapido di applicazioni destinate all’automazione robotica.

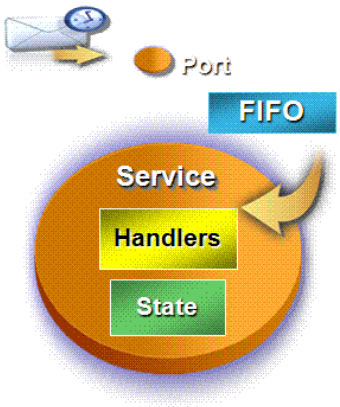


È costituito da quattro componenti principali.
1. VPL.
2. DSS.
3. VSE.
4. CCR.

VPL (*VISUAL PROGRAMMING LANGUAGE*)

È un linguaggio di flusso dei dati, significa che è possibile creare applicazioni disegnando diagrammi sullo schermo.
In fase di run-time, i messaggi si spostano da un blocco all'altro nel diagramma e il flusso dei dati è in realtà il percorso di esecuzione dell'applicazione.
VPL è destinato ai programmatori principianti.
Funziona come una lavagna, si disegna una sorta di catena di montaggio che collega e fa funzionare i diversi componenti di un'automazione.

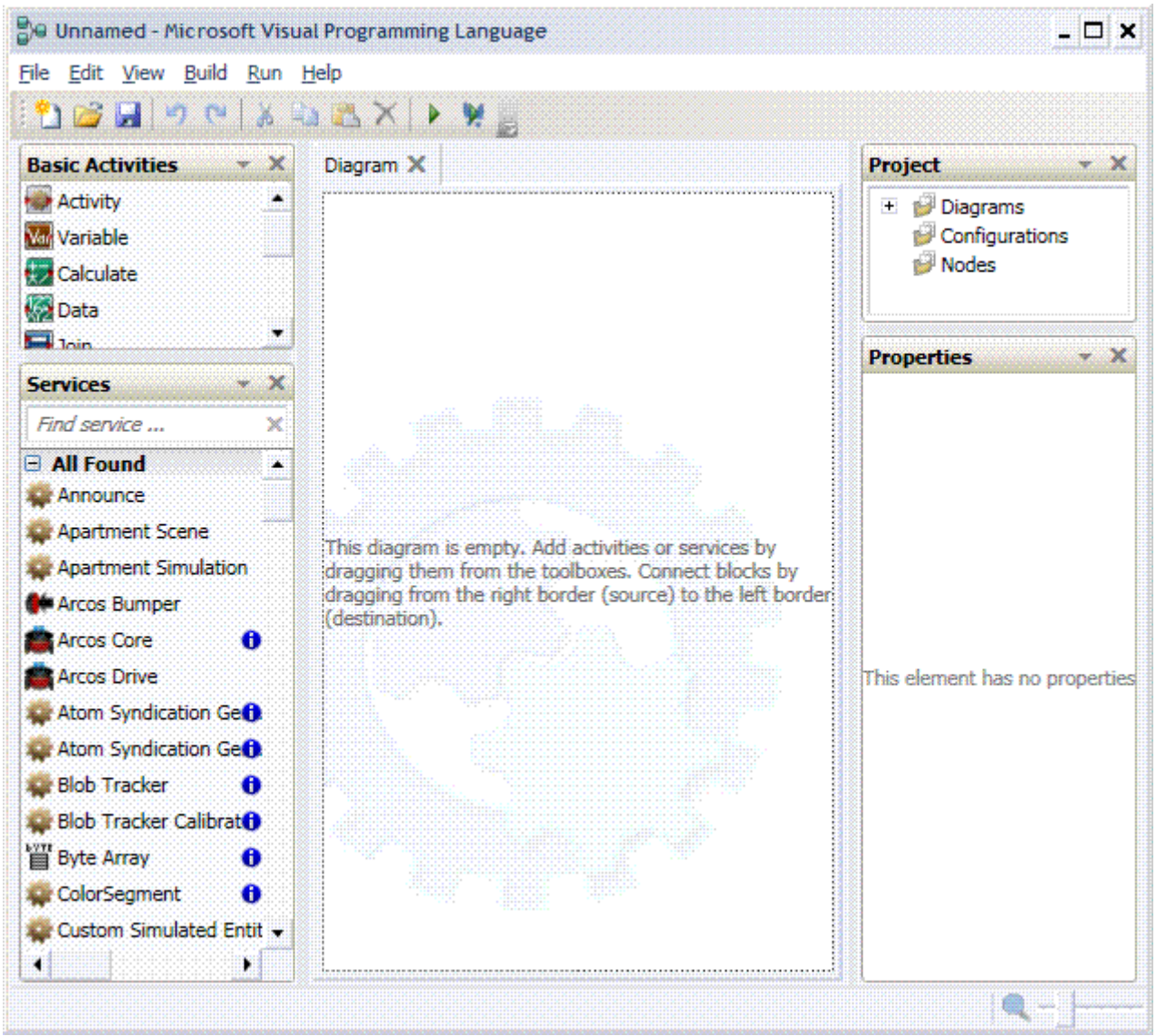
Finestra ***Services***
È un elenco di "pezzi di robot", scorrendoli si scoprono oggetti come i mattoncini Fishertechnics e LEGO MindStorm, joystick per videogiochi, ma anche hardware semi professionale come il MobileRobot Pioneer fabbricato da MobileRobots o la componentistica della Parallax.
I ***Services***, "pezzi di robot", rappresentano l'hardware ai quali fanno riferimento, ne avvolgono la complessità e la astraggono, rendendola trasparente allo sviluppatore; per iniziare a lavorare non si deve fare altro che cliccarci sopra e trascinarli nella finestra centrale ***Diagram***.
Dispone di stati con proprietà definite e interagisce tramite scambio di messaggi.
Operazioni di base: recupero/manipolazione degli stati e notifica di eventi.



Lo stato dei ***Services*** è osservabile a tutti i livelli dell'applicazione e possono fornire una ricca rappresentazione dei dati.
Le specifiche dei "pezzi di robot" sono fornite principalmente dai fabbricanti di robot e sono descritte in file Manifest, un file di configurazione in formato **XML** (*eXtensible Markup Language*).

Finestra ***Basic Activities***
Una volta disposti i ***Services***, è necessario fornirgli istruzioni e farli dialogare tra loro; le istruzioni sono allocate e manipolate attraverso oggetti chiamati ***Basic Activities*** che sono rappresentati graficamente attraverso quadratini con segmenti verdi sui lati sinistro e destro; il nome in codice di questi segmenti verdi è ***pin activities***.
Nel diagramma dell'automazione i dati che "rendono elettrici" i vari ***Services*** entrano nel pin posto a sinistra del quadratino ed escono, dopo essere stati elaborati e instradati, dal

pin posto a destra.



Ogni finestra ***Basic Activities*** possiede wizard di configurazione per impostare le proprietà.

Variabile
Crea una variabile e la tipizza; le variabili sono tipizzate attraverso i medesimi tipi previsti in C#.

Data
Alloca i dati dell'automazione in variabili.

List
Crea e tipizza una serie di dati.

List Functions
Consente di modificare un *List* preesistente.

Calcolate
Consente di sommare, sottrarre dividere o moltiplicare variabili numeriche oppure di concatenare stringhe.

If
Sposta il flusso dell'automazione verificando se una condizione è vera oppure è falsa.

Switch
Smista il flusso dell'automazione confrontando il messaggio in ingresso con un valore impostato dallo sviluppatore.

Merge
Fonde tra loro due o più flussi di automazione.

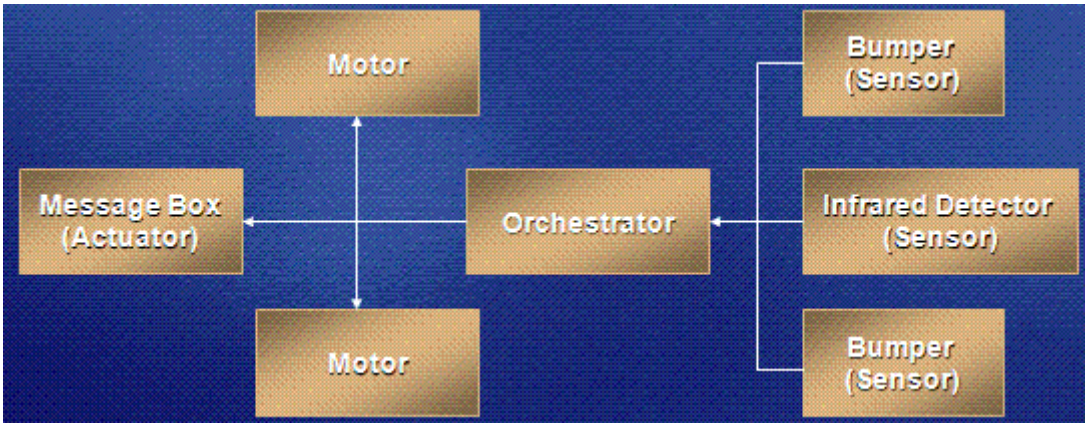
Join
Unisce due o più flussi di automazione dando ad ognuno di questi uno specifico nome.

Activity
È possibile racchiudere l'intero flusso di automazione descritto in un diagramma in un'*Activity*; è eseguita selezionando il menu ***Run***.

Comment
Inserisce un commento al flusso di automazione.

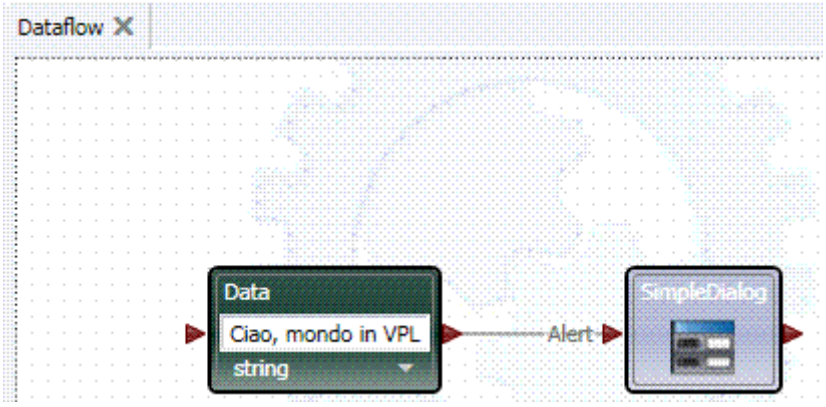
La finestra ***Project*** consente d'impostare il nome del diagramma, di aggiungere oppure o cancellare un diagramma.
La finestra ***Properties*** consente d'impostare le proprietà dell'oggetto ***Services*** oppure ***Activity*** selezionato.

Struttura
La struttura di una tipica applicazione robotica è la seguente.

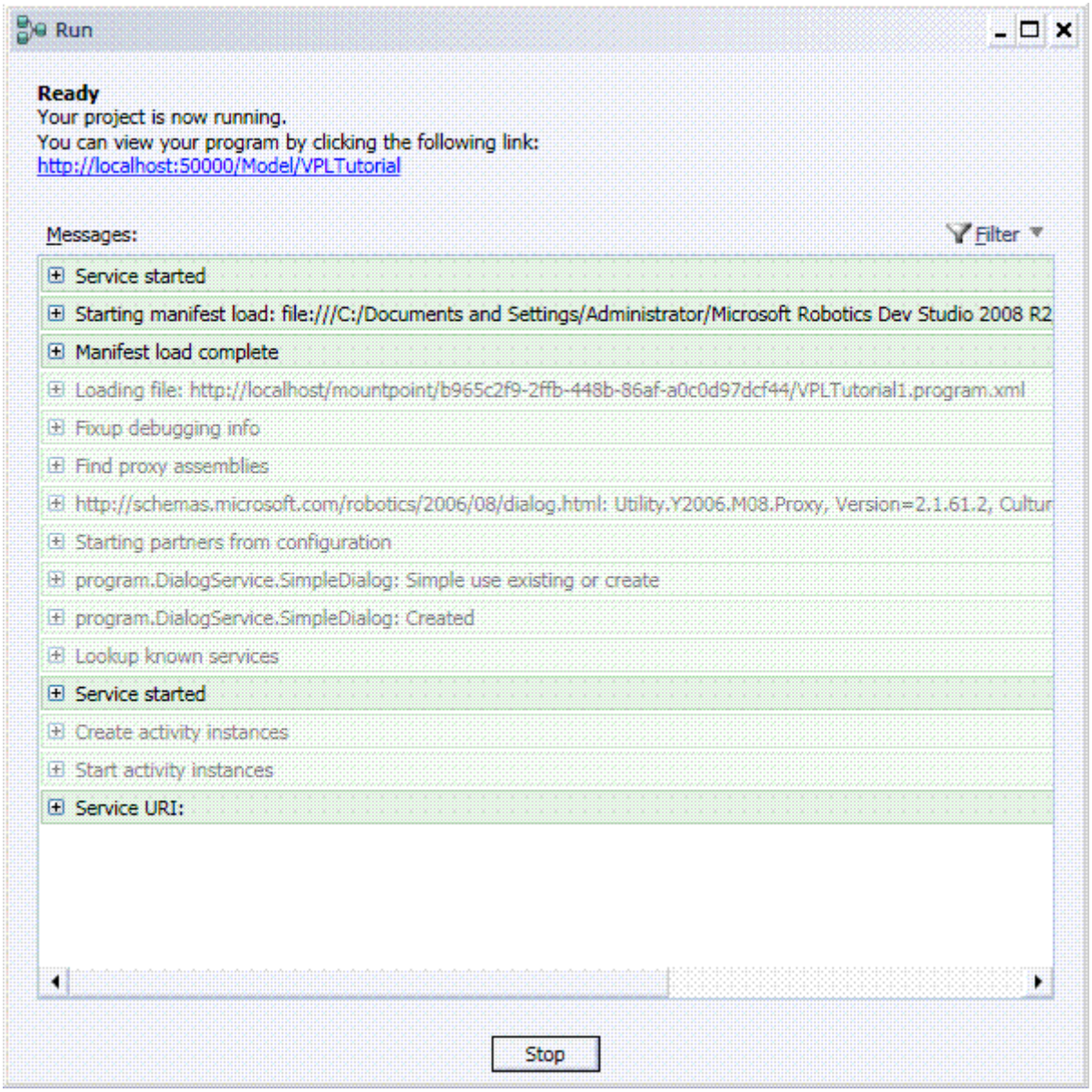


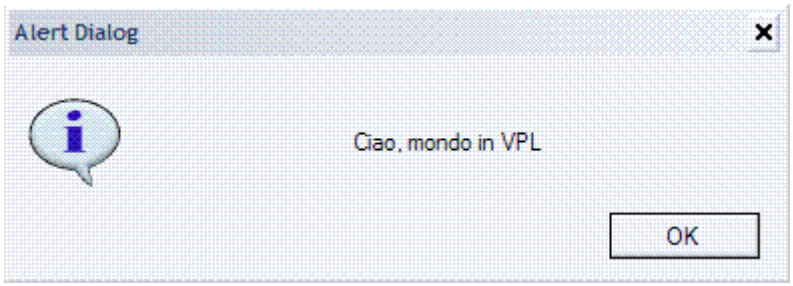
È composta da componenti debolmente accoppiati eseguiti concorrentemente.
Gestione di sensori/attuatori.
Interfaccia utente.
Robotics usa una libreria concorrente basata su .NET Framework e semplifica lo sviluppo di applicazioni asincrone.
Ha un'architettura di messaggistica modulare orientata ai servizi usata per determinare lo stato dei sensori/attuatori tramite browser.

DSS *(Decentralized System Services)*
Fornisce i servizi di run-time.
Scambio di messaggi asincroni fra i nodi.
Un meccanismo d'isolamento garantisce l'affidabilità e il parallelismo dei componenti.

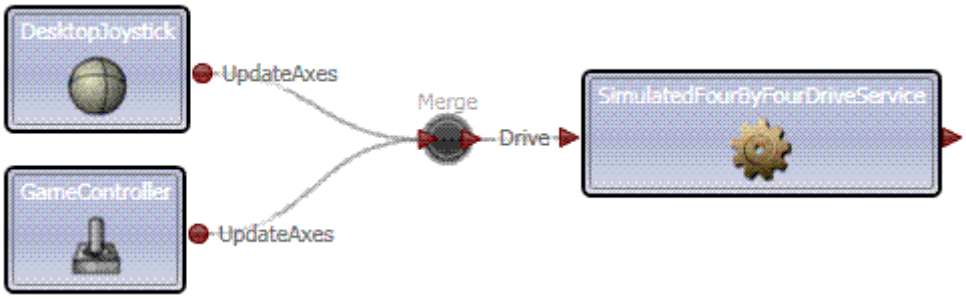


Fare clic sulla barra degli strumenti pulsante ***Start (F5).***

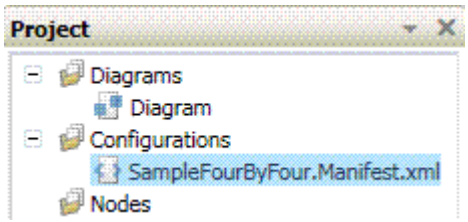




Una volta assemblata l'automazione è possibile telecomandare un robot, oppure avviare un'animazione che simula l'automazione.
Sono i file Manifest dei diversi ***Services*** che impostano il simulatore grafico piuttosto che l'hardware; in altre parole, il simulatore è avviato quando l'automazione utilizza ***Services*** le cui caratteristiche sono descritte in file Manifest che richiedono il simulatore grafico mentre automazioni assemblate con ***Services*** i cui Manifest richiedono hardware comunicano e richiedono la connessione con un robot vero e proprio, se si utilizza un file Manifest dedicato a un hardware reale il Framework cercherà i pezzi restituendo un messaggio di errore se non li trova, per esempio non ho il mattoncino Lego Mindstorm e non è intercettata alcuna connessione Bluetooth.



Nella finestra seguente è selezionato il file Manifest per il progetto corrente.



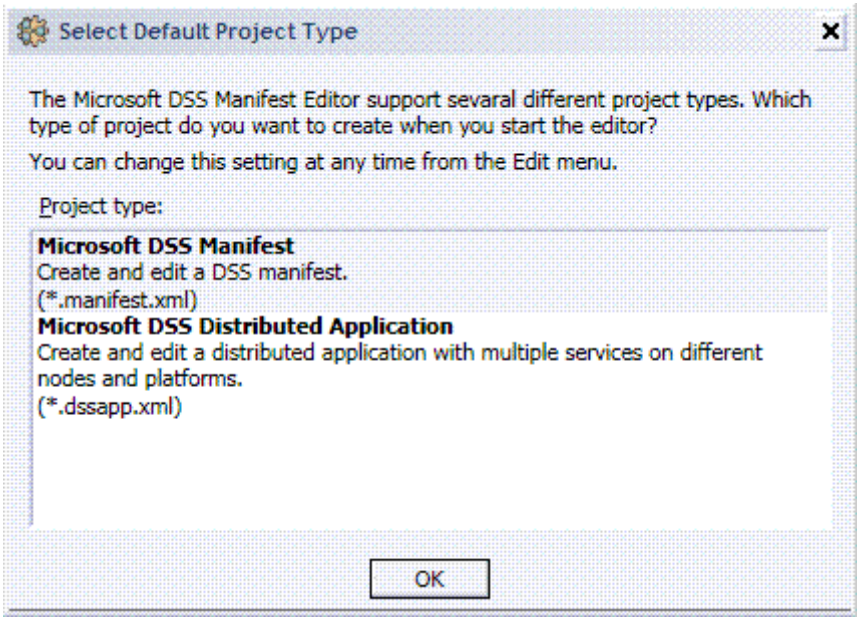
File SAMPLEFOURBYFOUR.MANIFEST.XML

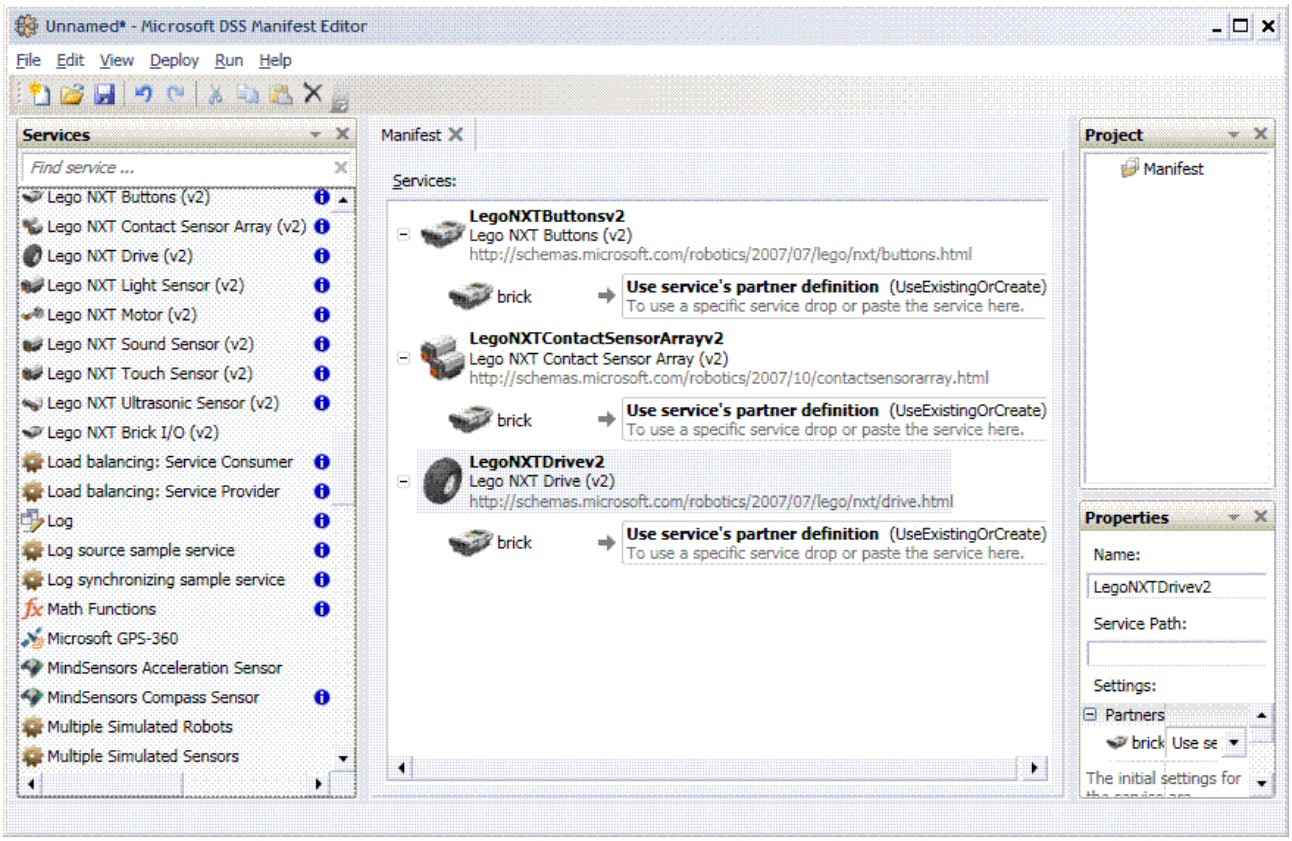
```
<Manifest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.microsoft.com/xw/2004/10/manifest.html">
  <CreateServiceList>
    <ServiceRecordType>
      <Contract
xmlns="http://schemas.microsoft.com/xw/2004/10/dssp.html">http://schemas.microsoft.co
m/robotics/2006/04/simulationengine.html</Contract>
      <AliasList xmlns="http://schemas.microsoft.com/xw/2004/10/dssp.html" />
      <PartnerList xmlns="http://schemas.microsoft.com/xw/2004/10/dssp.html">
        <Partner>
<Contract>http://schemas.microsoft.com/robotics/2006/04/simulationengine.html</Contrac
t>
          <Service>simulationengine (SampleFourByFour.Manifest).xml</Service>
          <AliasList />
```

```
        <PartnerList />
        <Name>StateService</Name>
      </Partner>
    </PartnerList>
    <Name xmlns:q1="urn:uuid:fe27ba25-2680-4932-958c-
2aae3bde3611">q1:SimulationEngine</Name>
  </ServiceRecordType>
  <ServiceRecordType>
    <Contract
xmlns="http://schemas.microsoft.com/xw/2004/10/dssp.html">http://schemas.microsoft.co
m/robotics/2007/10/simulatedfourbyfourdrive.html</Contract>
    <AliasList xmlns="http://schemas.microsoft.com/xw/2004/10/dssp.html" />
    <PartnerList xmlns="http://schemas.microsoft.com/xw/2004/10/dssp.html">
      <Partner>
        <Service>http://localhost/Sample4x4Vehicle</Service>
        <AliasList />
        <PartnerList />
        <Name
xmlns:q2="http://schemas.microsoft.com/robotics/2006/04/simulation.html">q2:Entity</Na
me>
      </Partner>
    </PartnerList>
    <Name xmlns:q3="urn:uuid:fe27ba25-2680-4932-958c-
2aae3bde3611">q3:SimulatedFourByFourDriveService</Name>
  </ServiceRecordType>
 </CreateServiceList>
</Manifest>
```

DSSME (*Decentralized Software Services Manifest Editor*)

Permette di creare, editare i file Manifest.
Fare clic sul menu ***File/Nuovo… (CTRL+N)*** per scegliere il tipo di progetto.





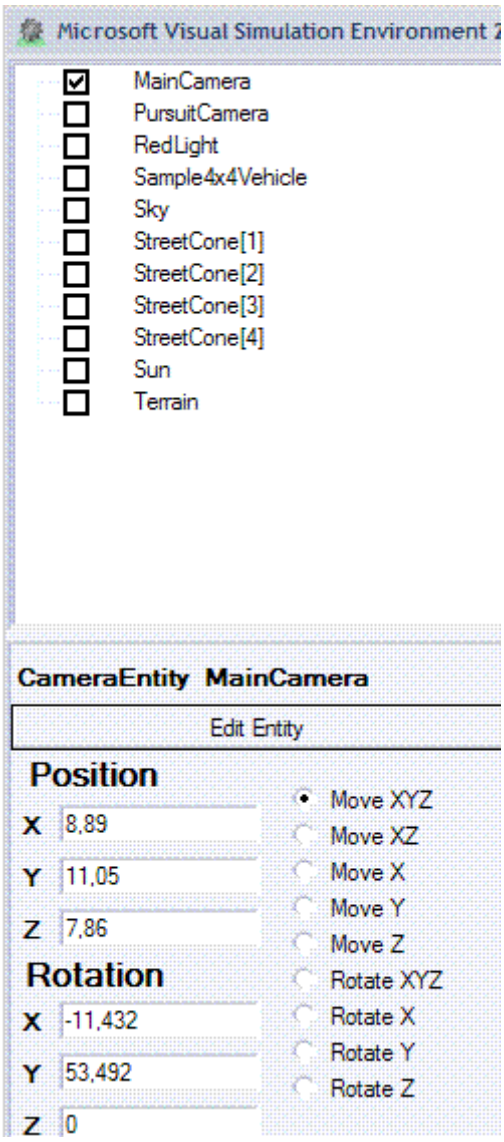
VSE (*VISUAL SIMULATION ENVIRONMENT*)

La tecnologia impiegata per il VSE è quella del Game Engine PhysX™ by Ageia Technology, è un software “a basso livello”, utilizzato nella programmazione dei videogiochi, che si occupa della grafica, delle leggi fisiche, delle animazioni e dell’intelligenza artificiale fornendo un modello coerente sul quale lo sviluppatore può lavorare, è programmato utilizzando l’ambiente di sviluppo XNA Game Studio.

Strumenti

Controllo tramite browser.
Scripting.
Windows Gadgets.
Linguaggi di programmazione: Microsoft Visual Studio C#, VB.NET, IronPython, Jscript.

Tra le funzionalità del simulatore è possibile personalizzare lo scenario delle simulazioni, è possibile aggiungere, posizionare o eliminare le figure geometriche che popolano lo scenario della simulazione, basta fare clic sul menu ***Mode/Edit (F5).***

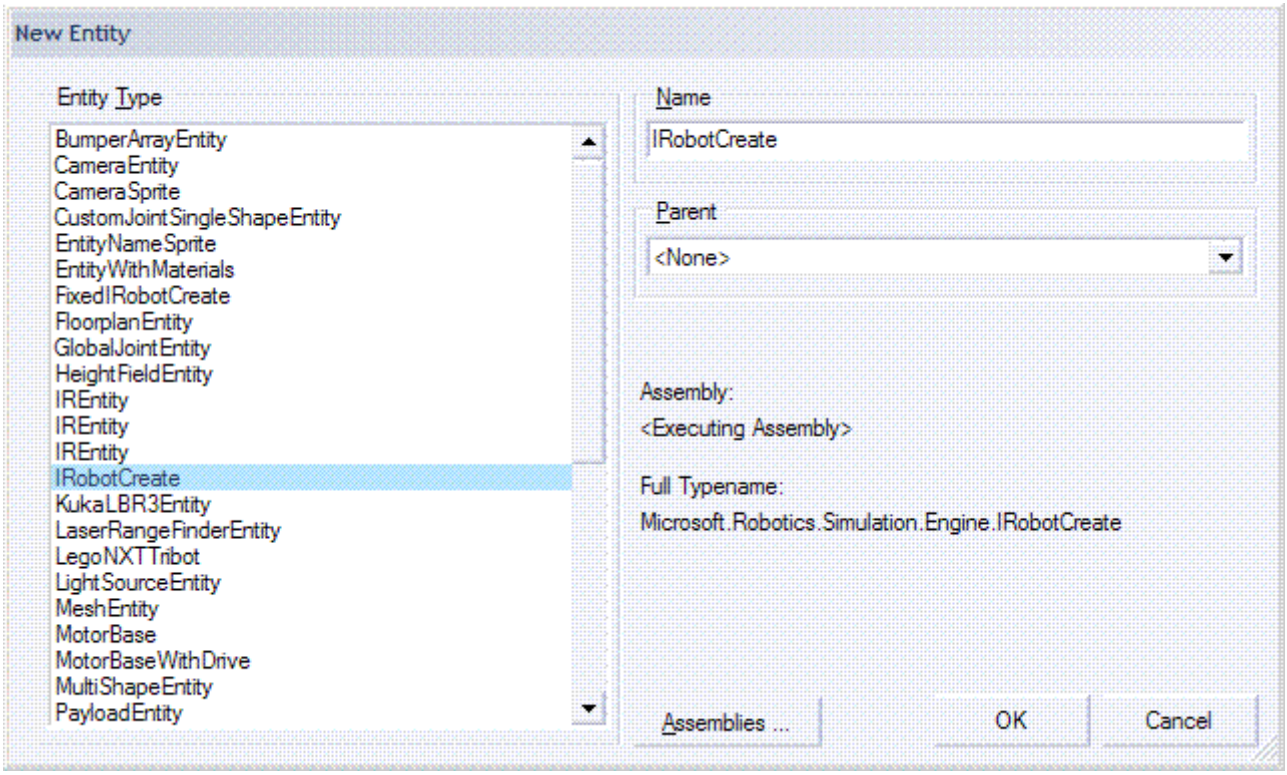


A sinistra della finestra del simulatore si va ad aggiungere un'area che enumera tutti gli elementi che popolano la simulazione, per esempio ***MainCamera*** consente di posizionare l'elemento selezionato secondo i tre piani cartesiani e anche di ruotarlo.
In questa finestra è possibile modificare le proprietà associate ad ogni entità, tali proprietà controllano ogni elemento, dal nome dell'entità alla relativa posizione nell'ambiente di simulazione.
Inoltre, consentono anche di controllare in modo preciso la modalità di rendering cui è sottoposta l'entità, influenzando la visualizzazione di tale entità nella simulazione.
Se si torna alla modalità ***Mode/Run***, è possibile spostare la simulazione utilizzando il mouse o i tasti freccia.
In questo modo si modifica il punto di vista per la fotocamera principale, che rappresenta l'occhio dell'utente nella simulazione.
VSE consente di eseguire il rendering della scena in modalità diverse.

- ✓ ***Render/Visual*** la modalità visiva è la predefinita ed è una vista realistica della scena di simulazione.
- ✓ ***Render/Wireframe.***
- ✓ ***Render/Physics.***
- ✓ ***Render/Combined.***

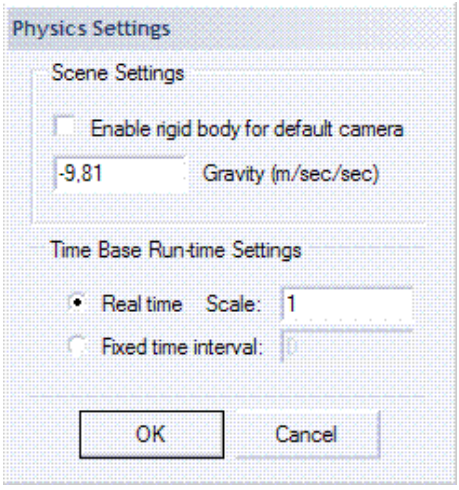
- ✓ ***Render/No Rendering*** è inclusa poiché il rendering è soltanto un aspetto di una simulazione, la caratteristica più preziosa dell'esecuzione di una simulazione è l'interazione fra varie entità, poiché le entità di rendering all'interno di una scena di simulazione sono costose in termini di risorse, questa opzione può essere utile quando sono coinvolte numerose entità.

Un tipo di entità consente di definire una nuova istanza di un particolare tipo di entità, in pratica agisce come modello per la nuova entità, nel senso che specifica proprietà associate ad un particolare tipo di entità.
I valori per tali proprietà possono essere modificati una volta creata l'entità, ma il tipo di entità definisce le proprietà contenute al suo interno.
Per aggiungere un'entità a una simulazione, è richiesto un tipo di entità.
Ad esempio, per aggiungere un robot come *Create* di iRobot, è necessario aggiungere una nuova entità facendo clic sul menu ***Entity/New…*** mentre è attiva la modalità ***Edit***.
In questo modo sarà visualizzata la finestra di dialogo ***New Entity***.

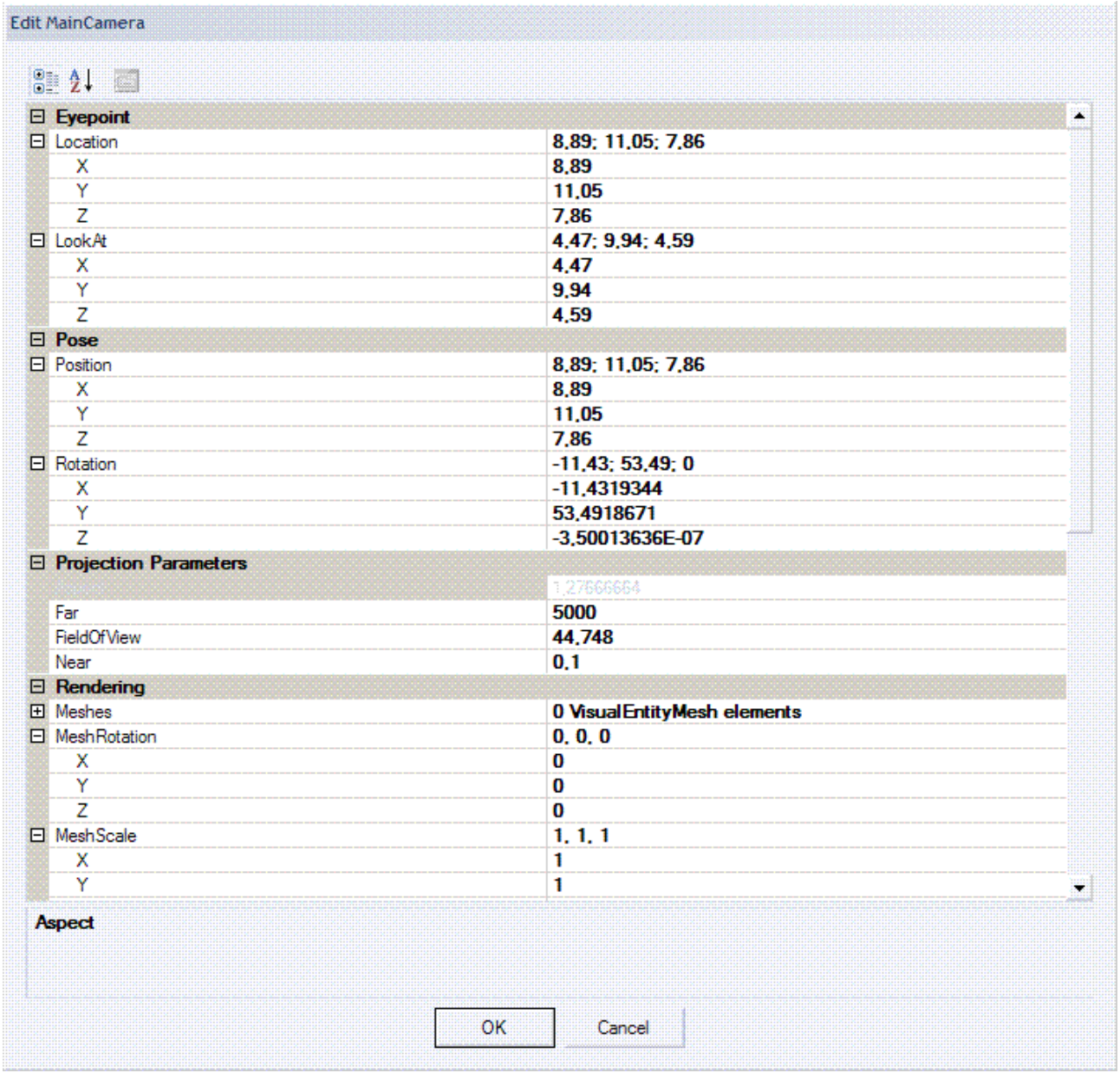


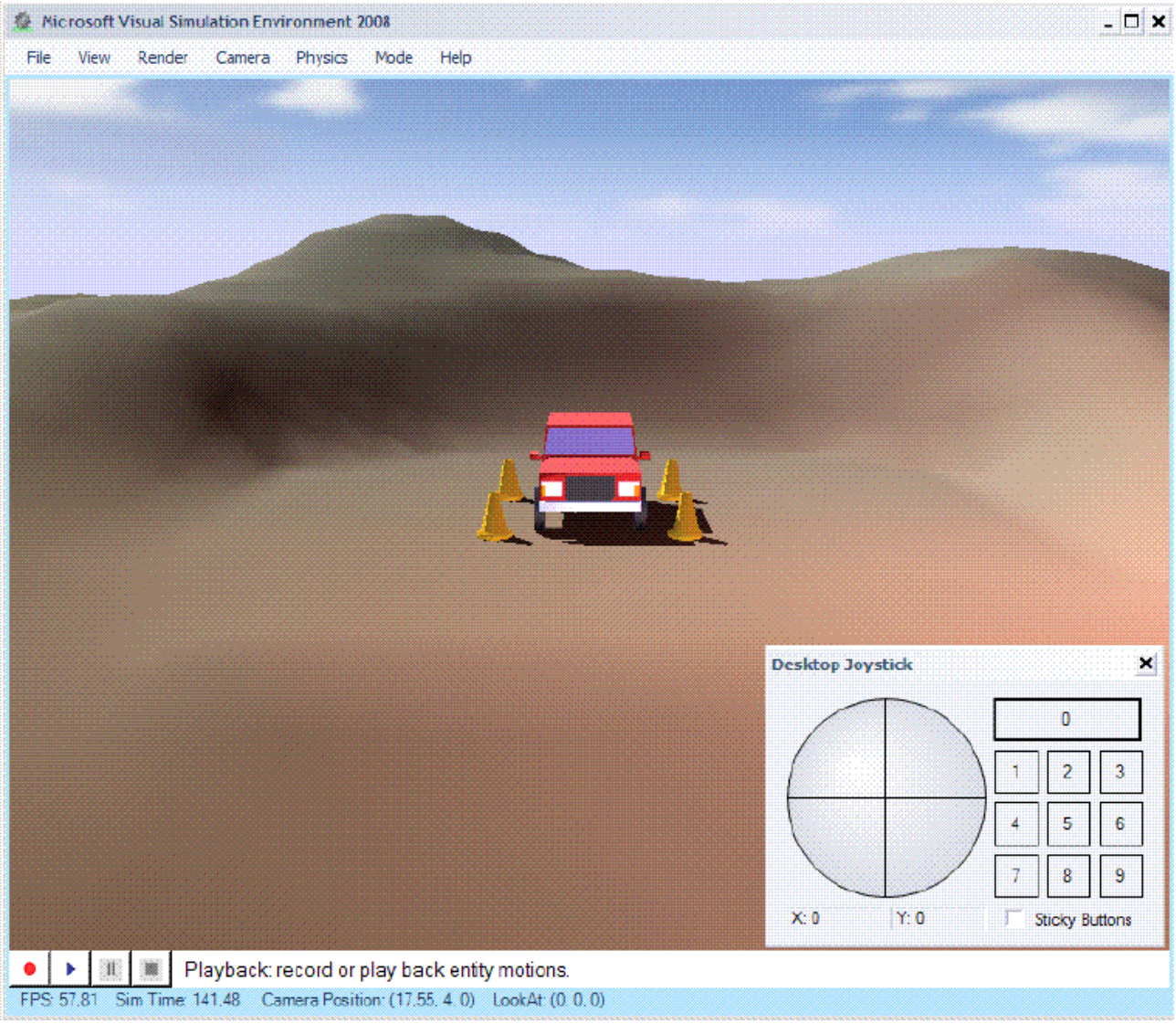
È possibile modificare gli elementi geometrici della simulazione e dal menu ***File*** è possibile salvare uno scenario Save ***Scene As…*** e caricarne uno con ***Open Scene…***

È possibile modificare la gravità della scena, un'opzione che si può rivelare utile qualora si dovesse progettare l'automazione per un robot esploratore su Marte, fare clic sul menu ***Physics/Settings…***



Fare clic sul pulsante ***Edit Entity***.





PROGRAMMAZIONE DEI **SERVICES**

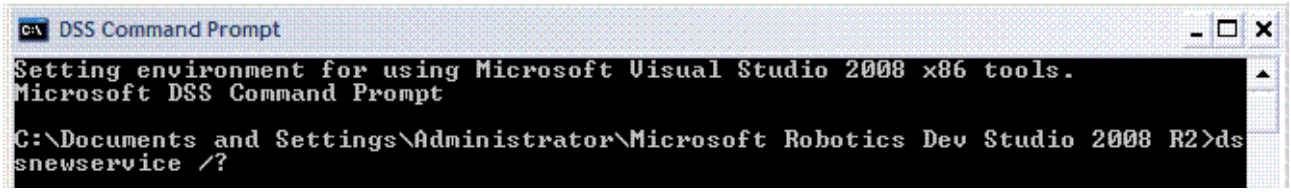
I ***Services*** sono stati creati utilizzando CCR e DSS, concettualmente sono simili ai servizi web, avviati monitorati e distrutti grazie ad un componente software il **DSSN** (*Decentralized Software Services Node*) che si occupa anche di farli comunicare attraverso un protocollo **SOAP** (*Simple Object Access Protocol*), basato su un modello di **REST** (*Representational State Transfer*) utilizzando **DSSP** (*Decentralized Software Services Protocol*) per la comunicazione tra i servizi.
In alternativa, è possibile inviare messaggi a un proxy che funge da interfaccia esterna del servizio, questo significa che servizi possono essere distribuiti in qualsiasi punto della rete.
L'utilizzo di un proxy ha due effetti.

1. I messaggi inviati tra i ***Services*** sono prima della trasmissione serializzati e dopo deserializzati all'altra estremità, XML è il formato per i dati trasmessi.
2. Il proxy definisce un contratto in modo efficace con il set di **API** (*Application Programming Interface*) esposto ad altri ***Services***.

Ogni ***Services*** dispone di uno stato interno, che è possibile recuperare o modificare mediante operazioni.
Questi implicano l'invio di un messaggio di richiesta e in attesa di un messaggio di risposta in un processo simile a quella utilizzata dalla maggior parte dei servizi web.
Quando è modificato lo stato del ***Services***, può inviare notifiche ai sottoscrittori.
Questo approccio pubblicazione e sottoscrizione rende i ***Services*** diversi dai servizi web tradizionali poiché i messaggi di notifica sono inviati in modo asincrono ai sottoscrittori.

Quando si genera un nuovo ***Services***, diventa automaticamente visibile in VPL e inizia a utilizzarlo immediatamente.

Orchestration
È la perfetta sincronia, dall'inizio alla fine, di tutti i ***Services*** che compongono l'automazione.
Nella simulazione non succede nulla se il robot urta un ostacolo per una disattenzione, mentre nella realtà gli errori casuali o le imprecisioni di una macchina utensile possono causare danni incalcolabili, per esempio la catena di montaggio di una fabbrica di automobili i cui robots, scoordinati, smaltano, saldano, assemblano pezzi che non sono ancora arrivati o sono sistemati male.
L'Orchestration è un problema assimilabile ai problemi classici di sincronizzazione dei threads nei sistemi operativi.
Per creare un ***Services*** utilizzando DSSN si deve lanciare l'Interfaccia a linea di comando, indicando il nome del servizio e il linguaggio di programmazione.



C:\Documents and Settings\Administrator\Microsoft Robotics Dev Studio 2008 R2>ds snewservice /?
Version 2.1.61.2
Copyright (c) Microsoft Corporation. All rights reserved.

/service:<string>	***The new service name. (short form /s)***
/assemblyname:<string>	*The name of the new service assembly (defaults to the new service name). (short form /a)*
/alt:<string>	*The alternate contracts to be implemented.*
/template:{<uri>|<path>}	*The Dss model template used to generate a new service. (short form /t)*
/dir:<string>	*Output directory for generated code (Default .\assemblyname) (short form /d)*
/force[+|-|:true|:false]	*Force overwriting the output directory and files (short form /f)*
/verbosity:<string>	*Display this amount of information. Available levels are: q[uiet], m[inimal], n[ormal], d[etailed], i[nfo]. Default value:'n' (short form /v)*
/org:<string>	*The prefix which will be used to construct a Dss Service Contract. ex: schemas.tempuri.org (short form /o)*
/namespace:<string>	*The CLR namespace which identifies your project or group and is unique for this service. ex: FabriKam.Robotics.HelloWorld (short form /n)*

/year:<int> *The 4 digit year when this project was created (default is current year) (short form /y)*

/month:<int> *The 2 digit month when this project was created (default is current month) (short form /m)*

/referencepath:<string> *Assembly reference path (short form /r)*

/language:{CSharp|VB|Cpp} ***Output language Default value:'CSharp' (short form /l)***

/documentation[+|-|:true|:false] *Generate an XML documentation file for this project. (short form /doc)*

/createCFproject[+|-|:true|:false] *Create a Compact Framework project. Default value:'-' (short form /ccfp)*

/generateCFproject:{<uri>|<path>} *Generate a Compact Framework project based on the project provided as input. (short form /gcfpf)*

/CompactFrameworkGlobalPropertyFile:{<uri>|<path>} *File containing properties directing which Compact Framework to target. (short form /cfgp)*

/ExtendProxy[+|-|:true|:false] *Allow proxy code extensions. This option is obsolete. Default value:'-' (short form /ext)*

/vstarget:{Default|VS2005|VS2008} *Target Visual Studio 2005 or 2008 Default value:'Default' (short form /vs)*

/UseSubscriptionManager[+|-|:true|:false] *Generated service supports subscritions and partners with a subscription manager. Default value:'-' (short form /SubMgr)*

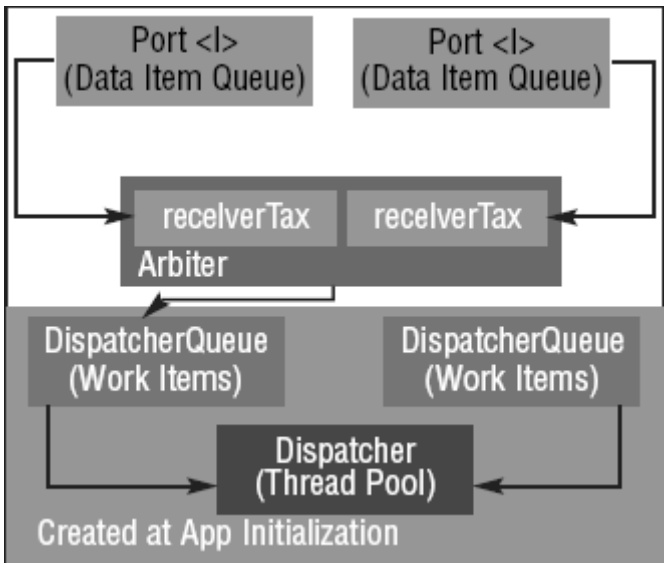
@<file> *Name of a text file containing arguments. Multiple arguments per line are allowed. An unquoted hash ('#') comments out the remainder of a line. An argument file is processed by removing comments, leading and trailing spaces and tabs, processing quotation marks and escape sequences, and then processing the arguments as if they were on the command line.*

CCR (*Concurrency and Coordination Runtime*)

È la libreria di costrutti di programmazione usati per coordinare le operazioni fra applicazioni, in pratica gestisce l'Orchestration, programmare un'applicazione con Microsoft Robotics significa gestire l'Orchestration tra gli input e gli output che si scambiano i ***Services***.

Se ad esempio si deve progettare un'intranet per una **LAN** (*Local Area Network*) composta da cento PC e si deve controllare se tutti sono collegati: si deve scrivere un

software che lancia un ping verso ogni singolo PC e ne attenda la risposta, gestendola poi in base alle esigenze dell'intranet.
Utilizzando la libreria CCR si deve solo dire che cosa fare, lanciare un ping e come gestire le risposte; ogni singolo ping è gestito dal CCR creando per ogni **IP** (*Internet Protocol*) un thread che utilizzerà la scheda di rete in più richieste parallele.
La libreria CCR è scritta in C#.



Classe *Port*
Il CCR definisce una classe di tipo *Port* che riceve in ingresso i parametri che dovranno essere elaborati tipizzandoli;
Il seguente codice istanzia una classe *Port* di nome *pF* che riceve in ingresso un parametro di tipo *float*, è possibile inserire in input 16 parametri di tipo diverso.

```
Port<float> pF = new Port<float>() ;
```

Il seguente codice passa due parametri di tipo *int* e *string*.

```
PortSet<int,string> pIS = new PortSet<int,string>();
```

La classe di tipo *Port* comprende una struttura di dati **FIFO** (*First In First Out*) il dato che entra per primo è il primo dato che è elaborato e restituito; i dati accodati nella porta sono poi "affidati" ad un arbitro che a sua volta chiama, facendo uso di delegate e metodi anonimi, una funzione da eseguire su questi dati; i dati sono elaborati chiamando un delegate, che è associato a un metodo denominato oppure un metodo anonimo che consentono di passare un blocco di codice come parametro del delegate.

Esempio.

```
// crea un gestore per un evento clic
button1.Click += delegate(System.Object o,System.EventArgs e)
{
System.Windows.Forms.MessageBox.Show("Clic!");
};
```

I metodi supportati dalla classe *Port* sono *Post* che invia un messaggio ad una porta.

```
Port<int> pi = new Port<int>();
pi.Post (42);
```

Test che verifica, attraverso un valore booleano di ritorno, se la porta è occupata.

```
int iv;
if (pi.Test (out iv))
   Console.WriteLine ("Leggi " + iv) ;
else
   Console.WriteLine ("Porta vuota.") ;
```

Classe *Arbiter*
Una volta che i messaggi sono passati attraverso la classe *Port* è necessario elaborarli e smistarli; questo compito è svolto dalla classe *Arbiter* che li elabora utilizzando un delegate, che chiama la funzione specifica e poi l'invia alla classe *Dispatcherqueue* che li smista.
Metodi della classe *Arbiter*.

```
public static class Arbiter {public static ITask FromHandler(Handler handler);
```

È un metodo che specifica che un valore può essere accodato verso una classe *DispatcherQueue*, questo semplice membro della classe *Arbiter* non è associato ad alcun membro della classe *Port*.

```
public static ITask FromIteratorHandler(Handler handler);
```

È un metodo che enumera diversi valori che possono essere accodati verso una classe *DispatcherQueue*, non è associato ad alcun membro della classe *Port*.

```
public static Receiver<T> Receive<T>(Boolean persist, Port<T> port, Handler<T>handler);
```

È un metodo che può essere chiamato per passare il risultato di un'elaborazione verso una singola porta, deve essere specificato sia il delegate che chiamerà l'evento sia la porta.

```
public static JoinSinglePortReceiver MultipleItemReceive<T>(Boolean persist, Port<T>
port, Int32 itemCount,VariableArgumentHandler<T> handler);
```

È un metodo che può essere chiamato per inviare diversi risultati dell'elaborazione verso una singola porta, il parametro inviato al delegate *VariableArgumentHandler* consiste in un array di oggetti *Port*.

```
public static JoinReceiver MultiplePortReceive<T>(Boolean persist, Port<T>[]
ports,VariableArgumentHandler<T> handler);
```

È un metodo che può essere chiamato per inviare un singolo risultato dell'elaborazione verso molteplici porte, gli item devono essere del medesimo tipo delle porte, il parametro inviato al delegate *VariableArgumentHandler* consiste in un array di oggetti *Port*.

Classe *DispatcherQueue*
È la coda delle richieste dell'elaborazione che la classe *Arbiter* desidera siano smistati dalla classe *Dispatcher*; i threads gestiti nella classe del *Dispatcher* sono in attesa degli input accodati nella classe *DispatcherQueue*.

```
public sealed class DispatcherQueue : IDisposable {
// usa il pool di thread gestito dal CLR non quello del Dispatcher più flessibile
public DispatcherQueue(string name);
```

```
public DispatcherQueue(String name, Dispatcher dispatcher);
public void Dispose();
... // altri membri
}
```

Classe *Dispatcher*

Il compito di sincronizzare i vari thread dell'elaborazione è affidato alla classe *Dispatcher*.
La classe *Arbiter* ha accodato alla *DispatcherQueue* delle funzioni da eseguire.
La classe *Dispatcher* trova nella queue che funzioni ci sono e le esegue attraverso il multithreading, in altre parole in parallelo.

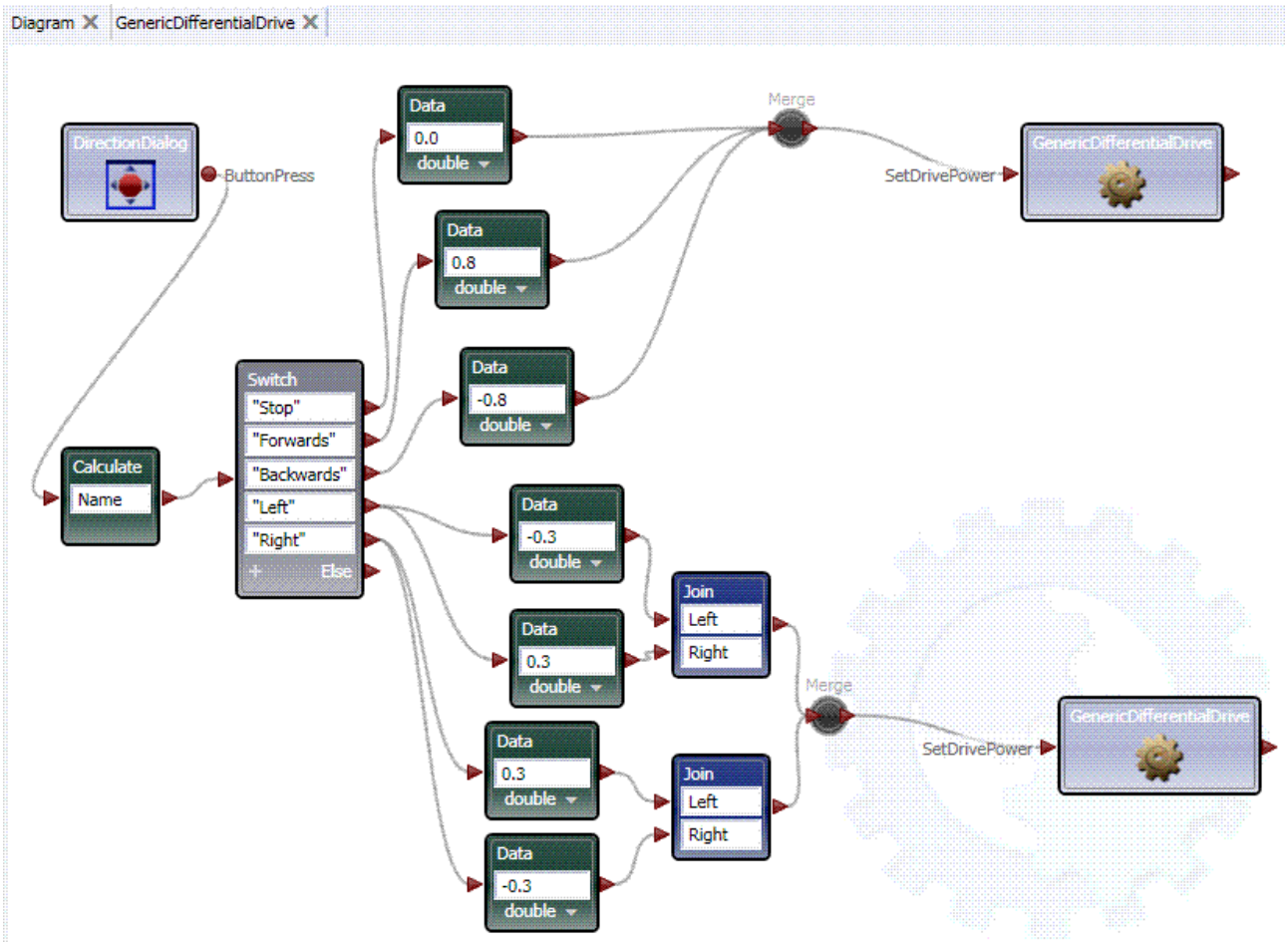
```
public sealed class Dispatcher : IDisposable {
public Dispatcher();
public Dispatcher(Int32 threadCount, String threadPoolName);
public Dispatcher(Int32 threadCount, ThreadPriority priority,
String threadPoolName);
public ICollection<DispatcherQueue>DispatcherQueues { get; }
... // altri membri
}
```

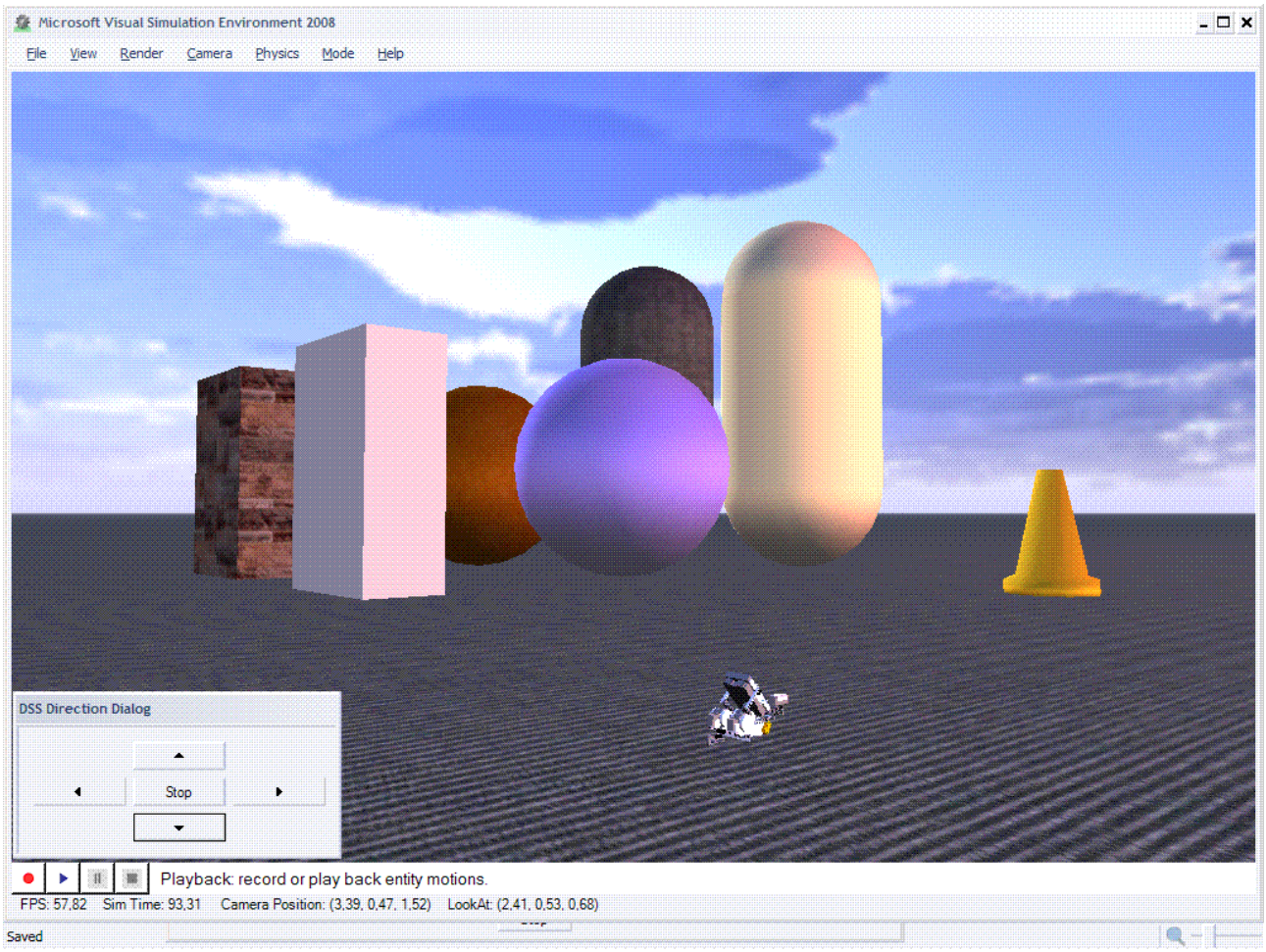
Per default ogni thread è indirizzato ad ogni singola CPU, è possibile identificare ogni singolo thread attraverso una stringa e settarne il livello di priorità per default, il Framework assegna ad ogni thread un nome che è poi utilizzato nel Visual Studio Debugger's Threads.
È possibile creare molteplici classi *Dispatcher*, ognuno con il suo pool di threads amplificando le possibilità di elaborazione dello stesso.

Il CCR è un componente difficile da programmare e gestire l'orchestrazione tra i vari ***Services*** è più difficile che costruire un robot.
Grazie a Robotics, non si deve scrivere una sola riga di codice; l'intera implementazione del CCR è gestita attraverso il VPL che genera tutto il codice del quale si ha bisogno.

PROGETTO

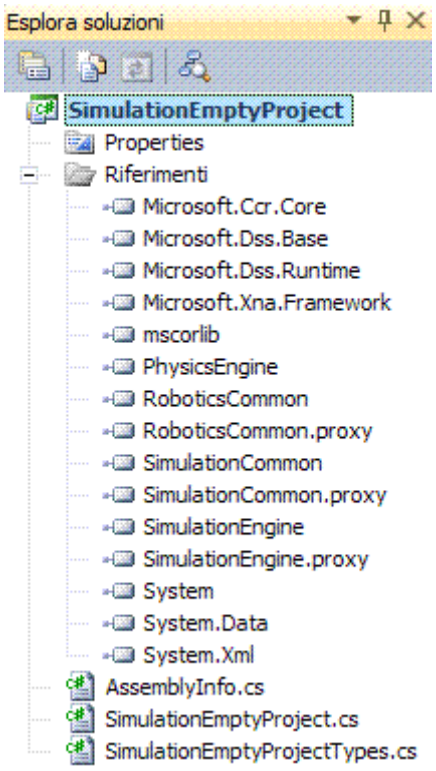
Automazione di un robot comandato da un joystick.
Durante la simulazione, se si preme il bottone Stop, le ruote non ricevono dati e si fermano se si spinge la leva avanti le ruote sono spinte in avanti grazie a dati di valore positivo se si spinge la leva indietro le ruote sono spinte indietro grazie a dati di valore negativo.
Le ruote sono fissate ad un asse e ruotano a sinistra ed a destra se una è spinta verso un valore positivo e l'altra è spinta verso un valore negativo equivalente o viceversa.
Il ***Services*** di tipo *Activity* incapsula tutto il flusso di automazione del diagramma.




Microsoft Visual Simulation Environment 2008
File View Render Camera Physics Mode Help
DSS Direction Dialog
Stop
Playback: record or play back entity motions.
FPS: 57,82 Sim Time: 93,31 Camera Position: (3,39, 0,47, 1,52) LookAt: (2,41, 0,53, 0,68)
Saved

MODELLO C#

Oltre a VSE, è possibile aggiungere le entità ad una simulazione creando un progetto di servizio DSS.
Modello da utilizzare per creare un nuovo servizio DSS.



La creazione di un nuovo servizio DSS utilizzando il modello comporterà la creazione di due file di classe.

FILE SIMULATIONEMPTYPROJECTTYPES.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using W3C.Soap;
namespace Robotics.SimulationEmptyProject
{
    /// <summary>
    /// SimulationEmptyProject contract class
    /// </summary>
    public sealed class Contract
    {
        /// <summary>
        /// DSS contract identifer for SimulationEmptyProject
        /// </summary>
```

```
        [DataMember]
        public const string Identifier =
"http://schemas.microsoft.com/2009/03/simulationemptyproject.user.html";
    }
    /// <summary>
    /// SimulationEmptyProject state
    /// </summary>
    [DataContract]
    public class SimulationEmptyProjectState
    {
    }
    /// <summary>
    /// SimulationEmptyProject main operations port
    /// </summary>
    [ServicePort]
    public class SimulationEmptyProjectOperations : PortSet<DsspDefaultLookup,
DsspDefaultDrop, Get, Subscribe>
    {
    }
    /// <summary>
    /// SimulationEmptyProject get operation
    /// </summary>
    public class Get : Get<GetRequestType, PortSet<SimulationEmptyProjectState, Fault>>
    {
        /// <summary>
        /// Creates a new instance of Get
        /// </summary>
        public Get()
        {
        }
        /// <summary>
        /// Creates a new instance of Get
        /// </summary>
        /// <param name="body">the request message body</param>
        public Get(GetRequestType body)
            : base(body)
        {
        }
        /// <summary>
        /// Creates a new instance of Get
        /// </summary>
        /// <param name="body">the request message body</param>
        /// <param name="responsePort">the response port for the request</param>
        public Get(GetRequestType body, PortSet<SimulationEmptyProjectState, Fault>
responsePort)
            : base(body, responsePort)
        {
        }
    }
    /// <summary>
    /// SimulationEmptyProject subscribe operation
    /// </summary>
    public class Subscribe : Subscribe<SubscribeRequestType,
PortSet<SubscribeResponseType, Fault>>
```

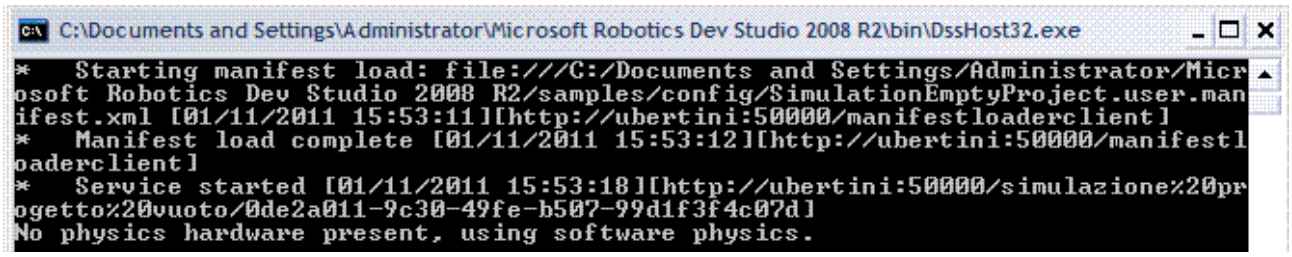
```
    {
        /// <summary>
        /// Creates a new instance of Subscribe
        /// </summary>
        public Subscribe()
        {
        }
        /// <summary>
        /// Creates a new instance of Subscribe
        /// </summary>
        /// <param name="body">the request message body</param>
        public Subscribe(SubscribeRequestType body)
            : base(body)
        {
        }
        /// <summary>
        /// Creates a new instance of Subscribe
        /// </summary>
        /// <param name="body">the request message body</param>
        /// <param name="responsePort">the response port for the request</param>
        public Subscribe(SubscribeRequestType body, PortSet<SubscribeResponseType,
Fault> responsePort)
            : base(body, responsePort)
        {
        }
    }
}
```

La classe d'implementazione, che per impostazione predefinita ha lo stesso nome del progetto, è quella in cui aggiungere il codice per creare una nuova entità.
File SIMULATIONEMPTYPROJECT.CS

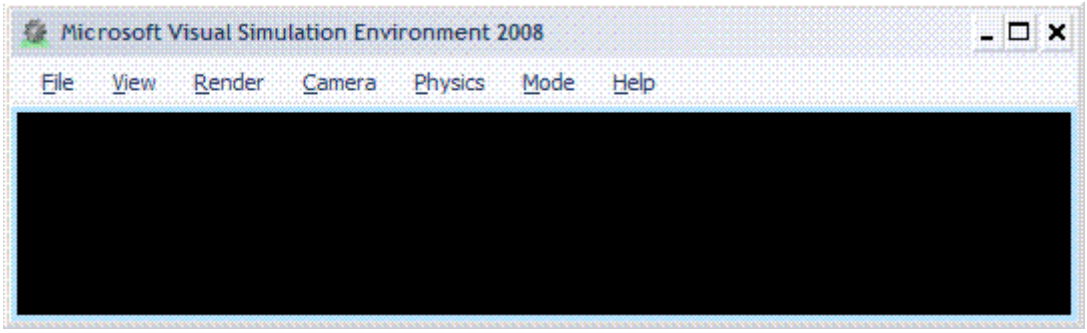
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using W3C.Soap;
using submgr = Microsoft.Dss.Services.SubscriptionManager;
using engine = Microsoft.Robotics.Simulation.Engine.Proxy;
namespace Robotics.SimulationEmptyProject
{
    [Contract(Contract.Identifier)]
    [DisplayName("(User) Simulazione Progetto vuoto")]
    [Description("Minimal base template for developing a service that uses the simulation
engine")]
    class SimulationEmptyProjectService : DsspServiceBase
    {
        /// <summary>
        /// Service state
        /// </summary>
        [ServiceState]
        SimulationEmptyProjectState _state = new SimulationEmptyProjectState();
```

```
        /// <summary>
        /// Main service port
        /// </summary>
        [ServicePort("/Simulazione Progetto vuoto", AllowMultipleInstances = true)]
        SimulationEmptyProjectOperations _mainPort = new
SimulationEmptyProjectOperations();
        [SubscriptionManagerPartner]
        submgr.SubscriptionManagerPort _submgrPort = new
submgr.SubscriptionManagerPort();
        /// <summary>
        /// SimulationEngine partner
        /// </summary>
        [Partner("SimulationEngine", Contract = engine.Contract.Identifier, CreationPolicy =
PartnerCreationPolicy.UseExistingOrCreate)]
        engine.SimulationEnginePort _simulationEnginePort = new
engine.SimulationEnginePort();
        engine.SimulationEnginePort _simulationEngineNotify = new
engine.SimulationEnginePort();
        /// <summary>
        /// Service constructor
        /// </summary>
        public SimulationEmptyProjectService(DsspServiceCreationPort creationPort)
            : base(creationPort)
        {
        }
        /// <summary>
        /// Service start
        /// </summary>
        protected override void Start()
        {
            //
            // Add service specific initialization here
            //
            base.Start();
        }
        /// <summary>
        /// Handles Subscribe messages
        /// </summary>
        /// <param name="subscribe">the subscribe request</param>
        [ServiceHandler]
        public void SubscribeHandler(Subscribe subscribe)
        {
            SubscribeHelper(_submgrPort, subscribe.Body, subscribe.ResponsePort);
        }
    }
}
```

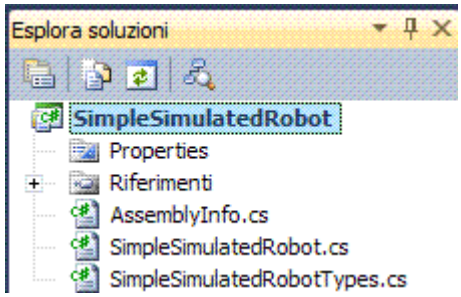
Per eseguire l'applicazione, fare clic sulla barra degli strumenti ***Avvia debug (F5***).
All'inizio, è visualizzata una finestra del prompt dei comandi.



Una volta caricato il servizio, è visualizzata la finestra VSE.



APPLICAZIONE C#



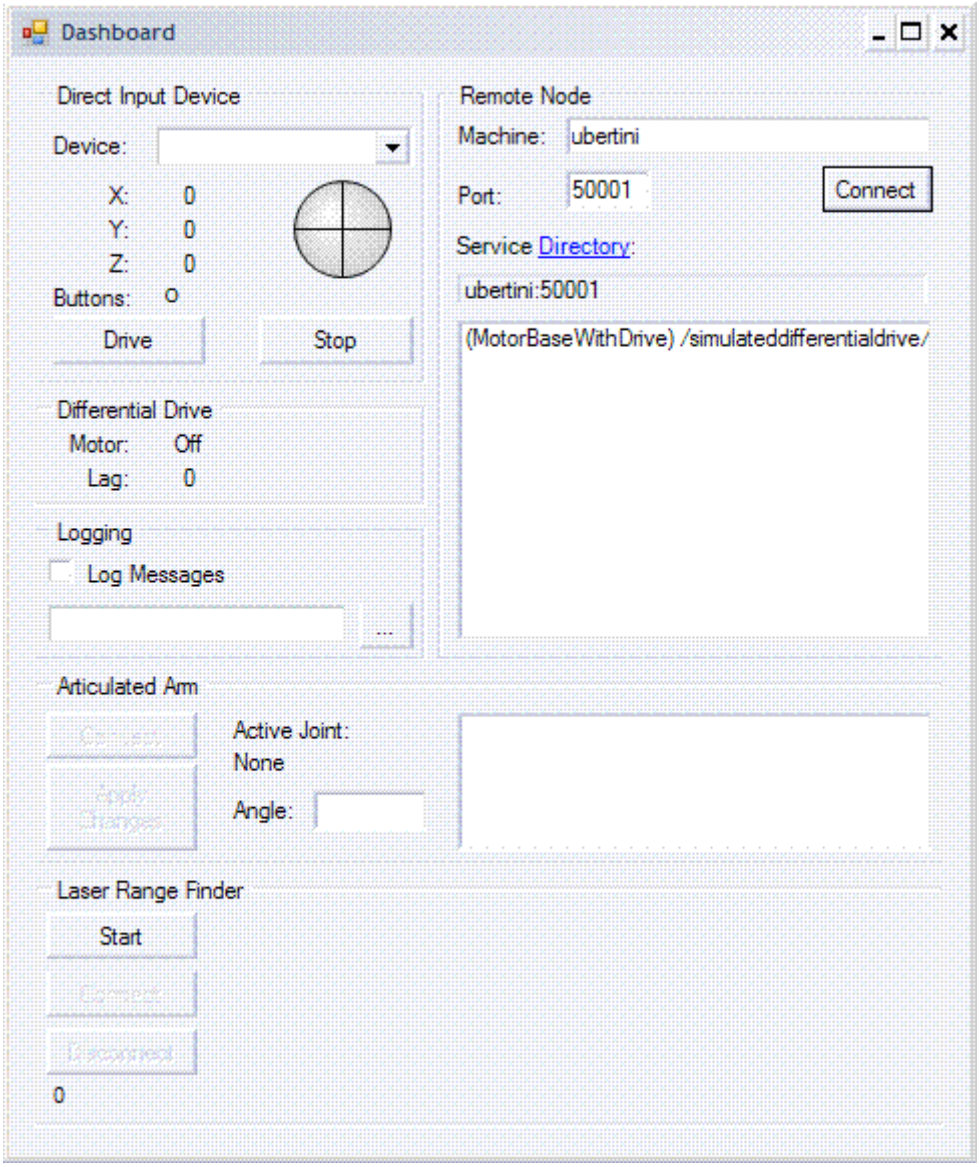
File SIMPLESIMULATEDROBOT.CS

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using Microsoft.Ccr.Core;
using Microsoft.Dss.Core.Attributes;
using Microsoft.Dss.ServiceModel.Dssp;
using Microsoft.Dss.ServiceModel.DsspServiceBase;
using W3C.Soap;
using submgr = Microsoft.Dss.Services.SubscriptionManager;
using engine = Microsoft.Robotics.Simulation.Engine.Proxy;
using Microsoft.Robotics.Simulation.Engine;
using Microsoft.Robotics.PhysicalModel;
using xna = Microsoft.Xna.Framework.Graphics;
using Microsoft.Robotics.Simulation.Physics;
using drive = Microsoft.Robotics.Services.Simulation.Drive.Proxy;
using simwebcam =
Microsoft.Robotics.Services.Simulation.Sensors.SimulatedWebcam.Proxy;
using System.Runtime.InteropServices;
namespace Robotics.SimpleSimulatedRobot
```

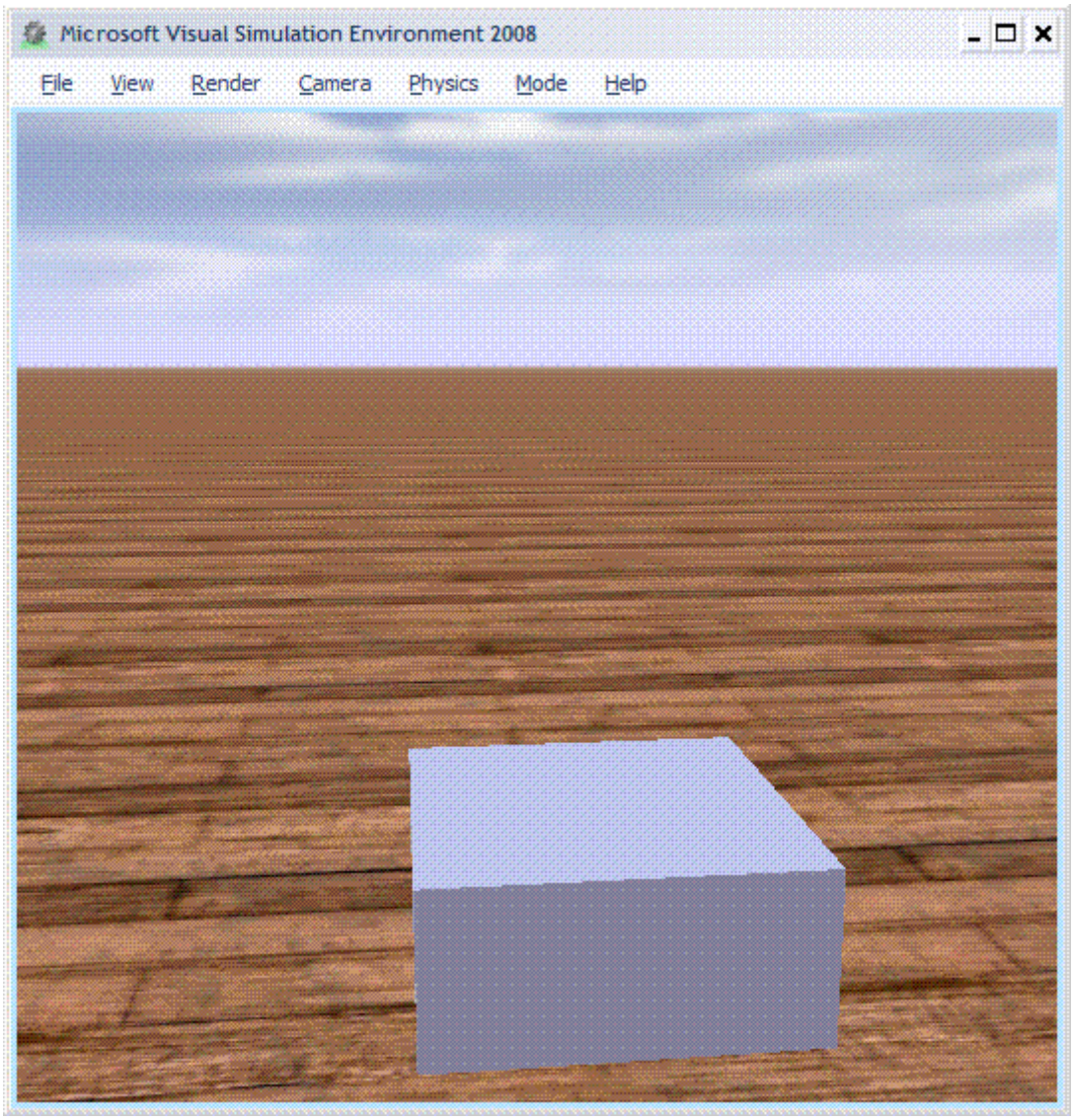
```
{
    #region Basic Simulation Service Template
   #region Motor Base
   /// <summary>
    /// MotorBaseWithSensors is an implementation of the differential drive entity
    /// that automatically starts the simulated differential drive service
    /// </summary>
    [DataContract]
    public class MotorBaseWithDrive : MotorBase
    {
        public MotorBaseWithDrive() { }
        public MotorBaseWithDrive(Vector3 position)
            : base(position)
        {
        }
        public override void Initialize(xna.GraphicsDevice device, PhysicsEngine
physicsEngine)
        {
            base.ServiceContract = drive.Contract.Identifier;
            base.Initialize(device, physicsEngine);
        }
    }
    /// <summary>
    /// SimulatedWebcamEntity is a CameraEntity that automatically starts the simulated
    /// web cam service
    /// </summary>
    [DataContract]
    public class SimulatedWebcamEntity : CameraEntity
    {
        public SimulatedWebcamEntity() { }
        /// <summary>
        /// Defaults to a 320x240 web cam with a 90 degree FOV
        /// </summary>
        /// <param name="viewSizeX"></param>
        /// <param name="viewSizeY"></param>
        /// <param name="halfViewAngle"></param>
        public SimulatedWebcamEntity(
            Vector3 initialPosition,
            [DefaultParameterValue(320)] int viewSizeX,
            [DefaultParameterValue(240)] int viewSizeY,
            [DefaultParameterValue((float)Math.PI / 4.0f)] float halfViewAngle
            )
            : base(viewSizeX, viewSizeY, halfViewAngle)
        {
            State.Pose.Position = initialPosition;
        }
        public override void Initialize(xna.GraphicsDevice device, PhysicsEngine
physicsEngine)
        {
            ServiceContract = simwebcam.Contract.Identifier;
            IsRealTimeCamera = true;
            base.Initialize(device, physicsEngine);
        }
    }
```

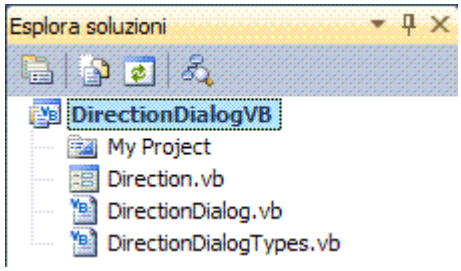
}

Per eseguire l'applicazione, fare clic sulla barra degli strumenti ***Avvia debug (F5***).
All'inizio, è visualizzata una finestra del prompt dei comandi.
Una volta caricato il servizio, sono visualizzate altre due finestre.
La finestra ***Dashboard*** è un modulo che include diverse caselle di gruppo, il lato destro del modulo contiene una casella di gruppo denominata ***Remote Node***, inserire il nome del PC, localhost, nella casella di testo ***Machine:*** e fare clic su ***Connect***.
Sarà avviata una connessione con il servizio *MotorBaseWithDrive* e saranno caricati i servizi partner di entità.



Il servizio di trasmissione è utilizzato per inviare comandi al robot, questo servizio consente di muovere il robot nella simulazione utilizzando il controllo del joystick virtuale o un joystick reale collegato al PC.
Per iniziare, si deve fare doppio clic sull'elemento *simulateddifferentialdrive* visualizzato nella casella di elenco, nella casella ***Differential Drive***, ***Motor: On***.
Si usa il mouse per trascinare il controllo joystick virtuale situato nell'angolo superiore sinistro della finestra.


Microsoft Visual Simulation Environment 2008
File
View
Render
Camera
Physics
Mode
Help



FILE DIRECTIONDIALOGTYPES.VB

```
Imports Microsoft.Ccr.Core
Imports Microsoft.Dss.ServiceModel.Dssp
Imports Microsoft.Dss.Core.Attributes
Imports System
Imports System.Collections.Generic
Imports W3C.Soap
Imports System.ComponentModel
Imports dssp = Microsoft.Dss.ServiceModel.Dssp
<Assembly: ServiceDeclaration(DssServiceDeclaration.ServiceBehavior)>
Namespace DirectionDialogVB
    Public Class Contract
        Public Const Identifier As String =
"http://schemas.microsoft.com/robotics/2006/10/vbdirectiondialog.user.html"
    End Class
    <DataContract()> _
    Public Class DirectionDialogState
        Private _buttons As List(Of Button) = New List(Of Button)()
        <DataMember(IsRequired:=True)> _
        Public Property Buttons() As List(Of Button)
            Get
                Return _buttons
            End Get
            Set(ByVal value As List(Of Button))
                _buttons = value
            End Set
        End Property
    End Class
    <DataContract()> _
    Public Class Button
        Private _name As String
        Private _pressed As Boolean
        Public Sub New()
        End Sub
        Public Sub New(ByVal name As String)
            _name = name
        End Sub
        <DataMember()> _
        Public Property Name() As String
            Get
                Return _name
            End Get
            Set(ByVal value As String)
```

```
            _name = value
        End Set
    End Property
    <DataMember()> _
    Public Property Pressed() As Boolean
        Get
            Return _pressed
        End Get
        Set(ByVal value As Boolean)
            _pressed = value
        End Set
    End Property
End Class
<DataContract()> _
Public Class ButtonPressRequest
    Private _name As String
    Private _pressed As Boolean
    Public Sub New()
    End Sub
    Public Sub New(ByVal name As String)
        _name = name
    End Sub
    <DataMember()> _
    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)
            _name = value
        End Set
    End Property
End Class
<DataContract()> _
Public Class ButtonReleaseRequest
    Private _name As String
    Private _pressed As Boolean
    Public Sub New()
    End Sub
    Public Sub New(ByVal name As String)
        _name = name
    End Sub
    <DataMember()> _
    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal value As String)
            _name = value
        End Set
    End Property
End Class
Public Class DirectionDialogOperations
    Inherits PortSet(Of DsspDefaultLookup, DsspDefaultDrop, GetOperation, Replace,
ButtonPress, ButtonRelease, Subscribe)
```

```
    End Class
    <DisplayName("Get")> _
    <Description("Gets the current state of the dialog as a list of the buttons on the dialog
and their current pressed states.")> _
    Public Class GetOperation
        Inherits dssp.Get(Of GetRequestType, PortSet(Of DirectionDialogState, Fault))
    End Class
    <DisplayName("DialogStateChange")> _
    <Description("Indicates when the dialog state changes.")> _
    Public Class Replace
        Inherits dssp.Replace(Of DirectionDialogState, PortSet(Of
DefaultReplaceResponseType, Fault))
    End Class
    <DisplayName("ButtonPress")> _
    <Description("Indicates when a button in the dialog is pressed.")> _
    Public Class ButtonPress
        Inherits dssp.Update(Of ButtonPressRequest, PortSet(Of
DefaultUpdateResponseType, Fault))
        Public Sub New()
        End Sub
        Public Sub New(ByVal body As ButtonPressRequest)
            MyBase.New(body)
        End Sub
    End Class
    <DisplayName("ButtonRelease")> _
    <Description("Indicates when a button in the dialog is released.")> _
    Public Class ButtonRelease
        Inherits dssp.Update(Of ButtonReleaseRequest, PortSet(Of
DefaultUpdateResponseType, Fault))
        Public Sub New()
        End Sub
        Public Sub New(ByVal body As ButtonReleaseRequest)
            MyBase.New(body)
        End Sub
    End Class
    Public Class Subscribe
        Inherits dssp.Subscribe(Of SubscribeRequestType, PortSet(Of
SubscribeResponseType, Fault))
    End Class
End Namespace
```

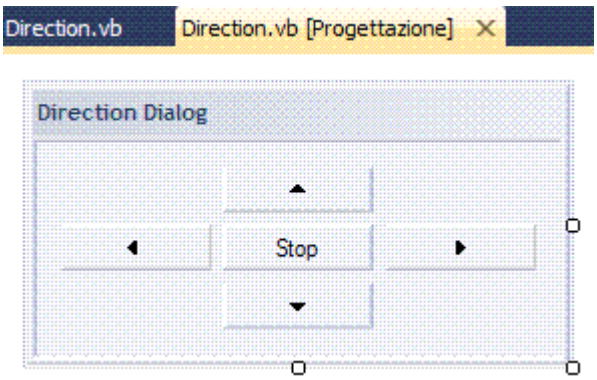
FILE DIRECTIONDIALOG.VB

```
Imports System
Imports System.ComponentModel
Imports Microsoft.Ccr.Core
Imports Microsoft.Dss.Core
Imports Microsoft.Dss.Core.Attributes
Imports Microsoft.Dss.ServiceModel.Dssp
Imports Microsoft.Dss.ServiceModel.DsspServiceBase
Imports System.Collections.Generic
Imports System.Security.Permissions
Imports W3C.Soap
Imports sm = Microsoft.Dss.Services.SubscriptionManager
Imports Microsoft.Ccr.Adapters.WinForms
```

```
Imports soap = W3C.Soap
Imports dssp = Microsoft.Dss.ServiceModel.Dssp
Namespace DirectionDialogVB
    <DisplayName("Direction Dialog (VB)")> _
    <Description("Displays a Windows dialog (compiled with Visual Basic.NET) with 5
buttons on it for direction control.")> _
    <Contract(Contract.Identifier)> _
    Public Class DirectionDialogService
        Inherits DsspServiceBase
        Private _state As DirectionDialogState = New DirectionDialogState()
        <ServicePort("/directiondialog", AllowMultipleInstances:=True)> _
        Private _mainPort As DirectionDialogOperations = New DirectionDialogOperations()
        <Partner("SubMgr", Contract:=sm.Contract.Identifier,
CreationPolicy:=PartnerCreationPolicy.CreateAlways, Optional:=False)> _
        Private _submgr As sm.SubscriptionManagerPort = New
sm.SubscriptionManagerPort()
        Public Sub New(ByVal creationPort As DsspServiceCreationPort)
            MyBase.New(creationPort)
        End Sub
        Protected Overrides Sub Start()
            _state.Buttons.Add(New Button("Forwards"))
            _state.Buttons.Add(New Button("Backwards"))
            _state.Buttons.Add(New Button("Left"))
            _state.Buttons.Add(New Button("Right"))
            _state.Buttons.Add(New Button("Stop"))
            ' Listen on the main port for requests and call the appropriate handler.
            'ActivateDsspOperationHandlers()
            Activate( _
                Arbiter.Interleave(New TeardownReceiverGroup( _
                    Arbiter.Receive(Of DsspDefaultDrop)(False, _mainPort, AddressOf
DefaultDropHandler) _
                ) _
                , New ExclusiveReceiverGroup( _
                Arbiter.Receive(Of Replace)(True, _mainPort, AddressOf ReplaceHandler) _
                    , Arbiter.Receive(Of ButtonPress)(True, _mainPort, AddressOf
ButtonPressHandler) _
                    , Arbiter.Receive(Of ButtonRelease)(True, _mainPort, AddressOf
ButtonReleaseHandler) _
                ) , New ConcurrentReceiverGroup( _
                    Arbiter.Receive(Of DsspDefaultLookup)(True, _mainPort, AddressOf
DefaultLookupHandler) _
                    , Arbiter.Receive(Of GetOperation)(True, _mainPort, AddressOf
GetHandler), Arbiter.Receive(Of Subscribe)(True, _mainPort, AddressOf
SubscribeHandler)  ) )  )
            ' Publish the service to the local service Directory
            DirectoryInsert()
            WinFormsServicePort.Post(New RunForm(AddressOf CreateWinForm))
            ' display HTTP service Uri
            LogInfo(LogGroups.Console, "Service uri: ")
        End Sub
        Function CreateWinForm() As System.Windows.Forms.Form
            Return New Direction(ServiceForwarder(Of
DirectionDialogOperations)(ServiceInfo.Service))
        End Function
```

```
        Sub GetHandler(ByVal getrequest As GetOperation)
            getrequest.ResponsePort.Post(_state)
        End Sub
        Sub ReplaceHandler(ByVal replace As Replace)
            For Each curr As Button In replace.Body.Buttons
                For Each button As Button In _state.Buttons
                    If button.Name = curr.Name Then
                        button.Pressed = curr.Pressed
                    End If
                Next
            Next
            replace.ResponsePort.Post(DefaultReplaceResponseType.Instance)
            SendNotification(Of Replace)(_submgr, _state)
        End Sub
        Sub ButtonPressHandler(ByVal buttonPressed As ButtonPress)
            Dim button As Button = Nothing
            For Each test As Button In _state.Buttons
                If test.Name = buttonPressed.Body.Name Then
                    button = test
                End If
            Next
            If Object.ReferenceEquals(button, Nothing) Then
                buttonPressed.ResponsePort.Post(Fault.FromCodeSubcodeReason( _
                    soap.FaultCodes.Receiver, DsspFaultCodes.UnknownEntry, _
                    "No such button: " + buttonPressed.Body.Name ) )
            Else
                button.Pressed = True
                buttonPressed.ResponsePort.Post(DefaultUpdateResponseType.Instance)
                SendNotification(_submgr, buttonPressed)
            End If
        End Sub
        Sub ButtonReleaseHandler(ByVal buttonReleased As ButtonRelease)
            Dim button As Button = Nothing
            For Each test As Button In _state.Buttons
                If test.Name = buttonReleased.Body.Name Then
                    button = test
                End If
            Next
            If Object.ReferenceEquals(button, Nothing) Then
                buttonReleased.ResponsePort.Post(Fault.FromCodeSubcodeReason( _
                    soap.FaultCodes.Receiver, DsspFaultCodes.UnknownEntry, _
                    "No such button: " + buttonReleased.Body.Name ) )
            Else
                button.Pressed = False
                buttonReleased.ResponsePort.Post(DefaultUpdateResponseType.Instance)
                SendNotification(_submgr, buttonReleased)
            End If
        End Sub
        Sub SubscribeHandler(ByVal subscribe As Subscribe)
            SubscribeHelper(_submgr, subscribe.Body, subscribe.ResponsePort)
        End Sub
    End Class
End Namespace
```

FILE DIRECTION.VB




```
Imports System.Windows.Forms
Imports DirectionDialogVB.DirectionDialogVB
Imports winforms = System.Windows.Forms
Public Class Direction
    Private _mainPort As DirectionDialogOperations
    Public Sub New(ByVal mainPort As DirectionDialogOperations)
        _mainPort = mainPort
        InitializeComponent()
    End Sub
    Private Sub button_MouseDown(ByVal sender As System.Object, ByVal e As
MouseEventArgs) Handles btnStop.MouseDown, btnRight.MouseDown,
btnLeft.MouseDown, btnForwards.MouseDown, btnBackwards.MouseDown
        If TypeOf sender Is winforms.Button Then
            Dim button As winforms.Button = CType(sender, winforms.Button)
            If button.Name.StartsWith("btn") Then
                Dim name As String = button.Name.Substring(3)
                _mainPort.Post(New ButtonPress(New ButtonPressRequest(name)))
            End If
        End If
    End Sub
    Private Sub button_MouseUp(ByVal sender As System.Object, ByVal e As
MouseEventArgs) Handles btnStop.MouseUp, btnRight.MouseUp, btnLeft.MouseUp,
btnForwards.MouseUp, btnBackwards.MouseUp
        If TypeOf sender Is winforms.Button Then
            Dim button As winforms.Button = CType(sender, winforms.Button)
            If button.Name.StartsWith("btn") Then
                Dim name As String = button.Name.Substring(3)
                _mainPort.Post(New ButtonRelease(New ButtonReleaseRequest(name)))
            End If
        End If
    End Sub
End Class
```

MODULO 7

LIGHTSWITCH

RIA
LightSwitch

RIA (*RICH INTERNET APPLICATION*)

INTRODUZIONE

In Internet c'è sempre più bisogno di servizi applicativi di tipo classico, come i gestionali, la logistica e la produzione, rivisti però in un'ottica web.
Per rispondere a queste esigenze si sono sviluppate nel tempo.

- ✓ Applicazioni basate sulle architetture web.
- ✓ Tecnologie **AJAX** (*Asynchronous JavaScript and **XML** [eXtensible Markup Language]).*

Ma non si è riusciti a cambiare la percezione degli utenti finali: l'usabilità e l'interattività era migliore quando si usavano le versioni client/server.
Oggi si parla del nuovo modello applicativo: RIA, applicazioni web che possiedono usabilità e interattività pari o migliori alle tradizionali applicazioni desktop per PC.

STRUMENTI

Sono disponibili diverse tecnologie e ambienti di sviluppo.

- ✓ Adobe Flex.
- ✓ Microsoft Silverlight.

Sono punti di riferimento per realizzare RIA attraverso virtual machine grafiche che dovrebbero sostituire i browser, ritenuti datati e incompatibili fra loro.
La difficoltà nell'utilizzo di questi strumenti non è solo quella di dover apprendere nuovi linguaggi e nuove architetture, ma soprattutto quella di dover suddividere la logica applicativa in due contesti separati.

1. **Lato server** tanti servizi applicativi che svolgono operazioni atomiche.
2. **Lato client** la logica di controllo dell'interfaccia utente che utilizzerà questi servizi.

Questo rende più difficile la realizzazione di applicazioni sicure e il controllo dell'applicazione stessa, perché nel lato client è semplice alterare il funzionamento modificando la coordinazione e l'uso dei servizi server; inoltre anche il deploy delle versioni successive è più complesso e costoso.
Infine, un grave problema è che realizzare applicazioni così frammentate richiede molto tempo e quindi costi elevati per cui la realizzazione di RIA resta un'attività decisamente più onerosa in confronto allo sviluppo di applicazioni tradizionali.
Proprio per superare queste difficoltà e rendere possibile a tutti la creazione di RIA, si stanno progettando sistemi che non sono semplicemente dei **CASE** (*Computer Aided Software Engineering*) o framework, ma veri e propri sistemi di sviluppo che contengono gli strumenti necessari alla creazione di applicazioni RIA di qualunque grado di complessità.

PROGRAMMAZIONE RELAZIONALE

Il principio di funzionamento è la programmazione relazionale: i componenti del software non sono più descritti in tanti file di testo, le classi, senza legami di coerenza intrinseca, ma attraverso una struttura relazionale, un grafo, che nativamente scopre e memorizza le relazioni fra i vari componenti dell'applicazione.
Il tutto avviene automaticamente mentre i programmatori continuano a lavorare con le stesse dinamiche cui sono abituati e con il vantaggio ulteriore di poter utilizzare un solo strumento per tutto il ciclo di vita del software, dal disegno del database, al disegno delle interfacce grafiche, alla scrittura delle logiche di business, alle definizioni delle stampe e alla creazione di componenti aggiuntivi quali ad esempio i web service.
Le comunicazioni fra browser e server non sono più basate su HTML, ma su XML compresso e questo rende possibile ottenere l'interattività richiesta dalle RIA.
In conclusione, è un unico strumento per gestire tutto il ciclo di vita del software.

- ✓ La progettazione e la creazione degli schemi **E/R** (*Entity/Relationship*) nel database.
- ✓ L'implementazione degli oggetti di business tramite la **DO** (*Document Orientation*).
- ✓ La creazione del presentation manager di tipo RIA, dai form ai report.
- ✓ Il debug e il test delle applicazioni.
- ✓ La personalizzazione a run-time delle applicazioni, compreso il multilingua.
- ✓ La creazione della documentazione utente e tecnica.
- ✓ La gestione delle problematiche di assistenza agli utenti dell'applicazione.
- ✓ La gestione del versioning e dei gruppi di lavoro.

SILVERLIGHT

Moonlight è l'implementazione open source di Silverlight sulla piattaforma Linux.
Moonlight implementa il modello di sicurezza CoreCLR, che ha per cuore una sandbox in grado di limitare l'accesso del codice alle risorse di sistema in base a tre livelli di sicurezza.

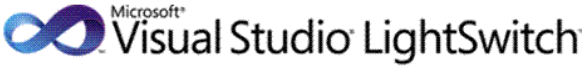
1. **Transparent** il più restrittivo.
2. **Safe-critical** restrizioni moderate.
3. **Critical** nessuna restrizione.

Gli sviluppatori di Moonlight sostengono che è possibile creare applicazioni Silverlight sulle piattaforme Linux utilizzando il Silverlight Unix SDK: per fare ciò è necessario Mono, implementazione open source del MS.NET Framework.
Moonlight utilizza, inoltre, i controlli Silverlight che Microsoft ha rilasciato sotto licenza open source.
Da Silverlight, Moonlight prende diverse, nuove funzioni per l'animazione grafica, il supporto parziale alle applicazioni che girano fuori dal browser e la Pluggable Pipeline, che permette agli sviluppatori di "agganciare" le proprie applicazioni in qualunque punto della pipeline di decodifica: data fetching, demuxing e codecs.
Moonlight 2 consiste in 142000 linee di codice C/C++ e di 320mila linee di codice C#.

LIGHTSWITCH

INTRODUZIONE

È una piattaforma di sviluppo che permette di creare applicazioni gestionali, in altre parole un'applicazione **LOB** (*Line Of Business*), basate sui dati, con pochi clic e in modo semplice come premere un interruttore della luce: da cui il nome LightSwitch.



Si creano le tabelle.
Si creano le maschere.
LightSwitch prepara l'infrastruttura.
Si esegue l'applicazione funzionante senza scrivere codice.

- ✓ Le applicazioni gestionali, anche se si tratta di applicazioni "verticali", fanno un certo numero di attività che sono comuni a tutti gli altri gestionali.
- ✓ I programmatori progettano applicazioni gestionali seguendo una sequenza di azioni sempre uguali.

Creare un database o connettersi a un database esistente, creare l'interfaccia utente con varie modalità griglia, master-detail, scheda singola, modulo di ricerca con selezione dati, associare i controlli dell'interfaccia all'origine dati, eseguire le classiche operazioni **CRUD** (*Creazione, Selezione, Aggiornamento, Cancellazione*), esportare una tabella di dati verso Excel, sono tutte abilità che devono essere fornite all'applicazione.

Non può sostituire strumenti professionali come Visual Studio 2010, ma può essere utile in vari contesti, nascondendo all'utente molti dettagli dell'architettura, evitandogli di dover conoscere il linguaggio di programmazione, almeno per i progetti più semplici.

Le applicazioni LightSwitch sono però completamente configurabili e si può accedere all'editor di codice per aggiungere il codice che rappresenta le regole di business che si vogliono introdurre nell'applicazione.

STRUMENTI

Le applicazioni LightSwitch utilizzano tecnologie e pattern d'avanguardia.

- ✓ Silverlight 4.0.
- ✓ WCF.
- ✓ **RIA** (*Rich Internet Application*) Services e l'Office Automation per l'integrazione con Office 2010.
- ✓ SQL Server, SQL Server Express.
- ✓ SharePoint 2010.
- ✓ SQL Azure.
- ✓ Pattern MVVM

TIPI DI DATI

LightSwitch Type	**C# Type**
Binary	*byte[]*
Boolean	*bool*
Date, DateTime	*DateTime*
Decimal, Money	*decimal*

Double	*double*
EmailAddress, PhoneNumber, String	*string*
Int16	*short*
Int32	*int*
Int64	*long*
Sbyte	*SByte*
Single	*float*
TimeSpan	*TimeSpan*
Guid	*Guid*

Ci sono due nuovi tipi di dato specifici per *EmailAddress* e *PhoneNumber*, sono importanti perché incorporano funzionalità di validazione dei dati.

Architettura

Una LOB è organizzata in tre livelli.

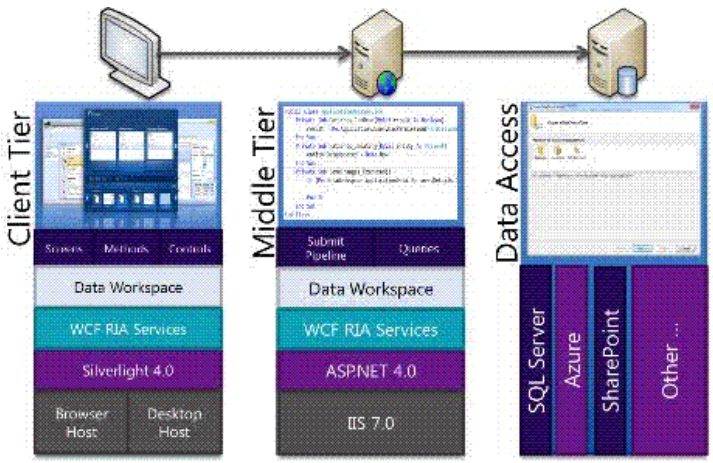
1. **Livello di presentazione** si occupa dell'interfaccia utente, recuperando i dati dagli altri livelli e inviando le modifiche.
2. **Livello relativo alla logica di business** è il livello che smista le richieste del livello di presentazione al livello dati, gestisce la sicurezza per garantire che l'utente abbia i permessi per visualizzare e/o modificare determinati dati e effettua le elaborazioni secondo le regole di business.
3. **Livello dati** è costituito dal **DBMS** (*Data Base Management System*) o da un'applicazione che si occupa di gestire la persistenza dei dati e dei documenti archiviati.

Questi tre livelli sono tra loro interconnessi, perché ciascuno dei livelli fornisce servizi che servono agli altri livelli.
Creare un'applicazione gestionale a tre livelli da zero non è un compito semplice, perché per ogni livello ci sono innumerevoli scelte da fare, sia dal punto di vista della tecnologia da utilizzare, sia da quello delle tecniche da adottare, per esempio il problema della sicurezza.

Tipologie di applicazione

È possibile creare tre tipologie di applicazione.

1. **Desktop client 2-tier** l'applicazione gira sul desktop, componenti server e database sono su un PC di rete.
2. **Desktop client 3-tier** l'applicazione gira sul desktop, componenti server e database girano in IIS.
3. **Browser client 3-tier** il client è in esecuzione in un browser, componenti server e database girano in IIS.

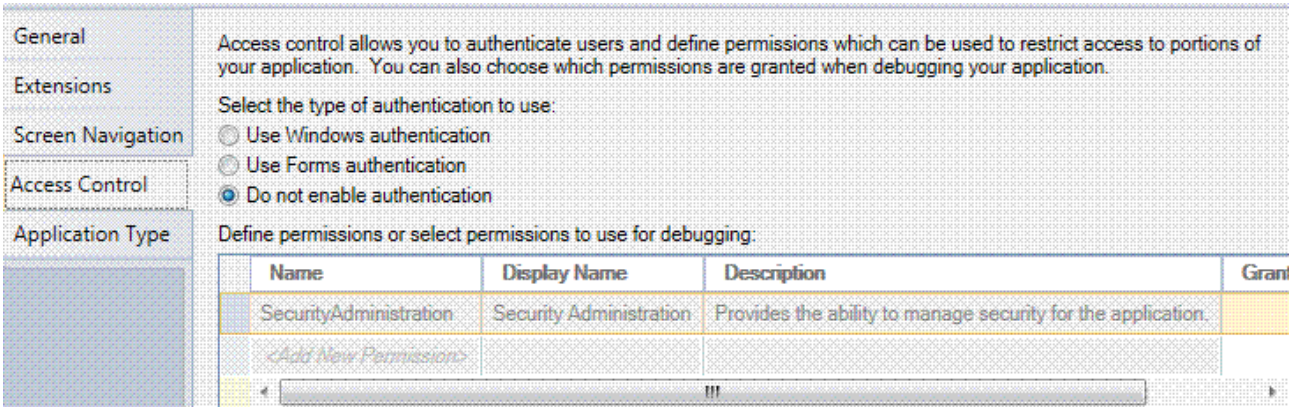


Sta iniziando una nuova “era di rottura” sulla divisione tra applicazioni desktop e web, perché ormai è possibile trasformare un’applicazione desktop in un’applicazione web o per il cloud, Windows Azure e viceversa con un semplice clic.
Questo era impensabile, fino a oggi, per le nette differenze delle piattaforme.

In LightSwitch si può passare da un’applicazione desktop a un’applicazione web, semplicemente con un clic del mouse.

È sufficiente aprire la finestra delle proprietà dell’applicazione, doppio clic sulla voce ***Properties***, nella finestra ***Esplora Soluzioni*** e spostarsi alla scheda ***Application Type***.
In questa scheda è possibile scegliere tre tipi di applicazione, ma più precisamente tre architetture diverse.

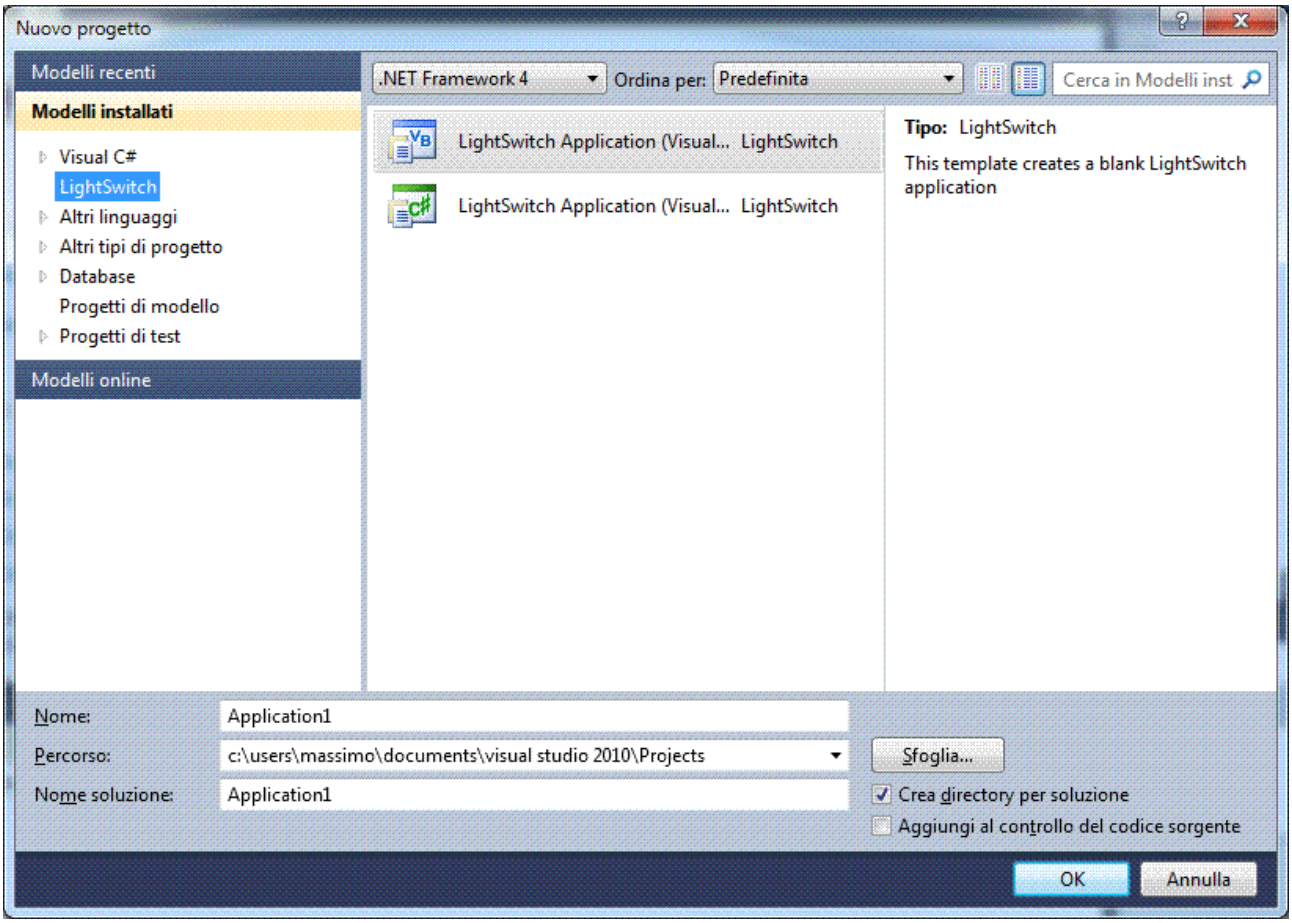
La scheda ***Access Control*** permette d’impostare il livello di sicurezza dell’applicazione.



Inoltre, è possibile definire i ruoli per esempio Utente, Superutente, Amministratore, gli utenti appartenenti a ciascun ruolo e i permessi da assegnare a ciascuno di essi o a ciascun ruolo.

APPLICAZIONE DESKTOP

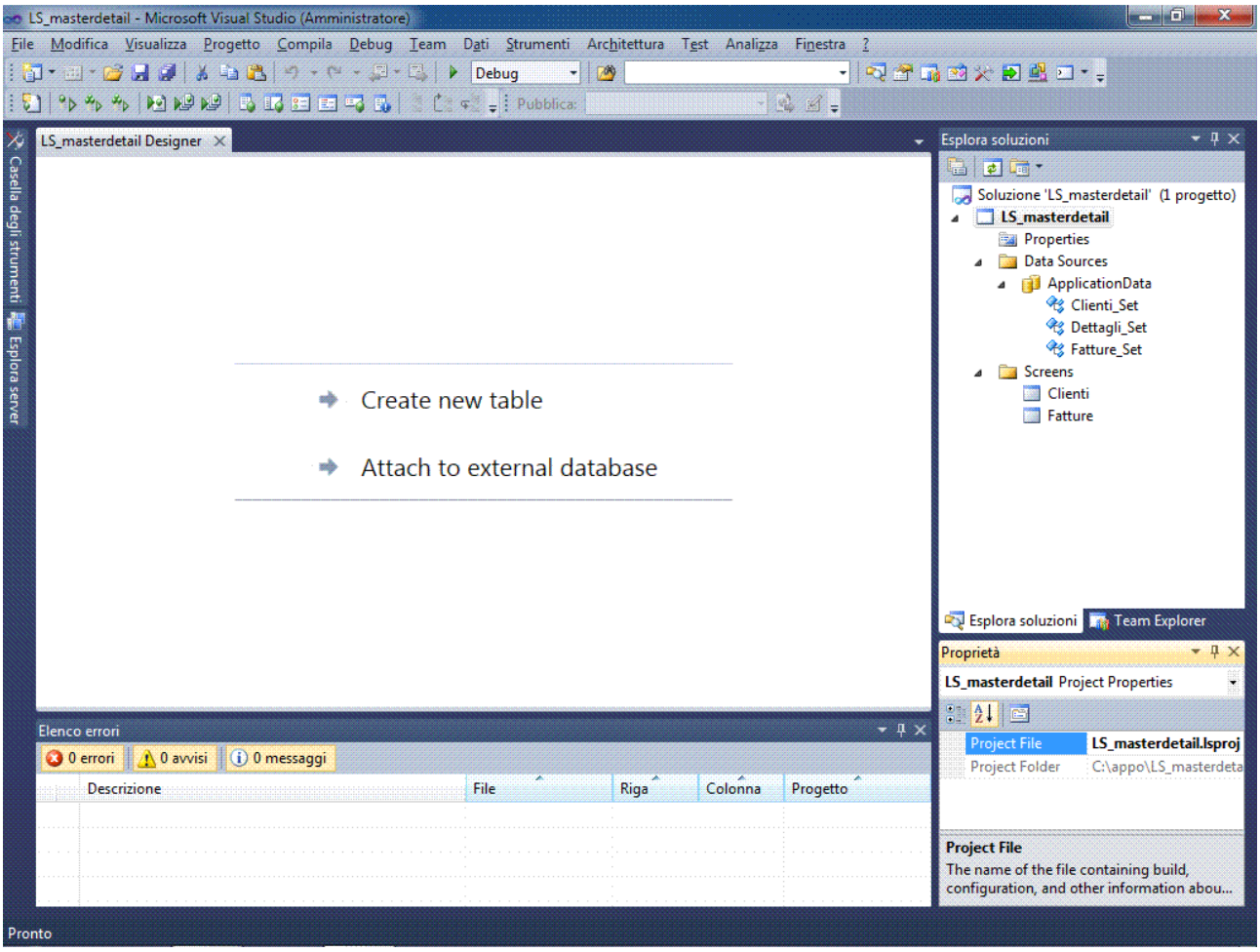
L'ambiente di sviluppo s'integra in Visual Studio 2010, aggiungendo ai template di progetto disponibili l'applicazione ***LightSwitch***, per la quale è possibile utilizzare sia il linguaggio Visual C# sia VB.NET.



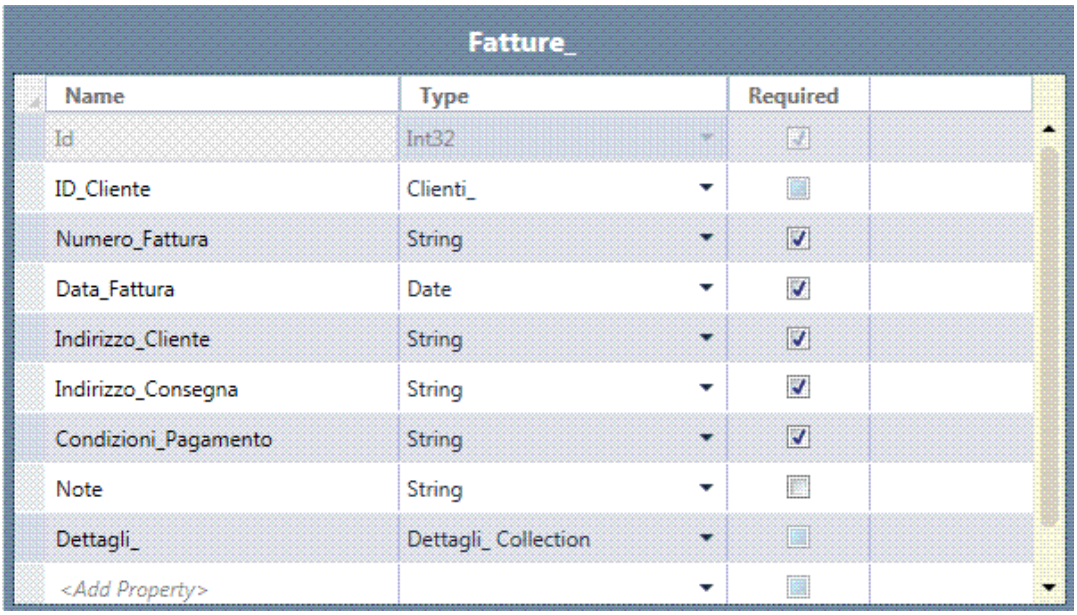
Gli sviluppatori si trovano di fronte a un ostacolo da affrontare che è costituito dalla necessità di collegare un database di SQL Server a un'istanza di SQL Server 2008 già installata sul PC.
Se l'applicazione non la installa il programmatore ma l'utente, come fa a collegare il database a SQL Server 2008, se esiste già un database con lo stesso nome, se SQL Server 2008 non è installato nel PC di destinazione o se ha un nome d'istanza diverso.
Con Visual Studio LightSwitch si ha la possibilità di creare un database locale, basato su file di SQL Server 2008, definendo opportunamente le tabelle e le relazioni che servono all'applicazione.
In questo modo si risolve completamente il problema dell'installazione, pur mantenendo alcune importanti caratteristiche dei file di database di SQL Server 2008: l'accesso protetto ai dati, la robustezza e la stabilità della struttura dei file.

Esempio, progettare un database e definire tabelle e relazioni per una struttura master-detail come può essere quella delle fatture.
La parte master corrisponde all'intestazione della fattura: numero e data fattura, dati del cliente, condizioni di pagamento.
La parte detail è formata dalle singole righe di ciascun prodotto o servizio fatturato: codice, descrizione, prezzo unitario, quantità, importo totale, aliquota IVA.
Aprire Visual Studio 2010 e fare clic su ***File/Nuovo progetto… (CTRL+N)***, selezionare nella finestra ***Nuovo progetto*** e in ***Modelli installati LightSwitch***.
Usare il template per VB.NET e chiamarlo ***LS_masterdetail***.

Creazione delle tabelle
Nella prima schermata, si può scegliere di creare una nuova tabella ***Create new table*** oppure di aprire una connessione a un database esterno ***Attach to external database***.

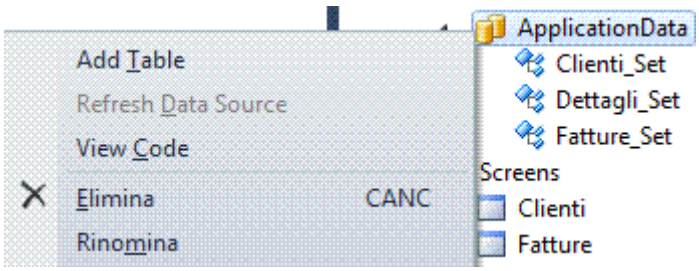


S'inizia da ***Create new table*** per creare una nuova tabella, selezionare l'intestazione ***Table1ItemSet*** e modificarla in ***Fatture_***
Quando si sposta il cursore dall'intestazione, si vede che nella finestra ***Esplora Soluzioni*** sarà modificato il nome della fonte dati in ***Fatture_Set***.
A questo punto, inserire i campi della tabella.



In ogni tabella è inserito di default un campo “*Id*” di tipo “*Int32*”, è la **PK** (*Primary Key*).

Creare la seconda tabella, con i dati delle righe di dettaglio della fattura.
Per avviare la creazione di una nuova tabella, in ***Esplora Soluzioni*** fare clic con il pulsante destro del mouse sul nodo ***ApplicationData*** e selezionate ***Add Table***.



Cambiare il nome della tabella in ***Dettagli_***.
A questo punto, inserire i campi della tabella.

Dettagli_

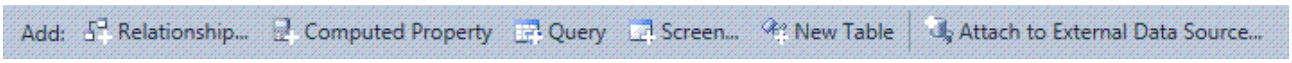
Name	Type	Required
Id	Int32	☑
Descrizione	String	☑
Qta	Int16	☑
Prezzo_Unitario	Money	☑
Prezzo_Totale	Money	☑
Aliquota_IVA	Double	☑
Fatture_	Fatture_	☐

Creare la tabella relativa ai dati dei clienti, cambiare il nome della tabella in ***Clienti_***.
A questo punto, inserire i campi della tabella.

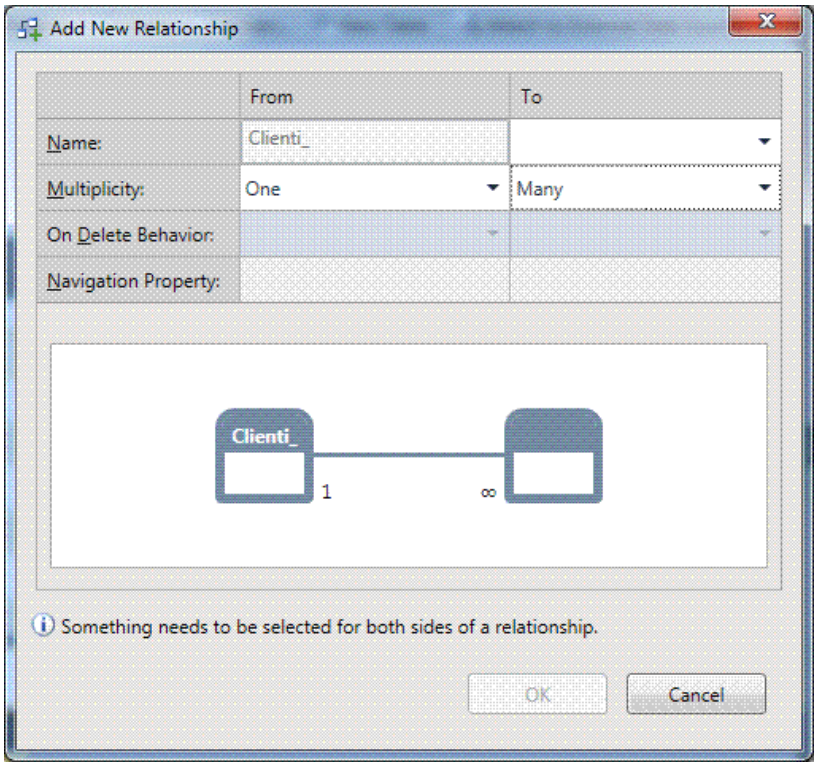
Clienti_

Name	Type	Required
Id	Int32	☑
RagioneSociale	String	☑
Partita_IVA	String	☑
Codice_Fiscale	String	☐
Sede	String	☑
Email	EmailAddress	☐
Fatture_	Fatture_ Collection	☐

Creazione delle relazioni tra tabelle
Nella parte superiore della finestra di definizione delle tabelle ci sono alcuni pulsanti.



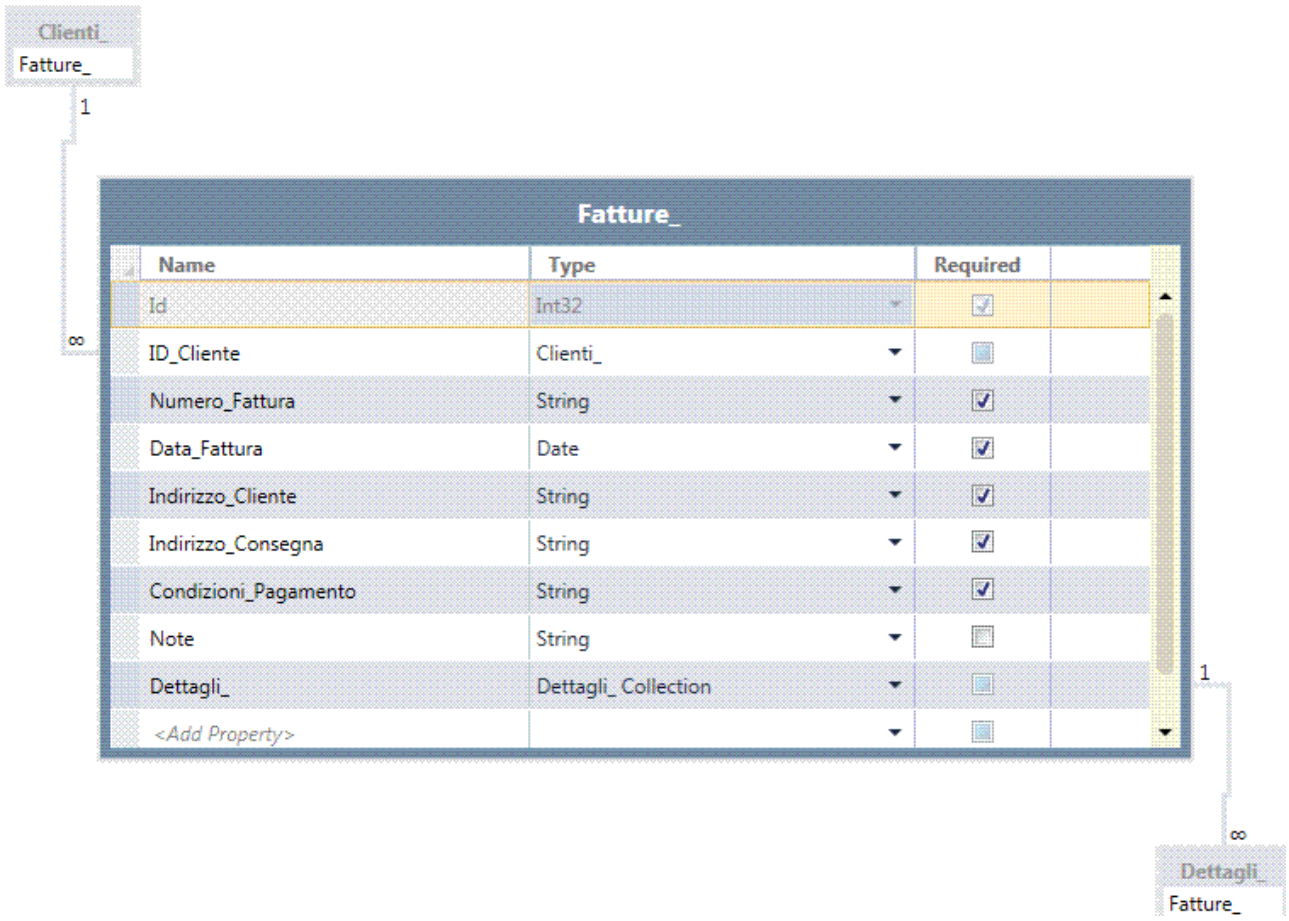
Per definire una nuova relazione, fare clic sul secondo pulsante ***Relationship…***.



Le relazioni da creare sono due.
1. Clienti e Fatture
2. Fatture e Dettagli.

In entrambi i casi, alla prima tabella è assegnata la proprietà ***Multiplicity*** pari a ***One***, mentre la seconda tabella ha la proprietà impostata a ***Many***, in altre parole la classica relazione “uno-a-molti”, indicato anche più sinteticamente con “1-N”.
La proprietà ***On Delete Behavior*** è la proprietà che definisce il comportamento da adottare in caso di cancellazione di un record: l’opzione preimpostata è ***Restricted***, ma se si vuole è possibile impostarla con l’opzione ***Cascade delete*** (cancellazione in cascata).
In pratica, con questa ultima opzione, se si cancella un record cliente, sono cancellate tutte le fatture associate a tale cliente e, ancora in cascata, tutti i record di dettaglio di ciascuna fattura.
Nel caso di un gestionale, in genere è meglio controllare in modo puntuale la cancellazione, prima di effettuare una cancellazione a cascata.

Nella figura si può osservare la struttura delle relazioni di tabella, focalizzata sulla tabella *Fatture*.



Per navigare tra le tabelle, è sufficiente fare un doppio clic sulla tabella desiderata e subito si vede cambiare la visualizzazione, con il focus su tale tabella.
Notare che le tabelle, dopo la definizione delle relazioni, sono state modificate: infatti, sono stati aggiunti automaticamente i campi necessari per inserire il riferimento al record associato nella tabella correlata: chiave esterna.
Le chiavi esterne sono implicite, poiché non sono mai visualizzate nelle schermate, ma sono gestite in modo totalmente trasparente all'utente.

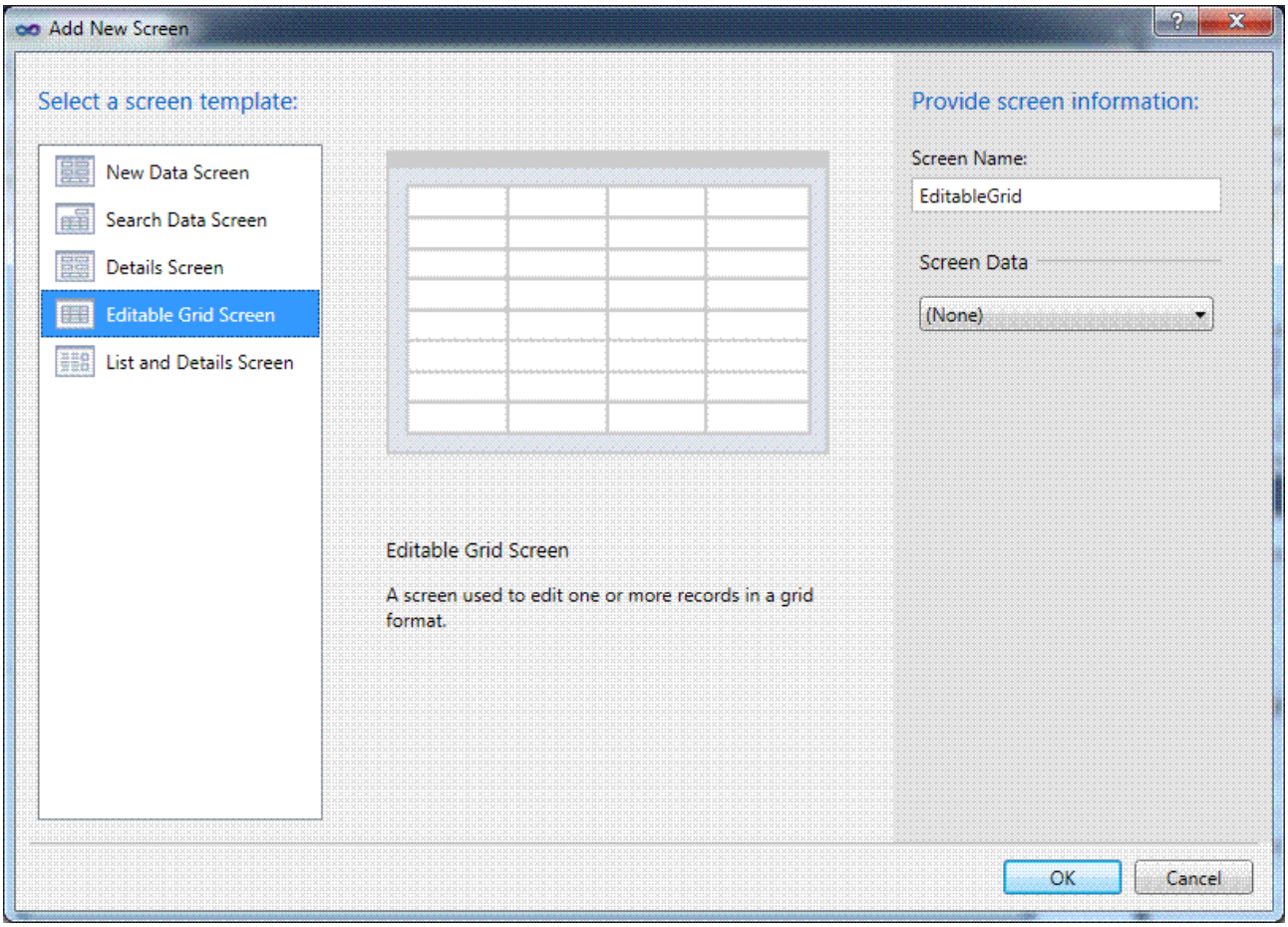
Creazione delle schermate per la gestione dei dati

Per definire un nuovo Screen, una visualizzazione, una schermata, fare clic sul quinto pulsante ***Screen…***, è possibile selezionare cinque tipi di schermate.

1. ***New Data Screen*** per l'inserimento di nuovi record.
2. ***Search Data Screen*** per la ricerca di dati.
3. ***Details Screen*** di tipo master-detail.
4. ***Editable Grid Screen*** una griglia per la visualizzazione e modifica di dati in forma tabellare, utilizzata per la visualizzazione di un cliente, nella casella ***Screen Name:*** inserire *Clienti*.
5. ***List and Details Screen***, di tipo master-detail espresso però come una lista, per la visualizzazione di una fattura.

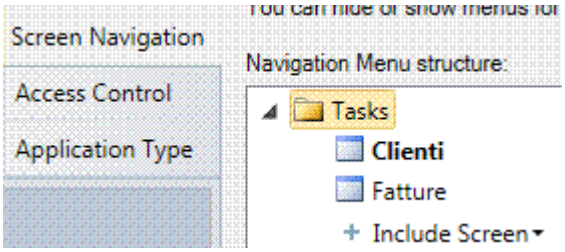
Anche se l'applicazione non è terminata, se si esegue si può vedere che questa parte è già funzionante, dato che ora è possibile inserire nuove righe nella tabella Clienti, modificarle e cancellarle.

Dopo aver inserito le righe di dati, ricordarsi sempre di cliccare sul pulsante ***Save***, per memorizzare i dati nel database locale.

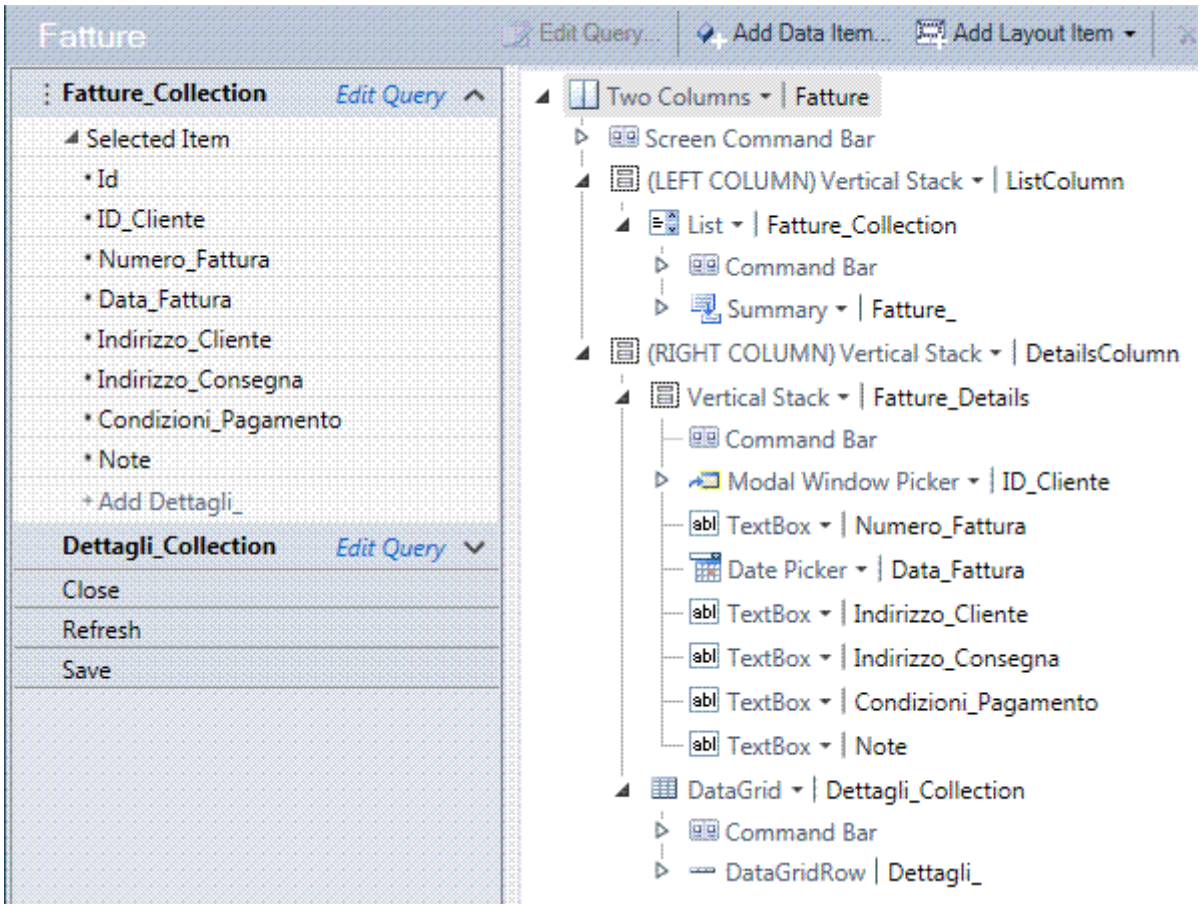


Durante lo sviluppo dell'applicazione, si può controllare se uno Screen potrà essere visualizzato nel menu laterale e quindi essere disponibile nell'applicazione in run-time, evitando così di avviare più volte l'applicazione per verificarne il corretto funzionamento.

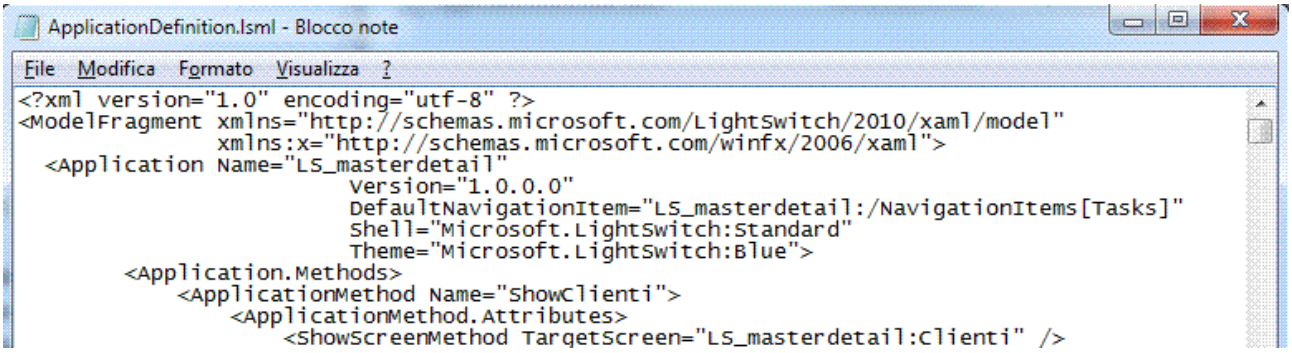
È sufficiente aprire la finestra delle proprietà dell'applicazione, doppio clic sulla voce ***Properties***, nella finestra ***Esplora Soluzioni*** e spostarsi alla scheda ***Screen Navigation***, nel nodo ***Tasks***, sono presenti entrambi gli Screen.



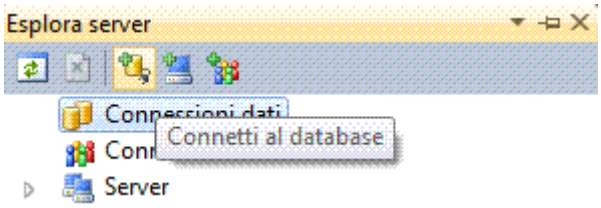
In ogni Screen ci sono dei pulsanti già predefiniti e funzionanti, tra cui anche un pulsante per l'esportazione del set di dati verso un foglio Excel.
Il designer mostra la composizione della maschera, elencando i controlli di comando e quelli data-bound, fare doppio su *Fatture*.



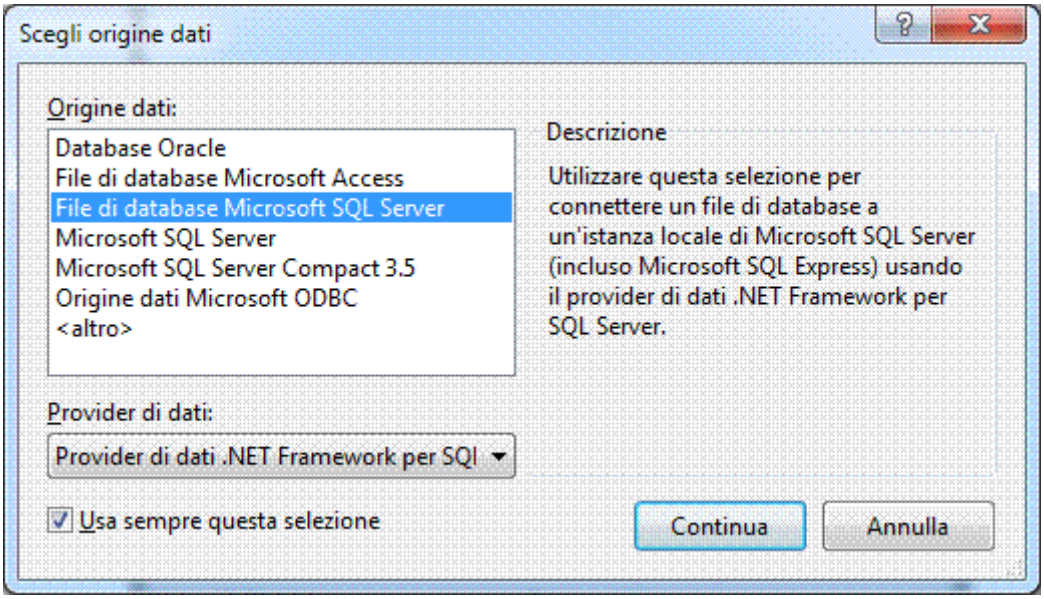
Database locale
Nella finestra ***Esplora Soluzioni***, i dati sono definiti all'interno del nodo ***Data Sources***, nel database chiamato ***ApplicationData***.
In quest'ultimo nodo sono contenute le tabelle, se si va a vedere all'interno dei file contenuti nel progetto, si scopre che gli oggetti di database sono mappati nel file APPLICATIONDEFINITION.LSML (*LightSwitch Markup Language*), contenuto nella cartella Data.



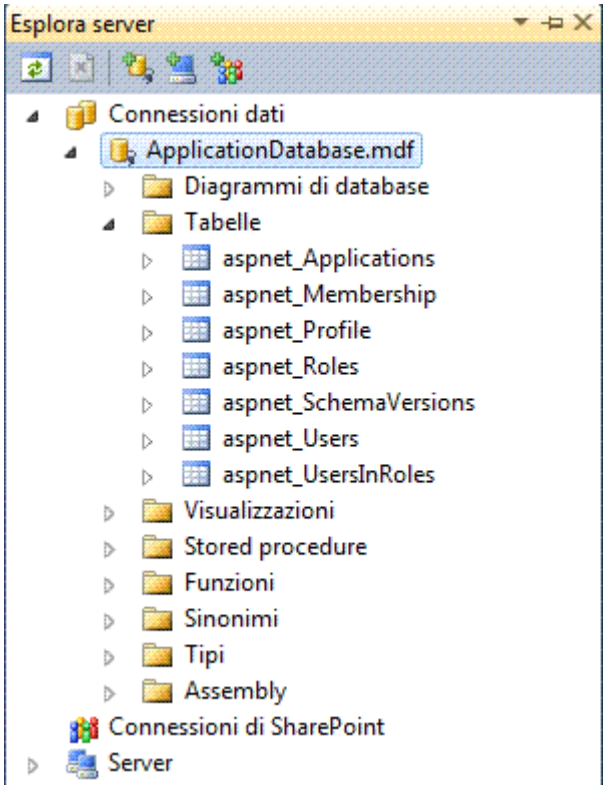
Questo però non è il file che contiene i dati inseriti dall'utente, ma è solamente il file che contiene la definizione della struttura dei dati: tabelle, campi, relazioni, vincoli.
I dati veri e propri si trovano nella sotto cartella bin\Data dove ci sono due file, rispettivamente con estensione MDF e LDF.
Il database locale non è altro che un file di database di SQL Server, scollegato da qualsiasi istanza di SQL Server eventualmente installata sul PC.
Esaminare la struttura del database: creare una nuova connessione al database, dalla finestra ***Esplora Server***, fare clic sulla terza icona ***Connetti al database***.



Usare il provider dati ***File di database Microsoft SQL Server***.

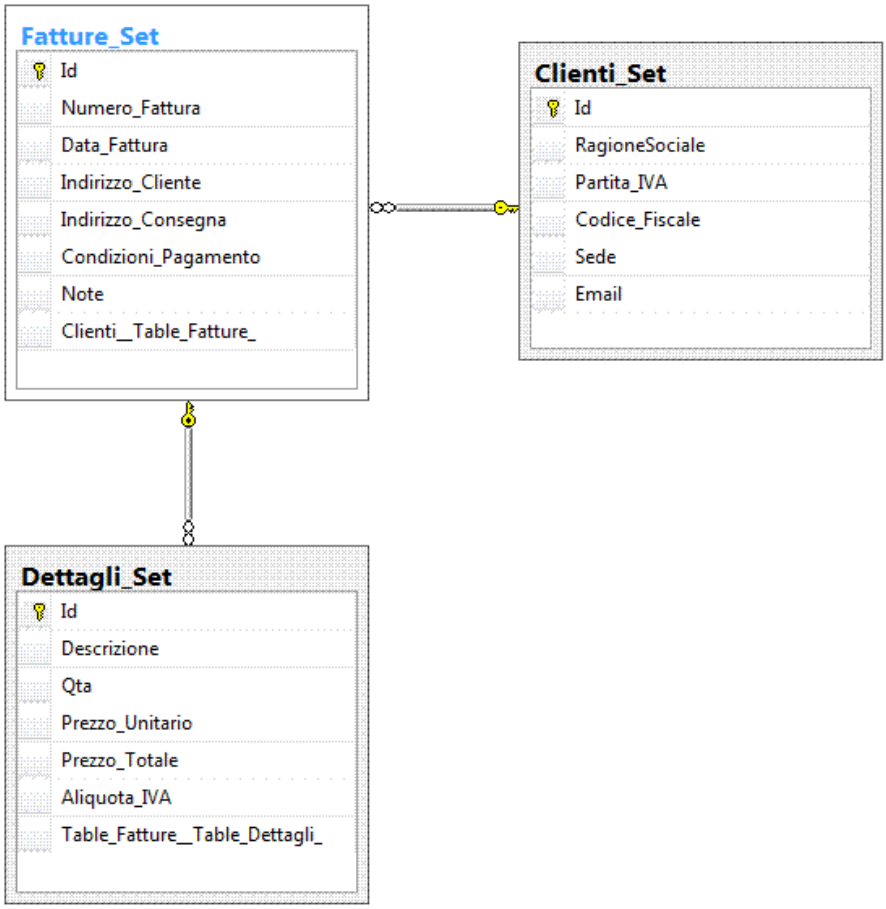


Una volta che aperta la connessione, si vede che il database contiene le seguenti tabelle.

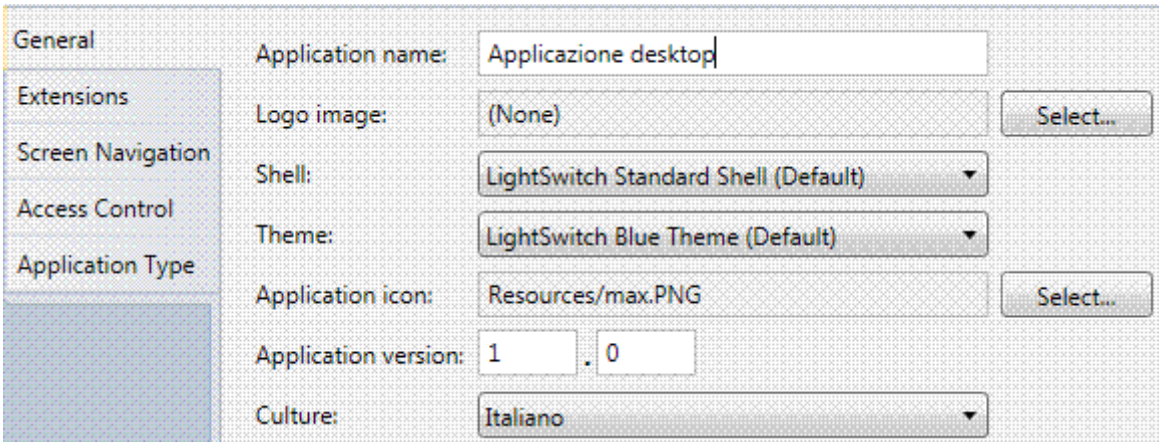


Il database include non soltanto le tabelle create nel corso dello sviluppo dell'applicazione, ma anche una serie di altre tabelle accessorie per la definizione di profili, ruoli, utenti, permessi.
Inoltre, c'è la presenza di numerose ***Visualizzazioni*** e ***Stored procedure***, definite automaticamente durante la creazione del database.

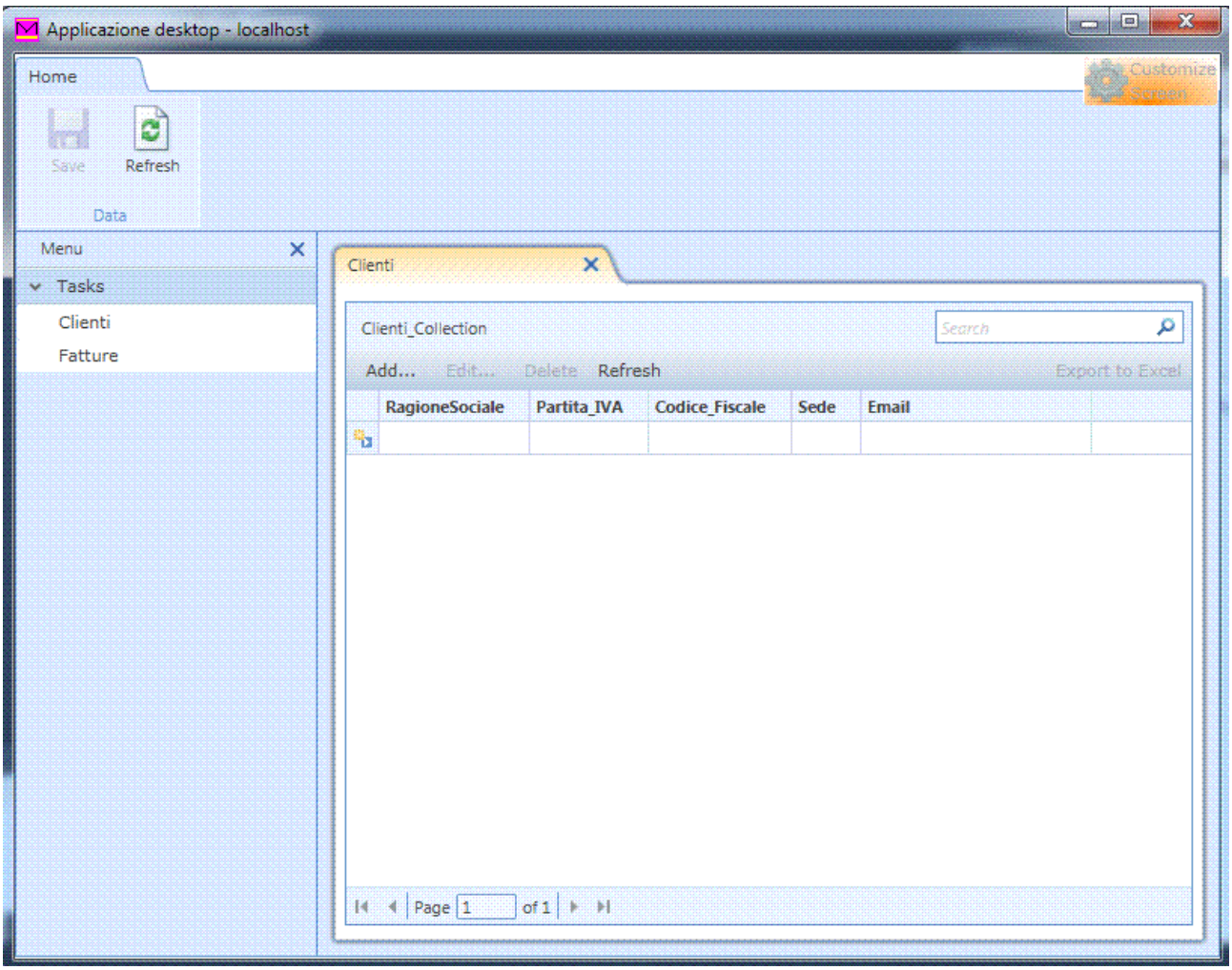
Creare il diagramma delle relazioni tra le tre tabelle, fare clic nella finestra ***Esplora Server*** sul pulsante ***Diagrammi di database***.



Aprire la finestra delle proprietà dell'applicazione, doppio clic sulla voce ***Properties***, nella finestra ***Esplora Soluzioni*** e spostarsi alla scheda ***General***.
Questa scheda è dedicata alle proprietà generali dell'applicazione, si può dare il nome all'applicazione, assegnare un'immagine per il logo e un'icona, indicare la "cultura" da utilizzare, per esempio per le impostazioni relative ai numeri, alle valute e alle date, il numero di versione distinto in major version e minor version.



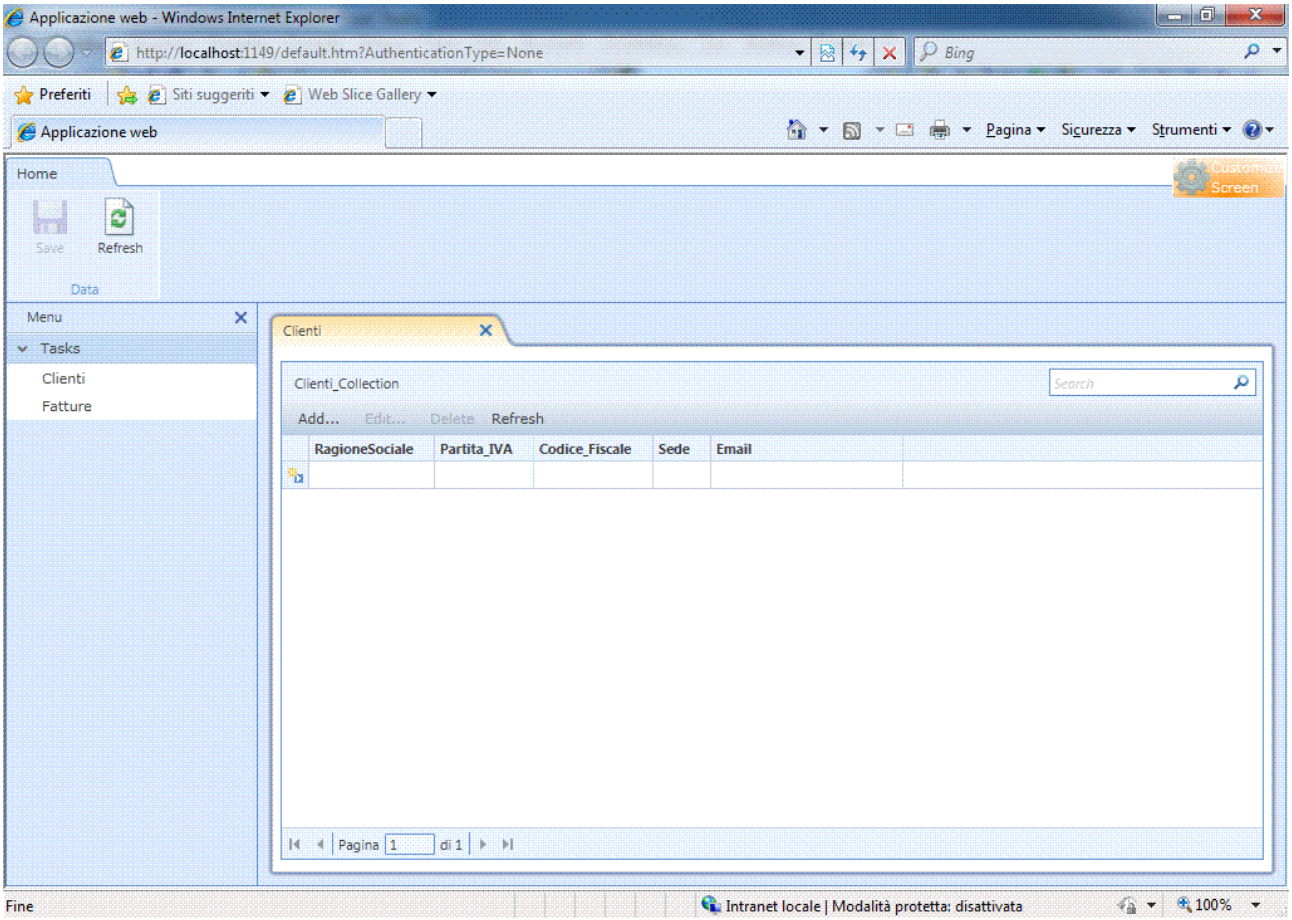
Esecuzione
Salvare, compilare e eseguire l'applicazione.
Nella finestra si notano i seguenti elementi.
- ✓ Un controllo ribbon con i comandi principali.
- ✓ Una scheda ***Task*** con i comandi per mostrare le maschere.
- ✓ La validazione automatica dei dati, senza scrivere codice.



Sul lato sinistro c'è l'elenco degli Screen inseriti nell'applicazione.
Nella parte alta è presente un controllo ribbon, con alcuni pulsanti per salvare le modifiche e per aggiornare la visualizzazione.
Infine, all'interno della finestra di visualizzazione, sono inseriti i pulsanti per aggiungere, modificare o cancellare i record, la casella di ricerca e perfino un pulsante già predisposto per esportare i dati dell'intera griglia in un foglio Excel.
Per esempio, se si vuole aggiungere un record alla tabella, basta cliccare sul pulsante ***Add…*** .
Appare una finestra di dialogo con tutti i controlli già predisposti per l'inserimento dei dati.
L'applicazione include anche già delle regole per la validazione dei dati, anche se non può certo prevedere tutte le regole di validazione che possono essere definite in contesti specifici.
L'applicazione, dopo la sua creazione attraverso i wizard, può essere completamente personalizzata e modificata, sia nell'aspetto grafico, sia nelle regole di business che devono essere definite dietro le quinte, sia aggiungendo funzionalità che possono essere implementate solamente attraverso il codice VB.NET o C#.

Applicazione Web

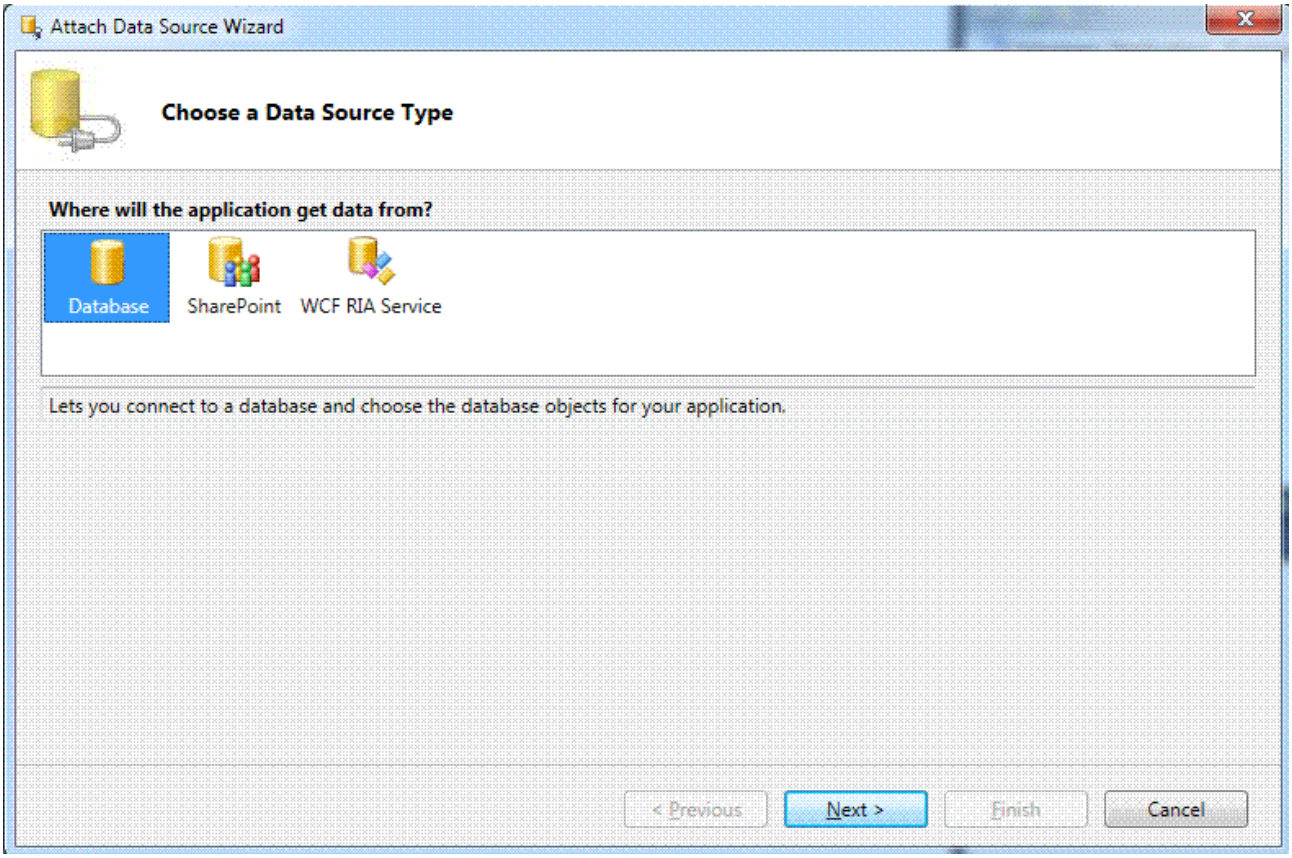
Aprire la finestra delle proprietà dell'applicazione, doppio clic sulla voce ***Properties***, nella finestra ***Esplora Soluzioni*** e spostarsi alla scheda ***Application Type***.
Fare clic sulla terza opzione, per selezionare il tipo di applicazione ***Browser client, 3-tier deployment***.
Dopo aver salvato le modifiche, eseguire l'applicazione.



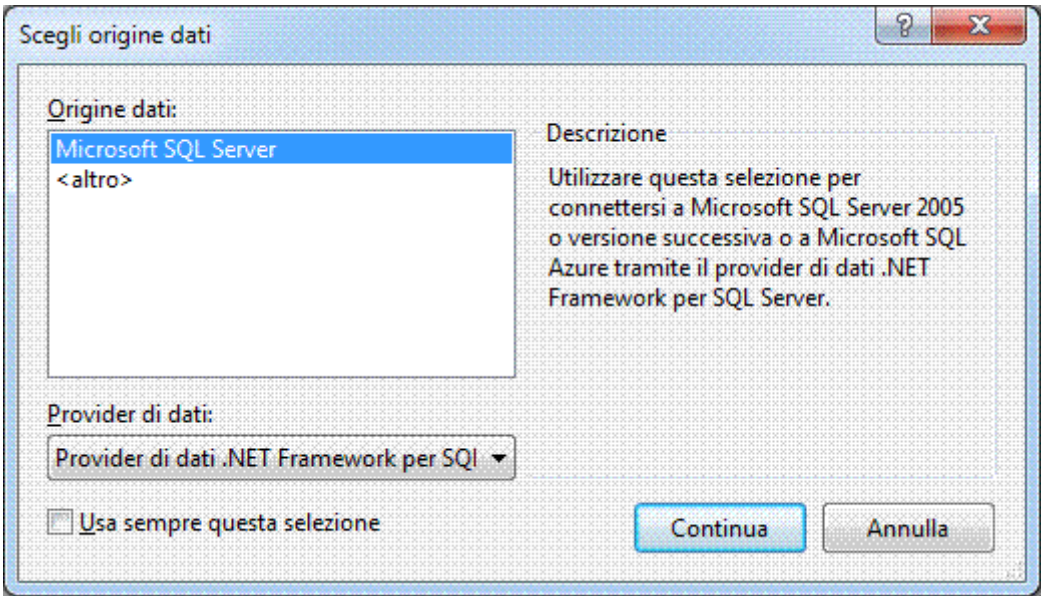
Si apre il browser con l'applicazione nella pagina web che ha lo stesso aspetto e le stesse funzionalità dell'applicazione per desktop.

Connessione a un database esistente

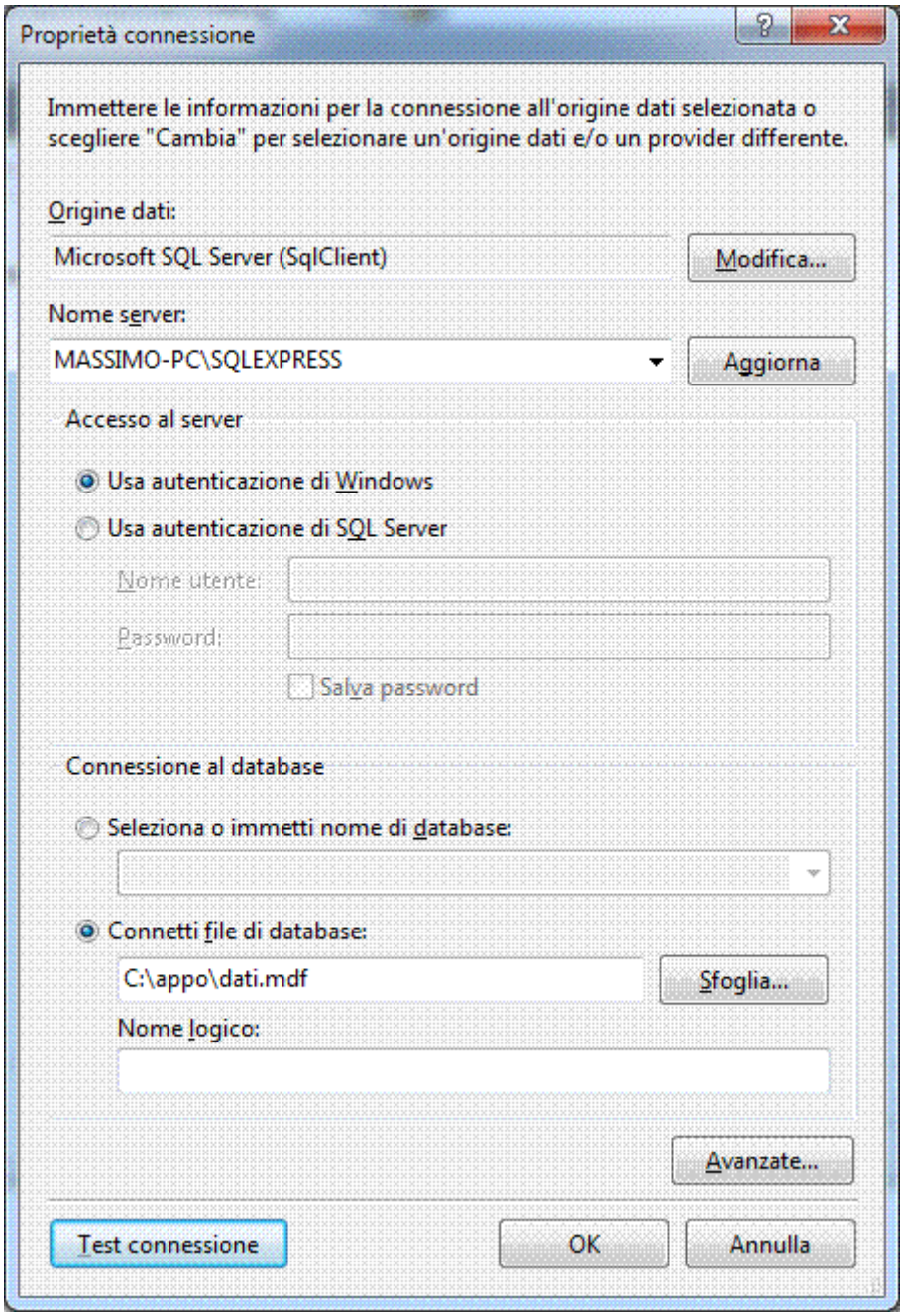
Fare clic sulla voce ***Attach to external database***, apparirà una finestra di dialogo relativa a un wizard, un'autocomposizione, per la definizione della connessione all'origine dati.



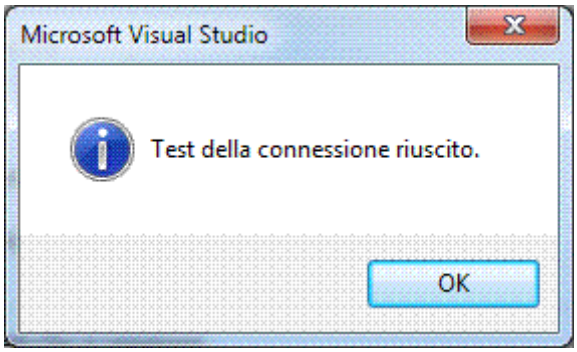
È possibile creare una connessione a un ***Database***, a un sito ***SharePoint*** o anche a un ***WCF RIA Service***, selezionare ***Database*** e fare clic sul pulsante ***Next***.



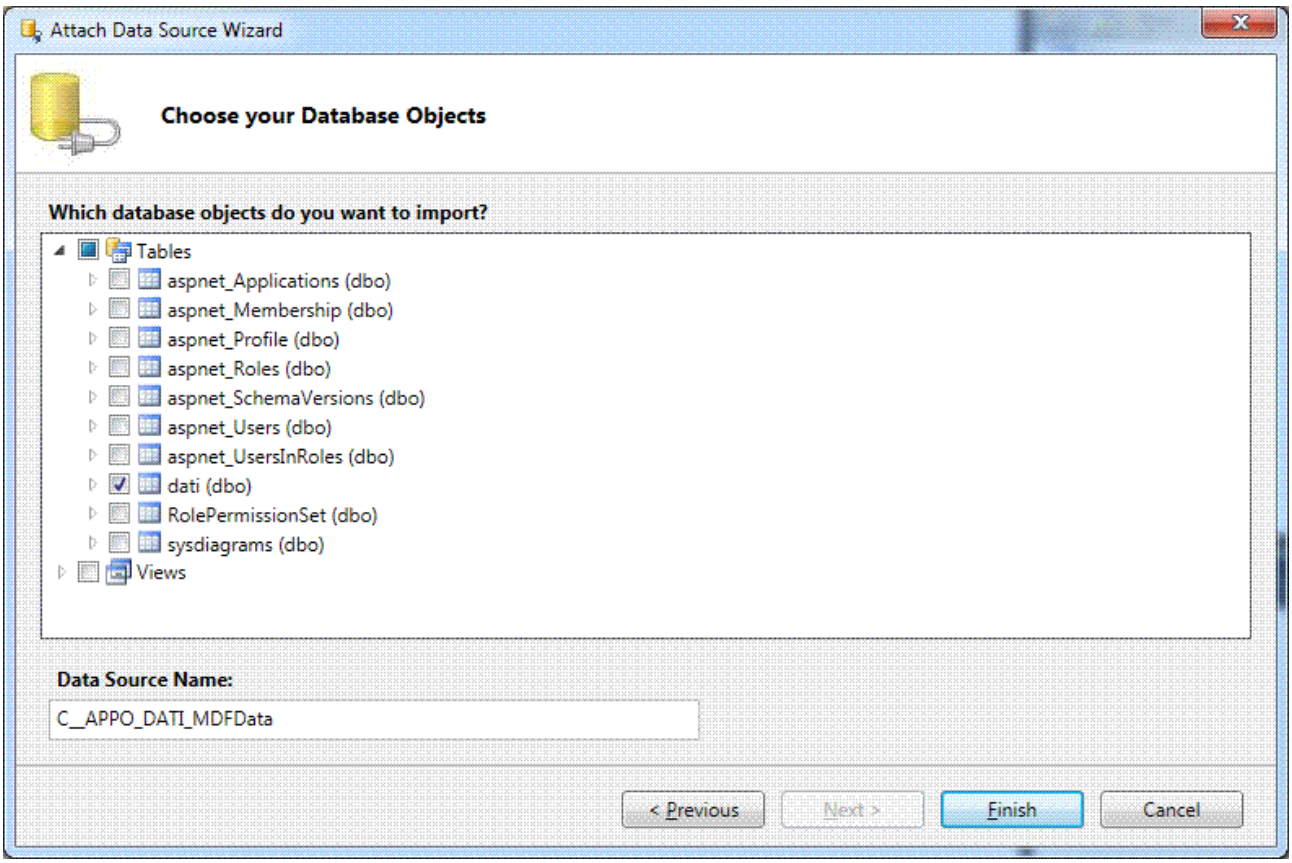
Il passo successivo definisce le proprietà della connessione, con la finestra di dialogo seguente.



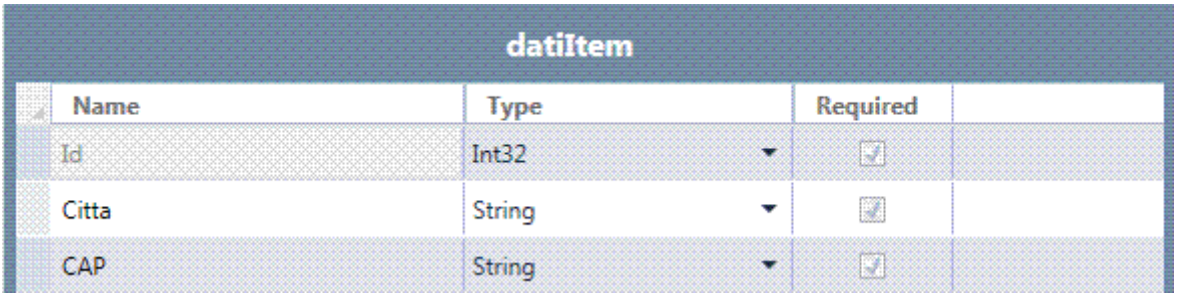
Lasciare invariato il provider dati ***Microsoft SQL Server (SqlClient)***, perché si ha a disposizione proprio un database di questo tipo.
Dopo aver selezionato il nome del server, del tipo di autenticazione e il nome del database che interessa, fare clic sul pulsante ***Test connessione***.



Avuta conferma della correttezza di tutte le proprietà di connessione, il passaggio successivo richiede la scelta delle tabelle che si vogliono inserire nel Data Source.
Selezionare la tabella *dati* e modificare, se si desidera, il nome del Data Source, da quello proposto automaticamente *C_APPO_DATI_MDFData* dovuto a una ridenominazione del database collegato all'istanza di SQL Server e fare clic sul pulsante ***Finish***.

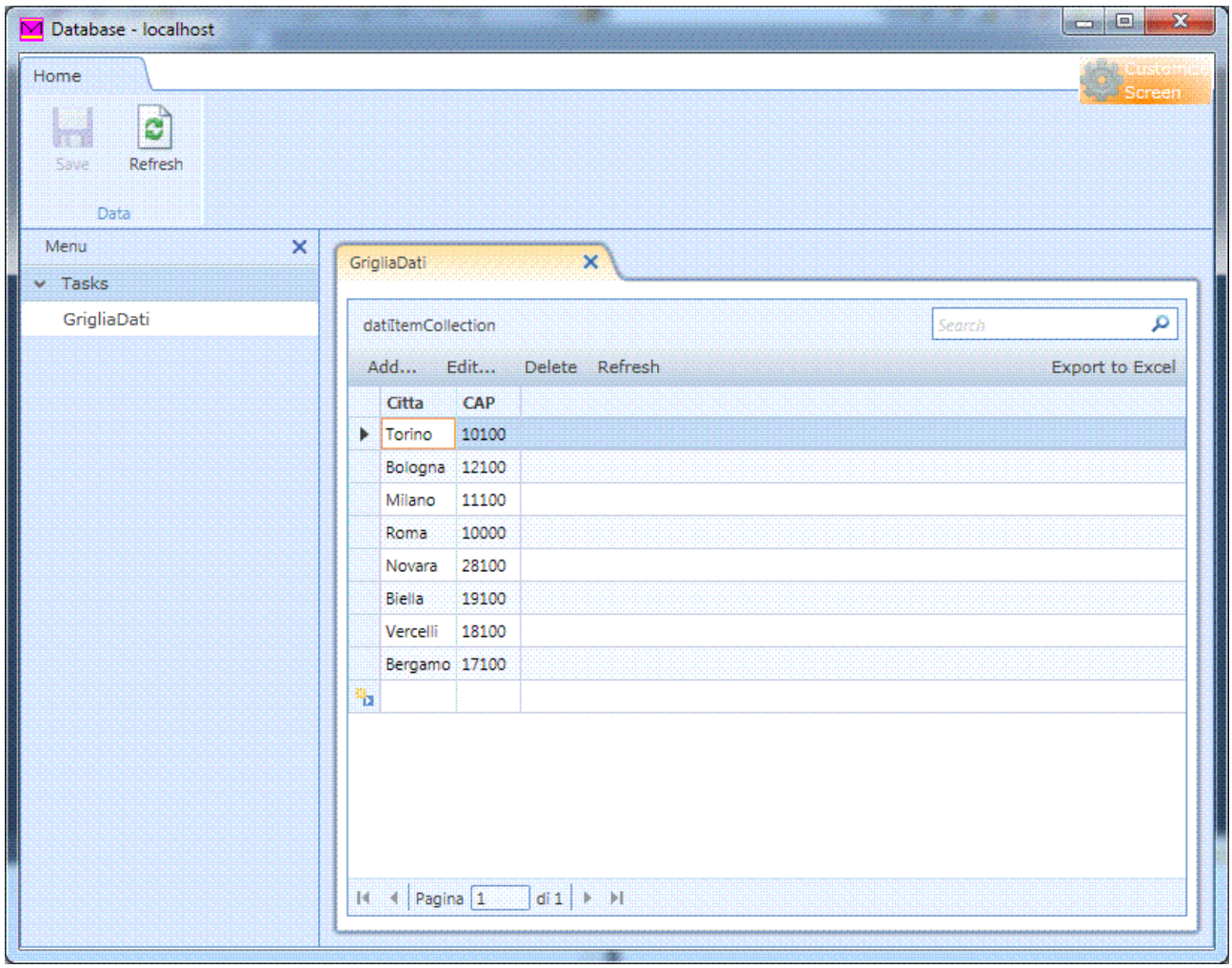


Sono visualizzate le tabelle e loro relazione, nell'esempio è una sola.



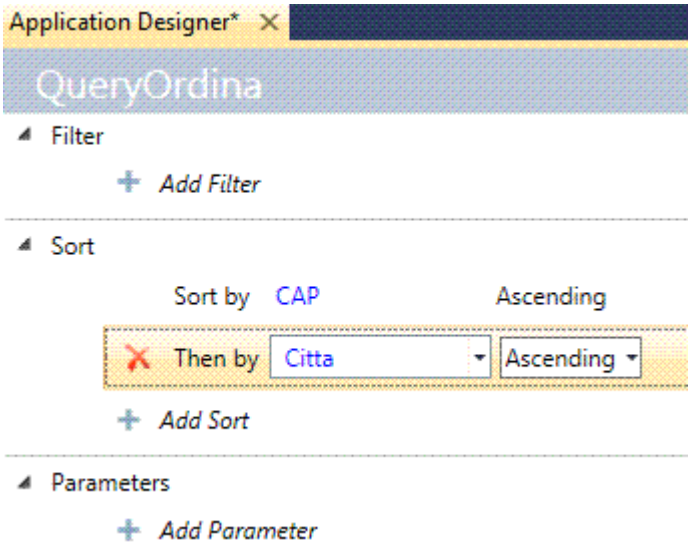
Definire uno Screen per la visualizzazione della tabella.

Salvare, compilare e eseguire l'applicazione.



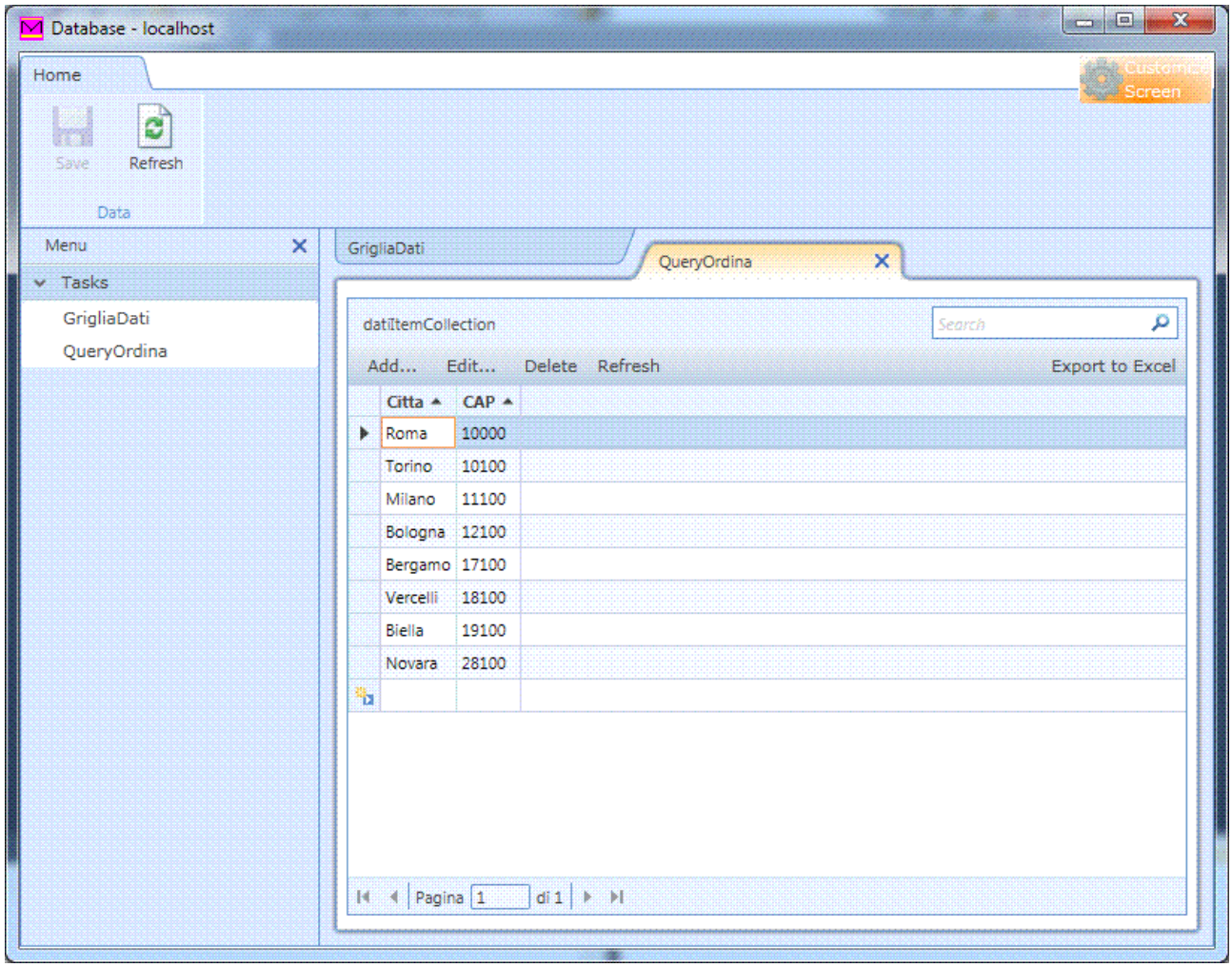
QUERY

Per avviare la creazione di una nuova query, in ***Esplora Soluzioni*** fare clic con il pulsante destro del mouse all'interno del nodo ***Data Sources/C_APPO_DATI_MDFData/dati/Add Query***.



Definire uno Screen per la visualizzazione della query.

Salvare, compilare e eseguire l’applicazione.



MODULO 8

CLOUD COMPUTING

Cloud computing
Windows Azure

CLOUD COMPUTING

INTRODUZIONE

Un browser non è più soltanto un software con cui accedere a una pagina web, ma un vero e proprio ambiente in cui far girare applicazioni di ogni tipo.
Se il browser diventa ambiente di lavoro, allora lo si può estendere fino a farlo diventare sistema operativo, per esempio Google Chrome OS.
Alcuni anni fa questo modello è stato chiamato **ASP** (*Application Service Provider*), poi **SAAS** (*Software As A Service*).
Il cloud computing è un insieme di risorse, hardware/software, disponibili e rese accessibili via rete, su richiesta, on demand, è di due tipi.

1. **Low-level**: memoria, spazio su disco, in pratica virtualizzazione dei sistemi.
2. **High-level**: accesso a un database, a un sistema di pagamento on-line, in pratica servizi di larga scala.

Le risorse sono selezionate in base alle reali necessità dei clienti, che quindi pagheranno solo il reale utilizzo delle stesse.

Quello che ieri si comprava, oggi si affitta, trasformando così i costi fissi in variabili, il cui ammontare è legato alle necessità dell'azienda.

Servizi su Internet da utilizzare quando servono e solo per il tempo necessario.
Esempi.

- ✓ Windows Azure: sistema operativo per il cloud computing.
- ✓ Office 365.
- ✓ Antivirus: F-Secure, McAfee.
- ✓ Google Docs e Gmail: Google.
- ✓ Micro**ERP** (*Enterprise Resource Planning*) di Zucchetti.
- ✓ La Nuvola Italiana: Telecom Italia.
- ✓ MyCustomerEasy: Olivetti.

WINDOWS AZURE

INTRODUZIONE

È la piattaforma per il Cloud Computing composta da due sistemi.

1. Un sistema operativo scalabile e fault-tolerant: Windows Azure.
2. Una piattaforma di servizi: le Azure Services Platform.

La realizzazione di un sito per la prima uscita pubblicitaria è un'operazione semplice: si sceglie la grafica e si pubblica l'home page e qualche pagina informativa.
Per la pubblicazione si devono svolgere i seguenti compiti.

1. Assegnare un IP libero al server web.
2. Creare una cartella sul server web.
3. Creare un nuovo sito in IIS.
4. Aprire un punto FTP per consentire la pubblicazione del progetto da parte degli sviluppatori.
5. Marcare il sito come applicazione e assegnare la versione corretta di ASP.NET.

I compiti descritti richiedono un intervento amministrativo sul server che comporta il coinvolgimento di una persona diversa dallo sviluppatore, sia per un problema di competenza, sia per una questione di permessi di accesso.
In conclusione, si possono verificare una serie di problemi quali la versione di ASP.NET non è stata configurata correttamente, manca spazio su disco per i file, IIS non è configurato bene.

Un sito web pubblico dovrebbe essere gestito da chi ha competenze sistemistiche e dovrebbe essere inserito in una topologia di rete che ne renda sicuro l'accesso.

Per le aziende piccole, dove lo sviluppatore "fa un po' di tutto", il problema è evidente in quanto una sola persona non può più essere in grado di conoscere i dettagli dell'intero spettro di tecnologie/prodotti/leggi.
Nelle grandi aziende che hanno una suddivisione netta dei ruoli fra sistemisti, esperti di sicurezza, sviluppatori e quindi una competenza elevata in tutti gli ambiti coinvolti dalla creazione di un'applicazione, il problema è ancora maggiore: entrano in gioco meccanismi e dinamiche strane, secondo le quali diventa complesso richiedere anche solamente un ambiente di test per le applicazioni.
La gestione dell'applicazione e del sistema stesso ha un costo, l'applicazione vedrà versioni successive da installare, così come devono essere aggiornati frequentemente le tecnologie di base, dal sistema operativo al motore database.
Queste operazioni richiedono molto tempo sia lato sviluppo sia lato sistemistico.
Un altro problema è rappresentato dai costi.
Acquistare hardware per una nuova applicazione è un'operazione costosa: oltre al costo iniziale, ci sono i costi di manutenzione e sostituzione componenti obsoleti, senza contare che spesso è difficile, soprattutto per le applicazioni web, dimensionare l'hardware.

Hosting

Si mette il sito web sul server del provider.

- ✓ Vantaggi: costo basso.
- ✓ Svantaggi: si condivide lo stesso server e la stessa banda con gli altri clienti.

Housing

Il server è di proprietà e gestito dell'azienda ma è collocato nei locali del provider.

- ✓ Vantaggi: si possono scegliere tutti i componenti desiderati.
- ✓ Svantaggi: è costoso e richiede esperienza nella gestione del server.

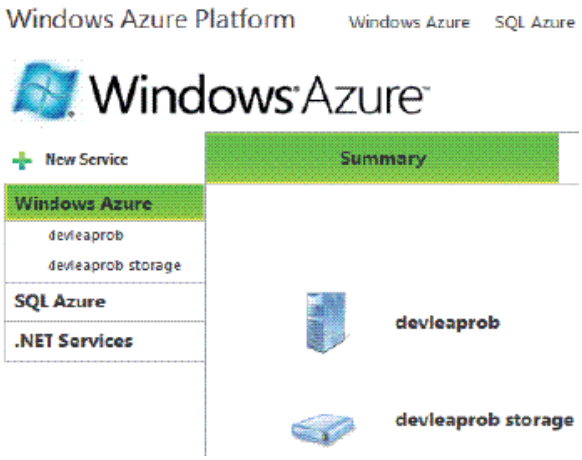
Se si portano le macchine fisiche da un provider, non si risolve il problema del costo iniziale e il problema della gestione dell'applicazione; non si risolve neanche il problema di competenze/passaggi d'informazioni: si è semplicemente spostato la locazione fisica dove accadono i problemi.
Se si creano macchine virtuali da mettere in hosting si risolve il problema del costo iniziale in quanto non si deve acquistare hardware, ma si possono sfruttare i server del provider; si abbatte anche il tempo di aggiornamento visto che molti upgrade saranno montati direttamente dal provider.
In ogni caso la gestione della macchina virtuale è un compito dell'azienda che si complica notevolmente quando si vuole garantire load-balancing o fault-tolerance: non solo si devono installare e manutenere più macchine, ma si rende necessario anche effettuare il deploy dell'applicazione e delle release/patch successive su tutte le macchine.

CLOUD COMPUTING

È un insieme di tecnologie informatiche che permettono l'utilizzo di risorse, storage, CPU distribuite.
La caratteristica principale di tale approccio è di rendere disponibili all'utilizzatore tali risorse come se fossero implementate da sistemi server o periferiche "standard".
L'implementazione effettiva delle risorse non è definita in modo dettagliato; anzi l'idea è proprio che l'implementazione sia un insieme eterogeneo e distribuito, *the cloud*, in inglese nuvola, di risorse le cui caratteristiche non sono note all'utilizzatore.
La definizione mette però in luce due aspetti.

1. L'utilizzo di risorse distribuite e risorse le cui caratteristiche non sono note all'utilizzatore.
2. Le caratteristiche fisiche dell'hardware che non sono note neanche allo sviluppatore della soluzione stessa.

Per risorse fisiche s'intendono i dischi su cui leggere e scrivere, i nomi delle cartelle sul file system, l'indirizzo IP della macchina, il nome della macchina stessa.



LS (*Local Storage*)
In Windows Azure è possibile configurare LS per sopravvivere ai recycle del processo, tramite l'attributo *cleanOnRoleRecycle* della configurazione *<LocalStorage>.*
Questa impostazione consente di evitare la cancellazione dello store locale in caso di riciclo del processo: ad esempio, a fronte di una modifica alla configurazione oppure in seguito ad un upgrade, è probabile che il processo dove sta girando l'applicazione debba essere riciclato, come accade in IIS nelle soluzioni on-premise.
Questo flag consente di evitare di perdere il contenuto del LS, in ogni caso lo storage è locale all'istanza ed è comunque perso in caso di spostamento del codice su un altro nodo del cloud.
Windows Azure è il sistema operativo "in the cloud", installato sui Data Center di Microsoft

risponde all'esigenza di semplificazione della gestione di applicazioni.
La scelta delle caratteristiche necessarie a garantire un determinato livello di servizio non solo è estremamente semplice, ma è anche modificabile nel tempo per gestire una potenza superiore o inferiore al cambiare del business o per adattarsi ai momenti di picco.
Le due icone dell'immagine fanno riferimento a due servizi distinti.

1. *devleaprob*, rappresenta un servizio di tipo Hosted Service, consente di gestire, testare e effettuare il deployment dell'applicazione.
2. *devleaprob storage*, rappresenta un servizio di tipo Storage Account, la definizione di uno spazio a disposizione per memorizzare dati dell'applicazione.

Per installare un'applicazione è sufficiente premere il pulsante *Deploy* e eseguire l'upload sul portale di due file che Visual Studio crea in automatico a partire da un progetto web.

1. Il primo file con estensione CSPKG rappresenta il package dell'applicazione, ovvero il codice compilato dell'applicazione.
2. Il secondo file rappresenta il modello che descrive le necessità dell'applicazione.

```
<?xml version="1.0"?>
<ServiceConfiguration
serviceName="DevLeap.AzureSample"
xmlns="http://schemas.microsoft.com/
ServiceHosting/2008/10/ServiceConfiguration">
<Role name="WebRole1">
<Instances count="1" />
<ConfigurationSettings />
</Role>
</ServiceConfiguration>
```

Il file che rappresenta il modello è considerato il file di configurazione del progetto e contiene la definizione del *WebRole1*, all'interno del TAG *WebRole1* si trova l'elemento *Instances* il cui attributo *count* è impostato a 1.
Con questa sintassi s'indica a Windows Azure che si vuole una sola istanza del servizio: Windows Azure segue le istruzioni e durante un deploy installerà l'applicazione in un solo nodo del Data Center.
In qualunque momento si può premere il pulsante *Configure* sul portale e modificare quest'impostazione per far sì che il sistema operativo distribuisca il carico su tanti nodi quante sono le *Instance* configurate.
Una volta premuto *Run* l'applicazione sarà raggiungibile al suo URL.
In pratica, il programmatore "deve" concentrarsi sul codice dell'applicazione, quindi scriverlo il meglio possibile e indicare a Windows Azure le necessità.
Maggiori sono le informazioni fornite nel modello applicativo, migliore sarà il supporto che il sistema operativo fornirà per gestire le politiche di ripartizione del carico, i costi sono proporzionali al modello di configurazione scelto.
L'onere della gestione fisica delle macchine, della manutenzione e degli strumenti di amministrazione è compito di chi offre la piattaforma.
Non occorre installare patch o configurare fisicamente le macchine e i servizi; è sufficiente descrivere "i bisogni" nel modello per configurare le applicazioni e non è necessario conoscere dove e neanche come sono memorizzate le informazioni o gestite le risorse.
Per esempio, in .NET non è necessario sapere come lavora il GC, l'importante è rilasciare correttamente gli oggetti in modo che il GC possa fare egregiamente il suo lavoro; non è necessario suggerire al GC quando intervenire.

CLOUD PRINT

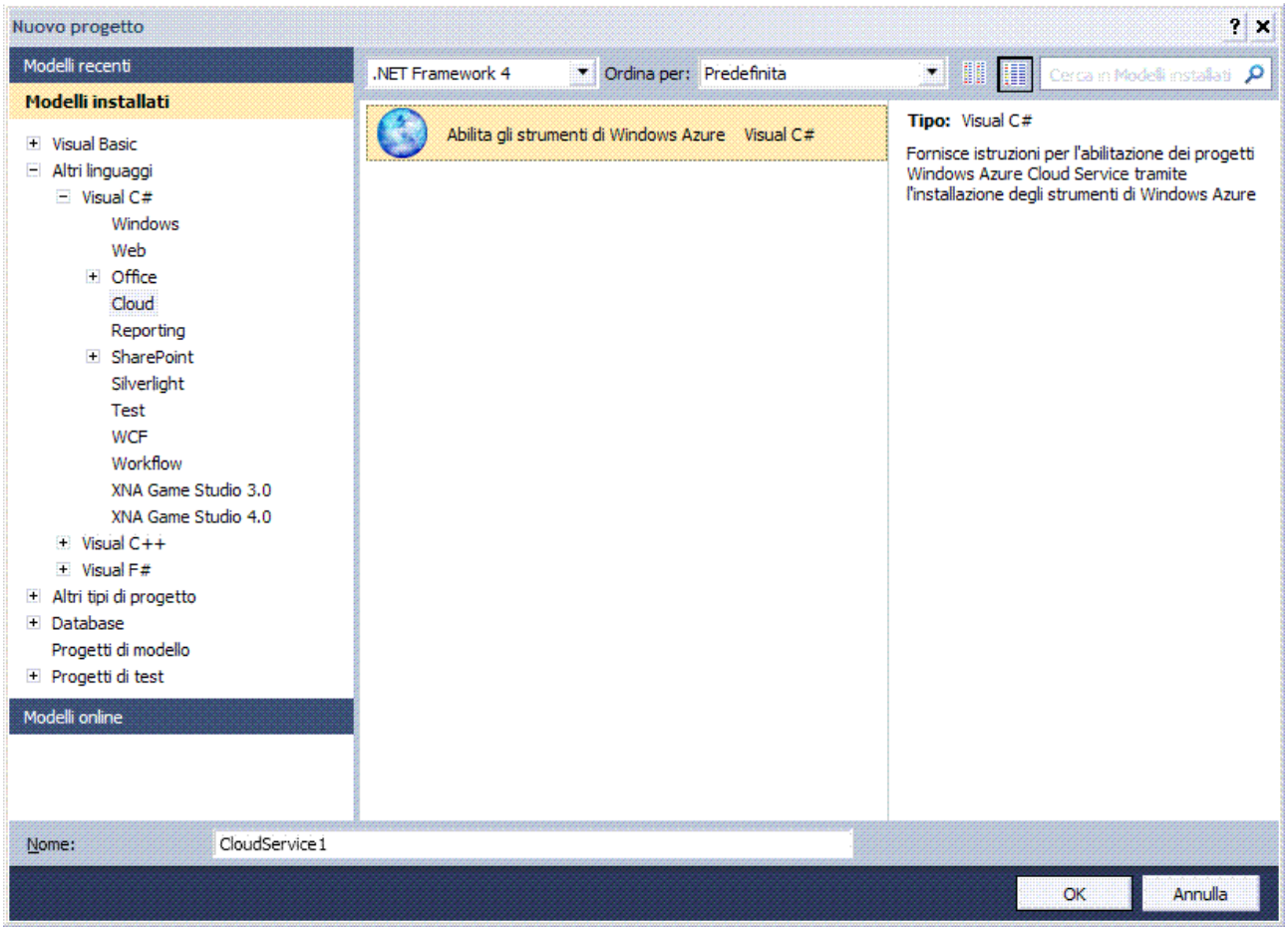
È una tecnologia concepita per le applicazioni web-based e mobili, stampare da qualsiasi dispositivo e applicazione su qualsiasi stampante, senza l'uso di alcun driver locale.

Utilizzando un componente che tutti i principali dispositivi e sistemi operativi hanno in comune, l'accesso alla cloud, il servizio permette a qualsiasi applicazione web, desktop o mobile su qualsiasi dispositivo di stampare su qualsiasi stampante.
Anziché affidarsi al driver e all'interfaccia di stampa del sistema operativo sottostante, con Cloud Print le applicazioni possono inviare le richieste di stampa a un proxy remoto insieme ad alcune informazioni sulla stampante locale: il server utilizza queste informazioni per tradurre opportunamente le richieste e inoltrarle direttamente alla stampante attiva.

Il sistema, che si premura altresì d’informare costantemente l'applicazione sullo stato della stampa, funziona con tutte le stampanti collegate all’USB o direttamente alla rete.
Il grande vantaggio fornito dalla stampa via cloud è che si bypassa completamente il driver nativo: questo consente di stampare anche dai dispositivi e dai sistemi operativi per i quali non sono disponibili i driver o che non supportano le funzioni di stampa.
Il principio di funzionamento di Cloud Print è basato su stampanti **cloud-aware**, ossia capaci di comunicare direttamente con la cloud senza l'intermediazione di un PC o di un router: tali stampanti permetteranno, ad esempio, di stampare direttamente da uno smartphone anche quando il PC è spento.

SVILUPPO

Si può gestire lo sviluppo dell'applicazione per Windows Azure in Visual Studio 2010.
Per creare un nuovo progetto, fare clic su ***File/Nuovo/Progetto… (CTRL+N)***



Nella finestra di dialogo ***Nuovo Progetto*** selezionare un linguaggio a scelta dall'elenco, per esempio ***Visual C#***, quindi selezionare ***Cloud.***
Immettere il nome del progetto nel campo ***Nome:***, per esempio ***CloudService1***, quindi per creare il progetto, premere ***OK***.
Apparirà una pagina che permette di scaricare i tool e i modelli per la gestione di un progetto per Windows Azure.

Dopo avere scaricato e installato il pacchetto, operazione che si deve fare solamente la prima volta che si sceglie questo tipo di progetto, si può procedere alla creazione di un

nuovo progetto, selezionando il nuovo template presente nella categoria ***Cloud***.
Nella finestra di dialogo che appare, inserire nella soluzione un template di tipo ***ASP.NET Web Role*** e tutti gli altri template di cui si ha bisogno.
Ecco quindi che nella soluzione si vedono inseriti sia un progetto di tipo WebRole sostanzialmente identico ad un'applicazione ASP.NET e un progetto di tipo CloudService che, una volta compilata la soluzione, fornisce i file con estensione CSCFG e CSPKG che devono essere caricati in un'istanza di Windows Azure.
Dopo tale caricamento, si può accedere all'applicazione nello stesso modo in cui si accede ad altre applicazioni web.
Durante lo sviluppo dell'applicazione per Windows Azure Visual Studio 2010 deve essere avviato con i privilegi di amministratore e devono essere avviati i servizi di un'istanza SQL Server sulla macchina di sviluppo.
In caso contrario, nel primo caso si ottiene un messaggio che richiede il riavvio di Visual Studio 2010 con privilegi elevati, mentre nel secondo caso si ottiene un errore per la mancanza di un’istanza di SQL Server disponibile.

UBERTINI MASSIMO
http://www.ubertini.it
massimo@ubertini.it

Dip. Informatica Industriale
I.T.I.S. "Giacomo Fauser"
Via Ricci, 14
28100 Novara Italy
tel. +39 0321482411
fax +39 0321482444
http://www.fauser.edu
massimo@fauser.edu