

# File system

## Funzionalità di base per la gestione di file e cartelle

Ambiente: .NET 2.0+/C# 2.0+

Anno 2013/2014

## Indice generale

<b>1</b>	<b>Concetti generali: i file.....</b>	<b>4</b>
1.1	Dispositivi fisici per la memorizzazione dei file.....	4
1.2	File system (FS).....	4
<b>2</b>	<b>Organizzazione logica del file system.....</b>	<b>5</b>
2.1	Simbologia e nomenclatura del file system.....	5
2.2	Percorsi ( <i>path</i> ).....	6
2.2.1	Percorsi assoluti.....	6
2.2.2	Percorsi relativi.....	6
2.2.3	Cartella corrente.....	7
2.2.4	Unità corrente.....	7
2.2.5	Simboli <i>speciali</i> : “.” e “..”.....	7
2.2.6	Percorsi UNC (Universal Naming Convention).....	7
2.3	Uso di stringhe verbatim per specificare i percorsi in C#.....	8
2.4	Informazioni che caratterizzano file e directory.....	8
<b>3</b>	<b>Classi di accesso al file system.....</b>	<b>9</b>
3.1	Classe File.....	9
3.2	Classe Directory.....	10
3.2.1	Pattern di ricerca e caratteri jolly.....	10
3.3	Classe Path.....	11
3.4	Percorsi standard: metodo GetFolderPath().....	11
3.5	Cartella corrente dei programmi eseguiti in Visual Studio.....	12
3.5.1	Percorso relativo dei file memorizzati nel progetto.....	13
3.5.2	Modificare la cartella corrente.....	14
3.5.3	Copiare i file nella cartella di output (Debug).....	14
<b>4</b>	<b>Esempio applicativo di accesso al File system.....</b>	<b>16</b>
4.1	Scenario di esempio.....	16
4.2	Procedimento risolutivo.....	16
4.2.1	Preparazione del progetto.....	16
4.2.2	Versione preliminare del codice.....	17
4.2.3	Verifica del tipo di file.....	17
4.2.4	Copia del file.....	17
<b>5</b>	<b>Classi che rappresentano oggetti del file system.....</b>	<b>18</b>
5.1	Accedere da una cartella: DirectoryInfo.....	18
5.1.1	Accesso ad una cartella inesistente.....	18

5.2	Ottenere l'elenco di file e cartelle.....	19
5.2.1	Ottenere un unico elenco di file e cartelle.....	19
5.3	Accesso agli “attributi” di file e cartelle.....	20
5.3.1	Proprietà “Attributes” .....	20

# 1 Concetti generali: i file

---

Un file può essere definito come un:

***un contenitore di informazioni di natura persistente.***

Il termine ***persistente*** si contrappone al termine ***volatile***, che caratterizza i dati presenti nella memoria centrale (RAM), la cui esistenza non supera lo spegnimento del computer.

I file risiedono su ***supporti permanenti***, (*memorie di massa*), i quali garantiscono la persistenza dei dati.

## 1.1 Dispositivi fisici per la memorizzazione dei file

Le memorie di massa più comuni sono:

- dischi magnetici: *hard-disk*;
- *memorie solide*: SSD, pen-drive, flash card;
- dischi ottici: CD-ROM R, CD-ROM RW, DVD, BLU-RAY;
- nastri (*tapes*);

La modalità di persistenza dei dati dipende dalla tecnologia impiegata e dal materiale che caratterizza ogni supporto.

La memorizzazione viene eseguita dal ***controllore del dispositivo*** (*device controller*), l'hardware che comanda la parte meccanica e/o elettronica. Questo è guidato dal ***driver del dispositivo*** (*device driver*), il software progettato per gestire il dispositivo.

Il sistema operativo si interfaccia con i *device driver* e traduce la struttura a basso livello dei dati nel ***file system***: una struttura facilmente accessibile sia per il programmatore che per l'utente finale.

## 1.2 File system (FS)

Il termine *file system* ha vari significati:

- 1 la parte del sistema operativo che gestisce la memorizzazione dei file;
- 2 la modalità di organizzazione fisica dei file nei vari dispositivi;
- 3 l'insieme dei file memorizzati, la loro organizzazione logica, le informazioni accessorie che caratterizzano ogni file;
- 4 l'interfaccia applicativa (API) utilizzata per operare sui file.

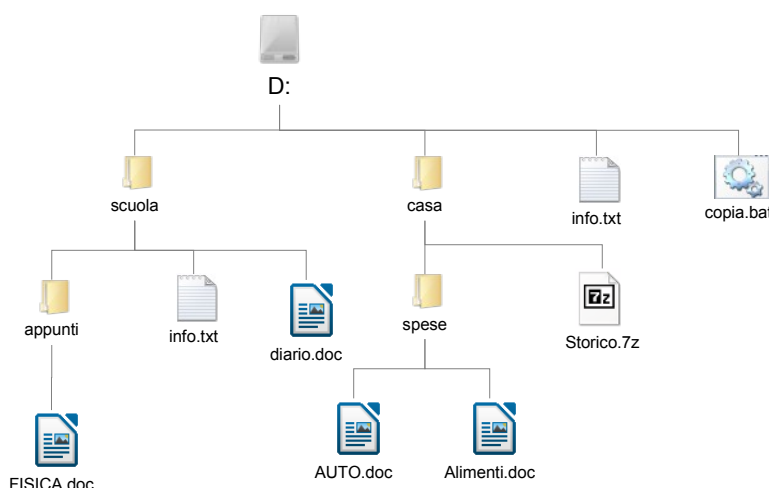
I primi due punti riguardano lo studio dei sistemi operativi. Gli altri due sono importanti sia per il programmatore che per l'utente finale.

## 2 Organizzazione logica del file system

L'organizzazione logica di un FS riguarda la collocazione dei file così come la vedono il programmatore e l'utente finale. Il termine "logica" si contrappone a "fisica", poiché non è importante come siano realmente memorizzati i file nel dispositivo.

I FS sono normalmente organizzati in una struttura chiamata **albero delle directory**. L'albero ha una **radice (root)**, dalla quale dipartono dei rami, dai quali possono dipartire altri rami e così via. In questa metafora, i file sono le foglie; le cartelle (*directory*) sono i rami che contengono le foglie e/o altri rami.

La *radice* è rappresentata dal dispositivo.



### 2.1 Simbologia e nomenclatura del file system

L'organizzazione logica dei FS è soggetta a delle regole e ad una simbologia. In Windows:

- i dispositivi vengono definiti **unità** e sono indicati con una lettera seguita dal carattere ":" (A: B: C:). Le prime due lettere sono riservate ai floppy disk, le lettere successive agli hard disk, ai dischi ottici, ai dischi removibili, etc.
- i nomi di file e cartelle possono contenere lettere, numeri, spazi e altri caratteri come `- + _`. I seguenti caratteri non sono ammessi: `/ \ * : ? " < > |`.
- Un nome può terminare con un'**estensione**, una sequenza di caratteri preceduta dal punto. Se si tratta di un file, l'estensione viene utilizzata dal SO per associarlo ad una determinata applicazione. (docx → Word; odt → Writer, xls → Excel, etc)
- Nei nomi non viene fatta alcuna distinzione tra lettere minuscole e maiuscole.
- File e cartelle sono caratterizzate da un **percorso (path)**: l'insieme delle cartelle che bisogna "attraversare" per raggiungere la cartella che li contiene.
- Il **nome completo (fullpath o fullname)** di un oggetto è rappresentato dal percorso più il nome.
- Il **nome semplice (o short name)** è dato dal nome privo di percorso.

- Due oggetti distinti non possono avere lo stesso nome completo. In altre parole: nella stessa cartella non possono esistere due file o due cartelle con lo stesso nome.

### Dispositivi virtuali, volumi e partizioni

Le cose sono più complicate:

- Le unità possono essere virtuali e non fare riferimento ad alcun dispositivo fisico.
- Un singolo dispositivo può essere partizionato in più unità.
- È possibile che un'unità sia “spalmata” su più dispositivi.

## 2.2 Percorsi (*path*)

Un percorso è rappresentato da una sequenza di nomi di cartella separati dal carattere `\` (o `/`). Opzionalmente, il percorso può indicare l'unità al quale si riferisce. Ad esempio:

```
D:\scuola\appunti
```

### 2.2.1 Percorsi assoluti

Un percorso si dice **assoluto** se inizia con il simbolo di unità seguito dal carattere `\`, oppure direttamente con `\`. L'esempio precedente è un percorso assoluto, che può essere riscritto come:

```
\scuola\appunti
```

Un percorso assoluto comprensivo di nome identifica univocamente un oggetto del FS. In altre parole: due percorsi assoluti uguali individuano sempre lo stesso oggetto (purché si riferiscano alla stessa unità).

### 2.2.2 Percorsi relativi

Un percorso si dice **relativo** se comincia direttamente con un nome di cartella o di file. Ad esempio:

```
casa\spese\auto.doc
```

Un *percorso relativo*, come suggerisce il nome, è sempre relativo ad un determinato punto di partenza, che può essere una cartella o l'unità.

Ad esempio, prendendo come punto di partenza la cartella “scuola”, il percorso del file “FISICA.doc” è:

```
appunti\FISICA.doc
```

### 2.2.3 Cartella corrente

Durante l'esecuzione del programma, i percorsi relativi sono relativi alla **cartella corrente**.

La *cartella corrente* può cambiare sia attraverso l'esecuzione di comandi espliciti, sia come effetto dell'esecuzione di alcune funzioni del SO.

### 2.2.4 Unità corrente

La *cartella corrente* esiste sempre nell'ambito dell'**unità corrente**.

Ad esempio, considera i seguenti percorsi, tutti relativi:

```
D:info.txt    C:auto.doc    appunti\FISICA.doc
```

Il primo referencia il file "info.txt" contenuto nella *cartella corrente* dell'unità D.

Discorso analogo vale per "C:auto.doc".

Il terzo si riferisce al file "FISICA.doc" contenuto nella cartella "Appunti", contenuta nella *cartella corrente* dell'*unità corrente*.

### 2.2.5 Simboli speciali: "." e ".."

I simboli speciali `.` e `..` sono degli alias per la *cartella corrente* e la *cartella padre* della *cartella corrente*<sup>1</sup>.

Supponiamo che "spese" sia la cartella corrente e che si desideri accedere al file "copia.bat". Usando un percorso assoluto non si può sbagliare:

```
D:\copia.bat
```

Ma è possibile utilizzare anche un percorso relativo (a "spese"):

```
..\..\copia.bat
```

Il SO interpreta il precedente percorso nel seguente modo:

`..` → *cartella padre di spese: casa*.

`..` → *cartella padre di casa: unità D*:

### 2.2.6 Percorsi UNC (Universal Naming Convention)

I percorsi UNC (*Universal Naming Convention*) referenziano gli oggetti in un computer remoto. Hanno la seguente struttura:

```
\\ computer \ condivisione \ percorso
```

Supponiamo che l'unità D: dell'esempio precedente si trovi in un server di nome "trixie" e che essa sia condivisa con il nome "Didattica"; ecco il percorso per accedere al file "info.txt" della cartella "scuola":

```
\\trixie\Didattica\scuola\info.txt
```

<sup>1</sup> La definizione non è esatta, anche se possiamo ritenerla corretta all'interno del discorso che stiamo facendo.

## 2.3 Uso di stringhe verbatim per specificare i percorsi in C#

In C# il carattere `\` usato all'interno di costanti stringa viene interpretato come l'inizio di una **sequenza di escape** (come `\n` ad esempio). Per usarlo in un percorso è dunque opportuno usare il prefisso `@`.

Ad esempio:

```
string path = @"D:\scuola";
```

L'uso di `@` identifica la stringa come *verbatim* (alla lettera); questa non viene elaborata alla ricerca di sequenze di escape.

## 2.4 Informazioni che caratterizzano file e directory

Gli oggetti memorizzati nel FS sono caratterizzati da informazioni che non sono collegate al loro contenuto:

- occupazione di memoria;
- data di creazione;
- data ultima modifica e data ultimo accesso;
- attributi: *nascondo* e *sola lettura*.

Alcuni FS memorizzano ulteriori dati, come i permessi di accesso all'oggetto ("chi" può fare "cosa") e alcune informazioni di riepilogo.

Queste informazioni prescindono dalla natura dell'oggetto e sono accessibili (fatta salva l'appropriata autorizzazione) sia da parte del programmatore che dell'utente finale.



## 3 Classi di accesso al file system

Il namespace **System.IO** definisce quattro classi per l'accesso agli oggetti del FS: **File**, **Directory**, **FileInfo**, **DirectoryInfo**.

Le prime due sono classi statiche, mentre le altre consentono di creare oggetti che si interfacciano con singole cartelle e file. Qui sono trattate soltanto le classi **File** e **Directory**.

**System.IO** definisce inoltre la classe statica **Path**. Questa fornisce metodi utili per manipolare i percorsi.

### 3.1 Classe File

La classe **File** definisce metodi che consentono di manipolare i file a prescindere dal loro contenuto. (Il contenuto del file non viene elaborato.)

Il seguente codice elabora un elenco di file, copiando nell'unità D: quelli che cominciano per il carattere *underscore* e cancellando gli altri

```
string[] fileList = {"_voti.txt", "_note.txt", "esercizi.doc"};
string destPath= @"d:\";

foreach (var fileName in fileList)
{
    if (File.Exists(fileName))
    {
        if (fileName.StartsWith("_"))
        {
            string destFullName = destPath + @"\" + fileName;
            File.Copy(fileName, destFullName);
        }
        else
        {
            File.Delete(fileName);
        }
    }
}
```

Alcune osservazioni:

- Il metodo **Exists()** ritorna true se il nome si riferisce ad un file effettivamente esistente.
- Il metodo **Copy()** richiede come secondo argomento il percorso completo del nuovo file; non basta specificare la cartella di destinazione (come invece accade per il comando "COPY" della *dos shell*.)

Nel codice sono utilizzati nomi relativi e dunque si suppone che i file siano memorizzati nella *cartella corrente*. (Vedi "*Cartella corrente dei programmi eseguiti in Visual Studio*")

## 3.2 Classe Directory

La classe **Directory** consente di elaborare le cartelle. È omologa alla classe File, ma non definisce un metodo di copia.

Tra i metodi più utili vi sono quelli che ritornano i nomi dei file e delle cartelle contenute in una cartella: **GetFiles()** e **GetDirectories()**.

L'esempio seguente ottiene i nomi dei file contenuti nella cartella "D:\Immagini":

```
string sourcePath = @"d:\immagini";  
string[] fileList = Directory.GetFiles(sourcePath);  
  
foreach (var fileName in fileList)  
{  
    Console.WriteLine(fileName);  
}
```

Ecco l'output:

```
Output  
d:\immagini\Lince nella neve.jpg  
d:\immagini\Cuccioli lince.jpg  
d:\immagini\hs-2013-18-a-full_tif.tif  
d:\immagini\Leopardo nevi_cucciolo.jpg
```

Nota bene: vengono ritornati i nomi completi, comprensivi di estensione.

### 3.2.1 Pattern di ricerca e caratteri jolly

I metodi **GetFiles()** e **GetDirectories()** accettano come secondo argomento una stringa che consente di restringere la ricerca a quei nomi che rispecchiano un determinato **pattern di ricerca**.

Di norma il pattern contiene uno o più caratteri jolly: **\*** e **?**.

Carattere	Descrizione
*	Qualsiasi sequenza di zero o più caratteri. Ad esempio: <b>*t</b> corrisponde a qualsiasi stringa che termina con la lettera "t". <b>s*</b> corrisponde a qualsiasi stringa che inizia con la lettera "s".
?	Un carattere qualsiasi. Ad esempio: <b>??t</b> corrisponde a qualsiasi stringa di 3 caratteri che termina con la "t". <b>mat???0</b> corrisponde a qualsiasi stringa di 6 caratteri che inizia per "mat" e termina con "0".

Usando il pattern di ricerca è possibile riscrivere il metodo precedente allo scopo di visualizzare soltanto i nomi delle immagini JPEG:

```
string sourcePath = @"d:\immagini";
string[] fileList = Directory.GetFiles(sourcePath, "*.jpg");

foreach (var fileName in fileList)
{
    Console.WriteLine(fileName);
}
```

### 3.3 Classe Path

I metodi della classe **Path** sono utili per:

- concatenare nomi di cartelle e file;
- estrarre il nome semplice da un percorso completo;
- estrarre il nome della cartella da un nome di file completo;
- estrarre l'estensione da un nome di file;
- ottenere il nome completo a partire dal nome relativo. Eccetera

Molti di essi si limitano a elaborare gli argomenti senza verificare che facciano riferimento a oggetti realmente esistenti.

L'esempio che segue riprende il codice precedente con l'obiettivo di visualizzare il nome semplice di ogni file.

```
string sourcePath = @"d:\immagini";
string[] fileList = Directory.GetFiles(sourcePath, "*.jpg");

foreach (var fileName in fileList)
{
    string shortName = Path.GetFileName(fileName);
    Console.WriteLine(shortName);
}
```

#### Output

```
Lince nella neve.jpg
Cuccioli lince.jpg
Leopardo nevi_cucciolo.jpg
```

### 3.4 Percorsi standard: metodo GetFolderPath()

Il SO definisce alcuni percorsi predefiniti, tra i quali:

- la cartella contenente il profilo dell'utente;
- all'interno della precedente, le cartelle **Documenti** e **Desktop**;
- la cartella contenente i programmi;
- la cartella contenente i file temporanei. Eccetera.

Questi percorsi variano da sistema a sistema, in base alla versione del SO, all'installazione

effettuata e all'utente collegato. Per accedervi è possibile usare il metodo **GetFolderPath()** della classe **Environment**.

Il metodo accetta un valore dell'enum **SpecialFolder** e ritorna il percorso della cartella specificata.

Ecco alcune costanti dell'enum **SpecialFolder**:

Costante	Descrizione
<b>Personal</b> (MyDocuments)	Percorso cartella Documenti.
<b>Desktop</b>	Percorso cartella Desktop
<b>Windows</b>	Percorso cartella d'installazione del SO.

Il seguente codice dimostra l'uso del metodo:

```
string personal = Environment.GetFolderPath(Environment.SpecialFolder.Personal);  
string desktop = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);  
string winPath = Environment.GetFolderPath(Environment.SpecialFolder.Windows);  
  
Console.WriteLine(personal);  
Console.WriteLine(desktop);  
Console.WriteLine(winPath);
```

#### Output

```
F:\Documenti - Paolo  
C:\Users\paolouser\Desktop  
C:\Windows
```

(Nota bene: nel mio sistema, il percorso di “Documenti” non è quello standard; questo dimostra l'importanza di avere un metodo che ritorni sempre il percorso corretto.)

## 3.5 Cartella corrente dei programmi eseguiti in Visual Studio

La *cartella corrente* di un programma è la cartella contenente il file eseguibile. I programmi eseguiti all'interno di Visual Studio vengono memorizzati nella cartella<sup>2</sup>:

```
<solution>\<progetto>\bin\debug
```

Dunque, se si desidera accedere ad un file specificando soltanto il nome è necessario che sia memorizzato nella cartella suddetta.

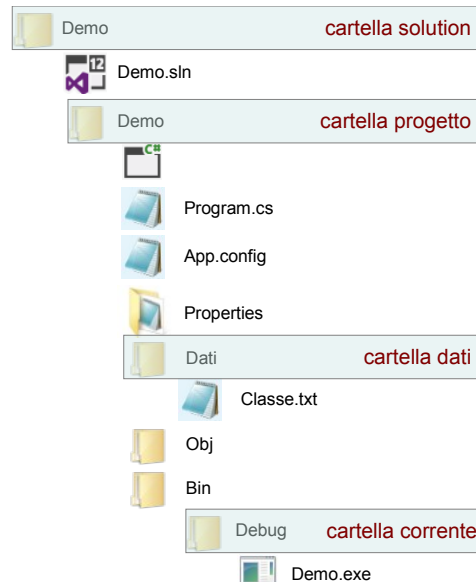
<sup>2</sup> Oppure nella cartella “...\bin\release”.

### 3.5.1 Percorso relativo dei file memorizzati nel progetto

Quando si sviluppa un programma può essere comodo memorizzare i file da elaborare direttamente nella cartella del progetto, in modo che possano “viaggiare” con esso.

Ipotizziamo di realizzare un programma che debba elaborare un file di testo di nome “Classe.txt”. Durante lo sviluppo potremmo collocare il file nella cartella di progetto, in una sotto cartella di nome “Dati”.

Segue l'albero delle cartelle della solution<sup>3</sup>:



Per accedere al file “Classe.txt” non basta usare il suo nome, poiché in questo caso Windows lo cercherebbe nella cartella Debug.

Si può utilizzare un percorso relativo rispetto a Debug:

```
..\..\Dati\Classe.txt
```

Che significa, partendo dalla cartella corrente Debug:

`..\` → cartella padre di **Debug**: **Bin**.

`..\` → cartella padre di **Bin**: **Demo** (cartella del progetto), etc

Il seguente codice copia il file in “D:\Backup”:

```
string sourcePath = @"..\..\Dati";
string fileName = "Classe.txt";
string destPath = @"D:\Backup";

string sourceFullPath = Path.Combine(sourcePath, fileName);
string destFullPath = Path.Combine(destPath, fileName);

File.Copy(sourceFullPath, destFullPath);
```

<sup>3</sup> Sono stati omessi alcuni file e cartelle.

Nota bene: per concatenare nomi di cartella con nomi di file viene usato il metodo **Path.Combine()**.

### 3.5.2 Modificare la cartella corrente

Esiste la possibilità di modificare da programma la cartella corrente, utilizzando il metodo **SetCurrentDirectory()**.

Il seguente codice rende "Dati" la cartella corrente. Dopodiché l'accesso al file Classe.txt può essere fatto utilizzando il nome corto del file:

```
string sourcePath = @"..\..\Dati";  
string fileName = "Classe.txt";  
string destPath = @"D:\Backup";  
  
Directory.SetCurrentDirectory(sourcePath);  
  
string destFullPath = Path.Combine(destPath, fileName);  
File.Copy(fileName, destFullPath);
```

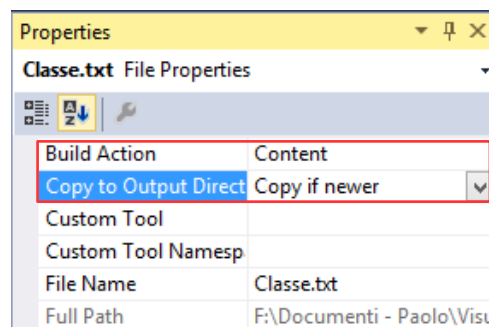
### 3.5.3 Copiare i file nella cartella di output (Debug)

Visual Studio fornisce una funzionalità che evita l'uso di percorsi complicati per accedere a file del progetto, oppure di dover modificare la cartella corrente.

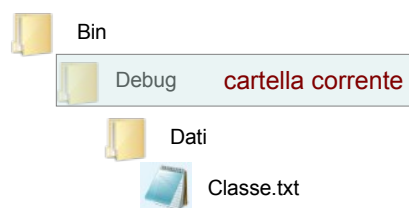
É possibile copiare automaticamente un file del progetto nella *cartella di output* (Debug) quando il programma viene compilato ed eseguito.

Per attivare questa funzionalità: fare clic destro sul file e selezionare **Properties** dal menù contestuale. Nella finestra Properties e:

- impostare **Build Action** su **Content** (lo è di default)
- impostare **Copy to Output Directory** su **Copy if newer** (oppure su **Copy always**)



Prima dell'esecuzione del programma viene prodotta la seguente situazione:



Nota bene:

- In Debug viene copiata anche la cartella Dati; viene cioè mantenuto il percorso di Classe.txt rispetto alla cartella del progetto.
- Esistono due file Classe.txt; l'originale non viene influenzato da questa operazione.

Dopo questa modifica si può accedere al file Classe.txt con il percorso:

```
Dati\Classe.txt
```

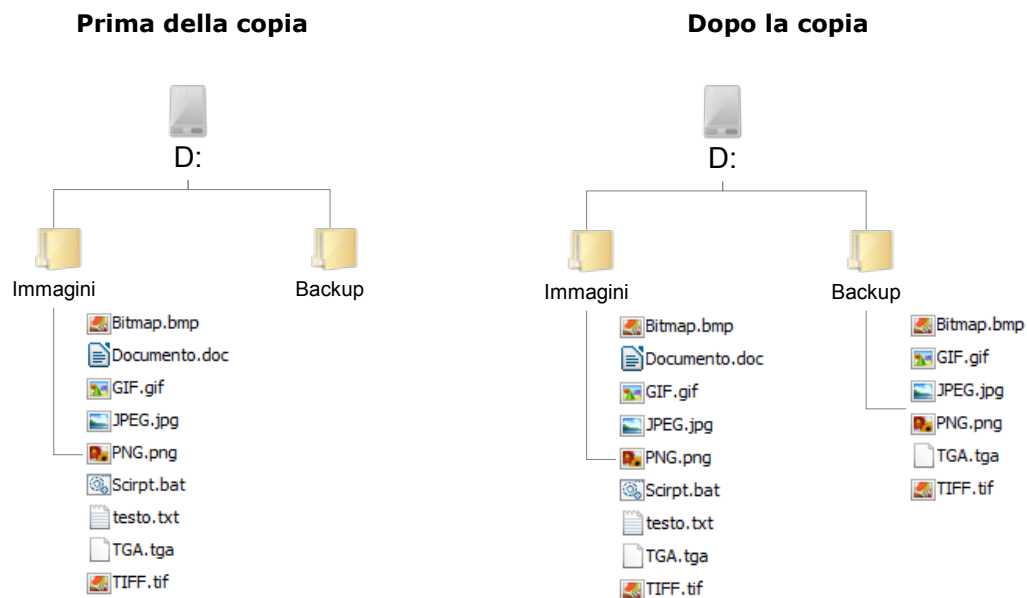
## 4 Esempio applicativo di accesso al File system

Realizzare un programma che, data una *cartella sorgente* e una *cartella destinazione*, copi tutti i file immagine dalla prima alla seconda.

I file immagine possono avere le seguenti estensioni: jpg, png, bmp, gif, tif.

### 4.1 Scenario di esempio

Si ipotizzi che la *cartella sorgente* sia “D:\immagini” e la *cartella destinazione* “D:\backup”. La cartella “Immagini” contiene vari tipi di file. Ecco ciò che vogliamo ottenere:



### 4.2 Procedimento risolutivo

Segue una pseudo codifica del procedimento:

```
crea la cartella destinazione se non esiste
carica l'elenco dei file della cartella sorgente
per ogni file
    se è un file immagine
        copia il file nella cartella destinazione
```

#### 4.2.1 Preparazione del progetto

Dentro la cartella di progetto creo le sottocartelle “Immagini” e “Backup”. Nella prima copio alcuni file, sia immagini che di altro tipo.

Definisco due costanti stringa che puntano alle cartelle:

```
const string sourcePath = @"..\..\Immagini";
const string destPath = @"..\..\Backup";
```



### 4.2.2 Versione preliminare del codice

Il codice seguente rappresenta una prima implementazione del procedimento.

```
if (Directory.Exists(destPath) == false)
    Directory.CreateDirectory(destPath);

string[] fileList = Directory.GetFiles(sourcePath);
foreach (string fileName in fileList)
{
    if (IsImageFile(fileName) == true)
        CopyFile(fileName, destPath);
}
```

### 4.2.3 Verifica del tipo di file

Per verificare se un file è un'immagine occorre ottenere la sua estensione e confrontarla con le estensioni corrispondenti alle immagini.

```
static string[] imagesExtList= { ".png", ".bmp", ".gif", ".jpg", ".tif" };

static bool IsImageFile(string filePath)
{
    string ext = Path.GetExtension(filePath);
    int index = Array.IndexOf(imagesExtList, ext.ToLower());
    return index > -1;
}
```

Nota bene: il metodo **GetExtension()** ritorna l'estensione comprensiva del punto.

### 4.2.4 Copia del file

La copia del file prevede innanzitutto la costruzione del nome completo di destinazione. Questo è dato dal percorso di destinazione più il nome semplice del file:

```
static void CopyFile(string sourceFileName, string destPath)
{
    string shortName = Path.GetFileName(sourceFileName);
    string destFullName = Path.Combine(destPath, shortName);
    File.Copy(sourceFileName, destFullName, true);
}
```

Nota bene: l'argomento "true" passato al metodo **Copy()** stabilisce che se la cartella di destinazione contiene un file già con lo stesso nome, questo viene sovrascritto dal nuovo file.

## 5 Classi che rappresentano oggetti del file system

I metodi delle classi **Directory** e **File** richiedono il percorso dell'oggetto (cartella o file) da elaborare. Ma esistono scenari nei quali si desidera accedere direttamente agli oggetti del file system: a questo scopo esistono le classi **FileSystemInfo**, **FileInfo** e **DirectoryInfo**.

La prima classe stabilisce le proprietà comuni ad ogni oggetto del file system. Le altre due sono specializzate per elaborare file e cartelle.

Di seguito sarà esaminata prevalentemente la classe **DirectoryInfo**; le classi **FileInfo** e **FileSystemInfo** sono del tutto analoghe.

### 5.1 Accedere da una cartella: DirectoryInfo

Un oggetto di tipo **DirectoryInfo** fornisce l'accesso a una cartella; il percorso della cartella viene specificato nel costruttore. Dopo la creazione dell'oggetto, ogni operazione eseguita su di esso si riflette sulla cartella corrispondente.

Il seguente codice accede alla cartella corrente e ne visualizza alcune proprietà:

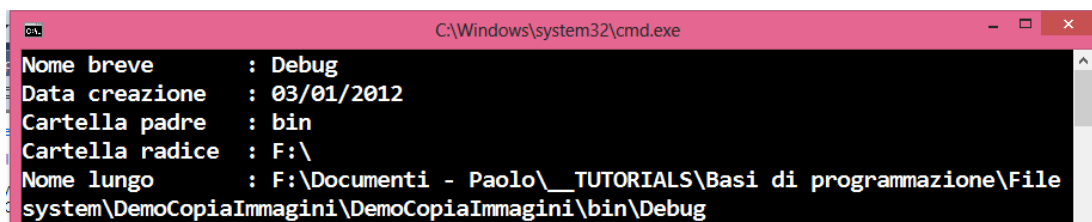
```
var di = new DirectoryInfo(".");           // "." -> cartella corrente
Console.WriteLine("Nome breve      : {0}", di.Name);
Console.WriteLine("Data creazione  : {0:d}", di.CreationTime);

var parent = di.Parent;                   // Parent è sempre di tipo DirectoryInfo
Console.WriteLine("Cartella padre   : {0}", parent.Name);

var root = di.Root;                       // Root è sempre di tipo DirectoryInfo
Console.WriteLine("Cartella radice  : {0}", root.Name);

Console.WriteLine("Nome lungo      : {0}", di.FullName);
```

Ecco l'output:



```
C:\Windows\system32\cmd.exe
Nome breve      : Debug
Data creazione  : 03/01/2012
Cartella padre   : bin
Cartella radice  : F:\
Nome lungo      : F:\Documenti - Paolo\__TUTORIALS\Basi di programmazione\File
system\DemoCopiaImmagini\DemoCopiaImmagini\bin\Debug
```

#### 5.1.1 Accesso ad una cartella inesistente

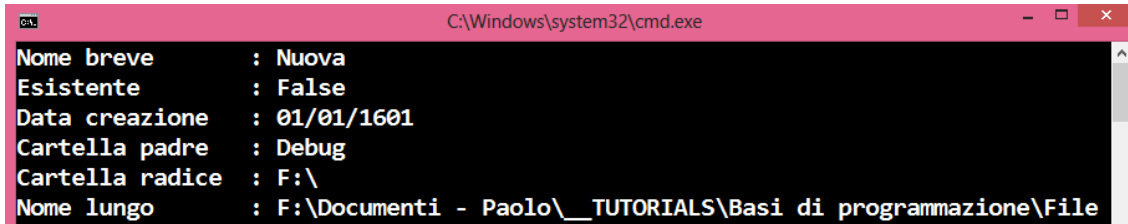
Un oggetto **DirectoryInfo** può accedere anche ad una cartella inesistente. (Di solito questo lo si fa allo scopo di crearla.)

Anche in questo caso, tutte le proprietà dell'oggetto restituiscono un valore valido. (**CreationTime** e le altre proprietà **DateTime** restituiscono "1/1/1601".)

La proprietà **Exists** restituisce vero se la cartella esiste, falso altrimenti.

Di seguito viene mostrato l'output del precedente codice, ma specificando una cartella inesistente durante la creazione dell'oggetto:

```
var di = new DirectoryInfo("Nuova");           // "Nuova" -> non esiste!  
Console.WriteLine("Nome breve      : {0}", di.Name);  
Console.WriteLine("Esistente       : {0}", di.Exists); // -> false!  
...
```



```
C:\Windows\system32\cmd.exe  
Nome breve      : Nuova  
Esistente       : False  
Data creazione  : 01/01/1601  
Cartella padre  : Debug  
Cartella radice : F:\  
Nome lungo      : F:\Documenti - Paolo\__TUTORIALS\Basi di programmazione\File
```

Nota bene: a parte **CreationTime**, le altre proprietà restituiscono valori coerenti, come se la cartella esistesse davvero e si trovasse in “Debug”.

## 5.2 Ottenere l'elenco di file e cartelle

**DirectoryInfo** definisce dei metodi che restituiscono l'elenco di file e cartelle contenuti nella cartella corrispondente: **GetFiles()** e **GetDirectories()**. Questi ritornano un vettore di oggetti **FileInfo** e **DirectoryInfo**.

Il seguente codice utilizza **GetFiles()** per visualizzare l'elenco dei file Access presenti nella cartella “Documenti”. Nota bene: il metodo accetta come argomento un pattern di ricerca:

```
string docPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);  
var di = new DirectoryInfo(docPath);  
FileInfo[] listaFile = di.GetFiles("*.mdb"); // "mdb" -> estensione file Access  
foreach (var file in listaFile)  
{  
    Console.WriteLine(file.Name);  
}
```

### 5.2.1 Ottenere un unico elenco di file e cartelle

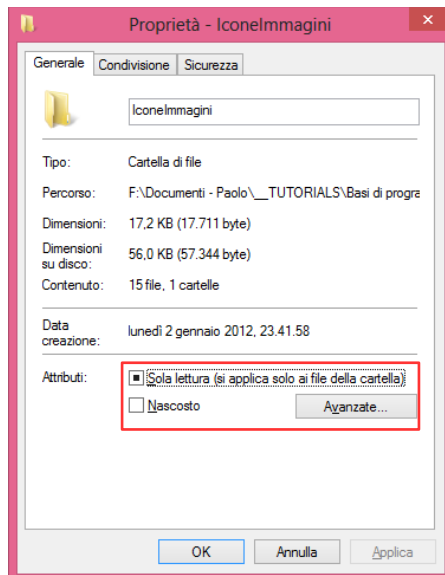
Il metodo **GetFileSystemInfos()** restituisce l'elenco completo di oggetti (file e cartelle) contenuti in una cartella. Il tipo degli oggetti restituiti è **FileSystemInfo** e dunque ogni oggetto può rappresentare un file o una cartella.

```
string docPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);  
var di = new DirectoryInfo(docPath);  
FileSystemInfo[] listaFile = di.GetFileSystemInfos();  
...
```

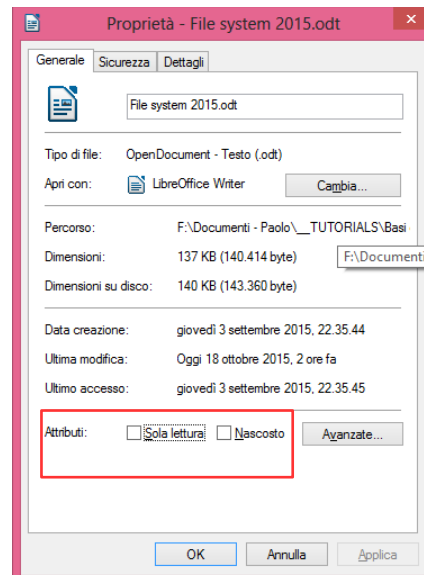
## 5.3 Accesso agli “attributi” di file e cartelle

File e cartelle definiscono degli attributi che riportano alcune informazioni aggiuntive. Alcuni di essi sono visualizzati dal SO quando si seleziona un oggetto e nel menù contestuale si sceglie “Proprietà”:

**Proprietà cartella**



**Proprietà file**



### 5.3.1 Proprietà “Attributes”

Gli attributi sono memorizzati nella proprietà **Attributes** dell'oggetto e sono del tipo enum **FileAttributes**. Segue l'elenco degli attributi più comuni:

Attributo	Descrizione
Compressed	File compresso.
Directory	È una cartella.
Hidden	File/cartella nascosto.
ReadOnly	File a sola lettura.
System	File di sistema.
Temporary	File temporaneo.

**Attributes** è un enum di tipo **bit flags**. Ciò significa che ogni valore è un multiplo di 2; dunque la proprietà **Attributes** può avere più attributi impostati contemporaneamente (esempio: un file può essere sia nascosto che di sola lettura).

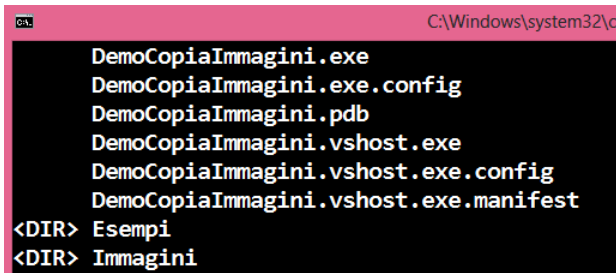
Di seguito viene mostrato come distinguere tra oggetti che rappresentano file o cartelle all'interno di un elenco di **FileSystemInfo**:

```
var di = new DirectoryInfo(".");
FileSystemInfo[] listaFile = di.GetFileSystemInfos();
foreach (var file in listaFile)
```

```
{  
    if (file.Attributes.HasFlag(FileAttributes.Directory))  
        Console.WriteLine("<DIR> {0}", file.Name);  
    else  
        Console.WriteLine("      {0}", file.Name);  
}
```

Il metodo **HasFlag()** restituisce vero se la proprietà ha impostato l'attributo **Directory**.

Presupponendo di aver creato le cartelle "Esempi" e "Immagini" nella cartella corrente (Debug), questo è il risultato:



```
C:\Windows\system32\cmd.exe  
DemoCopiaImmagini.exe  
DemoCopiaImmagini.exe.config  
DemoCopiaImmagini.pdb  
DemoCopiaImmagini.vshost.exe  
DemoCopiaImmagini.vshost.exe.config  
DemoCopiaImmagini.vshost.exe.manifest  
<DIR> Esempi  
<DIR> Immagini
```