Unità Didattica: FILE TESTO

Lavorare con i File

Visual Studio e il suo .NET Framework mette a disposizione diverse classi per la Gestione e la Scrittura e/o Lettura di Dati su **File** *memorizzati permanentemente su Memorie di Massa* quali Hard Disk, Pen Drive, ecc.

Per utilizzare tali classi, è necessario attivare un *NameSpace* (ossia una "libreria" di classi) chiamato **System.IO** (dove "IO" sta per Input/Output).

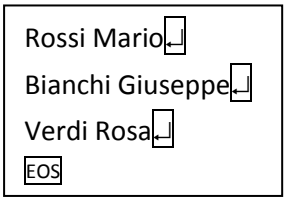
Per **Attivare un NameSpace** si utilizza la direttiva **using** da posizionare, all'inizio del codice.

Ciò significa che, tutte le tue *pagine di codice* (form, classi, ecc.) che utilizzano files su disco dovranno includere la seguente direttiva using:

```
using System.IO
```

File Testo: Struttura, Accesso Sequenziale, Apertura

I **File Testo** sono particolari file costituiti da una **Sequenza di Stringhe** separate da un segnale di **Ritorno a Capo** che indicheremo con il simbolo ↵.



Il "*Ritorno a Capo*" è, in realtà, composto dai 2 caratteri **CR+LF**: *Carriage Return* (codice ASCII = 13) e *Line Feed* (codice ASCII = 20).

I *File Testo* sono file ad **Accesso Sequenziale**, ossia *la loro lettura (o scrittura) avviene sempre a partire dalla prima riga e prosegue necessariamente in "sequenza"* (la prima, poi la seconda, poi la terza, ecc.). Non è possibile accedere "direttamente" a una determinata riga.

Per utilizzare un File è necessario anzitutto effettuare l'operazione di **Apertura del File**: con essa il programma comunica al Sistema Operativo che intende operare sul file e il Sistema Operativo effettua le necessarie operazioni perché questo possa avvenire (verifica l'accesso al file su disco, crea in RAM un "buffer" in cui far transitare i dati, ecc.).

I File Testo possono essere **Aperti in Lettura** o **Aperti in Scrittura**.

Se si **Apre un File Testo in Lettura**, da esso *si possono solo leggere dati* e non è possibile in alcun modo aggiungere, scrivere o modificare i dati presenti nel file.

Se si **Apre un File Testo in Scrittura**, esso *viene "svuotato" per consentire la scrittura di nuovi dati* (o, al limite, è possibile solo *aggiungere dati alla fine del file*) e non è possibile l'operazione di lettura, né la modifica dei dati esistenti.

Lettura di un File Testo e Classe StreamReader

La classe **StreamReader** consente di *Leggere sequenzialmente i dati da un File Testo*.

La *Creazione di un oggetto di classe StreamReader* non avviene, come di consueto, con un normale costruttore. E' necessario, invece, utilizzare la *classe* **File** che, con i suoi metodi statici *apre i files* nelle varie modalità.

Il metodo **File.OpenText (<*nome-file-con-percorso*>)** consente di *Aprire in Lettura il File Testo indicato e restituisce il relativo oggetto StreamReader* che consentirà di accedere ai dati in esso memorizzati.

Per "Aprire in Lettura" un File Testo di nome "Elenco.txt" memorizzato nella cartella "C:\Dati", scriverai così:

```
StreamReader FR = File.OpenText("C:\Dati\Elenco.txt");
```

Il metodo *File.OpenText* accede al file *"C:\Dati\Elenco.txt"*, lo *Apre in Lettura* e ***crea l'oggetto StreamReader*** che, in questo caso, viene *assegnato ad FR*. Come vedi, *non devi scrivere* ***new*** perché è il metodo OpenText che "crea" l'oggetto.

Attenzione: ***se il File indicato non esiste***, viene *generato un errore* durante l'esecuzione.

Una volta *aperto in lettura*, il File si predispone per la *lettura sequenziale* dei suoi dati, posizionando la **Posizione di Lettura** sul primo carattere della prima riga.

Il **metodo ReadLine** (Leggi la Riga) della classe StreamReader, *legge dal file tutti i caratteri dalla Posizione di Lettura fino al primo Carattere di Ritorno a Capo* e restituisce un *valore String* contenente tale sequenza.

```
FR.ReadLine( )      … restituisce una Stringa con la riga letta dal file e "va a capo"
```

Dopo la Lettura della Riga, la *Posizione di Lettura viene automaticamente "spostata" all'inizio della riga successiva* per predisporre la successiva lettura.

☞ Per leggere la *prima riga* dal file "Elenco.txt" memorizzato nella cartella "C:\Dati", scriverai:

```
string RigaLetta;
StreamReader FR = File.OpenText ("C:\Dati\Elenco.txt");
RigaLetta = FR.ReadLine( );
```

Eseguite queste istruzioni, nella variabile string *RigaLetta* è presente il valore *"Rossi Mario"* e la Posizione di Lettura si trova spostata sul primo carattere della seconda riga.

Per leggere le righe successive è sufficiente eseguire più volte il *metodo ReadLine* sull'oggetto F.

La **proprietà EndOfStream** (= Fine del Flusso) della classe StreamReader, assume il *valore TRUE* solo se *la Posizione di Lettura si trova Oltre l'Ultima Riga* del file testo. Assume invece il *valore FALSE* in tutte la altre situazioni.

FR.EndOfStream *… restituisce un valore Boolean (**true** se il file è finito, **false** se ci sono altri dati)*

✋ Verificare il valore della *proprietà EndOfStream*, consente di capire se, durante la lettura sequenziale, il file è terminato o se ci sono ancora altri dati da leggere.

Per **Leggere un intero File Testo** generalmente si ricorre ad un *ciclo* che ripete il *metodo ReadLine* fin quando la *proprietà EndOfStream* non assume il *valore TRUE*.

☞ Ad esempio, per visualizzare con dei MessageBox, *tutte le righe* del file testo "Elenco.txt" memorizzato in "C:\Dati", scrivi:

```
StreamReader FR = File.OpenText ("C:\Dati\Elenco.txt");
while ( ! FR.EndOfStream )
{
    MessageBox ( FR.ReadLine( ) )
}
```

Il *metodo ReadLine* "restituisce" una stringa, quindi può essere usato anche direttamente nel MessageBox: in tal modo si risparmia l'uso della *variabile RigaLetta*, presente nell'esempio precedente.

Al termine delle operazioni di Lettura (o Scrittura) su un File è necessario effettuare l'operazione di **Chiusura del File**. Essa è molto importante perché a seguito di essa, il Sistema Operativo provvede a *completare su disco eventuali operazioni* rimaste pendenti e a *liberare le risorse* destinate a gestire il file.

Il **metodo Close** della classe StreamReader, provoca la *Chiusura del File*.

FR.Close ( ) *… provoca la chiusura del File*

☞ Per chiudere il file dell'esempio precedente ti basta scrivere: **FR.Close ( )**

Scrittura di un File Testo e Classe StreamWriter

La classe **StreamWriter** consente di *Scrivere sequenzialmente dati in un File Testo*.

✋ Come già visto per la classe SteamReader, anche la *Creazione di un oggetto di classe StreamWriter*, avviene utilizzando *l'oggetto* **File** e i suoi metodi specifici per le modalità di scrittura.

Il metodo **File.CreateText (<*nome-file-con-percorso*>)** *crea su disco un nuovo File Testo, lo Apre in Scrittura indicato e restituisce il relativo oggetto StreamWriter* che consentirà di scrivere i dati nel file stesso.

✋ Per "Aprire in Scrittura" un File Testo di nome "Elenco.txt" creandolo nella cartella "C:\Dati", scriverai così:

```
StreamWriter FW = File.CreateText("C:\Dati\Elenco.txt");
```

Il metodo *File.CreateText* crea su disco un file *"Elenco.txt"*, nella cartella *"C:\Dati"*, lo *Apre in Scrittura* e ***crea l'oggetto StreamWriter*** che, in questo caso, viene *assegnato ad FW*.

Nel caso in cui **il File Testo indicato è già Esistente sul disco**, il metodo *File.CreateText lo "riscrive"*, ossia cancella tutti i dati in esso presenti, svuotandolo completamente e predisponendosi per riscrivere i nuovi dati dall'inizio del file.

Il metodo **File.AppendText (<*nome-file-con-percorso*>)**, invece, *apre in Scrittura il File Testo* indicato e *consente di aggiungere nuovi dati alla fine del file* stesso.

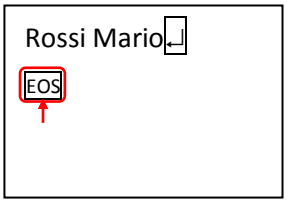
Quindi, attenzione: l'uso di *File.CreateText* "distrugge" i dati già presenti in un file testo! Se vuoi "aggiungere" dati a un File Testo già esistente, puoi farlo solo con *File.AppendText* e puoi *aggiungerli solo alla fine del file*. Devi invece utilizzare *File.CreateText* solo se vuoi creare un *nuovo* File Testo o riscriverne da zero uno già esistente, cancellandone il contenuto.

Una volta *aperto in scrittura*, il File, vuoto, si predispone per la *scrittura sequenziale* di nuovi dati.

In caso di AppendText, il file *non è vuoto* e la *Posizione di Scrittura* viene spostata *oltre l'ultima riga di testo*.

Il **metodo WriteLine** (Scrivi la Riga) della classe StreamWriter, *scrive nel file la stringa* indicata e aggiunge *automaticamente un Carattere di Ritorno a Capo.*

FW.WriteLine (<*stringa*>) *… memorizza nel file la Stringa indicata e "va a capo"*



Dopo la Scrittura della Riga, la *Posizione di Scrittura viene automaticamente "spostata" oltre l'ultima riga (posizione EOS = End Of Stream),* per predisporre la successiva scrittura.

Per scrivere la *stringa "Rossi Mario"* in un *nuovo file testo "Elenco.txt"* da creare *nella cartella "C:\Dati"*, scriverai:

```
StreamReader FW = File.CreateText("C:\Dati\Elenco.txt");
FW.WriteLine ( "Rossi Mario" )
```

Eseguite queste istruzioni, nel file viene memorizzato *"Rossi Mario"* e la Posizione di Scrittura si sposta oltre l'ultima riga (sull'EOS). Per scrivere altre righe è sufficiente eseguire più volte il *metodo WriteIn* sull'oggetto FW.

Per **Scrivere una intera Lista di Dati in un File Testo** generalmente si ricorre ad un *ciclo* che ripete il *metodo WriteLine scrivendo ad ogni passo un dato della lista*, fino alla memorizzazione di tutti i dati.

Ad esempio, per memorizzare in un file testo "Elenco.txt" da creare in "C:\Dati", tutti i dati contenuti in un *vettore ElencoNomi* contenente *N elementi* di tipo string, scrivi:

```
StreamReader FW = File.CreateText("C:\Dati\Elenco.txt");
for (int K = 0; K <= N-1; K++)
{
    FW.WriteLine ( ElencoNomi[K] );
}
FW.Close ( );
```

Ad ogni passo del ciclo, il *metodo WriteLine* scrive, nel file, un diverso dato del vettore, fino completare l'intera memorizzazione dei dati nel file testo. Come vedi, non devi dimenticare di "chiudere" il file, usando il metodo *FW.Close.*